

CanSat

poznámky

2.díl – programování palubního počítače s STM32

Praha 2011

Vladimír Váňa

Úvod

V současné době jsou k dispozici výkonné 32 bitové počítače s jádrem ARM. Cenově jsou srovnatelné s ATmega. Vzhledem k značnému rozšíření ARM se jejich použití stává levnější, než využití ATmega. Např. startkit **STM32VL Discovery** firmy **STMicroelectronics** je levnější, než obdobné startkity *Arduino*. Za méně peněz tak dostaneme výkonnější 32bitový počítač. Vyšší výkon využijeme např. při zpracování obrazu apod. Proto jsme pro palubní počítač zvolili právě STM32 firmy **STMicroelectronics** mající v Praze 8 vývojové laboratoře .

STMicroelectronics Prague

IBC Building, Pobrezni 3

186 00 Prague 8

Czech Republic



| Rank 2010 | Rank 2009 | Company | Country of origin | Revenue (million \$ USD) | 2010/2009 changes | Market share |
|-----------|-----------|-------------------------|--|--------------------------|-------------------|--------------|
| 1 | 1 | Intel Corporation |  USA | 40 394 | +24.3% | 13.2% |
| 2 | 2 | Samsung Electronics |  South Korea | 27 834 | +60.8% | 9.3% |
| 3 | 3 | Toshiba Semiconductors |  Japan | 13 010 | +26.8% | 4.3% |
| 4 | 4 | Texas Instruments |  USA | 12 944 | +34.1% | 4.3% |
| 5 | 9 | Renesas Electronics (1) |  Japan | 11 840 | +129.8% | 3.9% |
| 6 | 7 | Hynix |  South Korea | 10 577 | +69.3% | 3.5% |
| 7 | 5 | STMicroelectronics |  France  Italy | 10 290 | +20.9% | 3.4% |
| 8 | 13 | Micron Technology (2) |  USA | 8 853 | +106.2% | 2.9% |
| 9 | 6 | Qualcomm |  USA | 7 200 | +12.3% | 2.4% |
| 10 | 14 | Broadcom |  USA | 6 506 | +52.1% | 2.1% |
| 11 | 15 | Elpida Memory |  Japan | 6 678 | +74.2% | 2.3% |
| 12 | 8 | Advanced Micro Devices |  USA | 6 355 | +22.0% | 2.1% |
| 13 | 11 | Infineon Technoloaies |  Germany | 6 226 | +39.7% | 2.0% |

1.1 Procesory s jádrem ARM

Procesory ARM lze v současné době nalézt v mnoha elektronických zařízeních. Své uplatnění nalézají všude tam, kde je potřeba velký výpočetní výkon při malé spotřebě např. při zpracování obrazu, vyhodnocování GPS signálů, při mobilní komunikaci, atd.

Historie vzniku procesorů ARM sahá až ke konci 80-tých let 20. Století, kdy vznikl první RISC procesor **ARM** ve firmě *Acorn RISC Machine*. Následující úspěchy s procesory ARM1, ARM2 a ARM3 vedly k osamostatnění divize procesorů ARM pod novým názvem *Advanced RISC Machine*. Následovaly další procesory ARM6 -ARM 11 a také přejmenování firmy na ARM Ltd.

1.1.1 ARM1

Historicky prvním typem byl ARM1. Tento procesor byl použit v několika vývojových systémech pro BBC a PC ale především to byl prototyp, který byl rychle nahrazen typem ARM2. Procesoru ARM1 bylo vyrobeno řádově 100 ks, neobsahovaly instrukce násobení ani koprocessor a jeho instrukce.

1.1.2 ARM2

Nedlouho po ARM1 byl vyvinut typ ARM2, který již obsahoval 27 registrů z toho zároveň přístupných 16. Měl již čtyři pracovní módy (USR, IRQ, FIQ a SVC), umožňoval adresování až 64MB, měl třístavové zřetězení při zpracovávání instrukcí a jeho hodinový kmitočet byl 8MHz. Tomu odpovídá výpočetní výkon 4 -4,7 MIPS.

1.2.3 ARM3

Tento typ byl jen malým vylepšením předcházejícího typu. Oproti typu ARM2 byla přidána paměť cache (4k), koprocessor a zvýšena rychlost hodinového kmitočtu na 12 – 33 MHz. Výpočetní výkon při 12 MHz byl 7MIPS, při 24MHz 13,26 MIPS a při 33MHz 17,96 MIPS.

1.2.4 ARM4 a ARM5

Tyto procesory nebyly nikdy vyrobeny.

1.2.5 ARM6

První komerční procesor s plnou 32-bitovou adresovou sběrnici (adresovat lze až 4 GB), 32 32bitových registrů, 6 pracovních módů (USR, IRQ, FIQ, SVC, Abort a Undefined). Procesoru ARM6 bylo vyráběno několik verzí.

1.2.6 ARM7

Identický s ARM6 ale s větší rychlostí. Některé varianty obsahují hardwarovou násobičku. Většina změn nastala v časování různých signálů. Typ ARM700 má zvětšenou paměť cache (8k) a větší účinnost s ohledem na spotřebu. Na 40 MHz má typ ARM710 výkon 36MIPS. Dostupné verze jsou:

| | |
|---------|---|
| ARM7 | – holý chip |
| ARM7D | – jádro s podporou ladění |
| ARM7DM | – ARM7D s hardwarovou násobičkou |
| ARM7DMI | – ARM7DM s podporou rychlého zpracování přerušení |
| ARM70DM | – ARM7DMI jako chip |
| ARM700 | – ARM7 + MMU + cache + WriteBack Buffer |
| ARM7500 | – ARM7 + MMU + cache + Writeback Buffer + IOMD + VIDC20 |

Uvedené typy mohou být i ve verzi Thumb (možnost zpracování 16-ti bitových instrukcí)

1.2.7 ARM8

Přímo kompatibilní s ARM6 a 7. Obsahuje 5-ti stavové zřetězení zpracovávání instrukcí. Cache paměť zůstává stejně velká ale slouží pro WriteBack, dále je přidána 64-bitová násobička. Chipy jsou vyráběny technologií 0,5 um, která umožňuje dosažení výkonu až 80MIPS při 3,3V a 80MHz.

1.2.8 StrongARM

Rychlá varianta rodiny ARM. Architektura je podobná jádru ARM8, také využívá 5-ti stavové zřetězení. Rozdíl je v rozdělení paměti cache na dvě části. Na cache pro instrukce a pro data (Harvardská architektura). Každá cache má velikost 16kB. Při výrobě je použita technologie 0,35 um a výkony jsou od 115 MIPS při 100MHz až po 230 MIPS při 200MHz.

1.2.9 ARM9

Opět vychází z předchozího typu ARM8, ze kterého si bere 5-ti stavové zřetězení ale vnitřní architektura je Harvardská, jako u StrongARM. Velikosti pamětí cache může být 0kB – ARM9TDMI, 4kB/4kB – ARM940T, 8kB/8kB – ARM922T a 16kB/16kB – ARM920T. Použitá technologie je opět 0,35 um a výkon je až 200MIPS při 180 MHz.

1.2.10 ARM10E

Cesta za dalším zrychlení vedla k procesoru se šestistupňovým zřetězením, rozdělenou pamětí cache (32k/32k) a technologií 0,13 um

1.2.11 ARM11

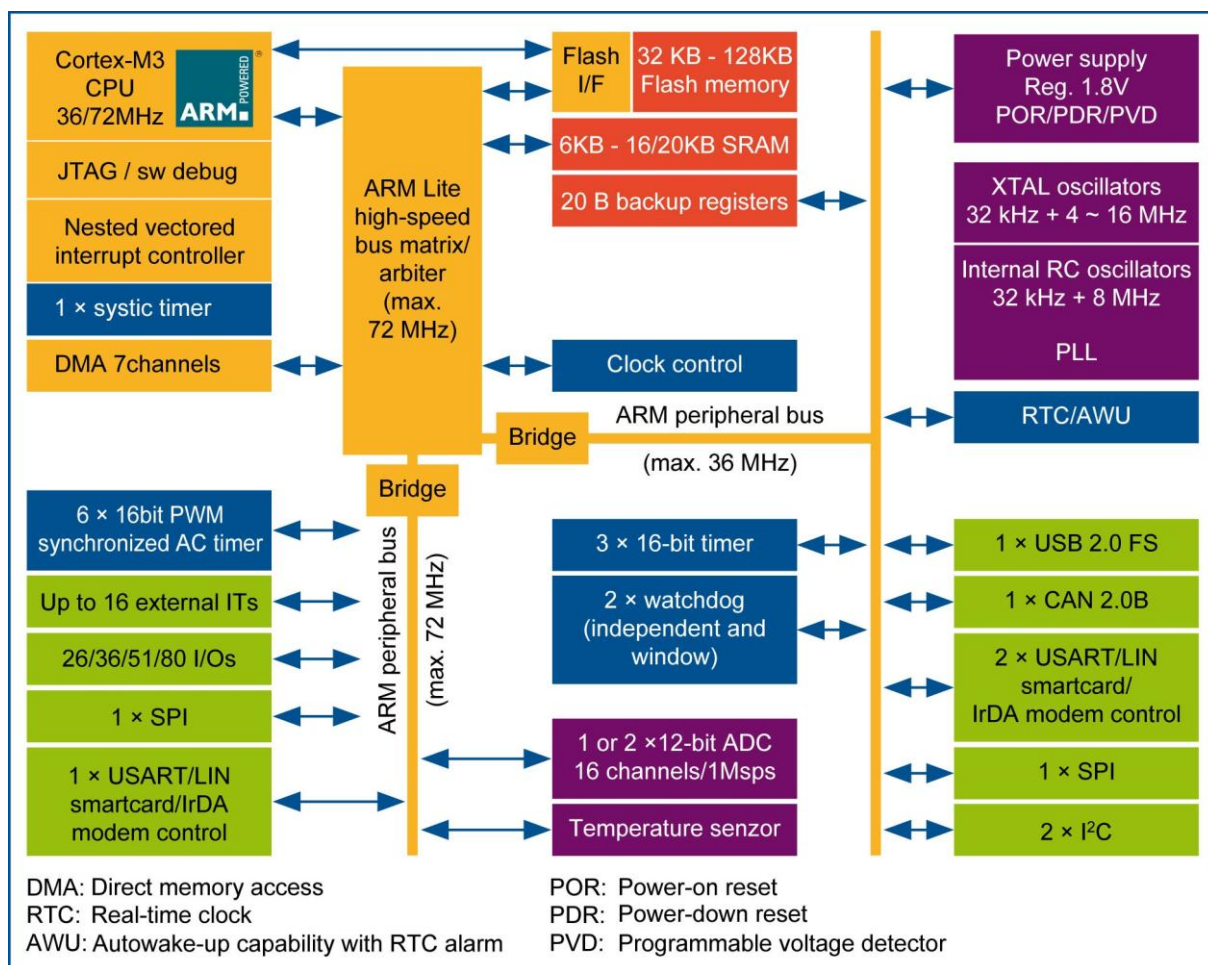
Tento typ má již osmi-stavové zřetězení, podporu cache paměti (4 – 64 kB) a technologii výroby opět 0,13 um. Hodinová frekvence je přes 1GHz a výkony jsou 740 Dhrystone 2.1 MIPS. Spotřeba je 0.6mW/MHz (0.13um, 1.2V).

1.2.12 Cortex M3

Dalším vylepšením procesorů ARM7 jsou procesory s novým jádrem ARM Cortex M3. Oproti ARM7 nabízejí zejména vyšší výpočetní výkon při stejné frekvenci hodin, rychlejší odezvu na přerušení, až 240 úrovní přerušení, výkonnější a paměťově úspornější instrukční soubor atd. Jádro založeno na tří-stupňovém zřetězení s Harvardskou architekturou a s inovovanou instrukční sadou Thumb (Thumb-2). Technologie výroby je 0,18 um. Při testu Dhrystone dosahuje výkonu 1,25 DMIPS/MHz a spotřeba je 0,19mW/MHz. Rodina ARM Cortex zahrnuje typy ARM Cortex-A8, ARM Cortex-A9 MPCore, ARM Cortex-A9 Single Core Processor, ARM Cortex-M1, ARM Cortex-M3 a ARM Cortex-R4(F). Tyto procesory se vyrábí ve třech sériích, u každé z nich je implementován Thumb-2 instrukční soubor. Série ARM Cortex-A aplikační procesory určeny pro komplexní operační systémy a uživatelské aplikace, podporuje 32-bit ARM, Thumb a Thumb-2 instrukční soubory. Série ARM Cortex-R Jsou určeny do segmentu embedded systémů pro práci v reálném čase, podporují 32-bit ARM, Thumb a Thumb-2 instrukční soubory. Série ARM Cortex-M Procesory této série jsou určeny zejména do segmentu, kde je kladen důraz především na nízkou cenu, podporuje pouze Thumb-2 instrukčním souborem.

1.3 Procesory STMicroelectronics

STMicroelectronics patří k firmám nabízejícím 32bitové mikrokontroléry s jádrem od firmy ARM. Novinkou v sortimentu jsou mikrokontroléry STM32 založené na nejnovějším jádru ARM[®] Cortex[™] - M3. Toto jádro je navrženo speciálně pro embedded aplikace vyžadující vysoký výkon, běh v reálném čase a nízký příkon (pozn.: určeno písmenem M za pomlčkou). Další předností tohoto jádra jsou jeho nízké nároky na plochu čipu. Díky tomuto faktu je i výsledná cena mikrokontroléru nízká, takže může konkurovat i některým výkonným 8bitovým mikrokontrolérům. To, že je tento mikrokontrolér směřován do oblasti jednodušších, nebo chcete-li lacinějších, aplikací je vidět i podle použitých LQFP pouzder s relativně nízkým počtem pinů (48pinů, 64pinů a 100pinů). To však neznamená, že jde o „ošizený“ mikrokontrolér, spíše naopak. Jádro pracuje s tříúrovňovým překrýváním fází vykonávání instrukcí se spekulativním vykonáváním skokových instrukcí. Oboje techniky minimalizují nutný počet taktů k vykonání instrukcí, takže i instrukce dělení a násobení jsou vykonávány během jednoho taktu. Pro lepší porovnání je vhodné uvést dosahovaný počet instrukcí za sekundu. Jádro se může „pyšnit“ 1,25 DMIPS/MHz (Dhrystone 2.1 Benchmark). To však není vše co nové jádro přináší. Další novinkou je nový instrukční soubor Thumb[®]-2. V podstatě jde o pokračování úspěšné myšlenky s redukcí 32bitové délky instrukcí tak, aby se délka programu zmenšila a výkon jádra zůstal co nejvíce zachován. Používáním kratších instrukcí se zmenšuje velikost výsledného kódu programu a protože může být použito menší, a tedy lacinější, paměti programu, cena řešení se snižuje. Že investice do „vývoje“ Thumb[®]-2 se vyplatila, je možné posoudit i ze „suchých“ čísel: jádro Cortex[™]-M3 je při vykonávání Thumb[®]-2 o 70% efektivnější než ARM7TDMI-S vykonávající instrukce Thumb[™] přičemž kód využívající Thumb[™]-2 je o cca 10% kratší než Thumb. Zde se sluší připomenout, že nové jádro dokáže vykonávat i původní sadu Thumb. Další novinkou je možnost vnořování přerušení (nesting interrupts), přičemž počet úrovní přerušení je maximálně 256 a tyto priority se navíc dají dynamicky měnit.



V sortimentu firmy STMicroelectronics najdeme tři řady mikrokontrolérů-základní, výkonnou a nízkopříkonovou. Rozdíly spočívají v maximální frekvenci jádra (36 MHz oproti 72 MHz), menší paměti SRAM (max. 16 KB oproti 20 KB) a menšímu počtu periférií. V základní řadě mikrokontrolérů není u žádného mikrokontroléru implementováno rozhraní USB 2.0FS ani řadič sběrnice CAN ani PWM. Ostatní parametry a funkce jsou u obou řad shodné. I u nejmenšího představitel základní řady, typu STM32F101C6, najdete velké množství periférií, neboť i o zdroji hodinového kmitočtu se dá, díky širokým možnostem, hovořit jako o periférii. Vyjma standardního krystalového oscilátoru pro externí krystal 4 MHz až 16 MHz, najdeme u této periférie taktéž interní, kalibrovaný, RC oscilátor s kmitočtem 8 MHz, interní RC oscilátor 32 kHz (pro watchdog i obvod RTC) i oscilátor pro 32 kHz krystal (pro obvod RTC). Oba rychlejší oscilátory jsou napojeny na jednotku PLL, která dokáže „vynásobit“ kmitočet 2x až 16x. Tento vynásobený kmitočet je pak použit pro taktovní mikrokontroléru. Kmitočet z „pomaloběžných“ oscilátorů je pak použit pro obvod RTC. Obvod je dělán velmi jednoduše. Sestává se z programovatelné předěličky na jejímž výstupu jsou sekundové impulsy. Tyto impulsy jsou čítány 32 bitovým čítačem, jehož stav může být komparován s hodnotou uloženou v registru RTC_ALR. Už podle názvu jde poznat, že se tzv. „alarm“, tj. možnost vzbudit mikrokontrolér po uplynutí nadefinovaného počtu sekund. Interním 32 kHz oscilátorem pak jsou řízeny dva obvody watchdog, přičemž jeden je klasický, druhý je typu „windowed“, tj.

znovunastavení obvodu MUSÍ proběhnout v určitém časovém období, tzv. „oknu“, jinak, i když dojde k pokusu o restart (reload) tohoto obvodu, je generován reset.

I když jsou obvody zajišťující základní chod mikrokontroléru zajímavé, z hlediska uživatele budou obvody časovačů podstatně zajímavější. U zmiňovaného nejjednoduššího mikrokontroléru nové řady jsou implementovány celkem dva. Pokud je někdo překvapen nízkým počtem, měl by vědět, že obvody časovačů jsou u 32bitových mikrokontrolérů podstatně komplexnější než běžně používané časovače u 8 bitových mikrokontrolérů. Nejinak je tomu v tomto případě a tak v **jeden kanál** časovače se skládá z časové základny kterou tvoří 16 bitový up/down (nastavitelný) čítač, kterému je předřazen 16 bitový programovatelný předdělič. S tímto čítačem spolupracují celkem 4 jednotky „input capture/output compare/PWM“. Možnost provozovat čítač v režimu „up“ tak i „down“ dává tušit, že při generování PWM signálů budeme mít možnost si zvolit mezi PWM signály „zarovnanými“ na hranu nebo na střed („edge or center aligned PWM“). První z nich je vhodná pro výkonnové aplikace ve spínaných zdrojích, zatímco druhá jmenovaná je vhodná pro řízení elektromotorů. Po tomto, byť krátkém, výčtu možností jednoho kanálu časovače, je jasné, že možnosti časovačů, i u tohoto nejjednoduššího mikrokontroléru v řadě, jsou více jak postačující. Větší typy pak disponují celkem 3 kanály časovačů.

Další částí, která je u 32bitových mikrokontrolérů často používána, jsou řadiče přímého přístupu do paměti. Tyto řadiče mají za úkol odlehčit jádru mikrokontroléru od rutinní činnosti, jakými přenosy dat jsou. Druhým důvodem jsou možné vysoké komunikační rychlosti mikrokontroléru s okolím. Každý ze 7 kanálů dokáže přenášet data mezi dvěma místy v paměti, mezi pamětí a periferií a mezi dvěma periferiemi. Zajímavou vlastností je možnost volby rozdílné velikosti dat u zdroje a cíle přenosu (bytově orientovaná periferie a slovně orientovaná paměť). Samozřejmostí je pak podpora kruhového bufferu.

Další periferie jsou již dobře známé z 8 bitových mikrokontrolérů i když některé jejich vlastnosti byly přizpůsobeny novým podmínkám. Na čipu najdeme jedno rozhraní I²C, jedno rozhraní SPI a celkem dva kanály velmi populárního rozhraní USART. Dále je na čipu implementován 12 bitový (!) analogově-digitální převodník (ADC) s předřazeným analogovým multiplexerem 10:1.

Přestože výkon těchto mikrokontrolérů značně překonává možnosti běžných 8 bitových mikrokontrolérů, zůstává spotřeba mikrokontroléru na přijatelné úrovni. Při maximální frekvenci 36MHz, běžícím programu (z interní paměti Flash) a povolených všech periferiích je typický odběr přibližně 22mA. Příznivý vliv na spotřebu měla jednak moderní technologie a též nízké pracovní napětí, neboť mikrokontrolér pracuje s napájecím napětím 3,3V.

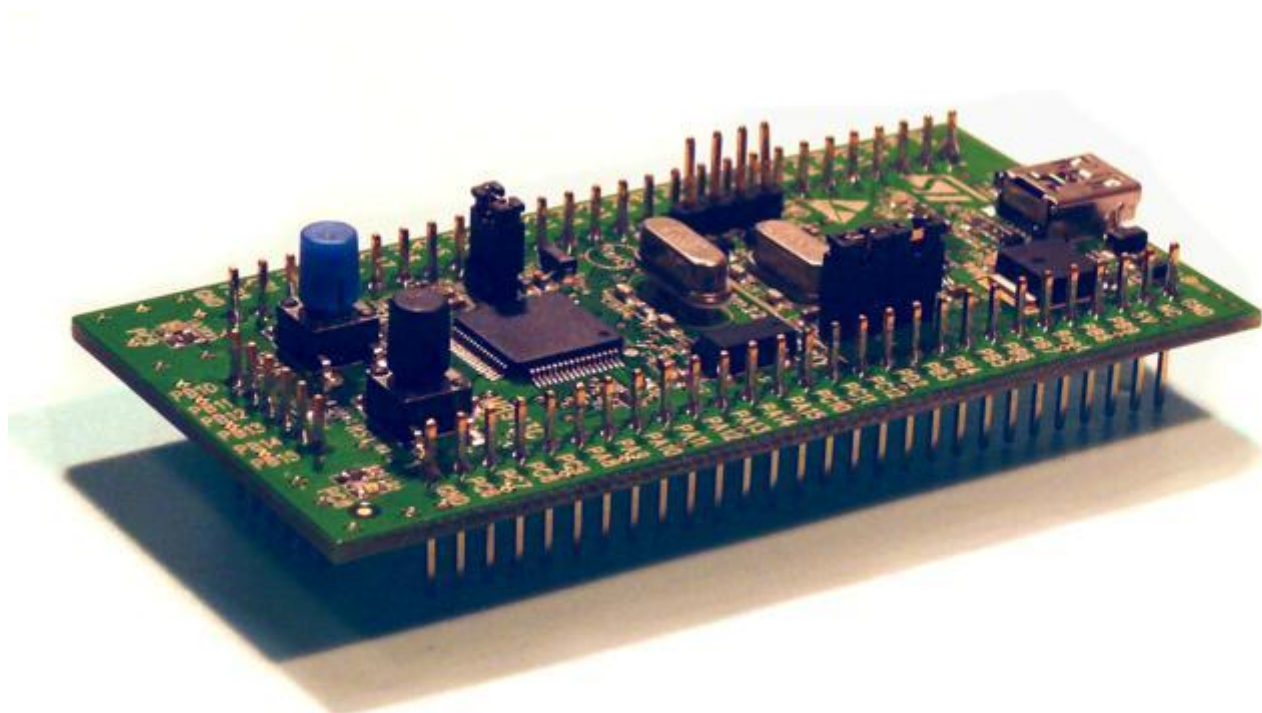
1.4 Startkit STM32VL Discovery

STMicroelectronics je výrobce startkitu STM32VL Discovery <http://www.st.com/stm32-discovery> (<http://www.st.com/internet/evalboard/product/250863.jsp>)

R/W je spojen s GND

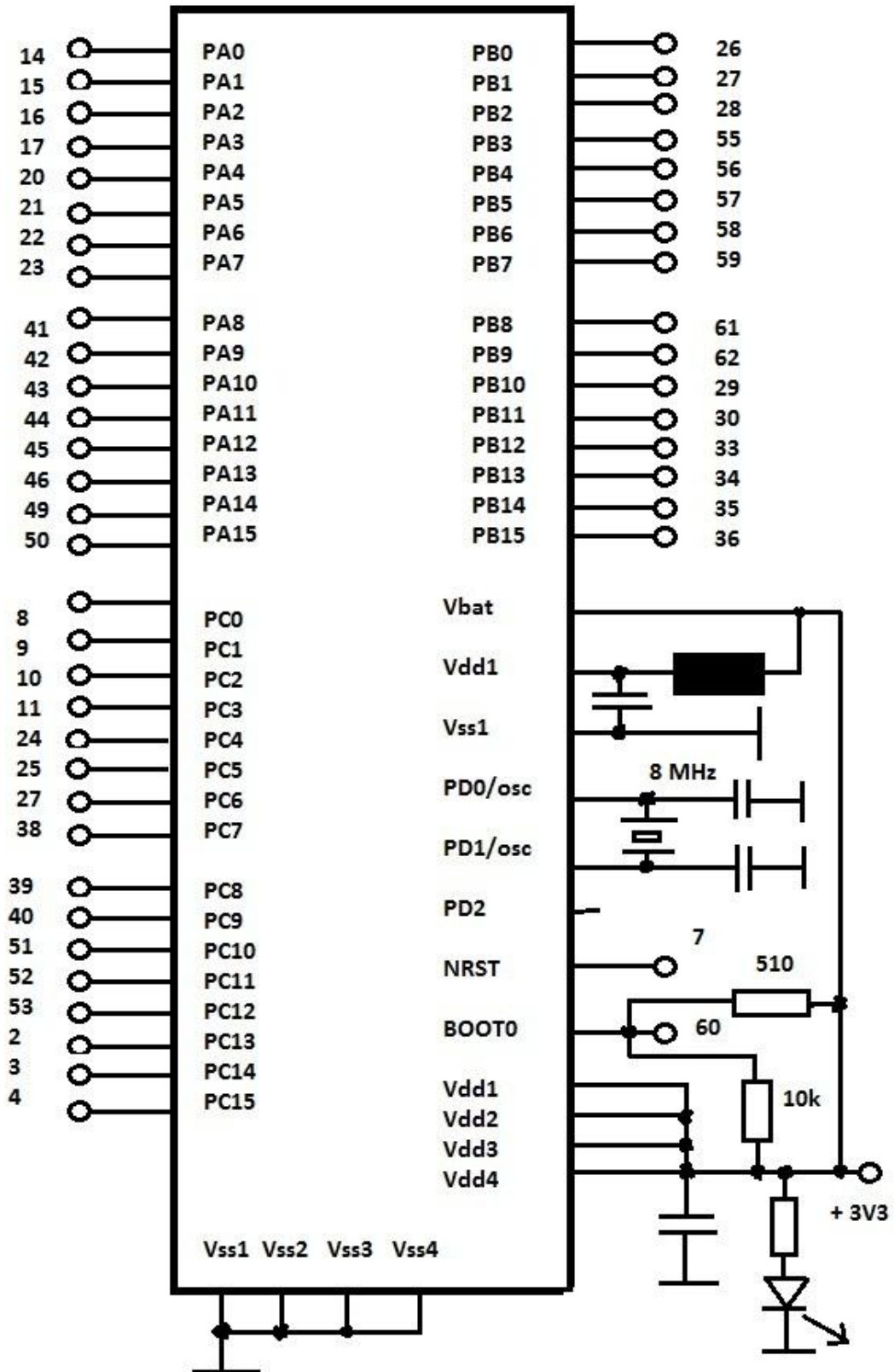
STM32F100RBT6 obsahuje 128 kB Flash, 8 kB RAM a je zapouzdřen v LQFP 64, tedy poměrně velkém pouzdře 10 x 10 mm.

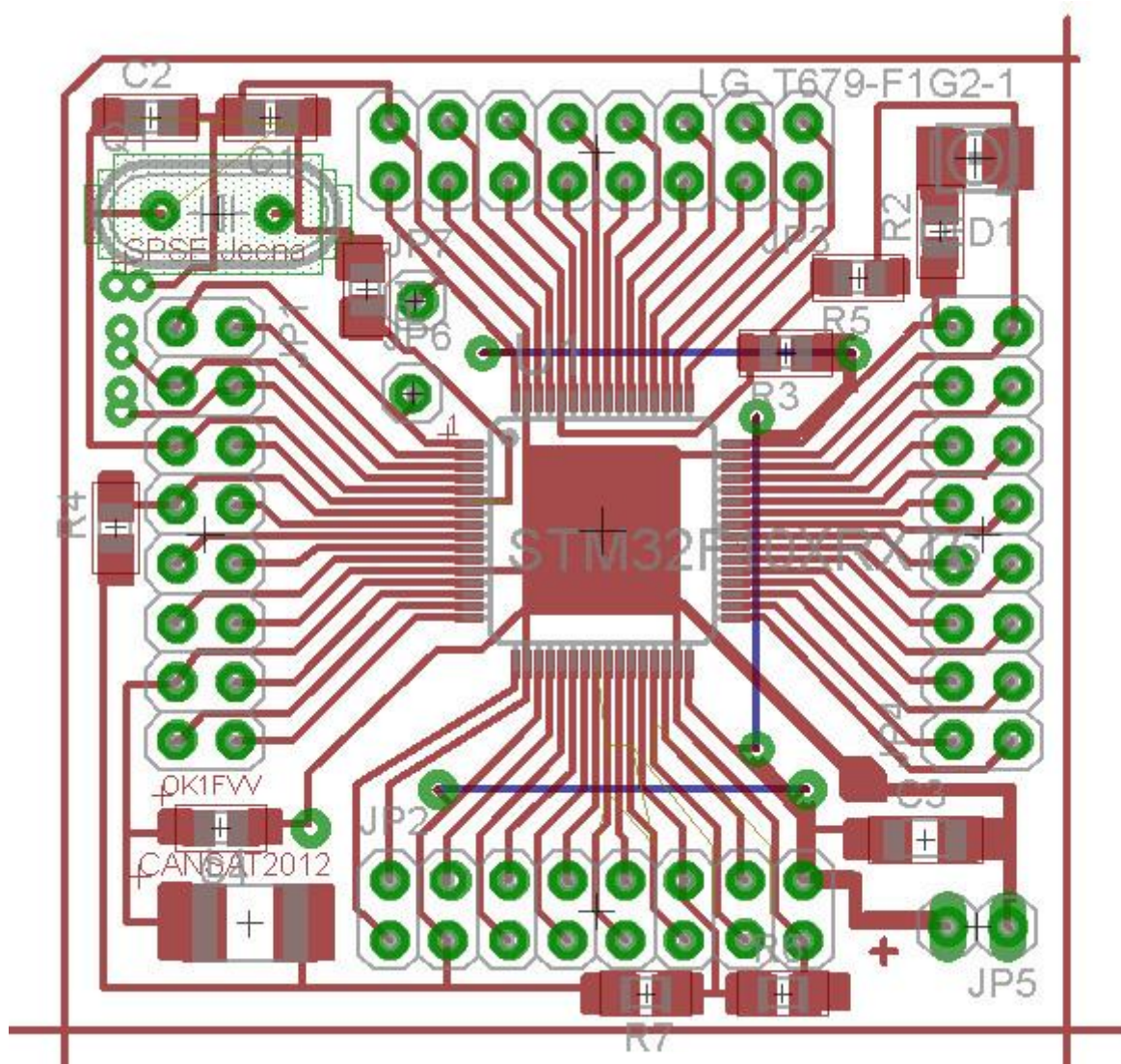
Kit obsahuje 4 LED, z toho 2 LED slouží pro ST-Link část, kde jedna signalizuje napájení (z USB konektoru) a druhá komunikaci prostřednictvím SWD. Další dvě LED jsou připojeny k Value Line mcu (barva zelená=PC9 a modrá=PC8).



1.5 Palubní počítač Cansatu SPŠE Ječná

STM32F100RBT6





Tento počítač obsahuje STM32F100RBT6, stejně jako startkit STM32 VL Discovery. Můžeme proto programy odladit na STM32VL Discovery a poté beze změn nahrát do palubního počítače.

Dále je uveden stručný návod jak nahrát vytvořený program do procesoru pro zde zmíněný palubní počítač.

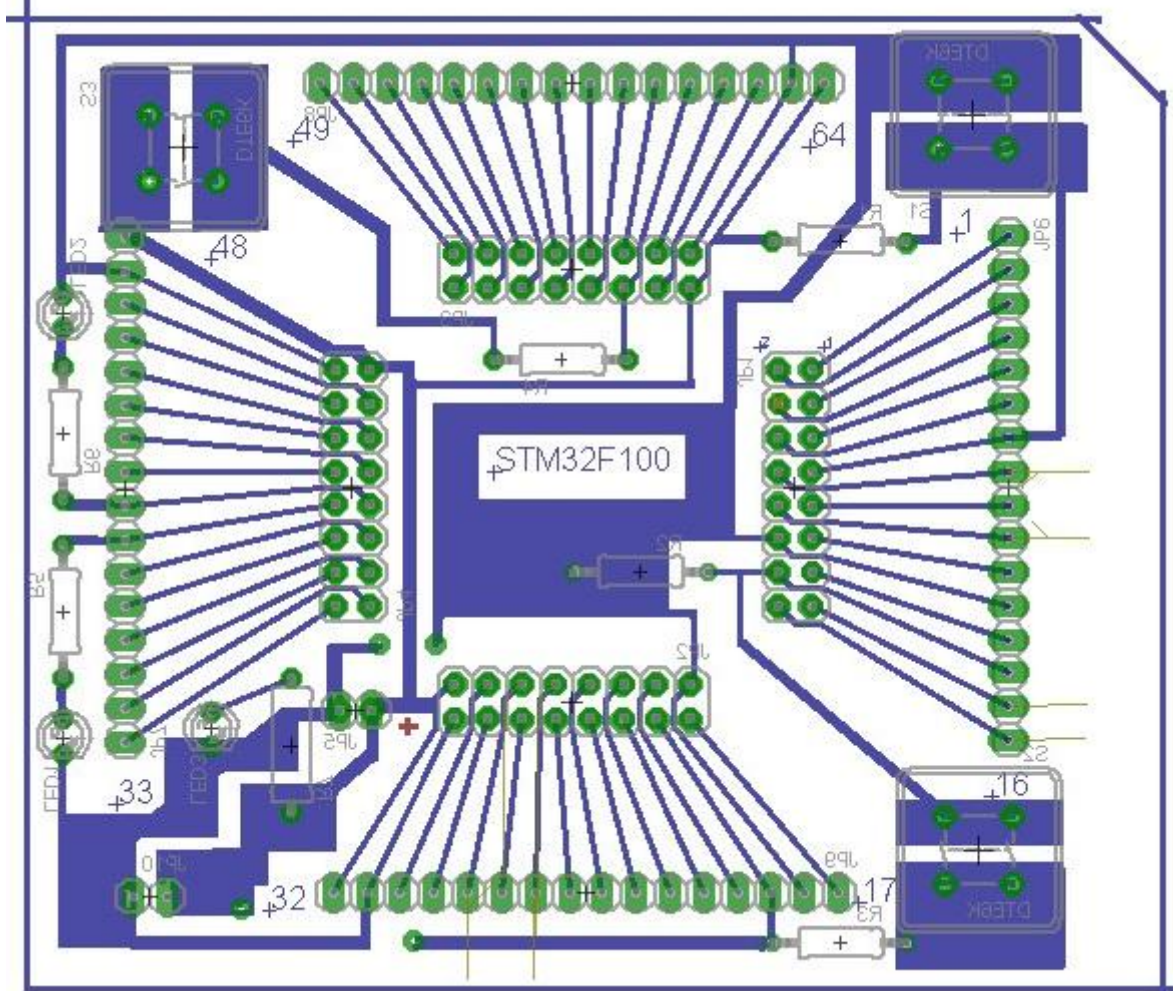
1. Propojíme palubní počítač s PC pomocí sériové linky RS232 (nemáme-li na PC RS232, lze použít i převodník USB -> RS232)
2. Připojíme napájení + 3,3 V palubního počítače

3. Stiskneme a držíme tlačítko BOOT připojené k palubnímu počítači a stiskneme a uvolníme tlačítko RESET, rovněž vně připojené k palubnímu počítači. Poté uvolníme i tlačítko BOOT. Takto je mikroprocesor připraven k nahrání programu.

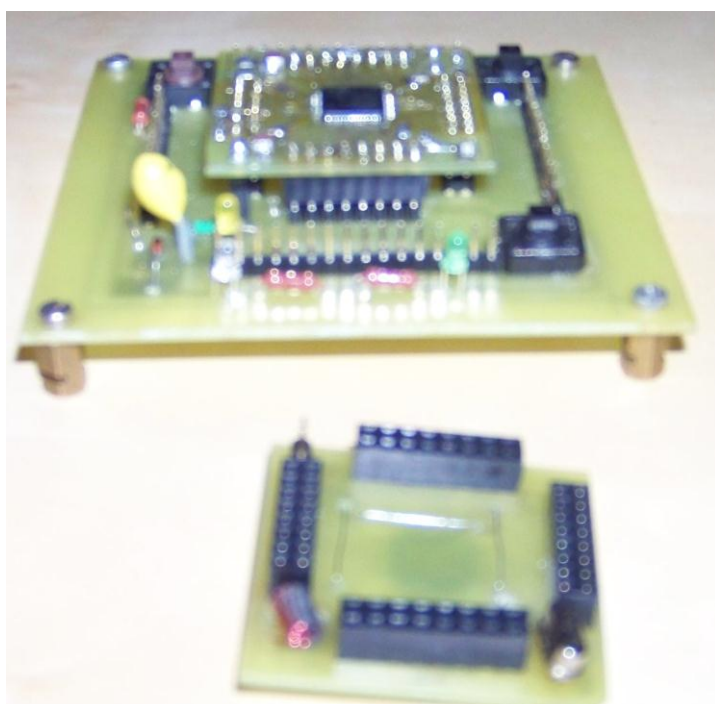
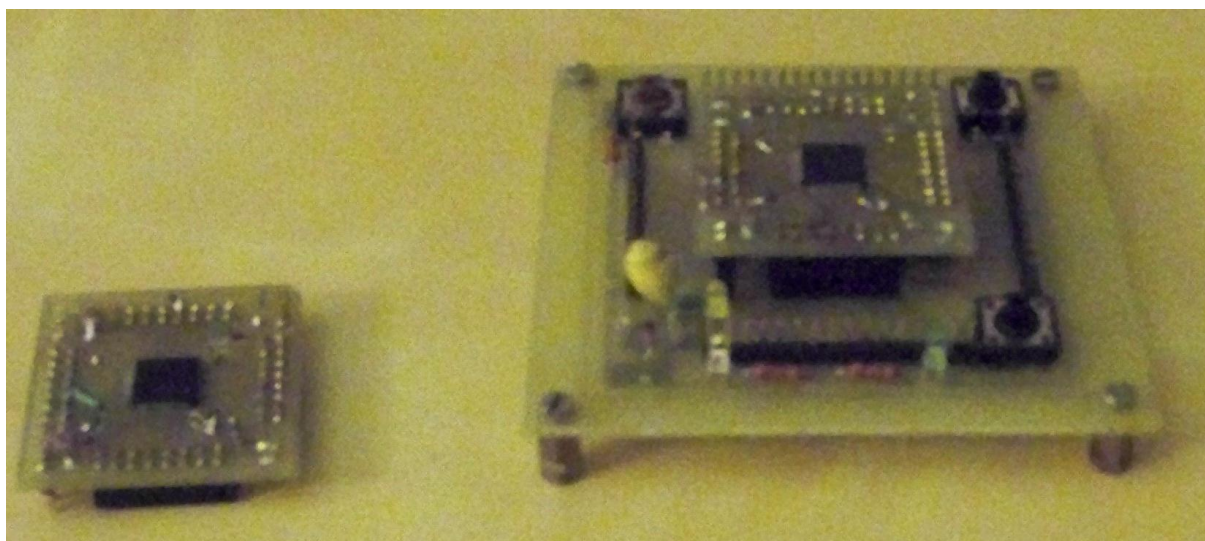
4. Na PC spustíme aplikaci STMicroelectronics Flash Loader Demonstrator V2.1.0 a na úvodní obrazovce vybereme komunikační port (COM0, COM1, ..), ostatní nastavení ponecháme a pokračujeme dále.

5. Další dvě okna stačí pouze potvrdit a zastavíme se až u okna s nabídkou *Download to device*, kde vyber náš přeložený a zkompilovaný soubor s příponou *.hex. Po potvrzení se již nahraje program do procesoru.

Výše uvedená vnější tlačítka Boot a Reset můžeme umístit např. na „základní desku“



Kromě již zmíněných tlačítek **Boot** a **Reset** je na desce i tlačítko **User** a dvě LED diody (modrá a zelená). Tlačítko user i obě LED jsou připojeny ke stejným pinům STM32F100, jako u startkitu **STM32VL Discovery**.

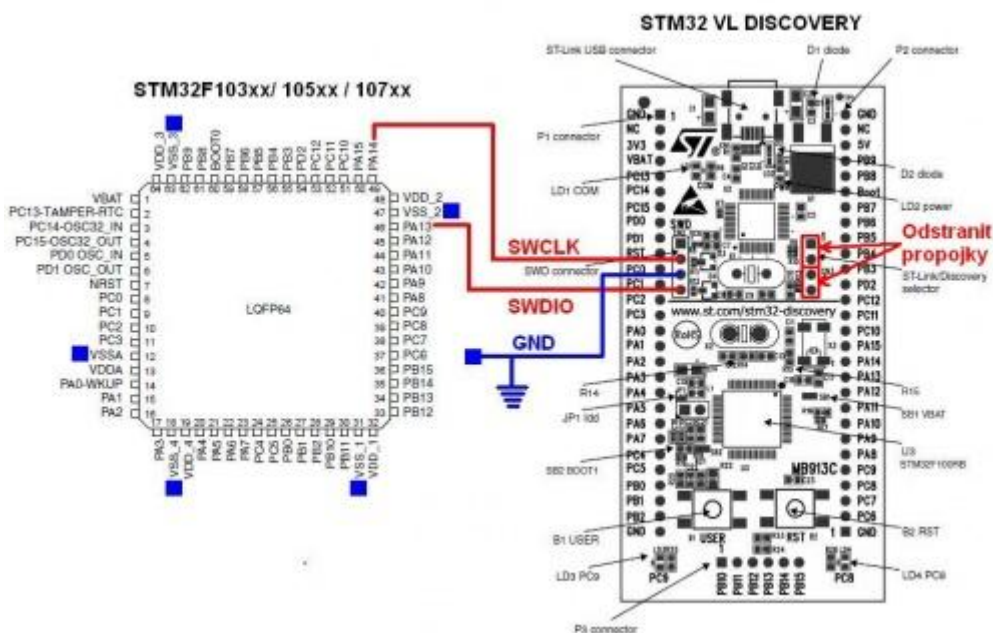


Podrobnější návod na obsluhu *Flash Loader Demonstrator* naleznete ve firemní literatuře STM.

Program můžeme nahrát také pomocí protokolu **SWD** z startkitu **STM32VL Discovery**. Propojit desky stačí třemi vodiči za předpokladu, že připojená deska má své vlastní napájení.

Připojení SWD:

```
SWD 2 <-> SWCLK (PA14),
SWD 3 <-> GND,
SWD 4 <-> SWDIO (PA13).
```

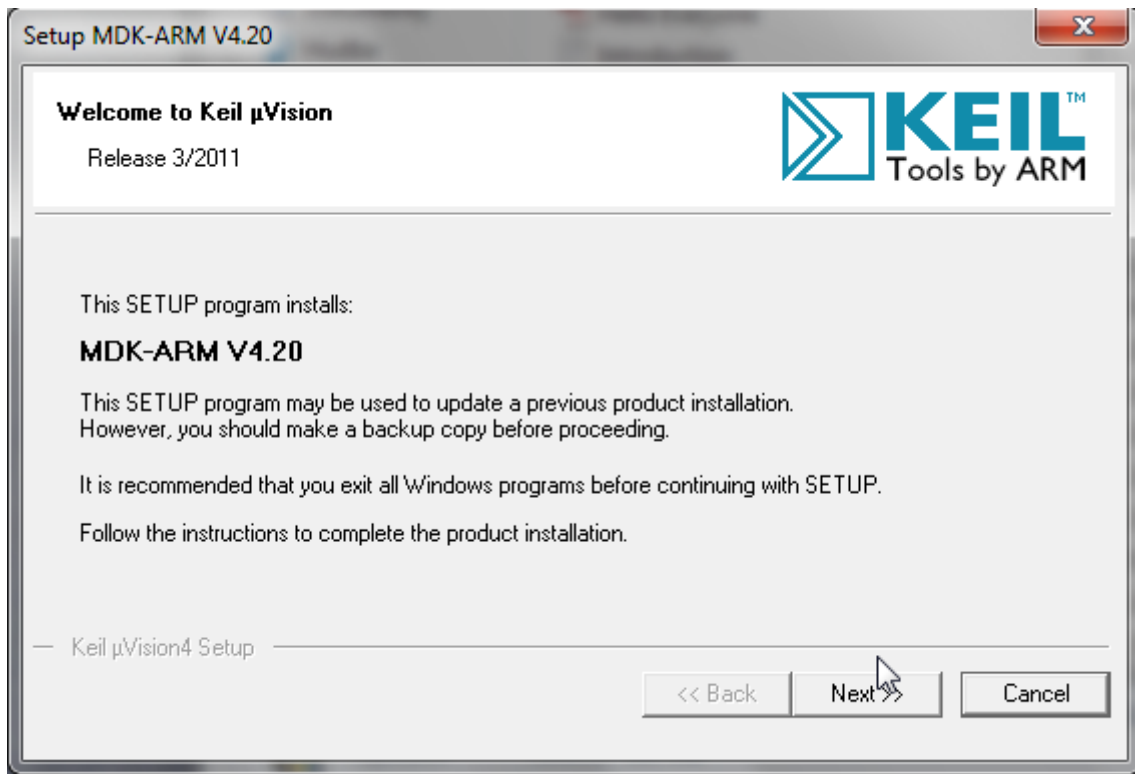


Podrobnější popis je na <http://neuron.feld.cvut.cz/publicwiki/STM32/STM32-VL-DISCOVERY> a na <http://neuron.feld.cvut.cz/publicwiki/Ulohy/STM32>

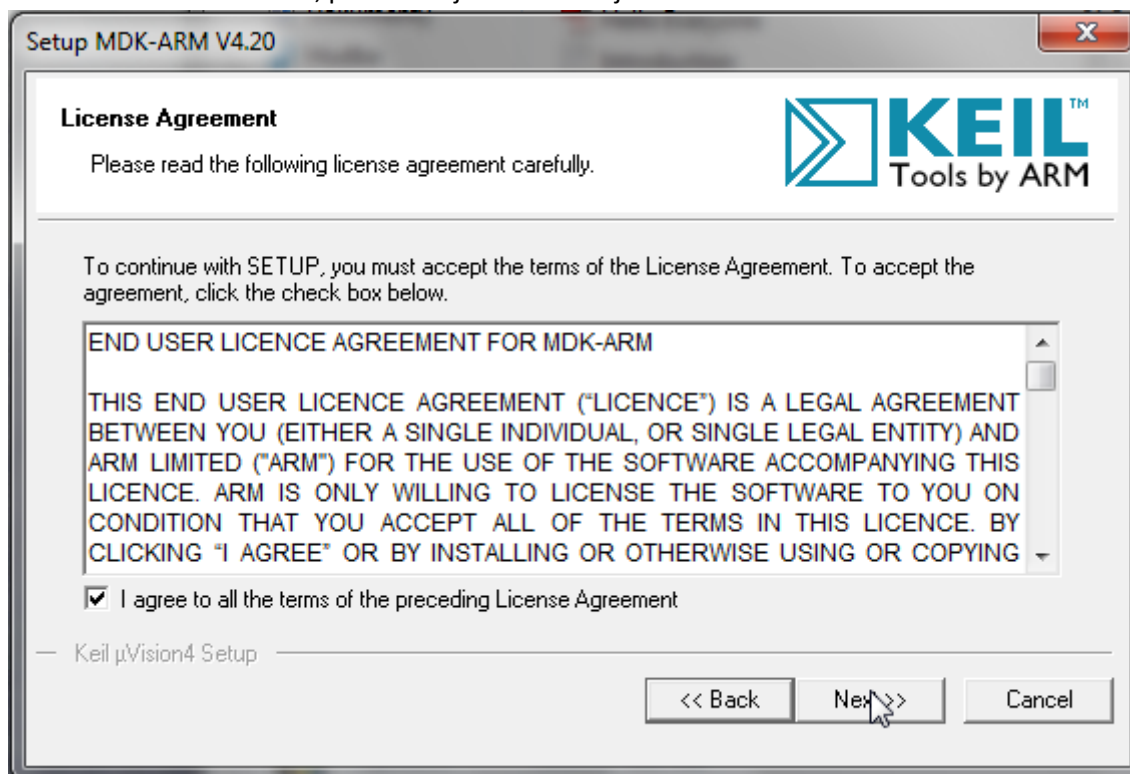
2. Práce v prostředí uVision4 Keil

2.1 Instalace prostředí Keil

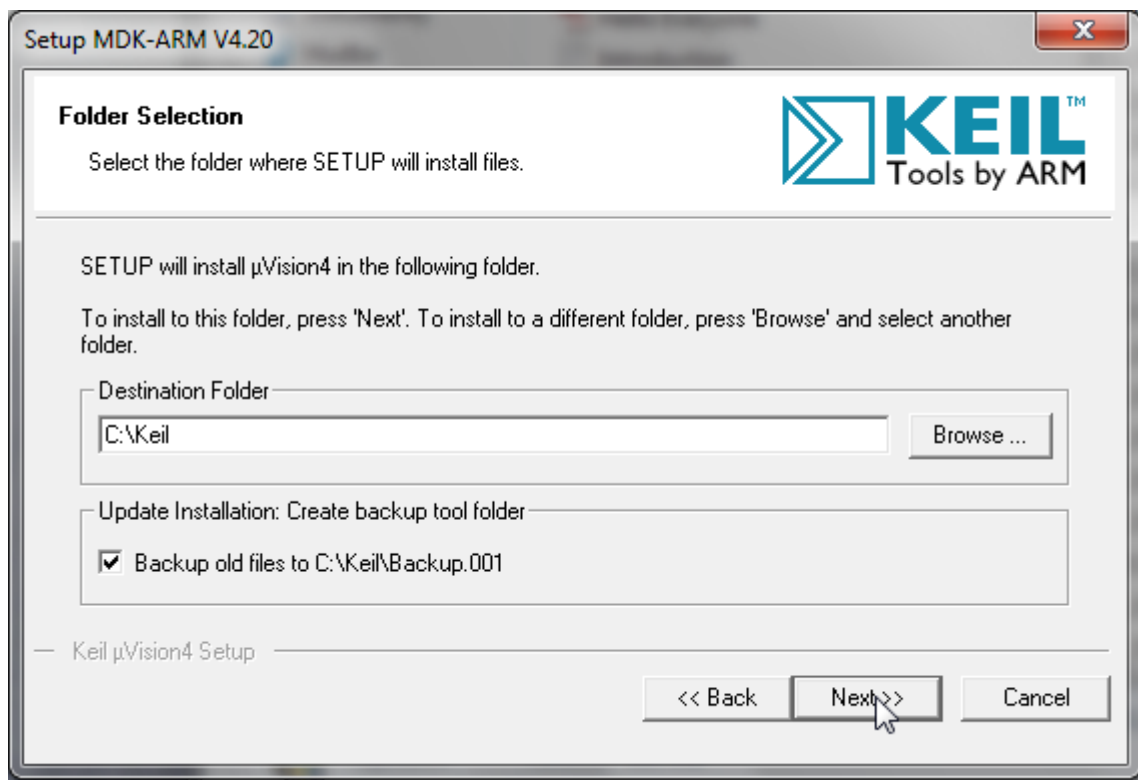
Spustíme instalační soubor **MDK420.exe**. Objeví se



Klikneme na tlačítko **Next**, poté se objeví licenční ujednání



Zaškrtneme souhlas s licenčním ujednáním a klikneme na tlačítko **Next**, tím postoupíme k výběru adresáře



Ponecháme navržené umístění a klikneme na tlačítko **Next**, dostaneme

Setup MDK-ARM V4.20

Customer Information

Please enter your information.

KEIL™
Tools by ARM

Please enter your name, the name of the company for whom you work, and your E-mail address.

First Name:

Last Name:

Company Name:

E-mail:

— Keil µVision4 Setup —

<< Back **Next >>** Cancel

Vyplníme výše uvedené informace a opět klikneme na **Next**. Potom začne instalace

Setup MDK-ARM V4.20

Setup Status

KEIL™
Tools by ARM

µVision Setup is performing the requested operations.

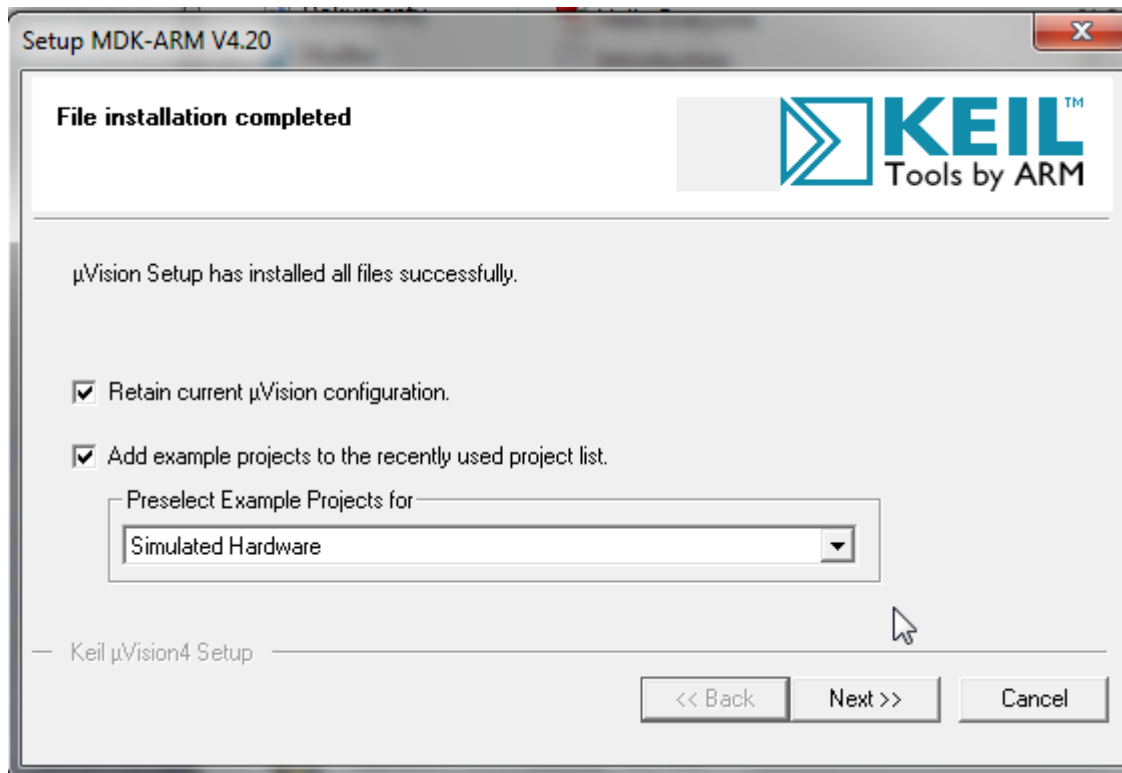
Install Files ...

Installing Clock.ini.

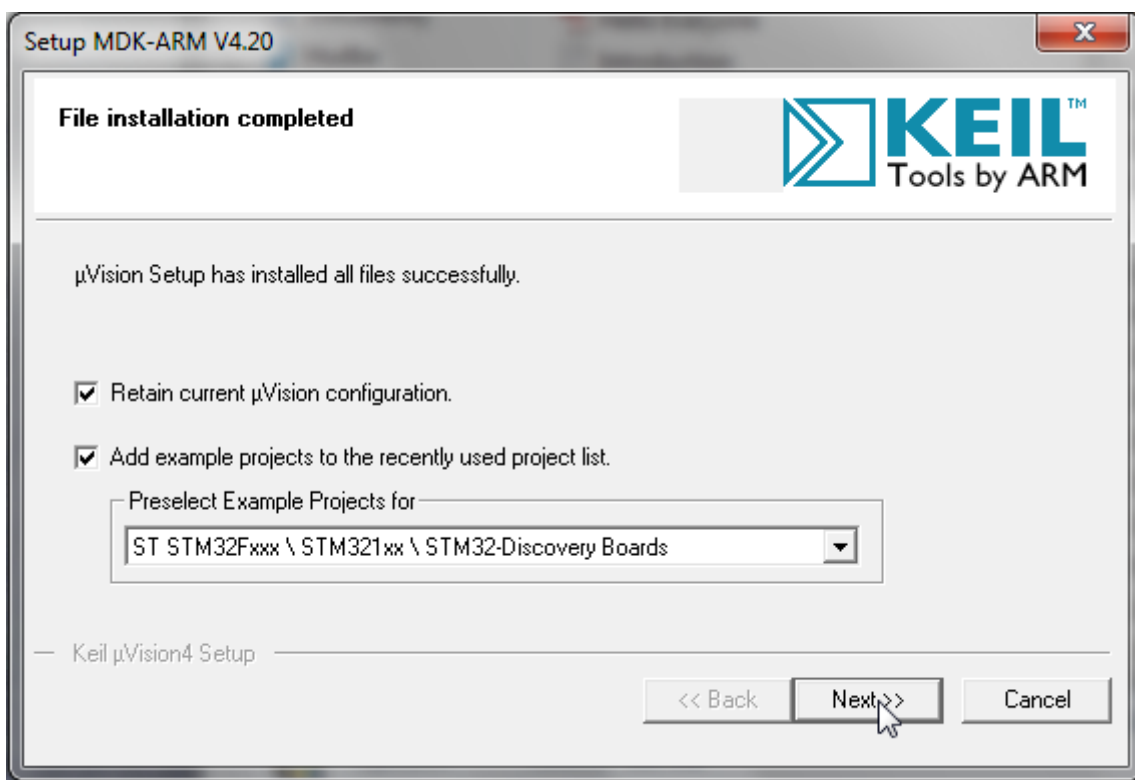
— Keil µVision4 Setup —

<< Back **Next >>** Cancel

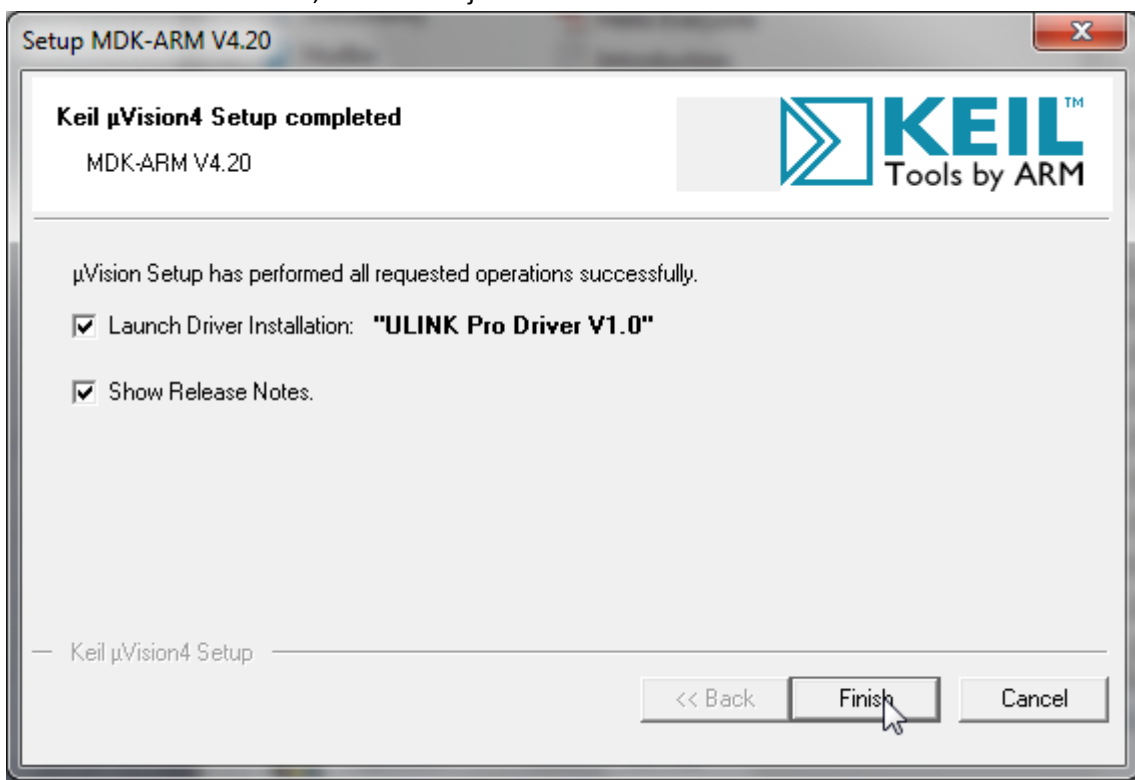
Po ukončení instalace



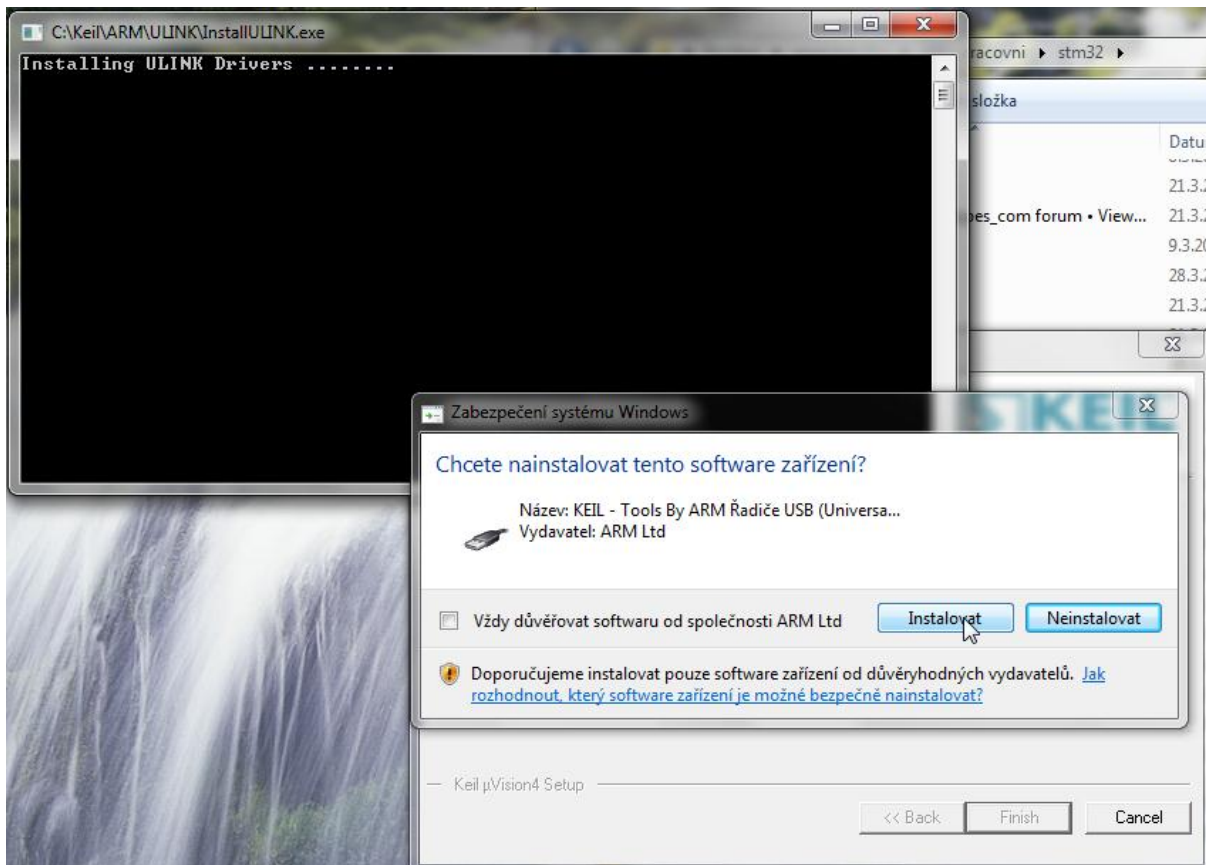
Změníme předpřipravené příklady projektů z Simulace hardware na **STM32 Discovery Boards**:



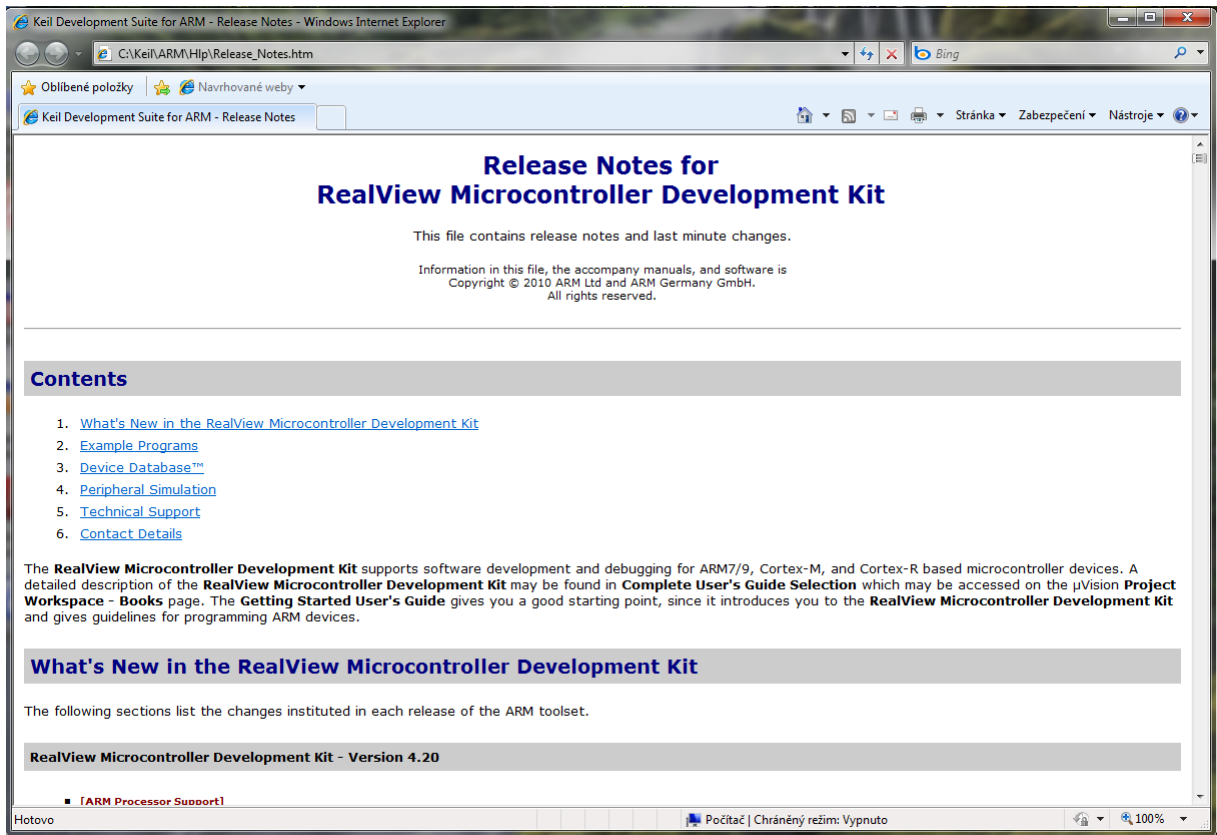
Potvrdíme tlačítkem **Next**, dostaneme ještě



Potvrdíme tlačítkem **Finish**. Poté se tento driver nainstaluje



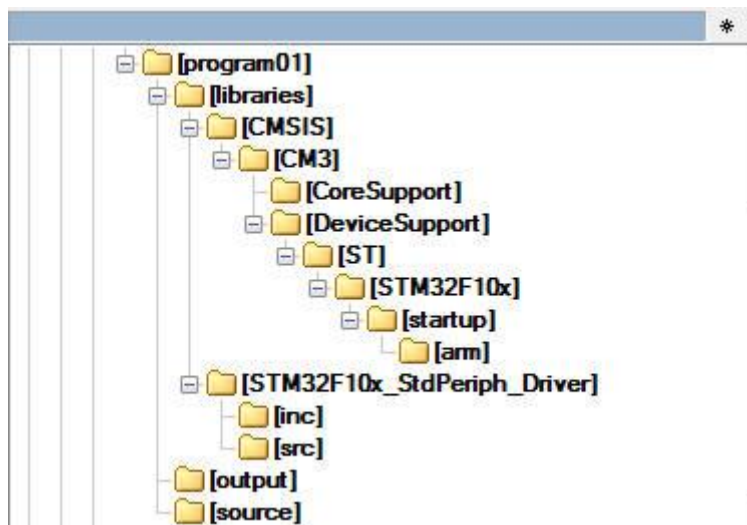
Nakonec se objeví informační stránka s poznámkami



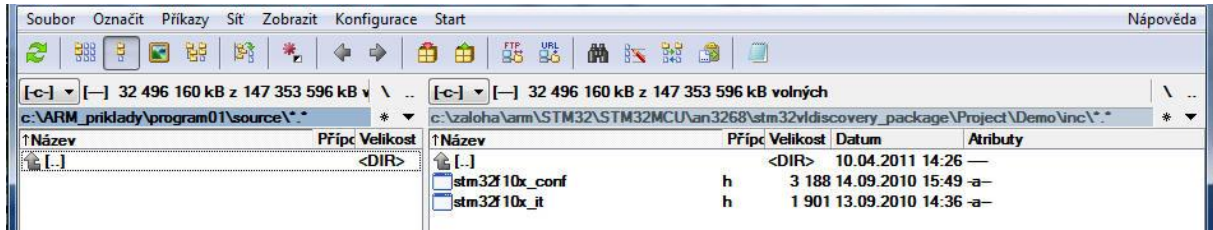
Stránku uzavřeme, uVision 4 Keil máme nainstalovaný.

2.2.1 Vytvoření našeho prvního programu – blikáč LED 1

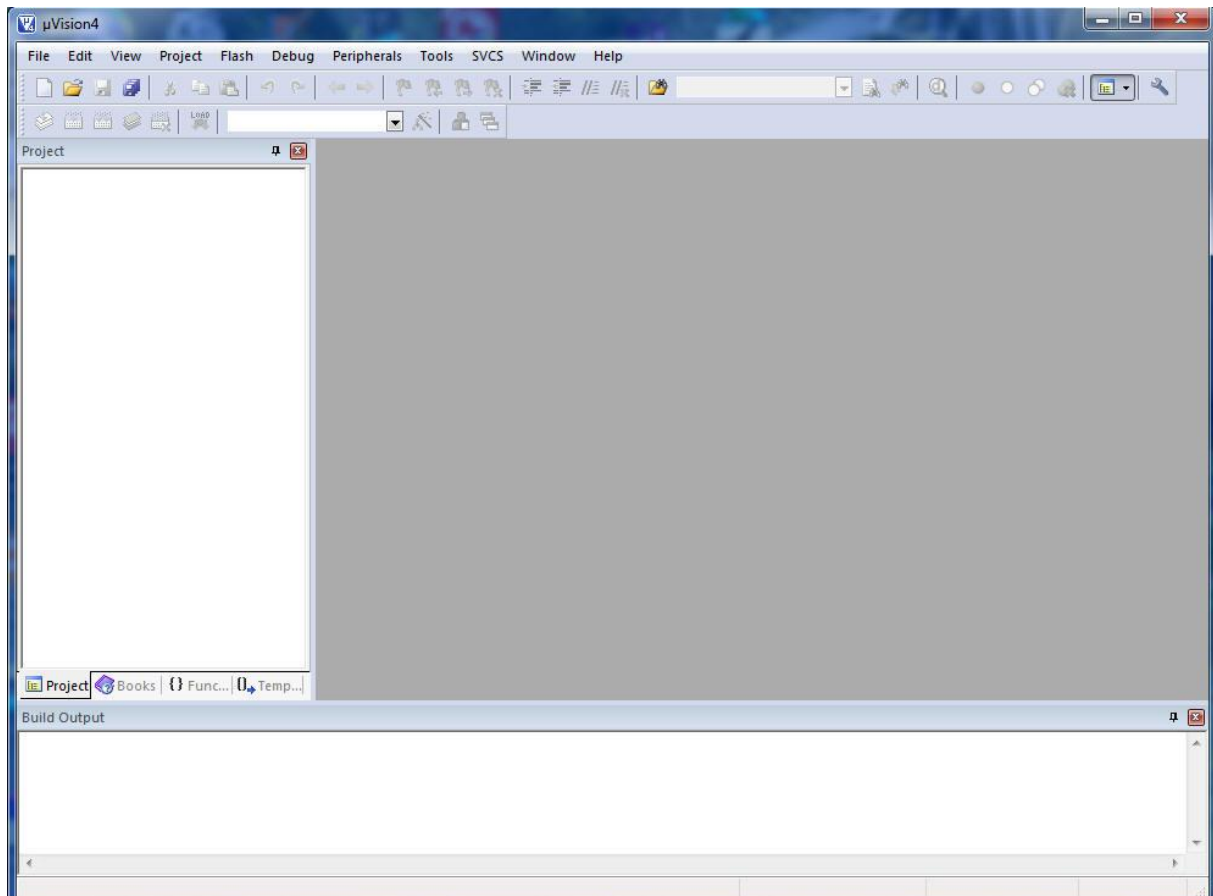
Nejprve si vytvoříme adresářovou strukturu pro snadnější orientaci v projektu. Tuto strukturu můžeme vytvořit z několika podadresářů. Jejich počet, pojmenování, stejně jako i v nich vnořených podadresářů si volí programátor sám. Nicméně se ustálilo několik doporučených struktur. Zvolíme např. následující strukturu.



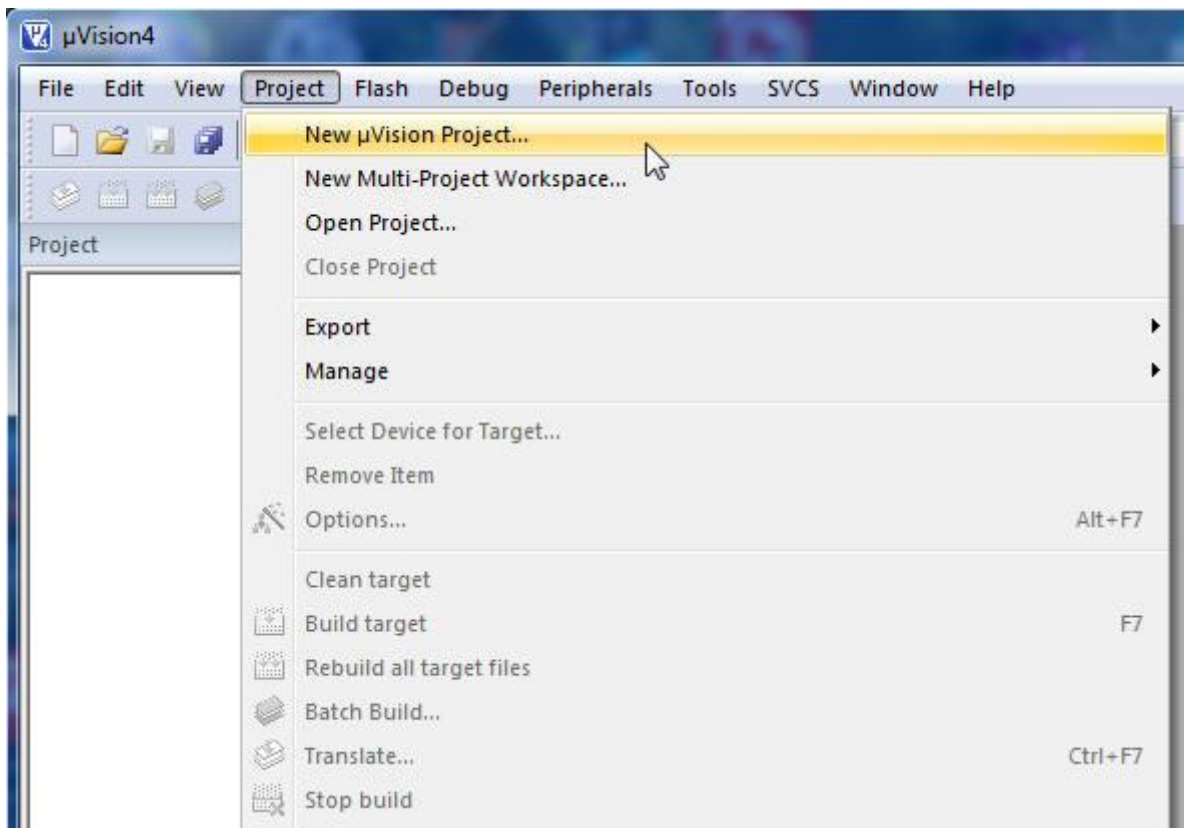
Náš první projekt nazveme **program01** a stejně pojmenujeme i adresář, který ho bude obsahovat. Složku **libraries**, včetně všech souborů zkopírujeme z **STM32VLDISCOVERY firmware package** (získáme z **AN3268** – aplikační poznámka /app. note/ 3268 stažením z firemních stránek STMicroelectronics). Do podadresáře **source** ještě zkopírujeme soubor **stm32f10x_conf.h**



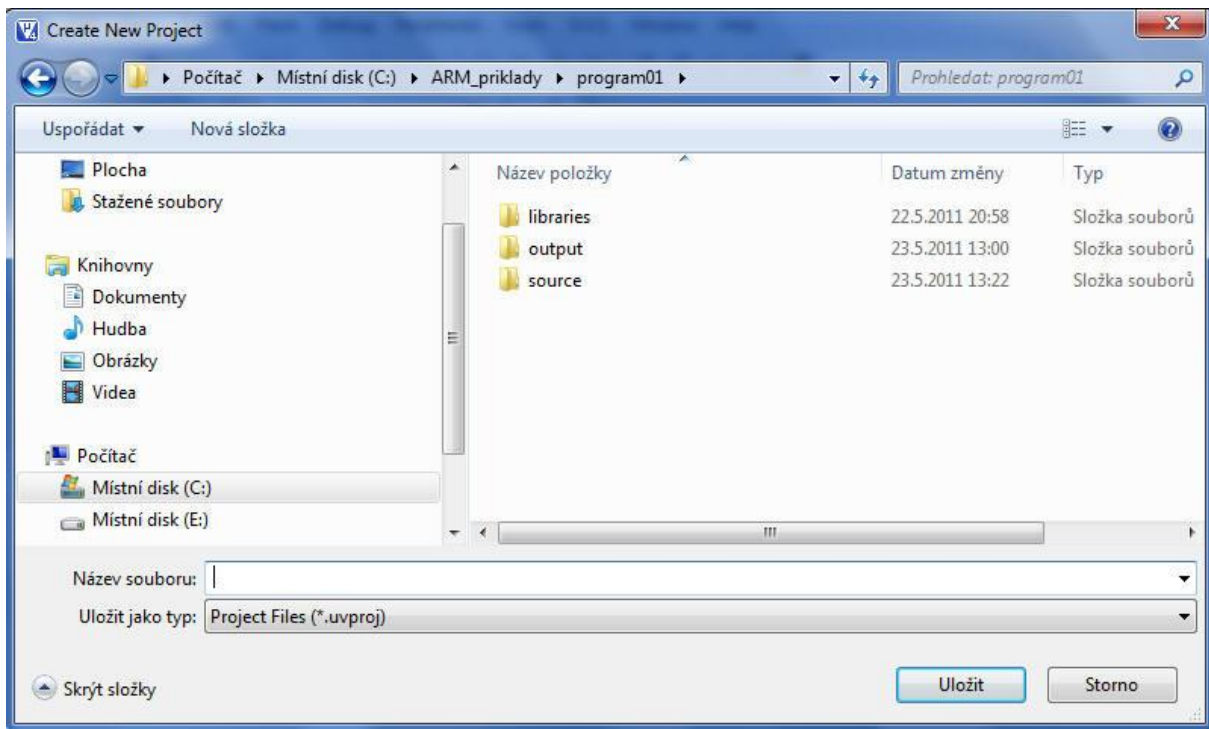
Nyní již spustíme vývojové prostředí **Keil uVision4**



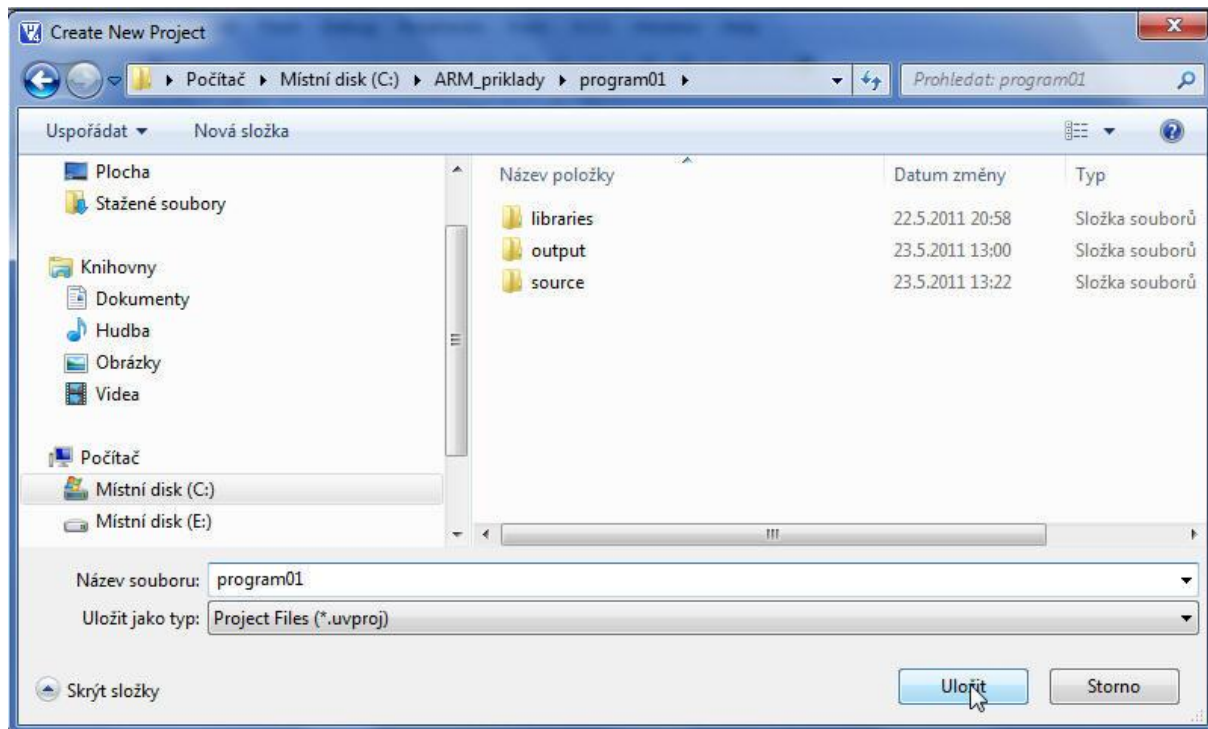
V menu vybereme **Project** → **New μ Vision Project ...**



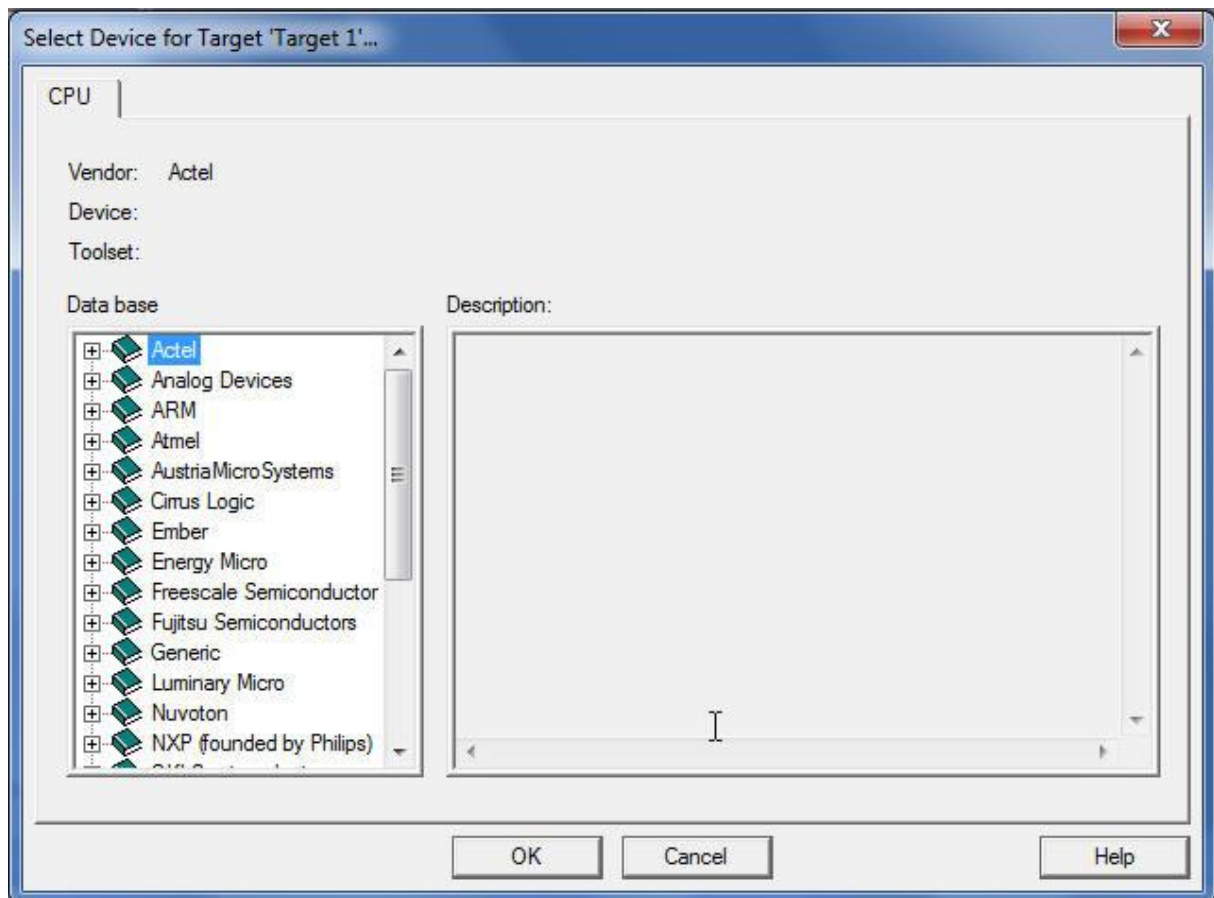
Dostaneme okno



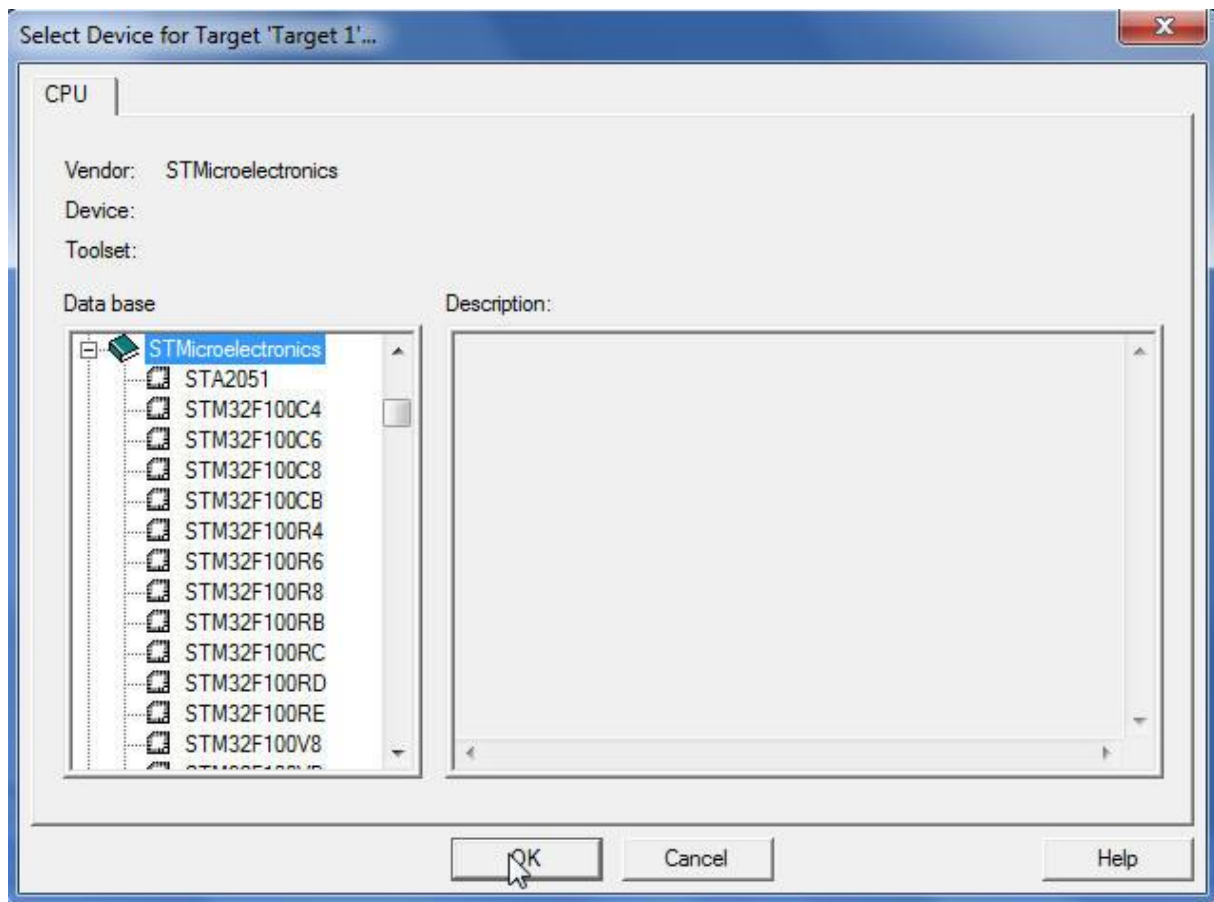
Zvolíme název souboru



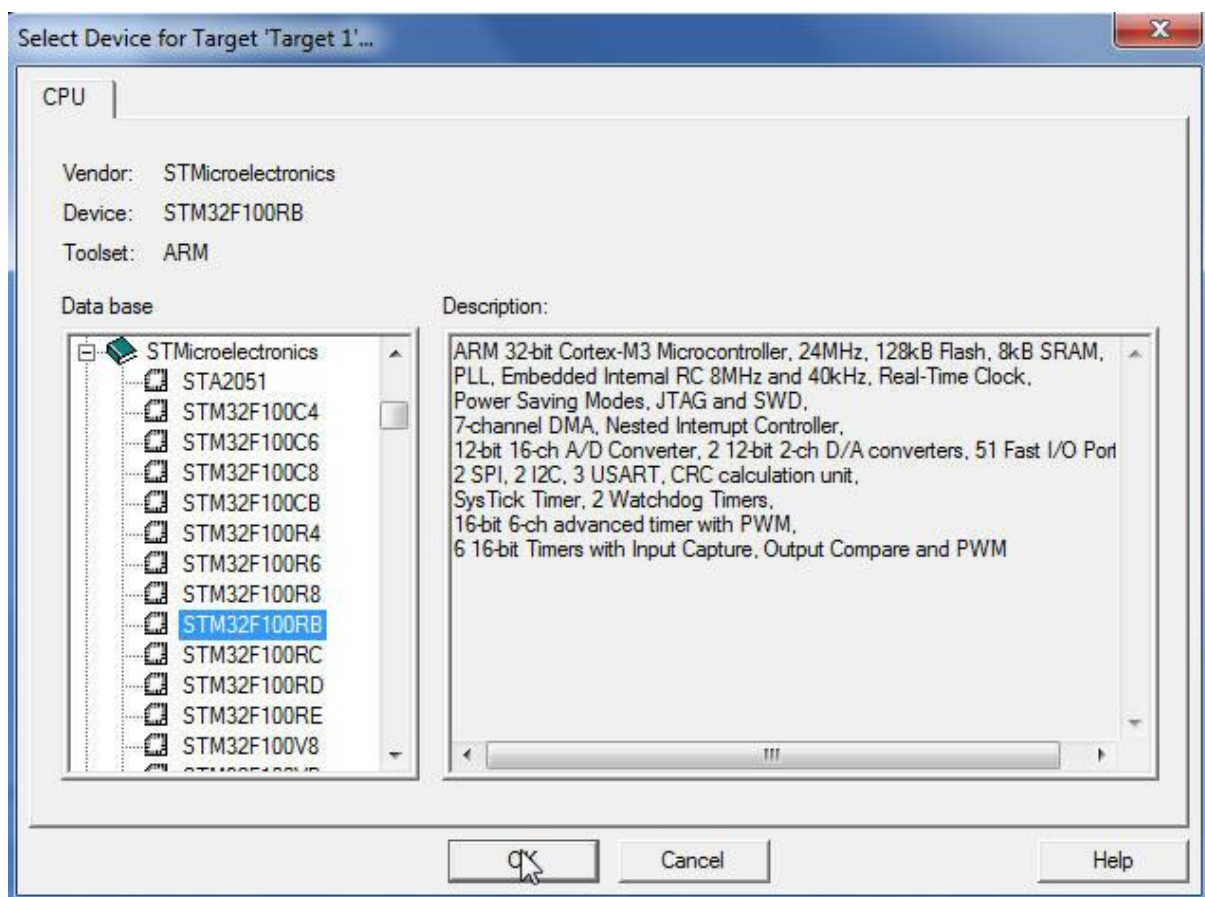
A potvrdíme tlačítkem **Uložit**. Dostaneme



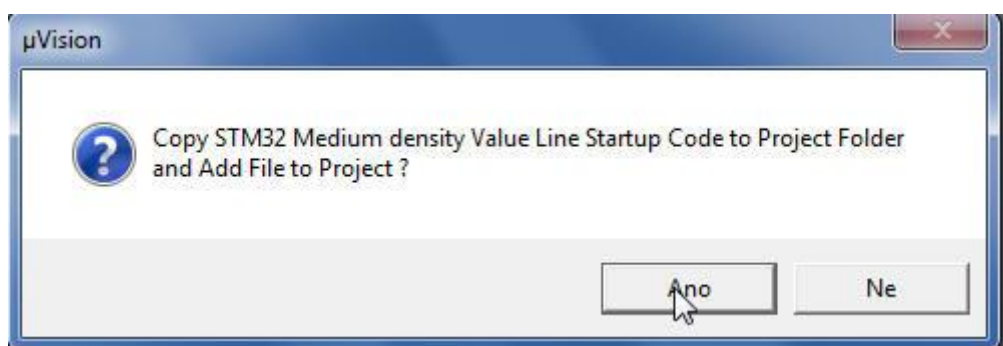
Tj. jsme dotázáni na konkrétní typ CPU. V Data base nejprve vybereme výrobce **STMicroelectronics** A kliknutím na křížek vlevo od **ikonky STMicroelectronics** dostaneme již seznam jednotlivých čipů



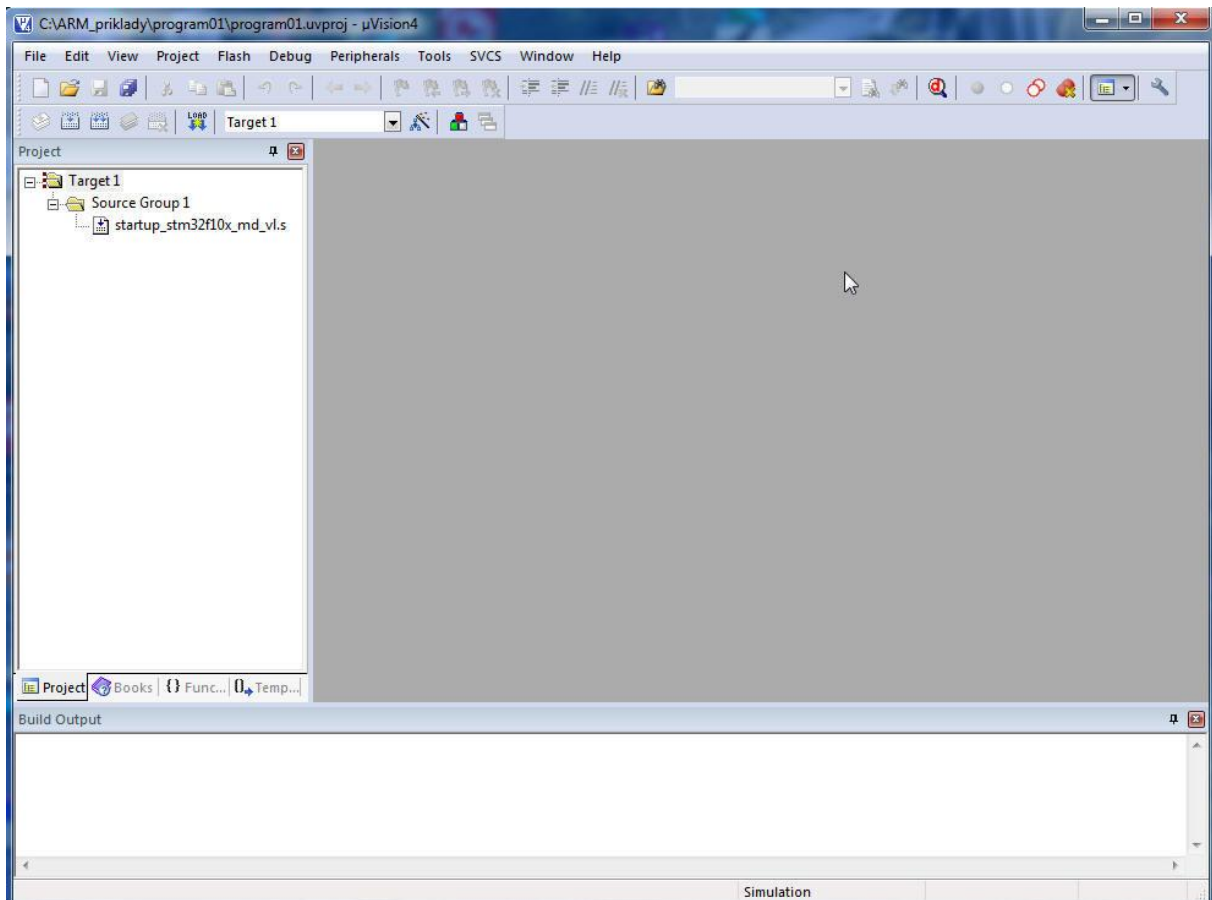
Vybereme **STM32F100RB**



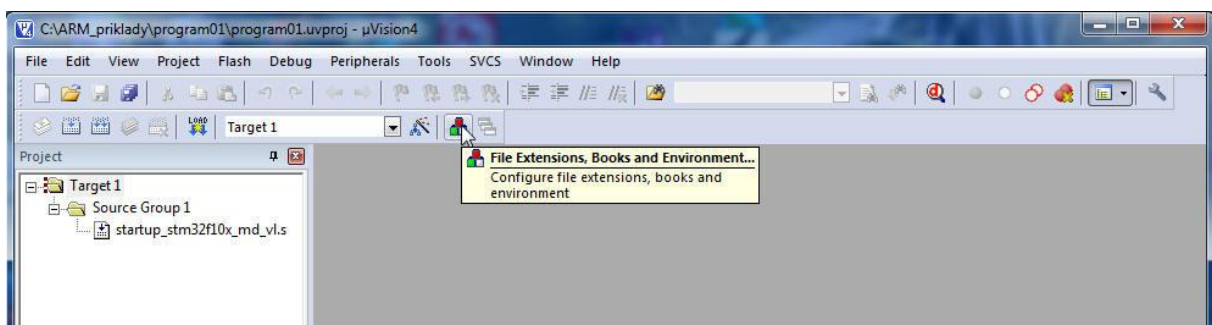
A výběr potvrdíme tlačítkem **OK**. Dostaneme dotaz



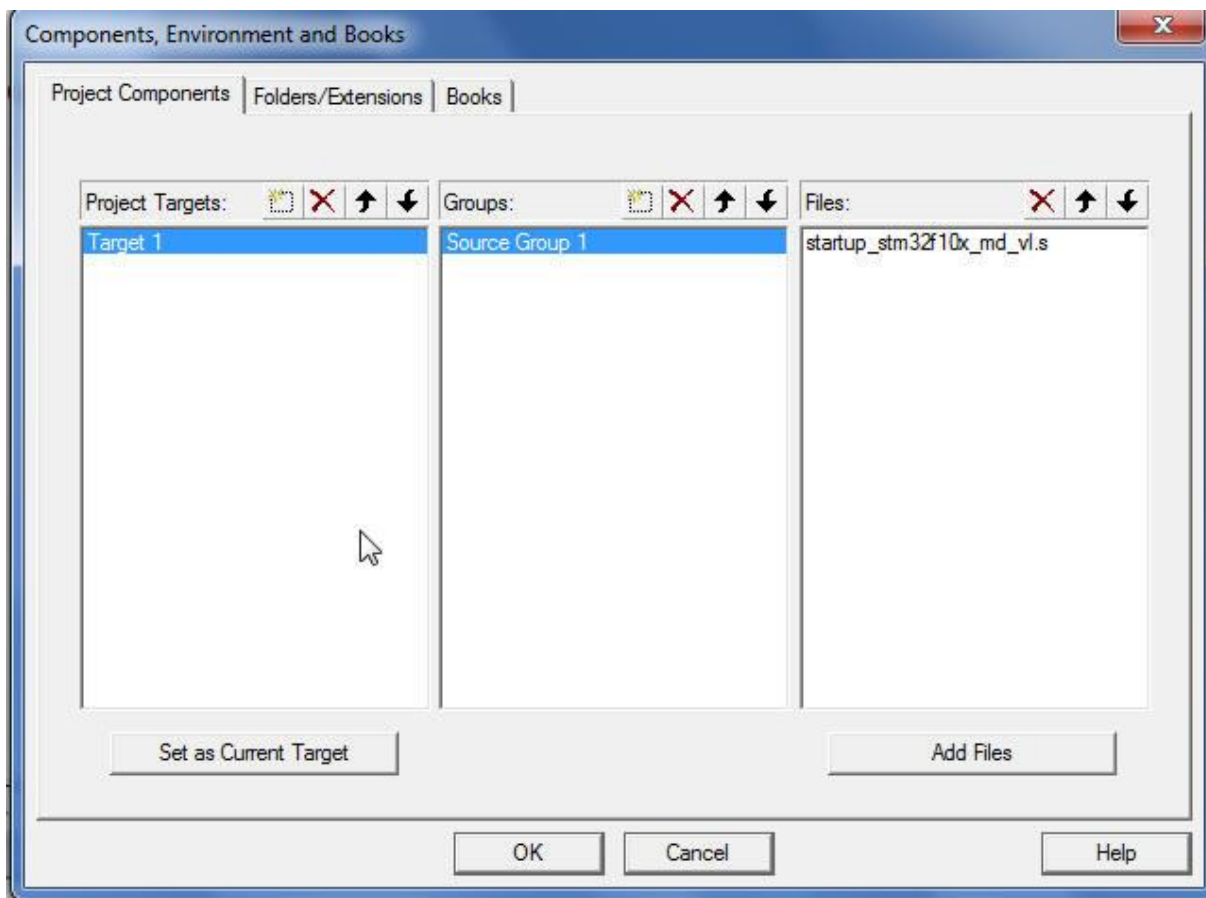
Potvrdíme tlačítkem **Ano**, dostaneme



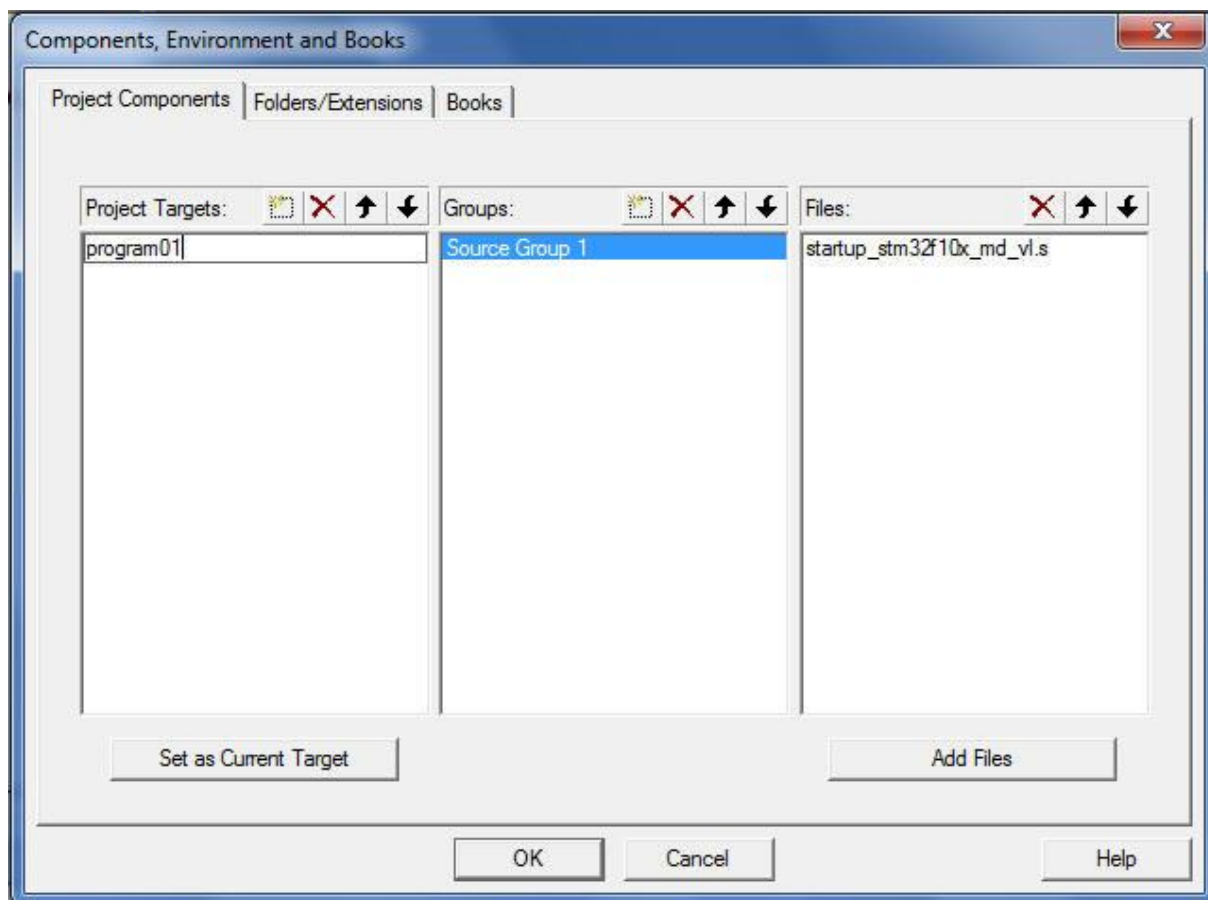
V podokně **project** se nám zobrazila struktura projektu. Nyní tuto strukturu upravíme, tj přejmenujeme název projektu **Target 1** na **program01**, složku **Source Group 1** přejmenujeme na **startup**. Dále vytvoříme složky **source**, **output** a **libraries**. Klikneme proto na ikonku **File Extensions, Books and Environment**



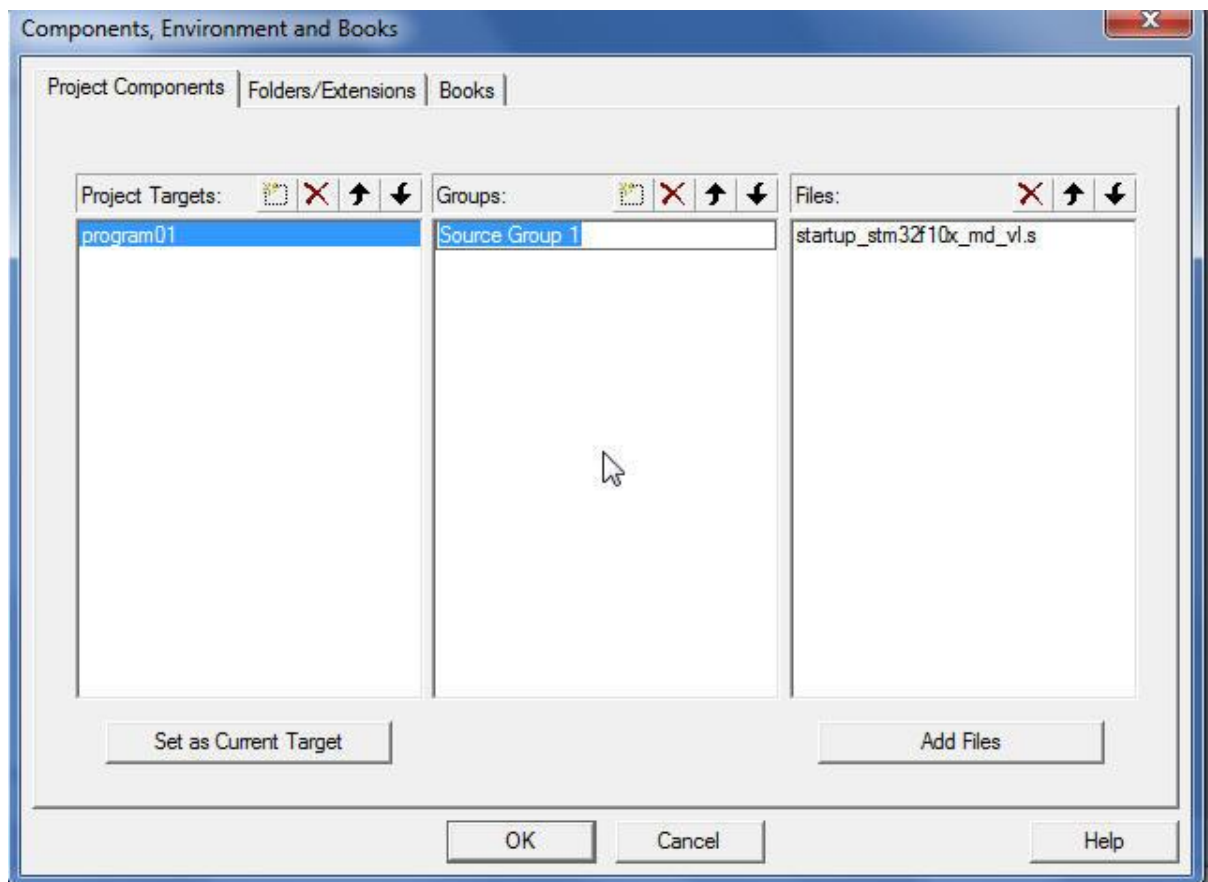
Dostaneme



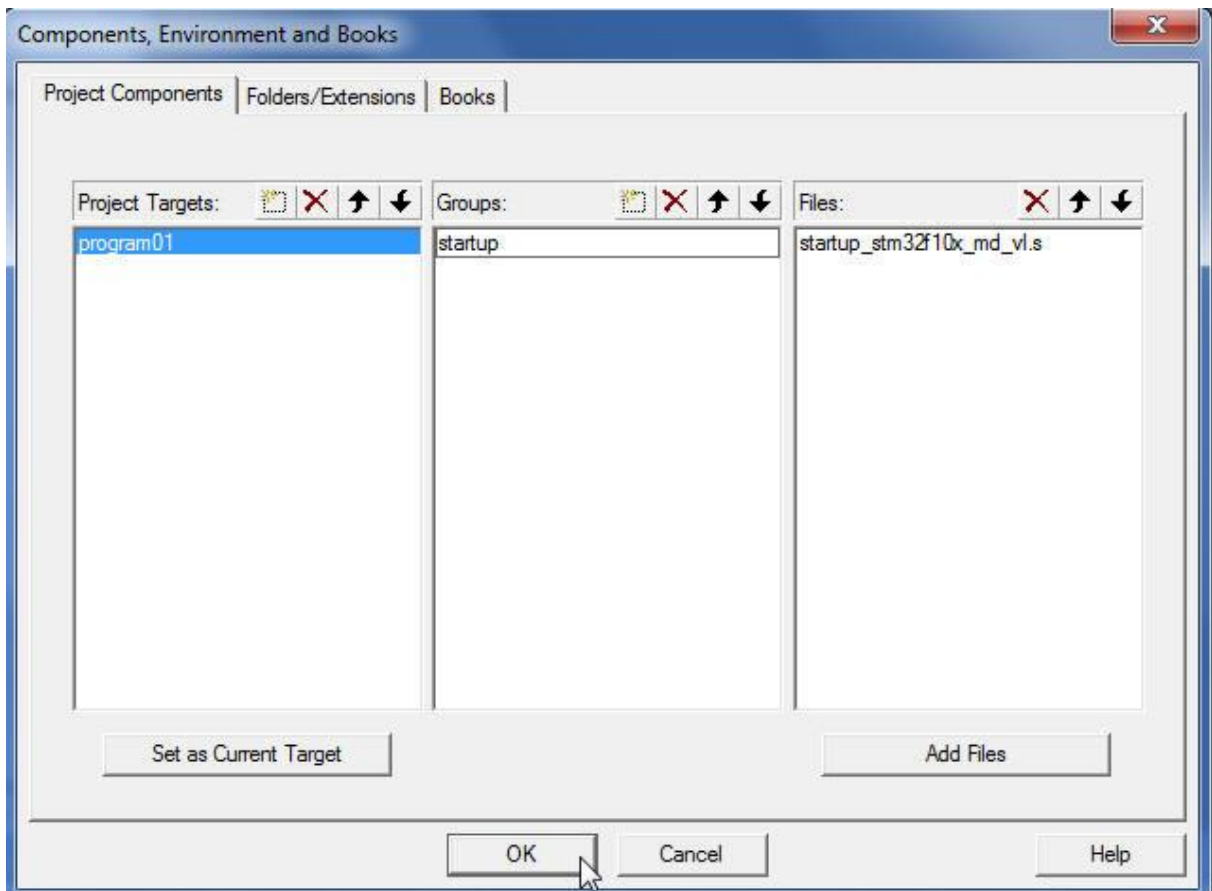
Kurzor umístíme (popř. ponecháme) v políčku **Project Targets** a zmáčkneme klávesu **F2**. Nyní již můžeme název projektu přejmenovat



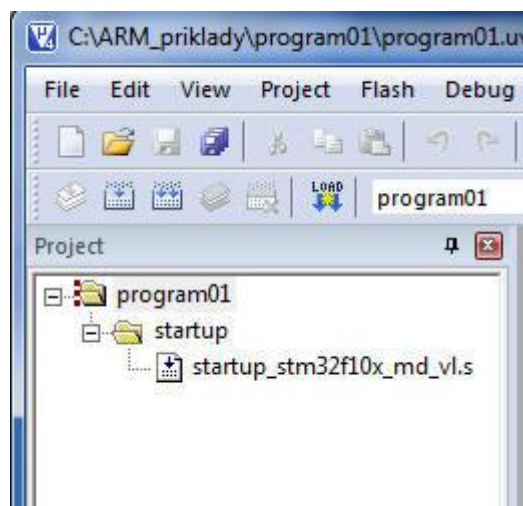
Obdobně přemístíme kurzor do políčka **Groups** a stiskneme klávesu **F2**



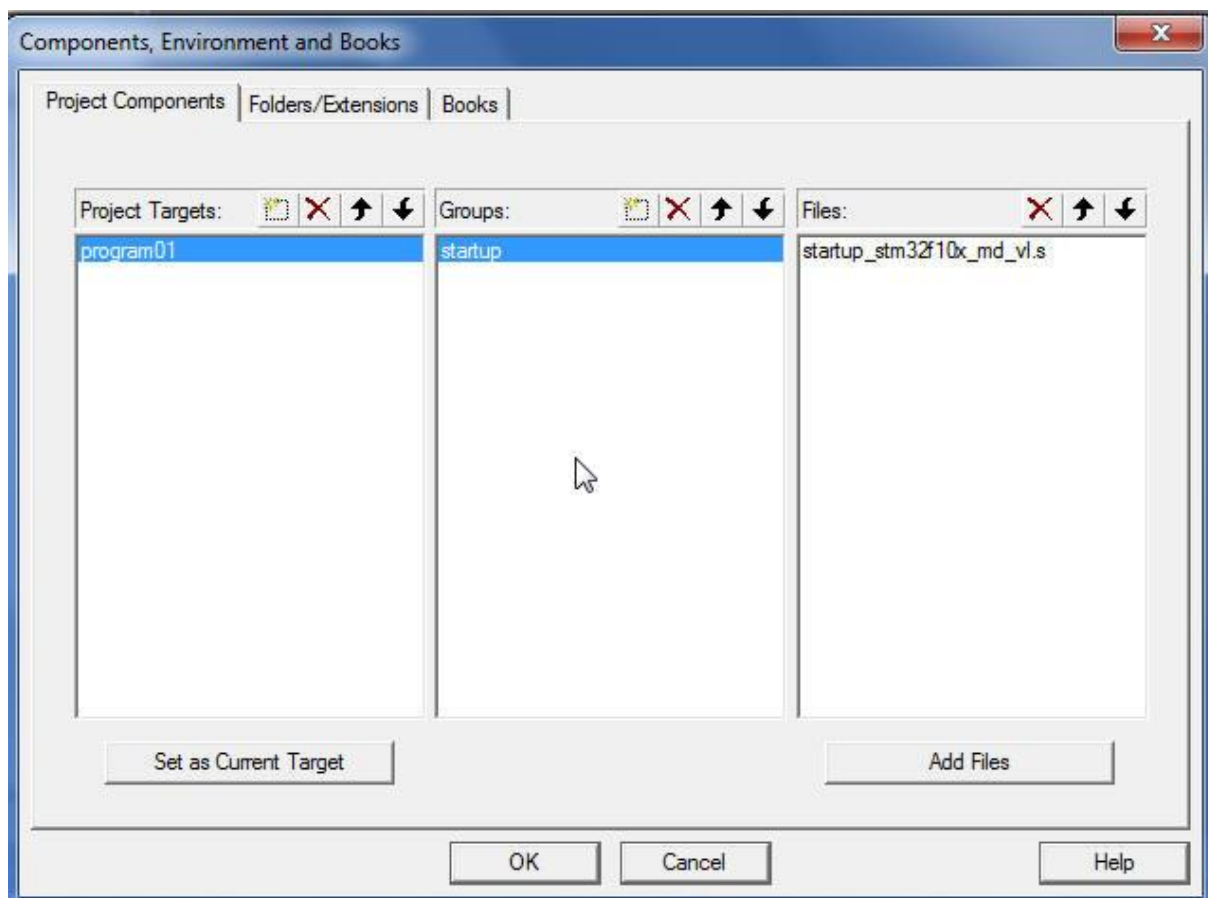
A adresář přejmenujeme na **startup**



A potvrdíme vše tlačítkem **OK**. Nyní máme strukturu



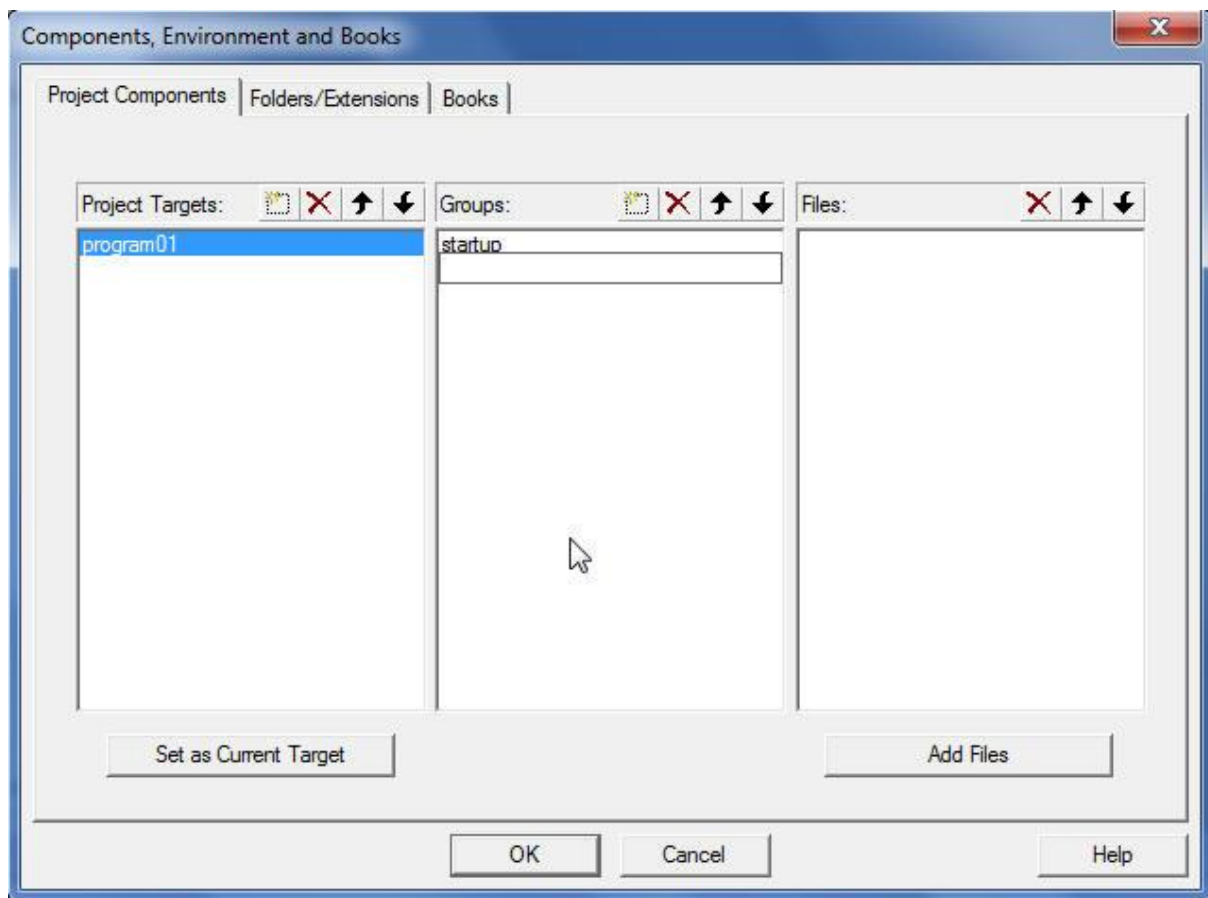
Nyní vytvoříme další adresáře struktury projektu. Proto znovu klikneme na ikonku **File Extensions, Books and Enviroment**



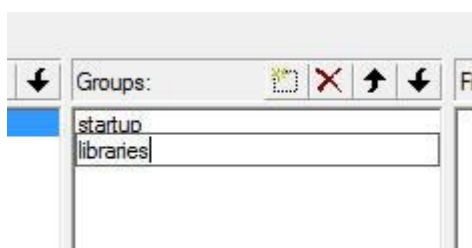
V části **Groups** (i **Project Targets**) si všimněme ikonek



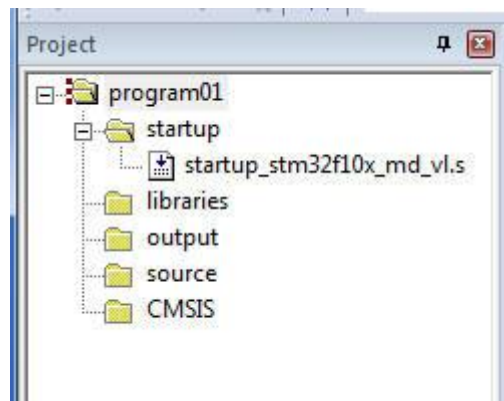
Umožňují přidávat, mazat a měnit pořadí složek. Klikneme na nejlevější ikonku



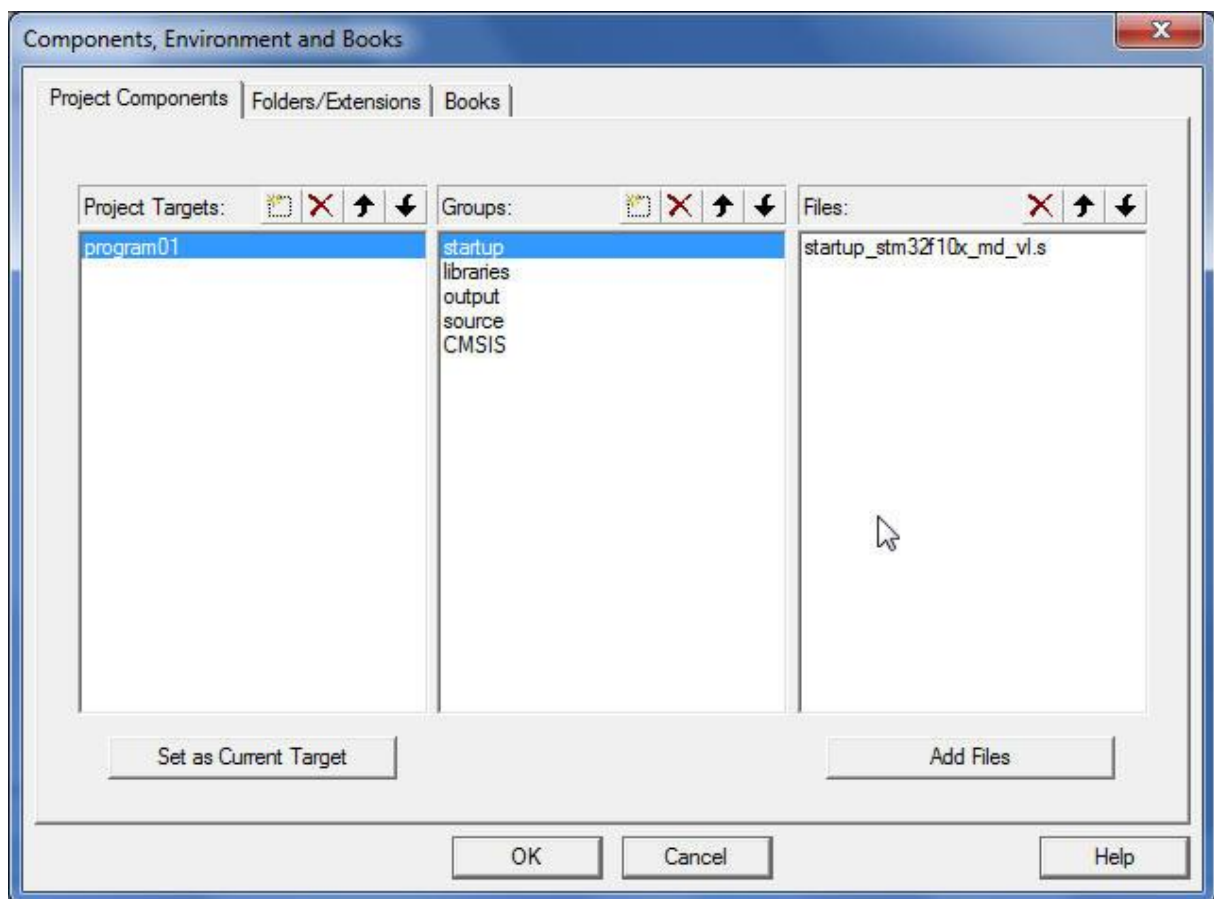
Nyní napíšeme jméno vytvářené složky



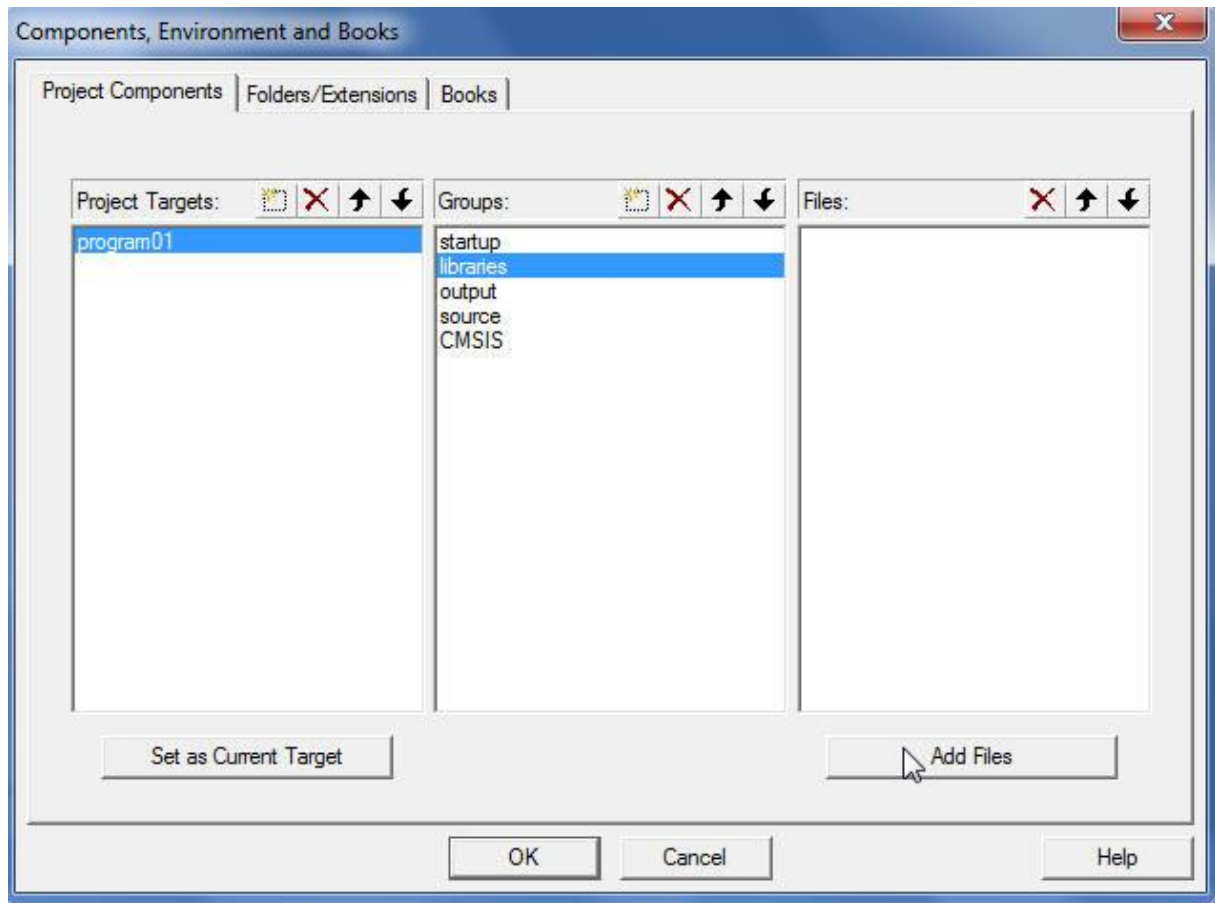
A stiskneme **Enter**. Tím máme tuto položku vytvořenou a opět klikneme na ikonku vytvořit a vytvoříme položku **output** a poté ještě **source** a **CMSIS**. Potvrdíme vše tlačítkem **OK**. Dostaneme



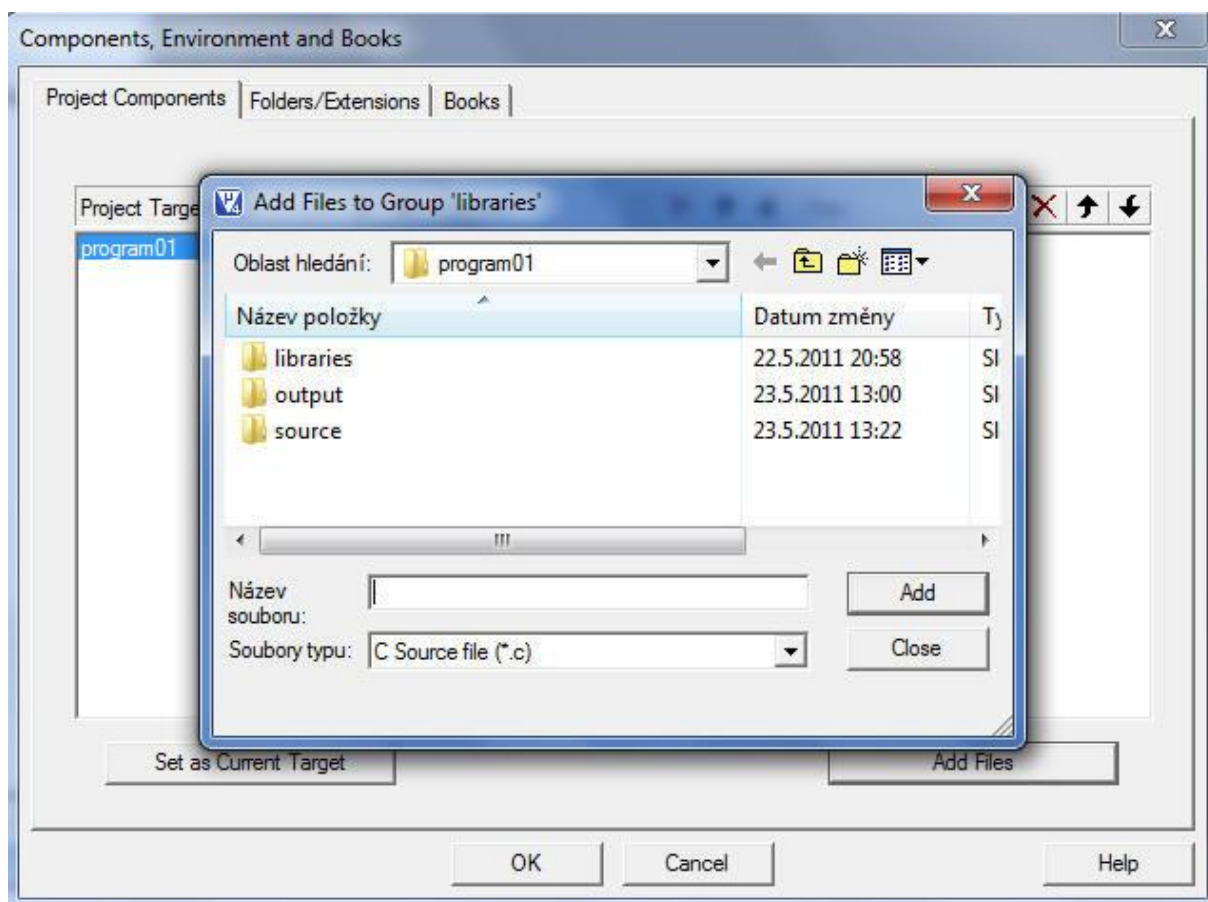
Ještě potřebujeme do adresářů projektu soubory. Takže opět spustíme **File Extensions, Books and Environment**



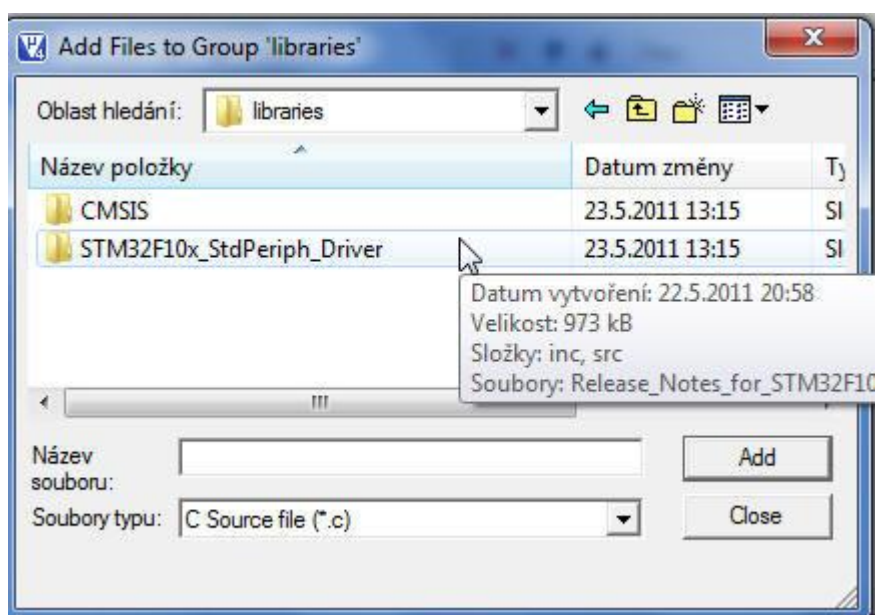
Označíme složku **libraries** a poté klikneme na tlačítko **Add Files**



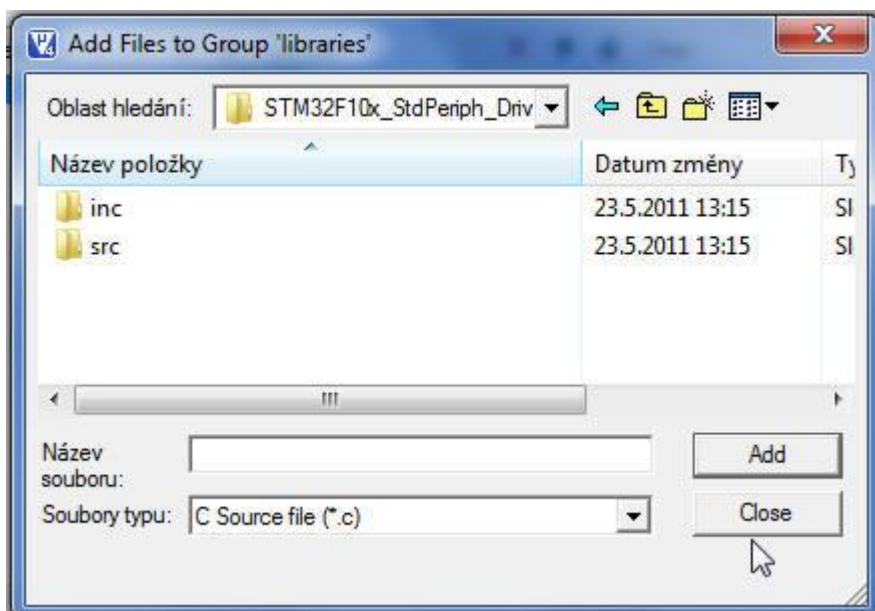
Dostaneme



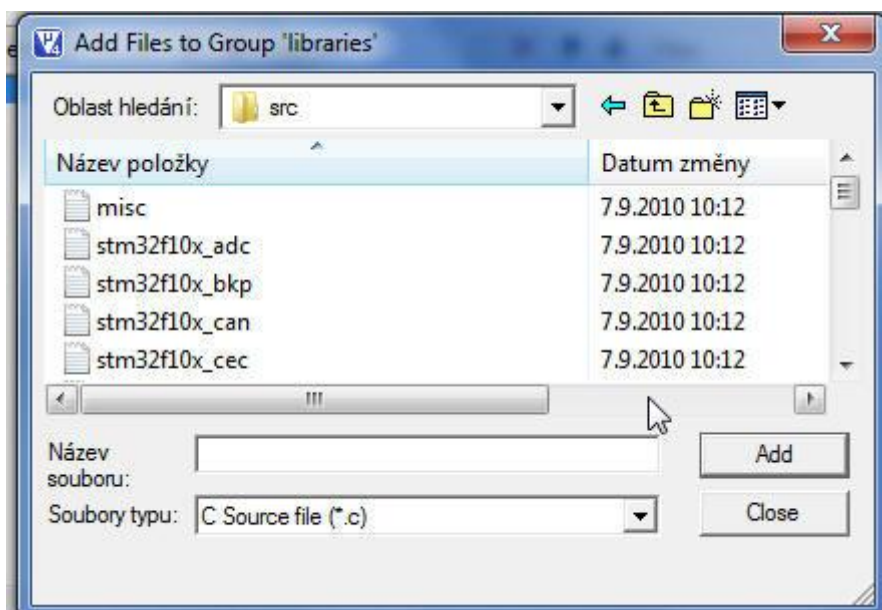
Dále v okně **Add Files to Group libraries** klikneme na položku **libraries**



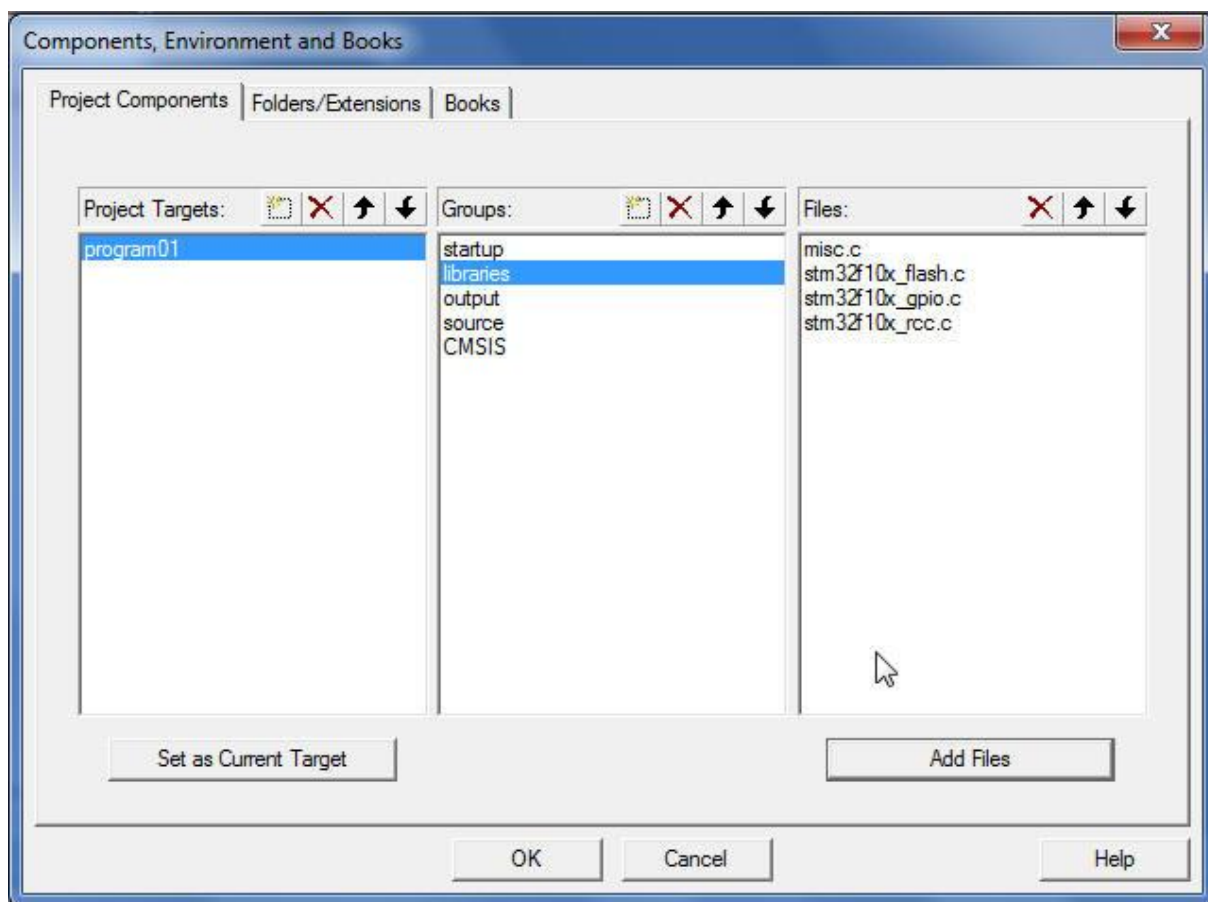
Nyní vybereme **STM32F10x_StdPeriph_Driver**



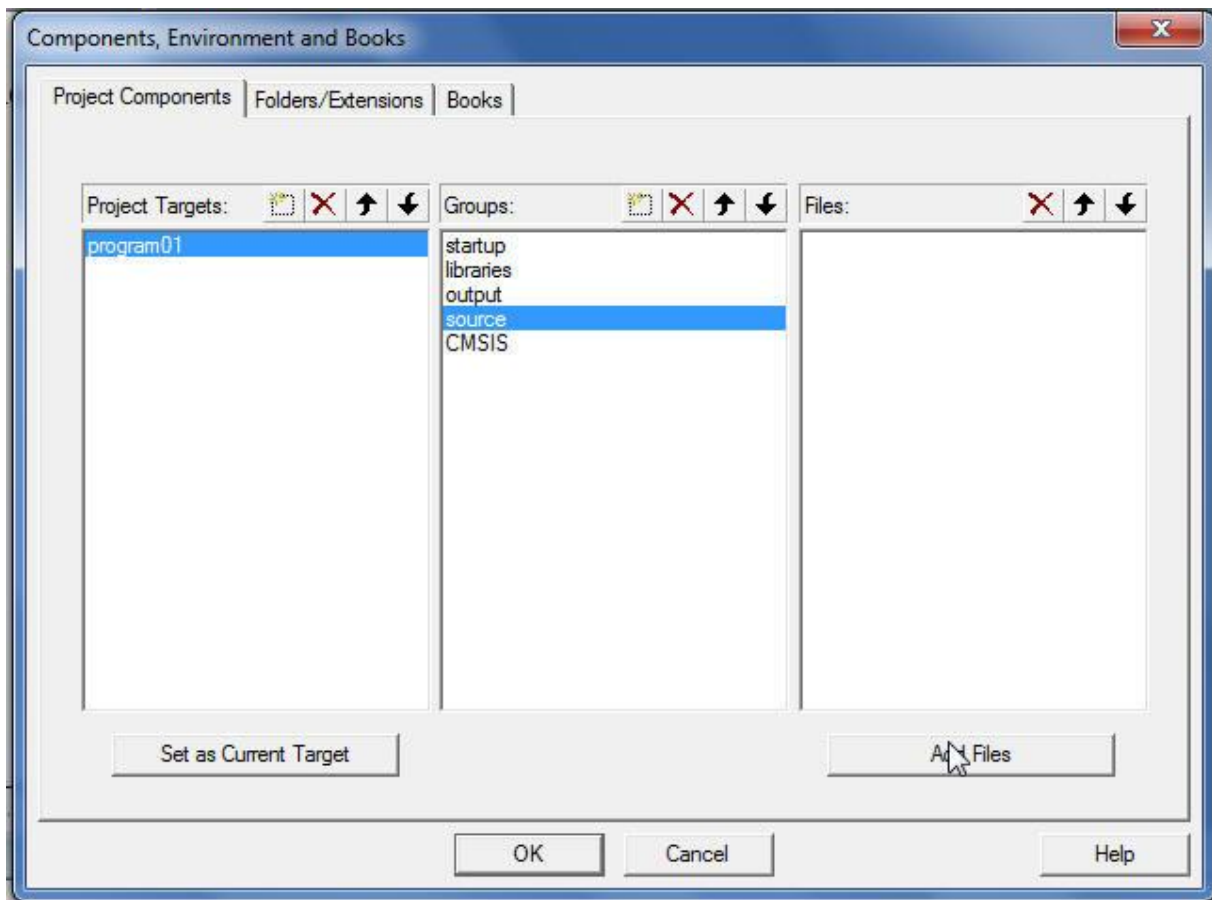
A konečně vybereme položku **src**



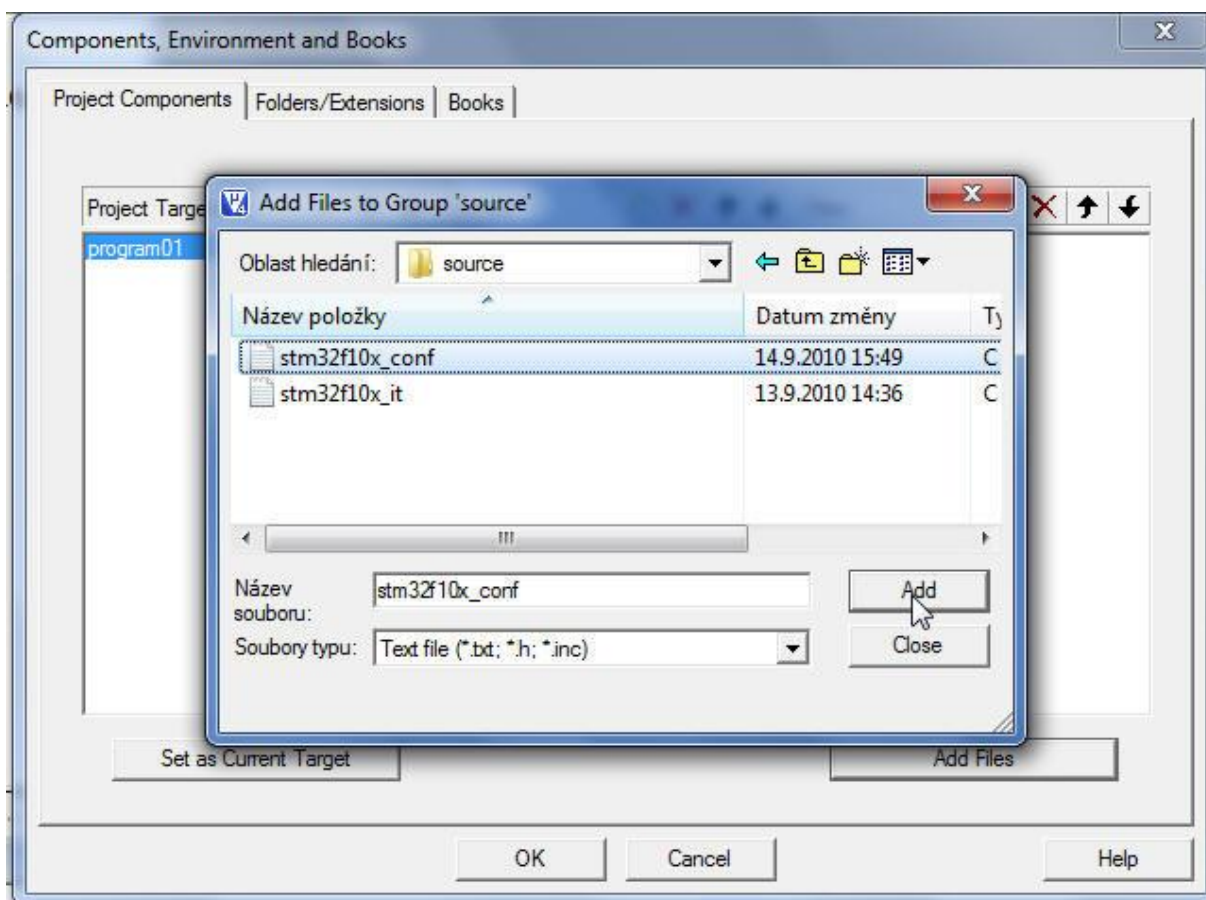
Tak jsme se konečně dopracovali k cíli a budeme vybírat potřebné knihovní soubory. Vybereme vždy soubor a potvrdíme tlačítkem **Add**. Takto vybereme následující čtyři soubory **misc.c**, **stm32f10x_flash.c**, **stm32f10x_gpio.c** a **stm32f10x_rcc.c** a nakonec stiskneme tlačítko **Close**. Dostaneme



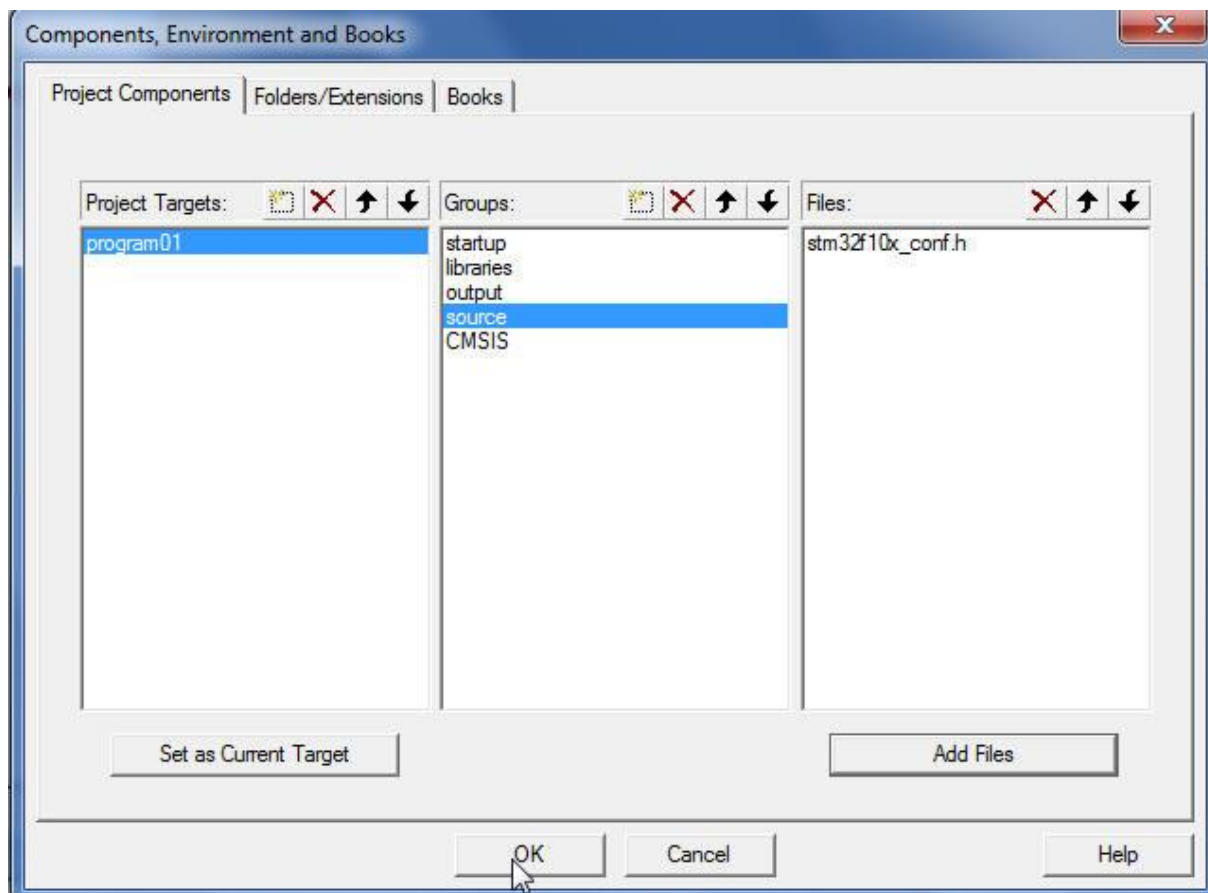
Dále přejdeme ke složce **source**



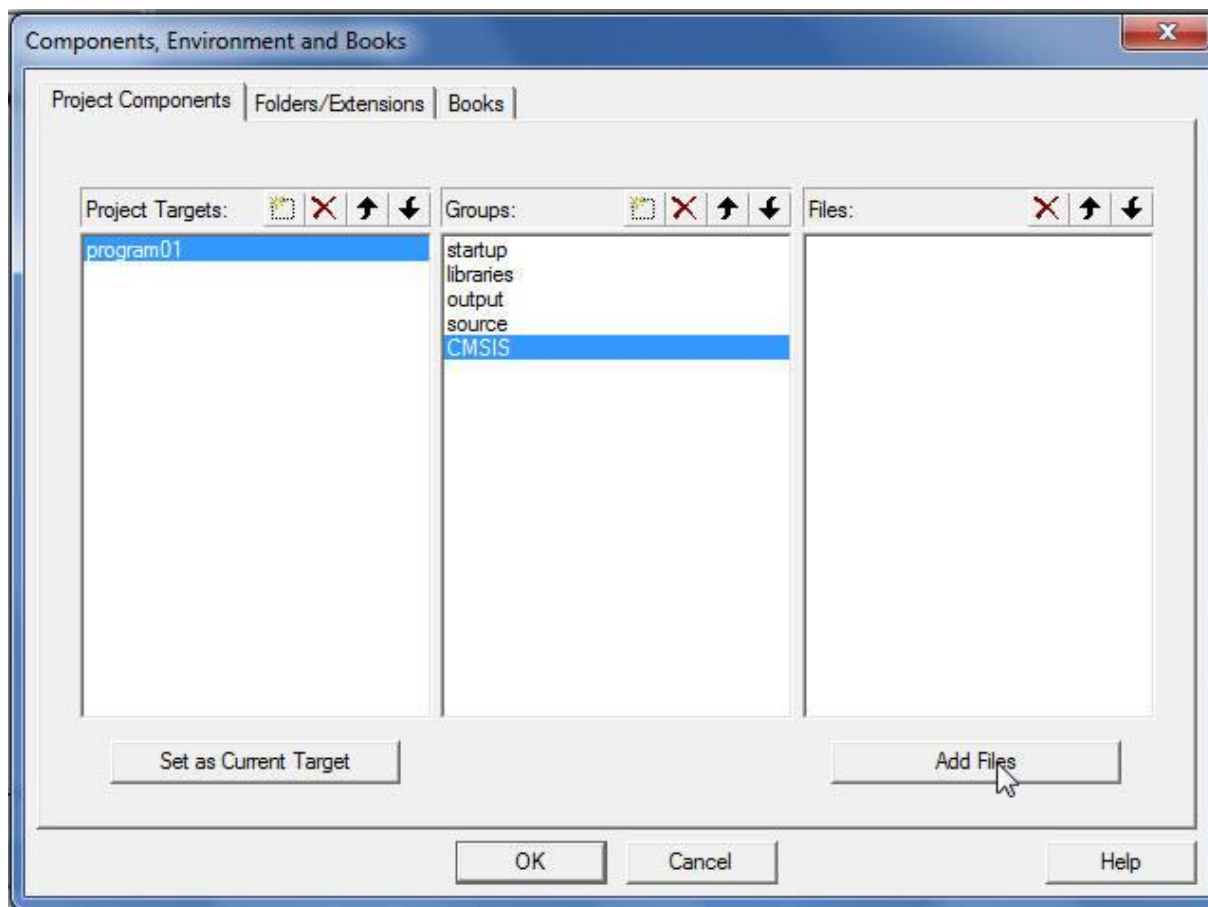
Klikneme na **Add Files**



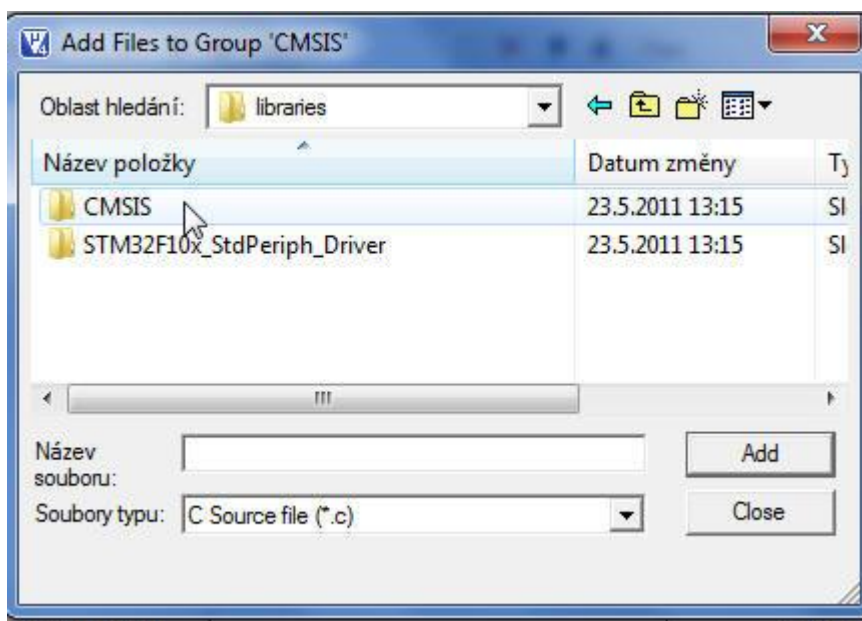
A vybereme soubor **stm32f10x_conf.h** a přidáme ho tlačítkem **Add** a poté klikneme na **Close**



Nakonec přidáme soubory do složky **CMSIS**



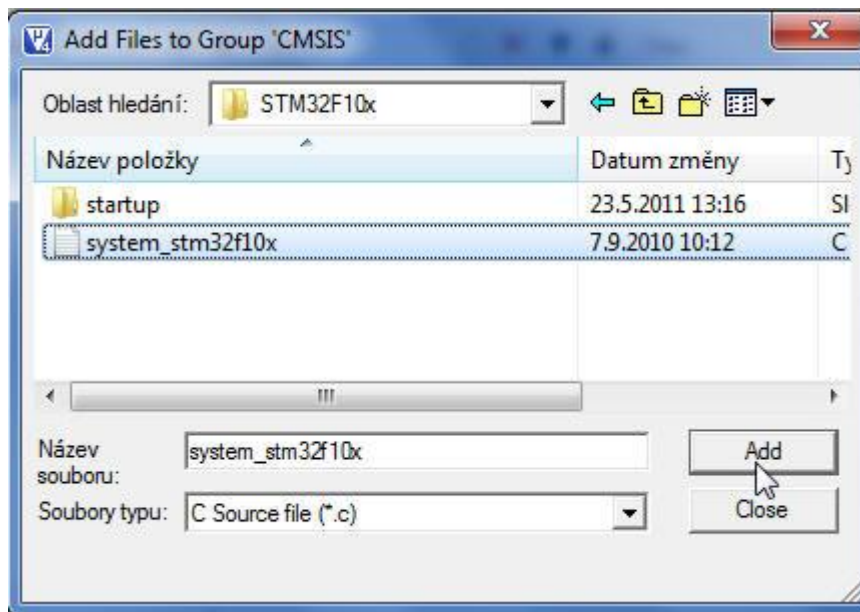
Klikneme na **Add Files** a v podadresáři **libraries** najdeme adresáře **CMISS** a **ATM32F10x_StdPeriph_Driver**



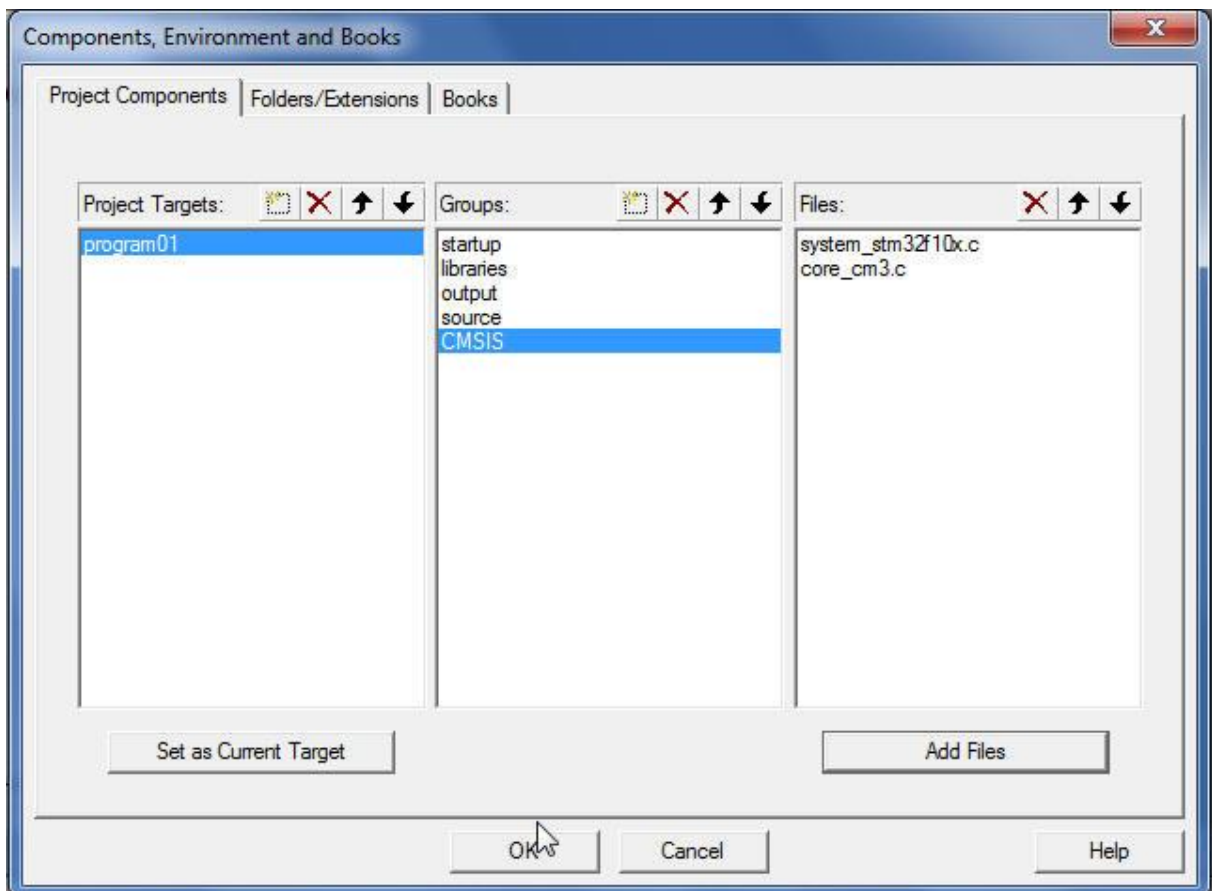
V **CMSIS** vybereme



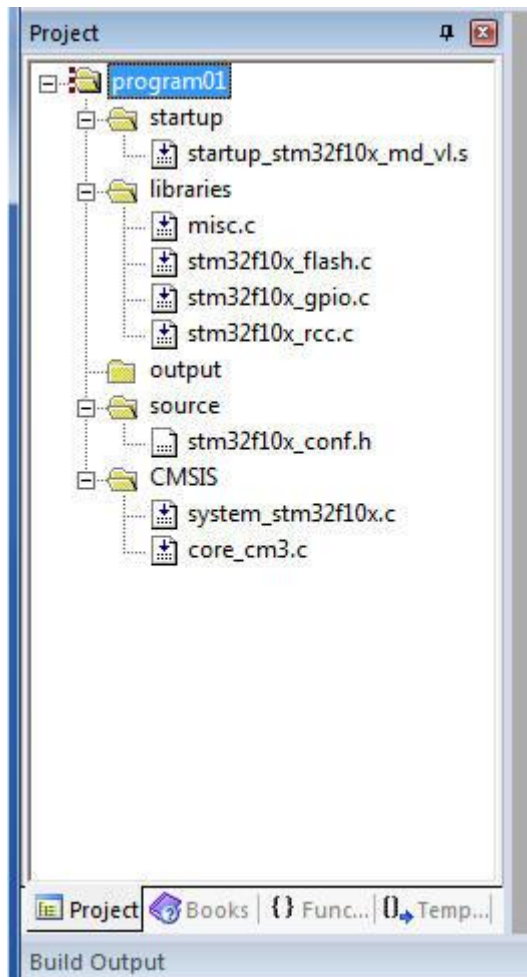
A dále



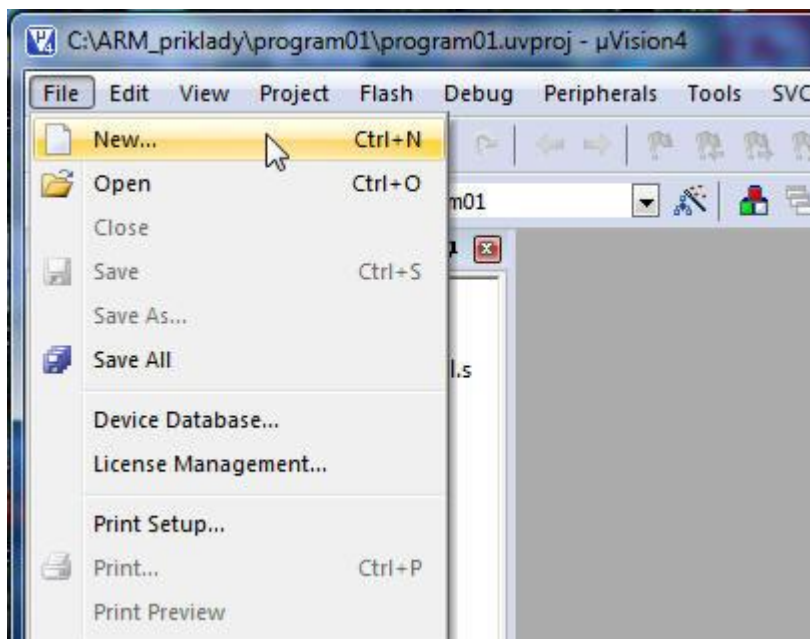
Dostaneme



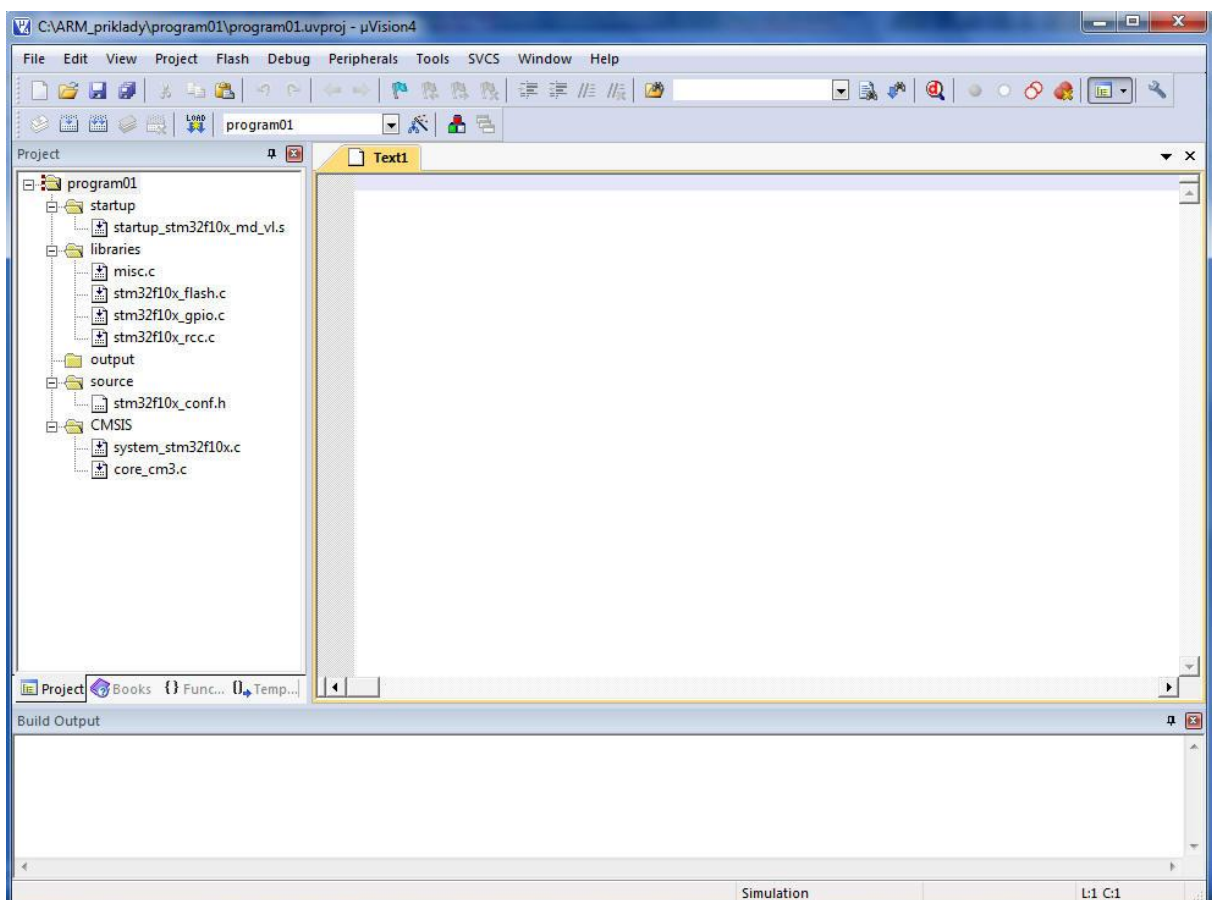
Potvrdíme **OK**



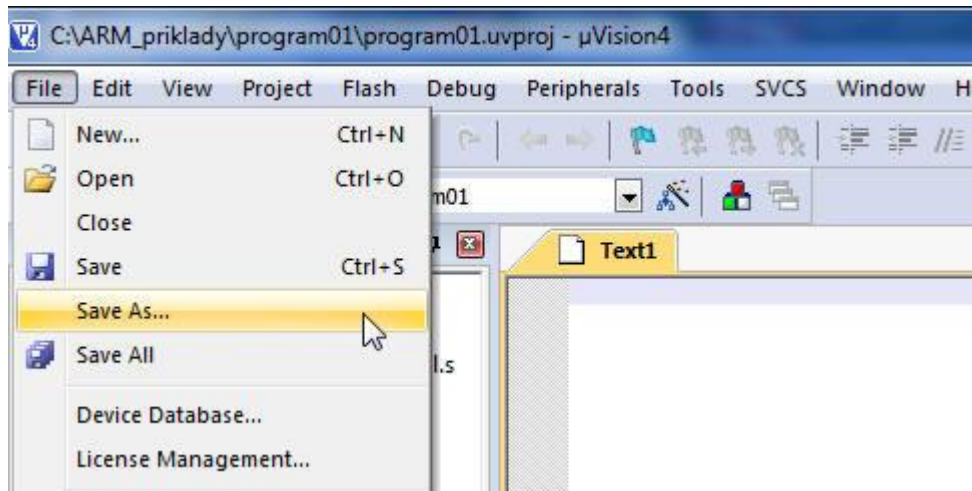
Nyní nám schází poslední soubor, který bude obsahovat zdrojový kód aplikace. V menu vybereme



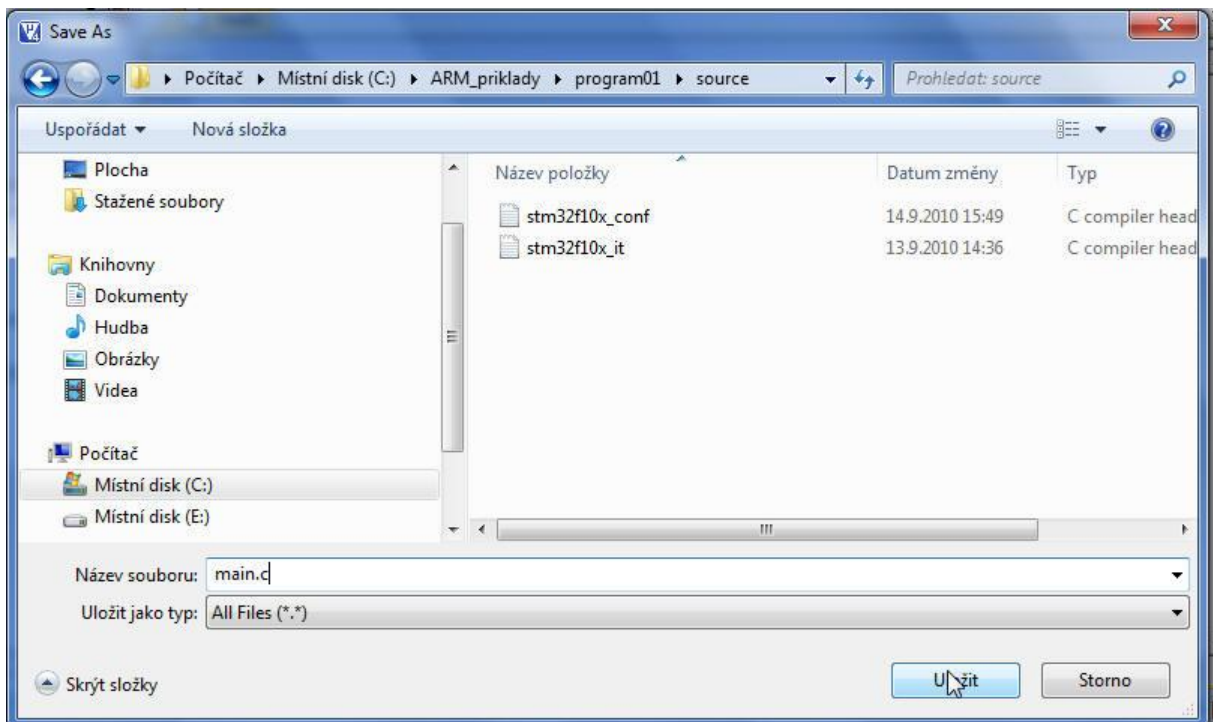
Dostaneme



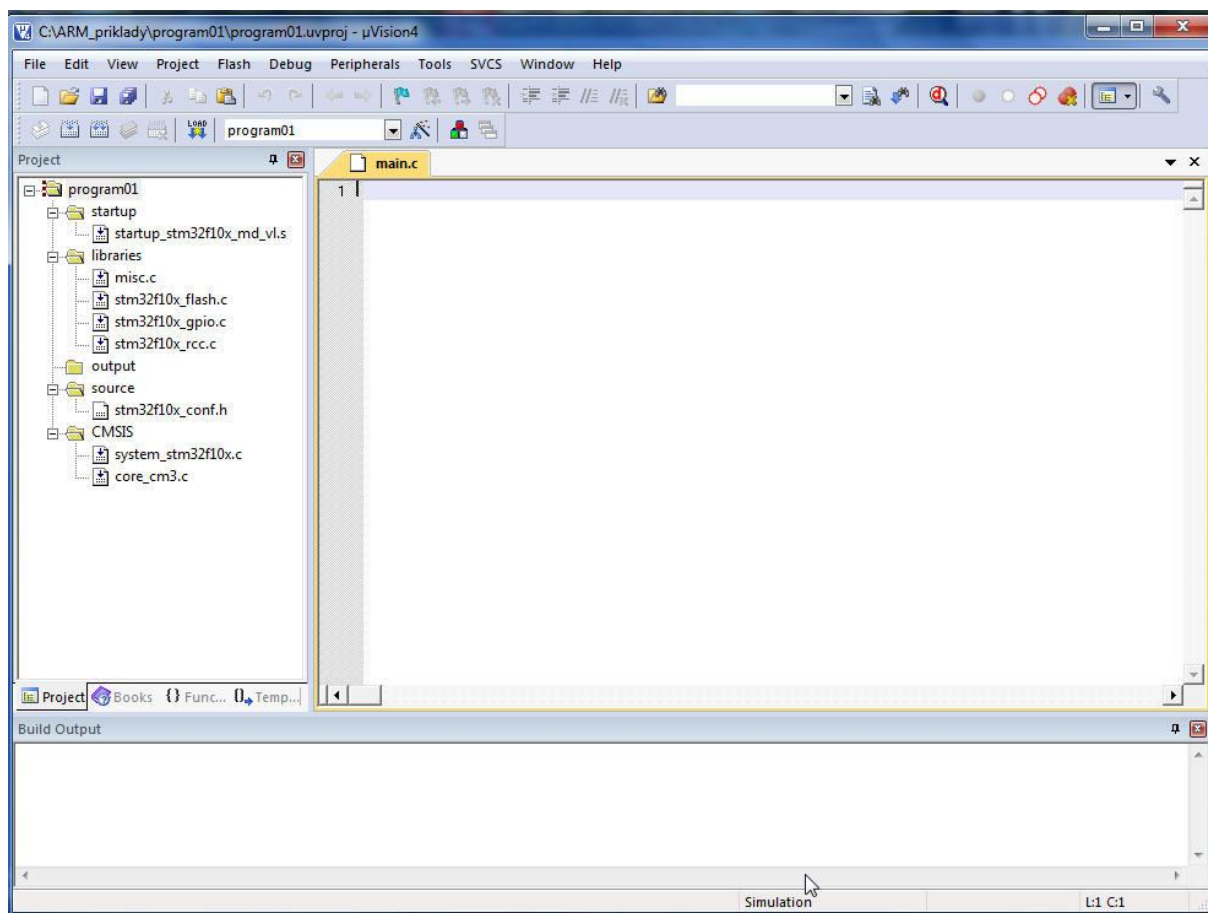
Tento soubor uložíme jako **main.c**



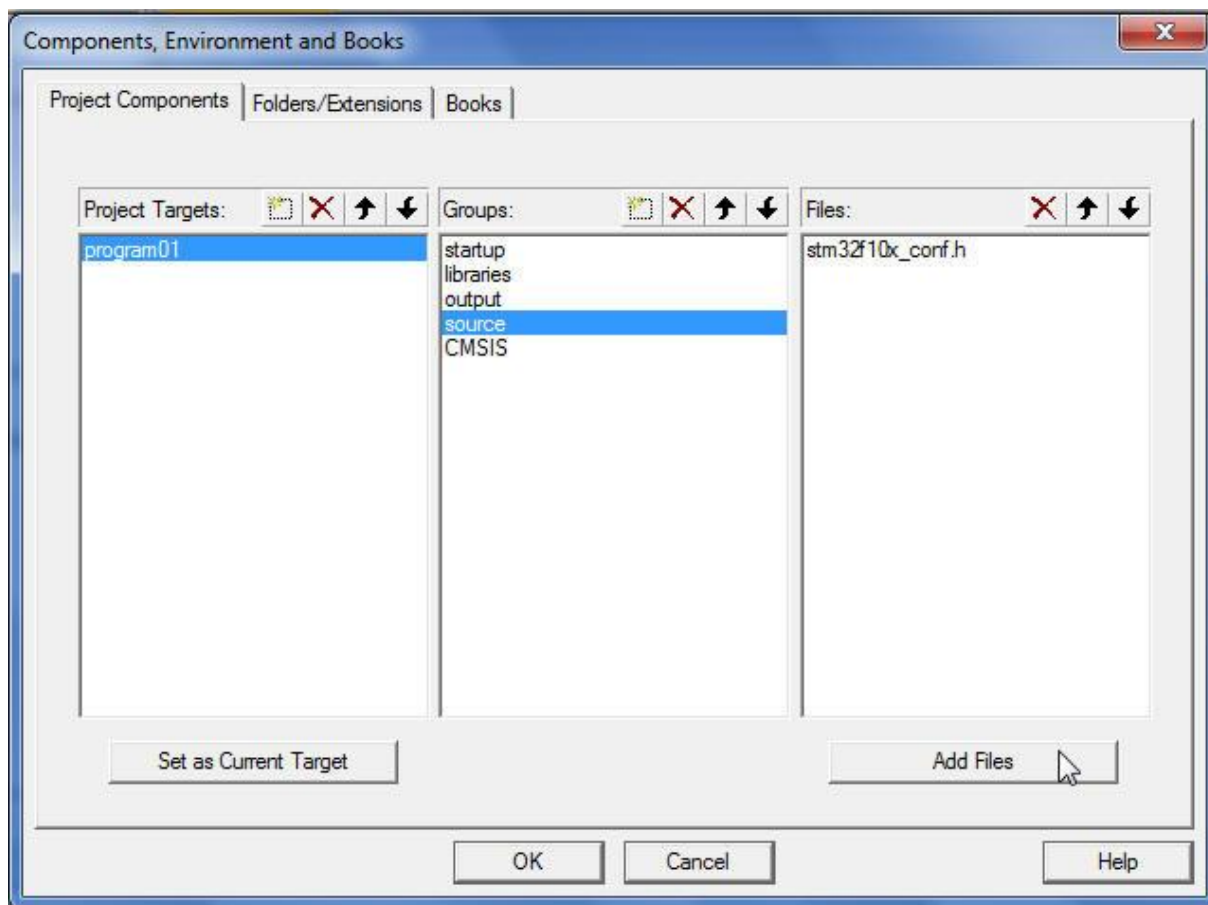
Dostaneme a vyplníme



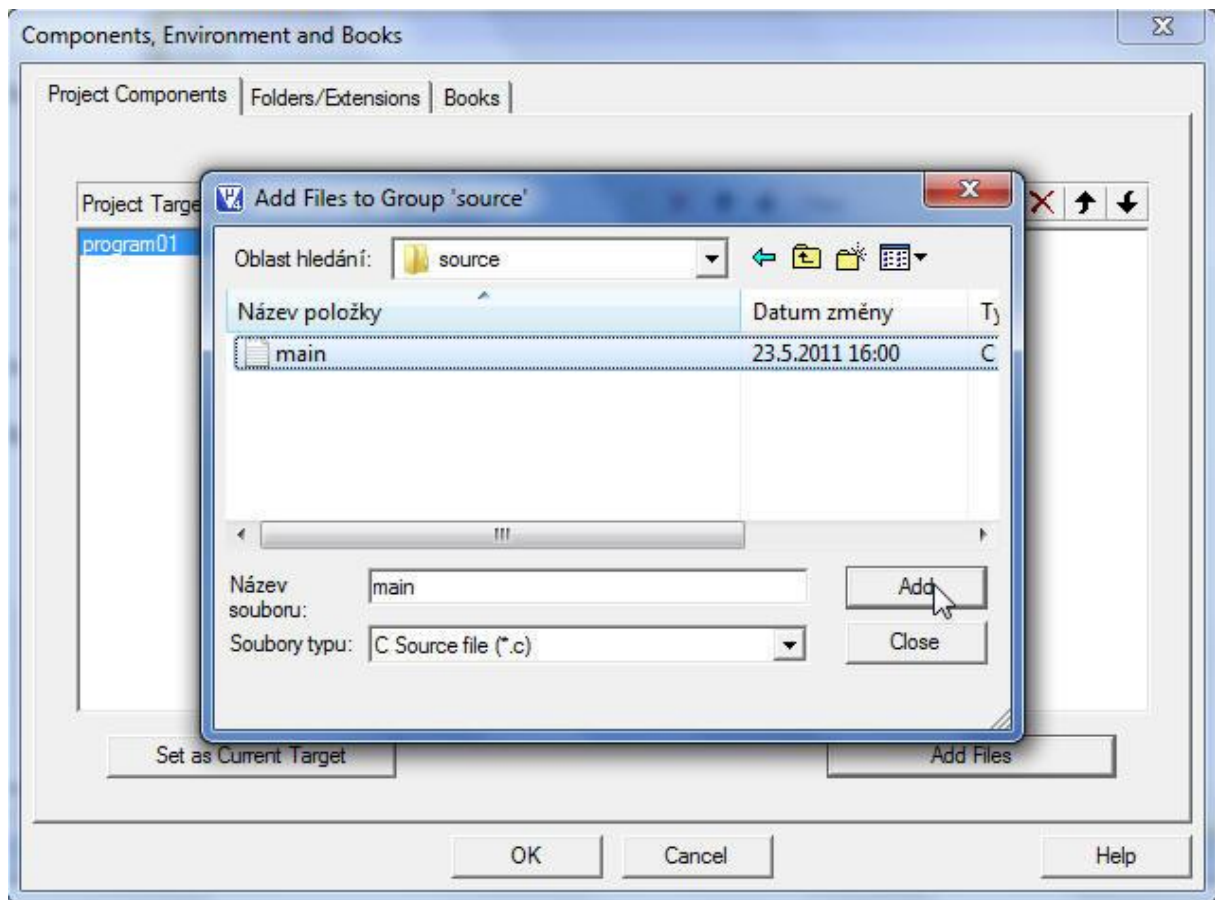
Klikneme na **Uložit**. Tak máme



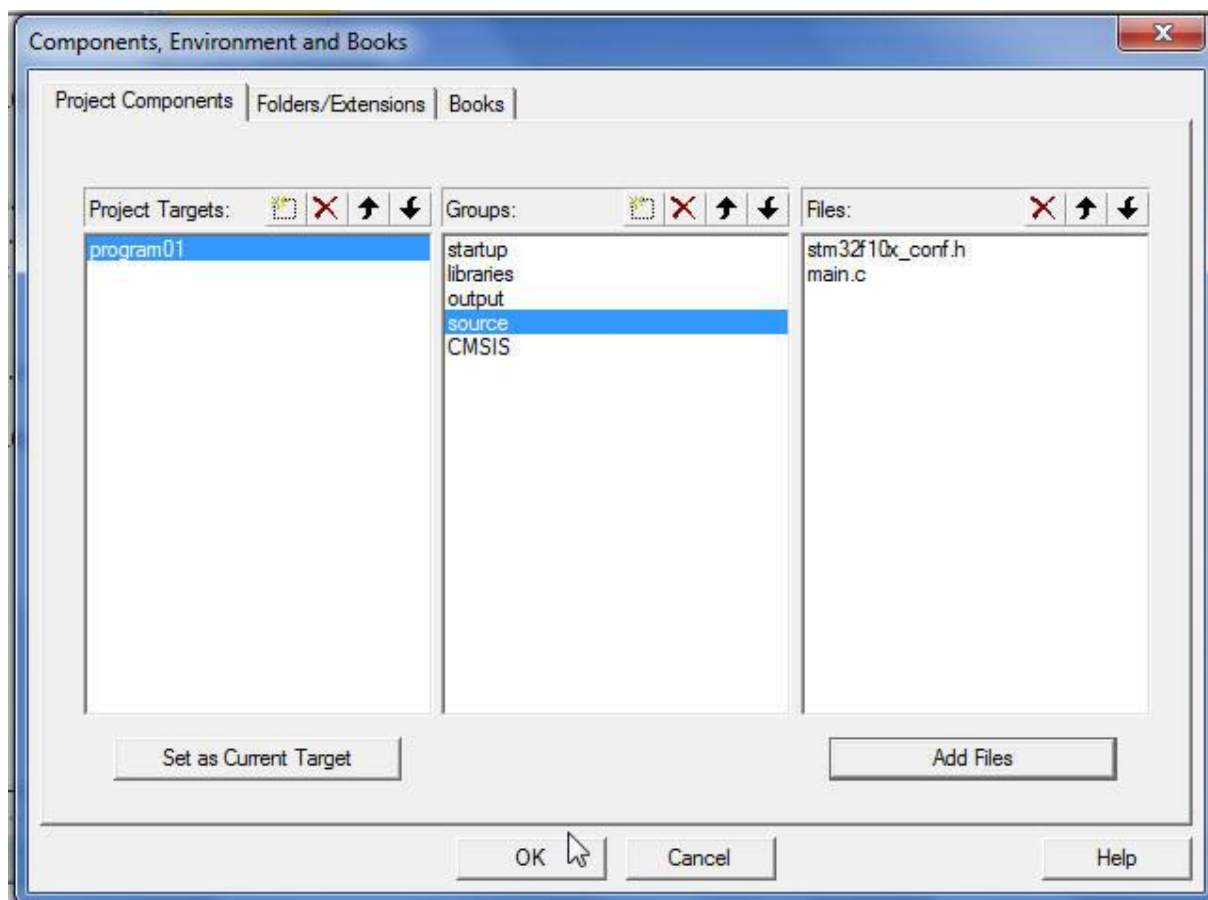
Ještě znovu spustíme **File Extensions, Books and Enviroment**



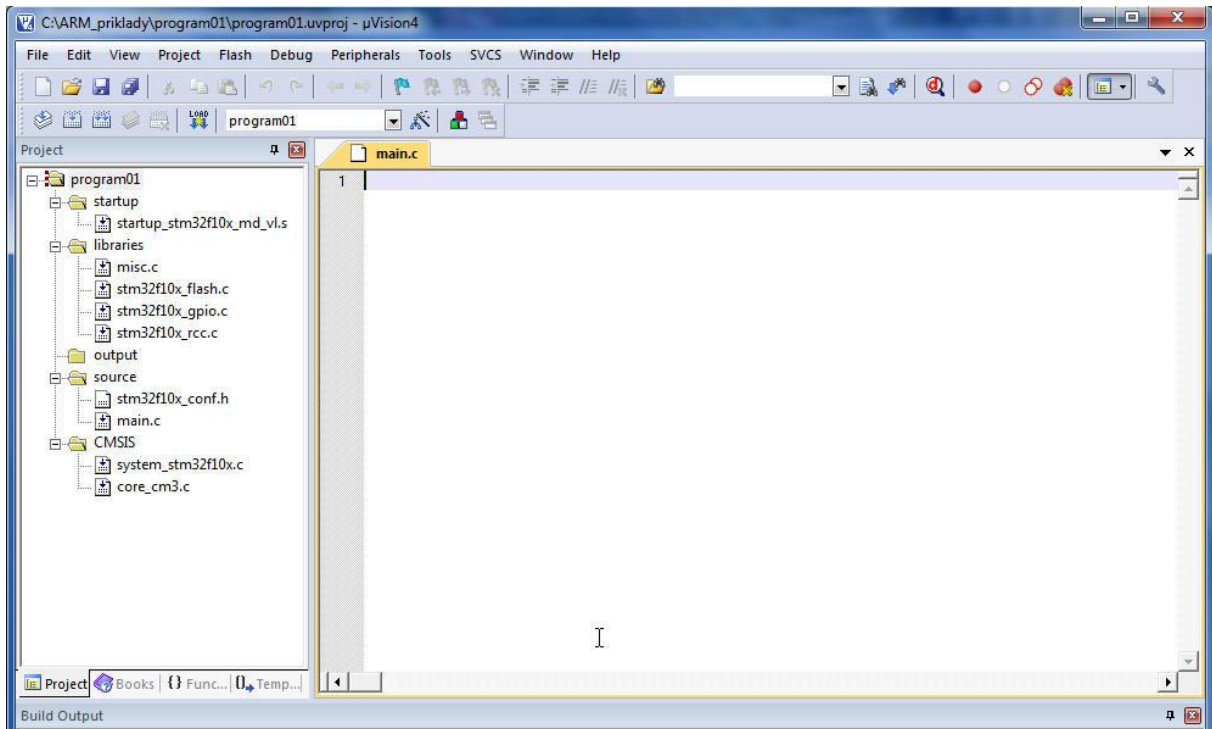
Klikneme na **Add Files**



A přidáme **main.c**



Potvrdíme **OK**. Máme nyní



Do zatím prázdného souboru **main.c** umístíme a uložíme zdrojový kód

```
#include "stm32f10x.h"
//#define HSE //Pouziti externiho oscilatoru
#define HSI //Pouziti interniho oscilatoru
/* funkčni prototypy */
void RCC_Configuration(void);
void GPIO_Configuration(void);
void Delay(vu32 nCount);

/* vstupni funkce programu */
int main(void) {
    /* konfigurace zdroju hodinoveho signalu */
    RCC_Configuration();
    /* konfigurace I/O portu */
    GPIO_Configuration();
    /* blikaci smycka */
    while (1) {
        /* prepneme stav pinu PC9 */
        GPIO_WriteBit(GPIOC, GPIO_Pin_9, (BitAction)(1 -
GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_9)));
        /* cekame */
        Delay(0x5FFFF);
        Delay(0x5FFFF);
        Delay(0x5FFFF);
        Delay(0x5FFFF);
    }
}
```

```

        Delay(0x5FFFF);
        Delay(0x5FFFF);

    }
}

/* nastaveni zdroju hodinoveho signalu (HSE) */
void RCC_Configuration(void) {
#ifdef HSE
    ErrorStatus HSEStartUpStatus;
    /* reset nastaveni */
    RCC_DeInit();
    /* aktivace signalu HSE (External High Speed oscillator) */
    RCC_HSEConfig(RCC_HSE_ON);
    /* kontrola stability oscilatoru */
    HSEStartUpStatus = RCC_WaitForHSEStartUp();
    if (HSEStartUpStatus == SUCCESS) {
        /*Nastaveni delicek HS*/
        RCC_HCLKConfig(RCC_SYSCLK_Div1); //HCLK = 24 MHz, AHB
        RCC_PCLK2Config(RCC_HCLK_Div1); //APB1 = 24 MHz
        RCC_PCLK1Config(RCC_HCLK_Div1); //APB2 = 24 MHz
        /* nastaveni PLL */
        RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_4); //PLLCLK =
24 MHz
    }
#endif
#ifdef HSI
    /* reset nastaveni */
    RCC_DeInit();
    /* aktivace signalu HSI (Internal High Speed oscillator) */
    RCC_HSIcmd(ENABLE);
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET);
    /* nastaveni delicek HS */
    RCC_HCLKConfig(RCC_SYSCLK_Div1); //HCLK = 24 MHz, AHB
    RCC_PCLK1Config(RCC_HCLK_Div1); //APB1 = 24 MHz
    RCC_PCLK2Config(RCC_HCLK_Div1); //APB2 = 24 MHz

    /* nastaveni PLL */
    RCC_PLLConfig(RCC_PLLSource_HSI_Div2, RCC_PLLMul_8); //PLLCLK = 24
MHz
#endif

    /* nastaveni FLASH */
    FLASH_SetLatency(FLASH_Latency_2);
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);

    RCC_PLLCmd(ENABLE);
    while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
    /*Zhroj hodin pro AHB*/
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
    while (RCC_GetSYSCLKSource() != 0x08);
    /* aktivace zdroje HS pro periferie */

```

```

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC , ENABLE);
    }

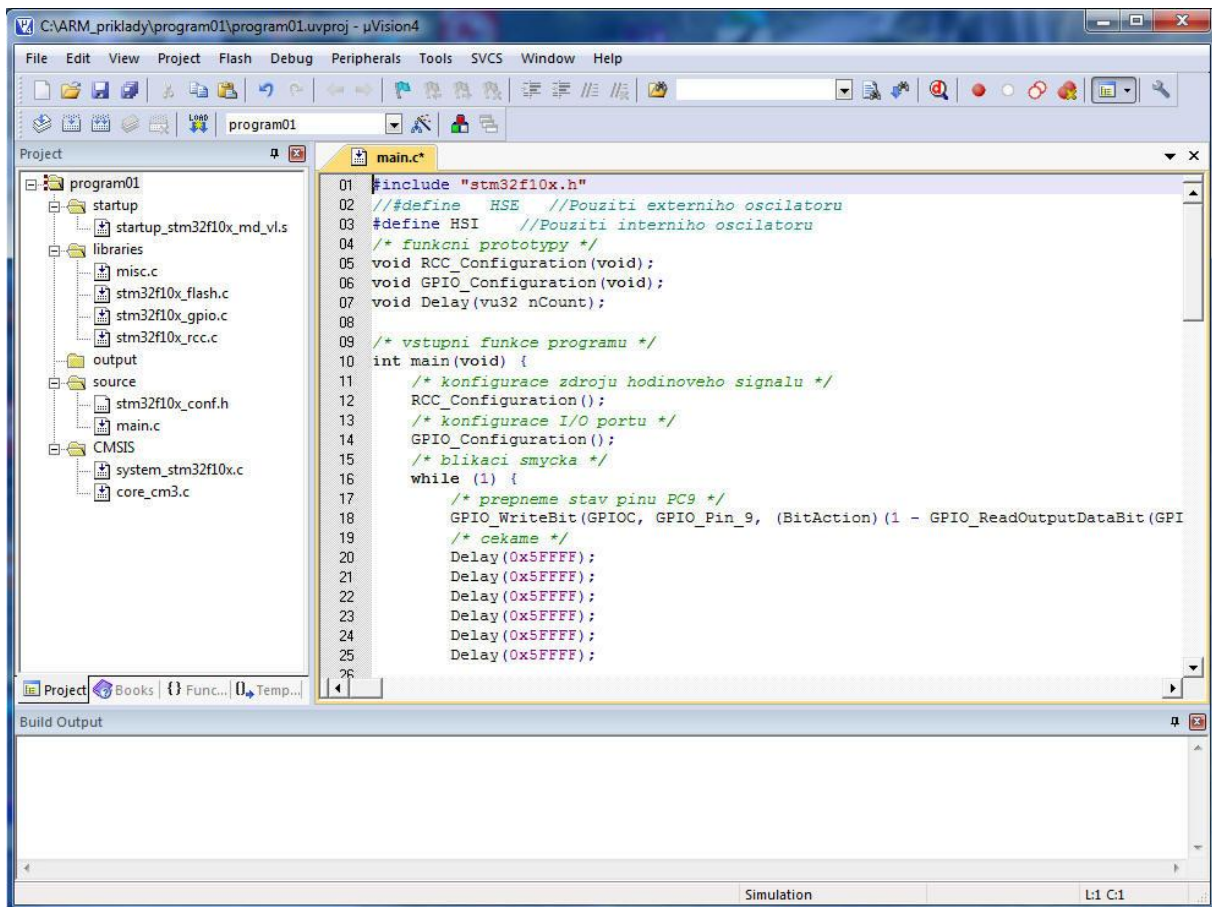
    /* konfigurace I/O portu */
    void GPIO_Configuration(void) {
        /* struktura s informacemi o konfiguraci pinu */
        GPIO_InitTypeDef GPIO_InitStructure;

        /* konfigurujeme pin PC7 jako output push-pull */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
    }

    /* cekani */
    void Delay(vu32 nCount) {
        while (nCount--);
    }

```

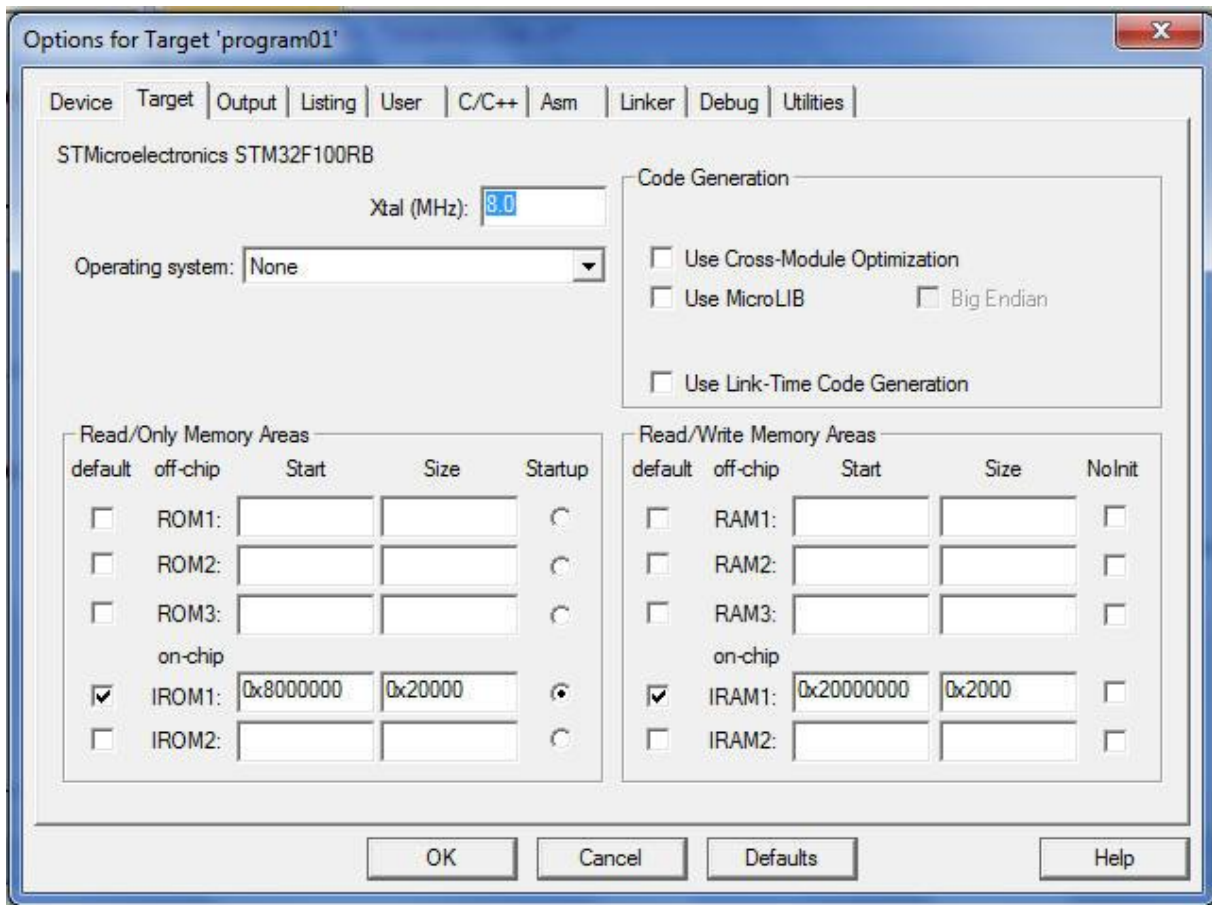
Takže máme



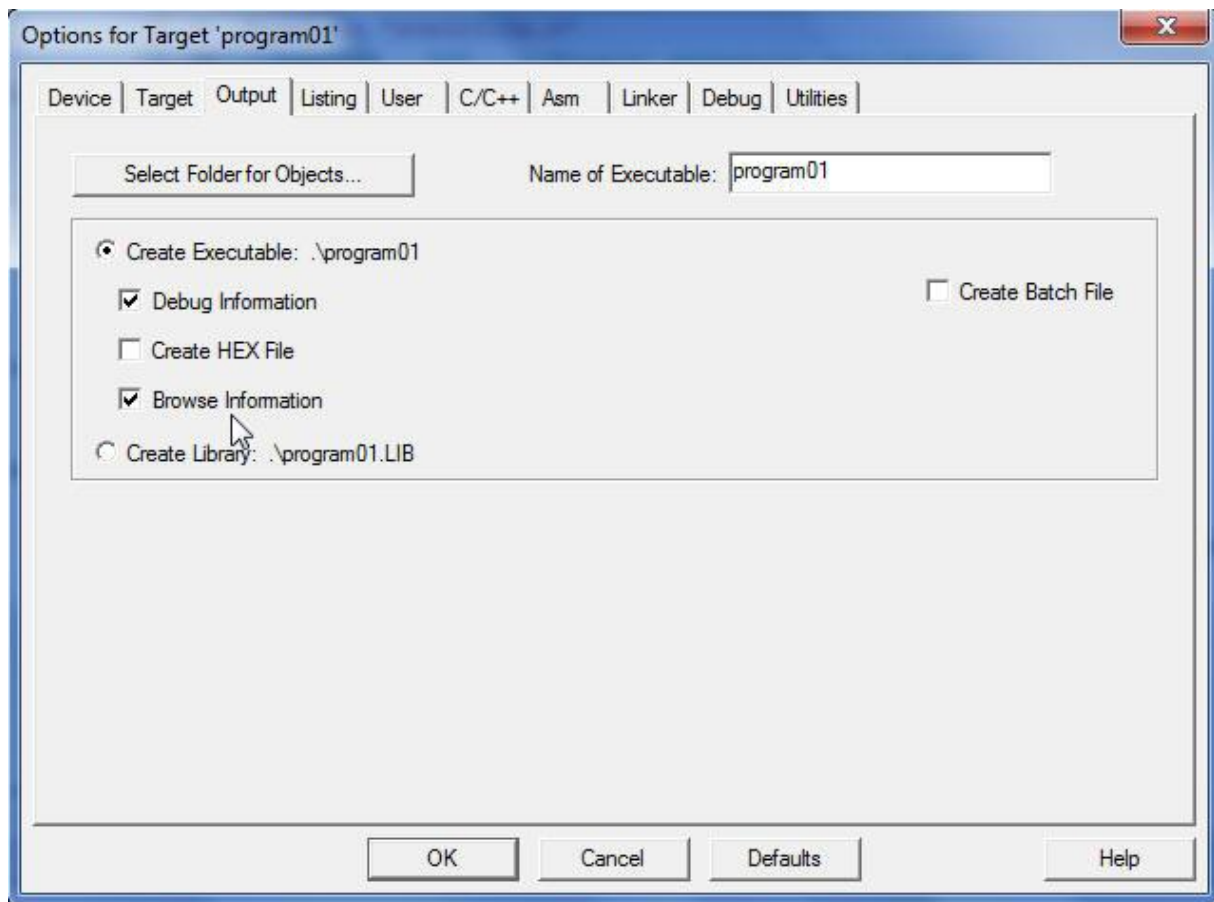
Ještě nastavíme „Target Options“ projektu, klikneme na ikonku



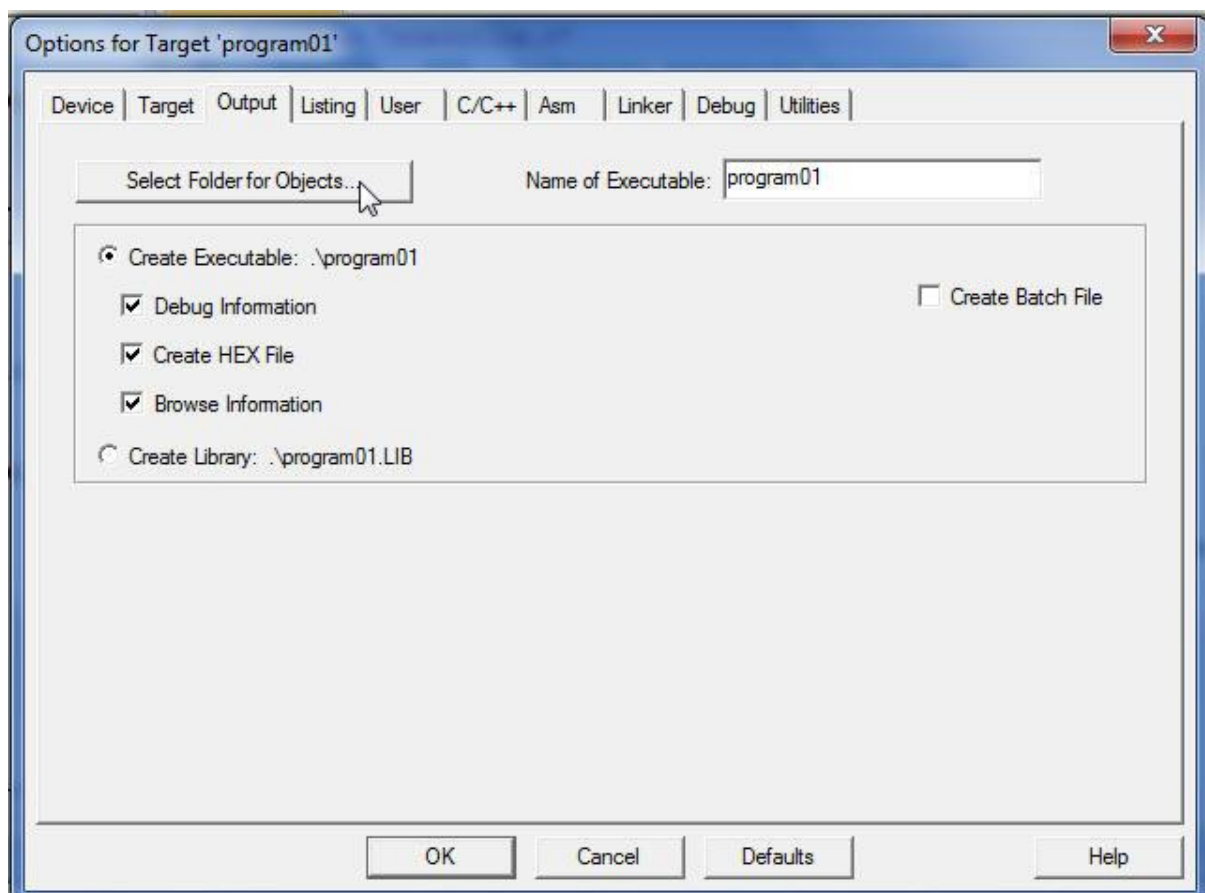
A dostaneme



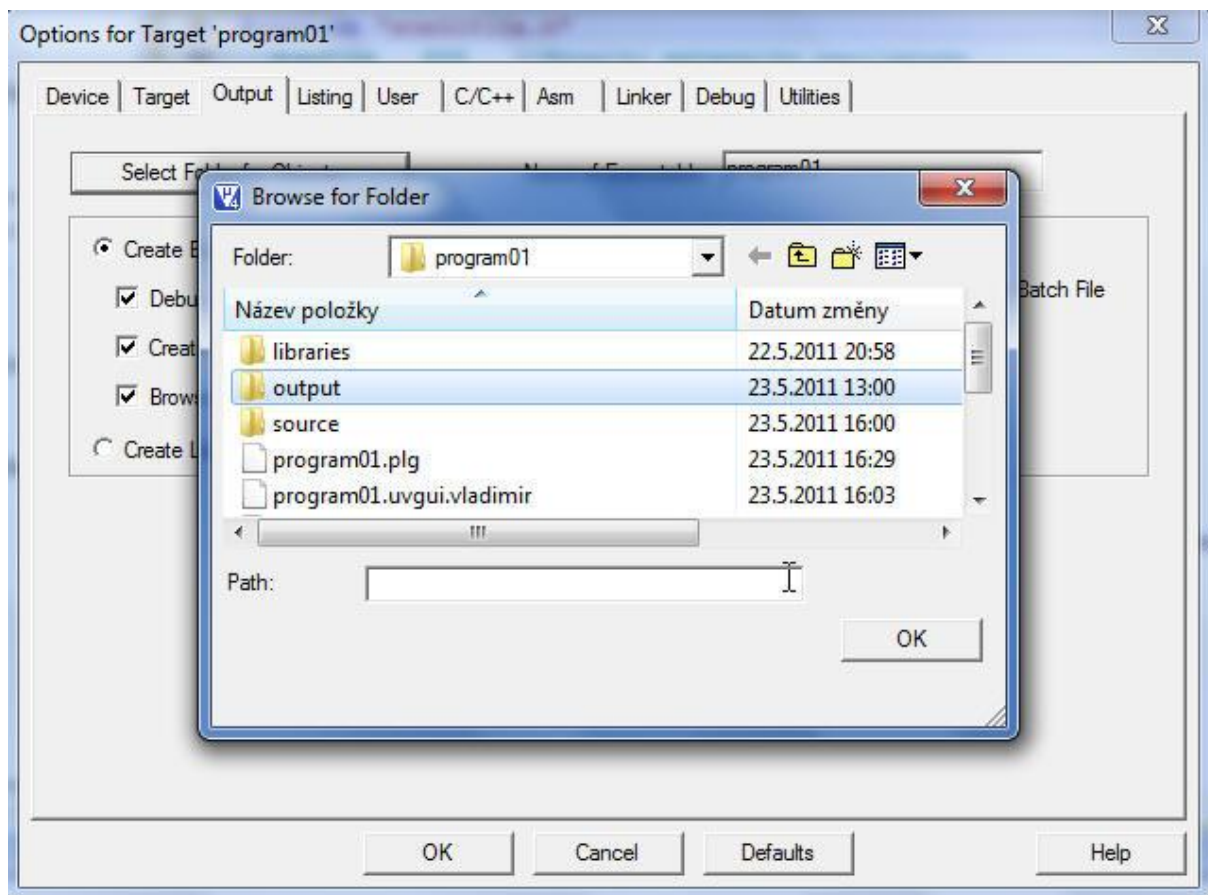
Vybereme záložku **Output**



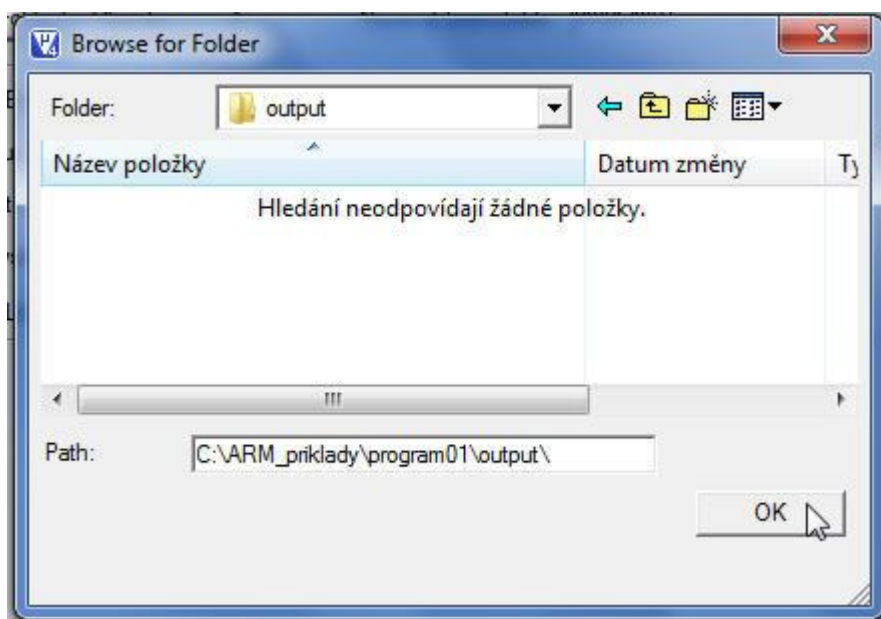
Zaškrtneme **Create HEX File**



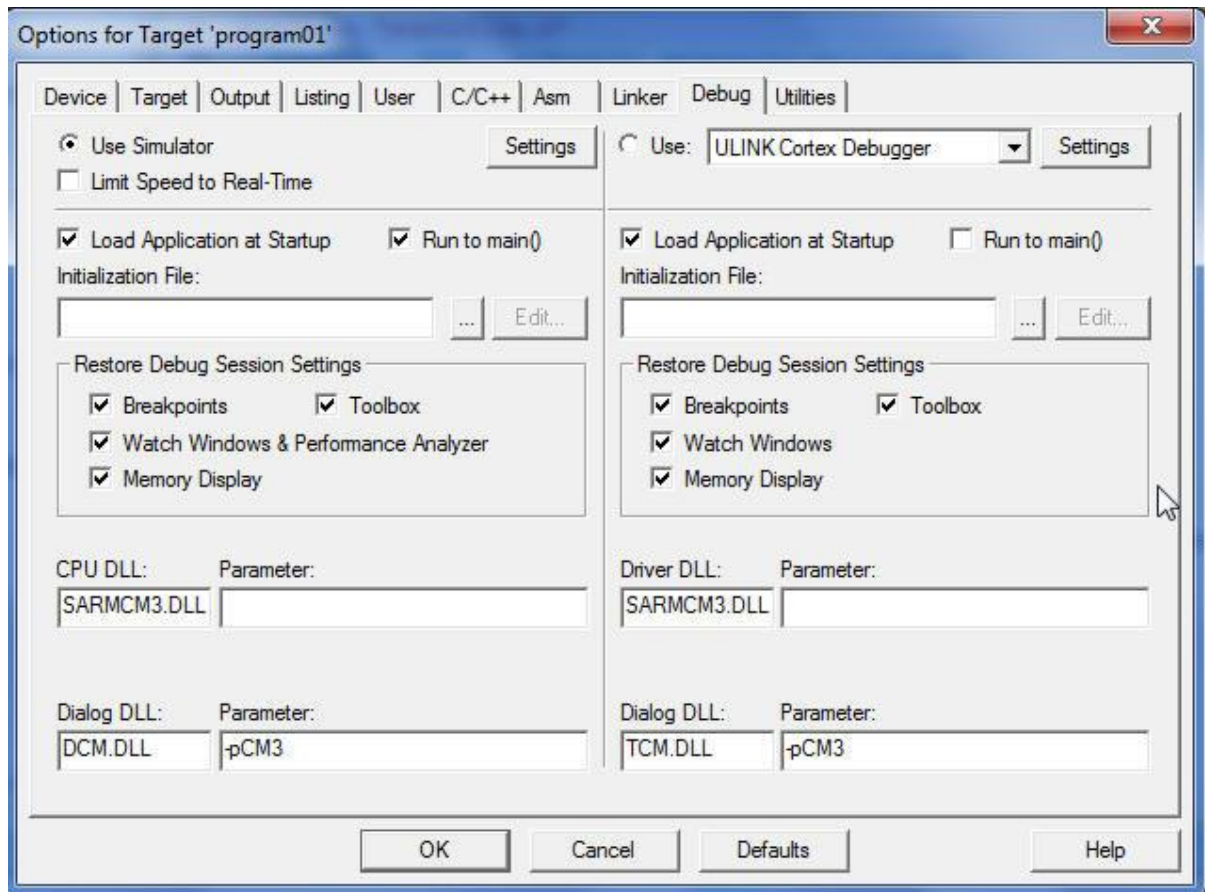
Dále klikneme na tlačítko **Select Folder for Object** a máme



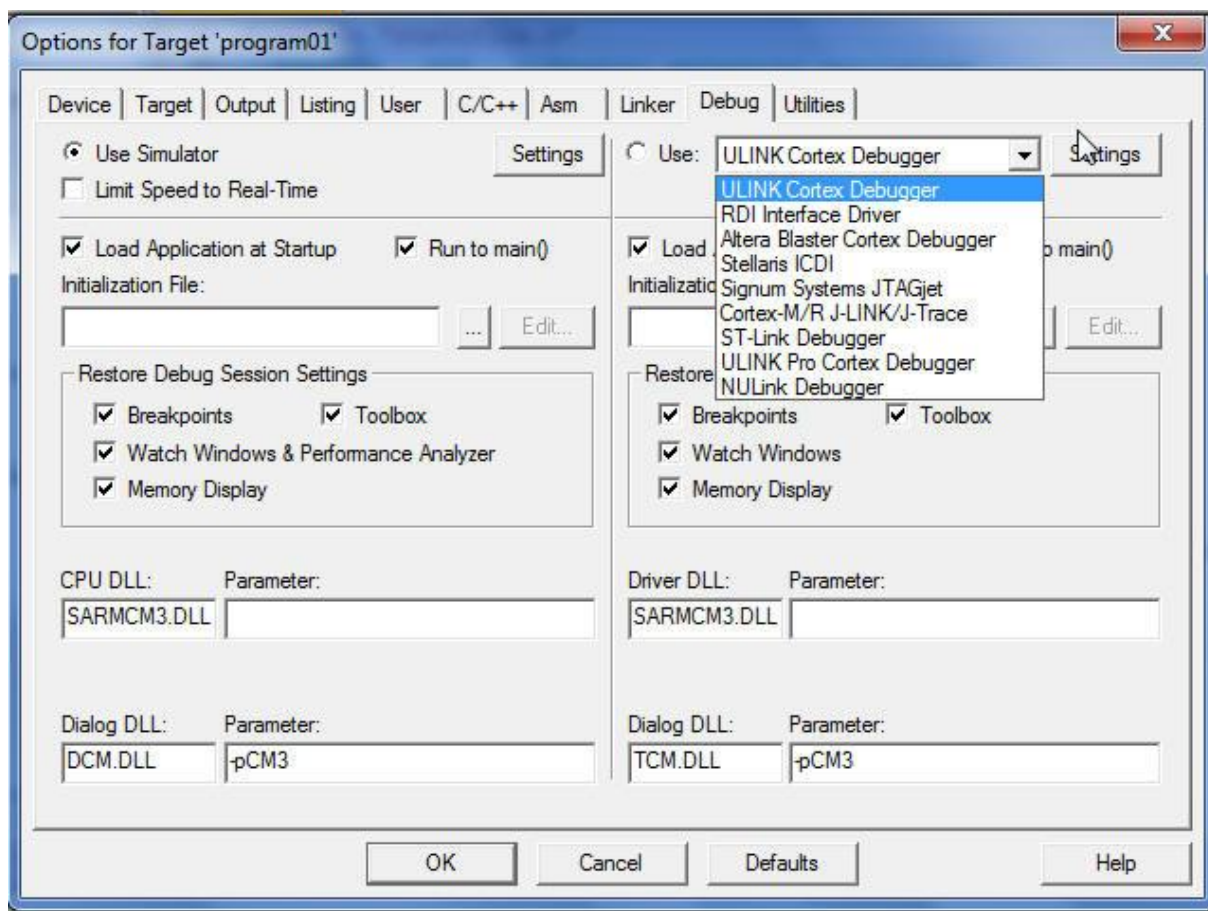
Vybereme složku **output**



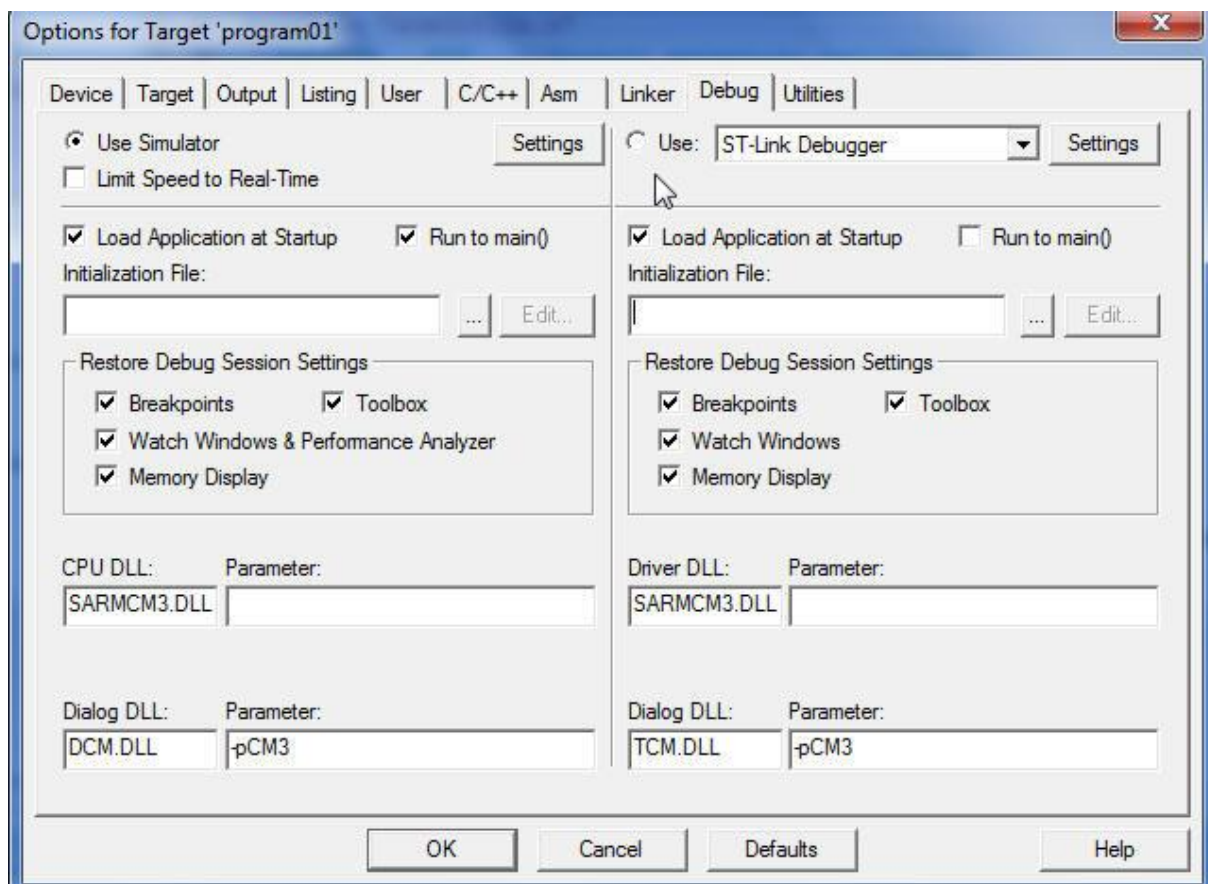
A potvrdíme **OK**. Poté v **Options for Target** vybereme záložku **Debug**



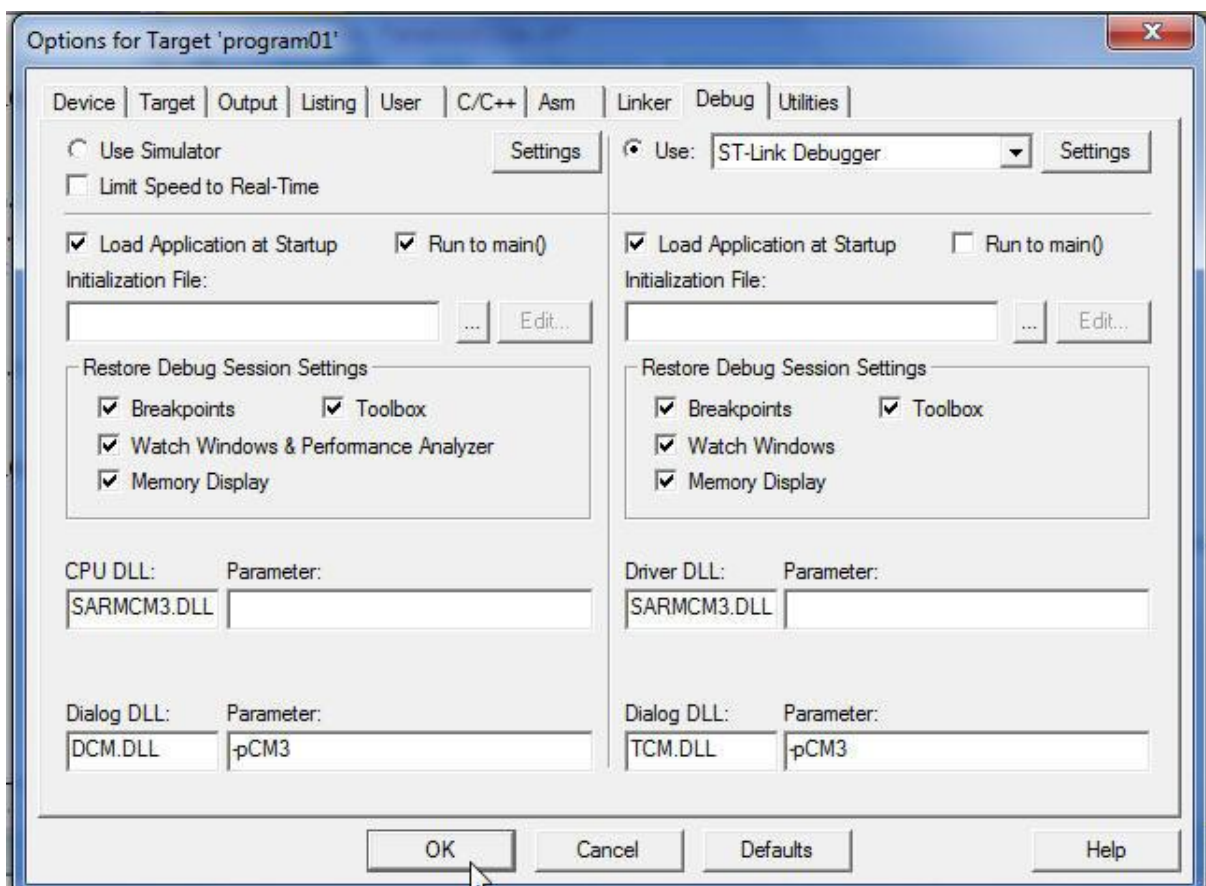
V pravé části nahoře rozvineme **listbox**



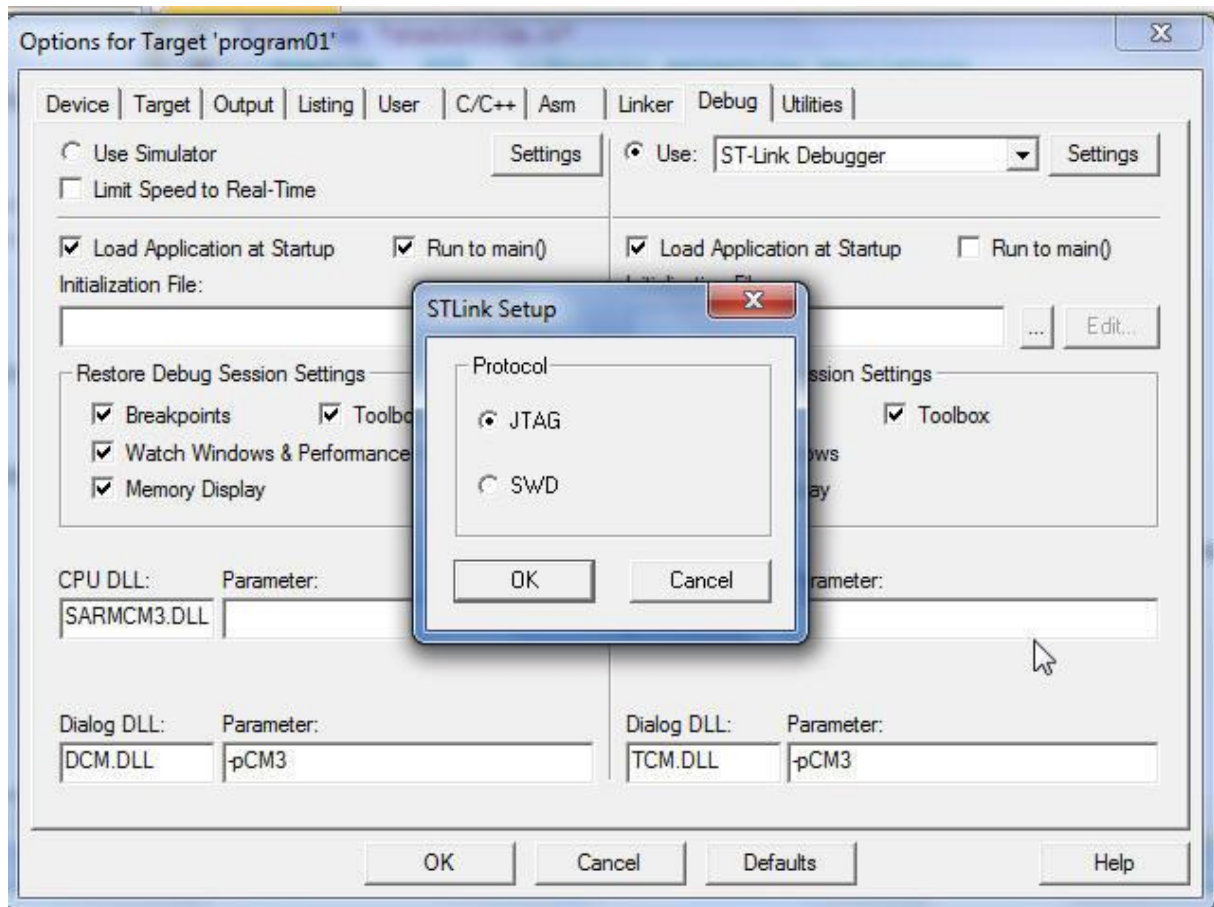
A z této nabídky vybereme **ST-Link Debugger**



A dále klikneme na *radiobutton use* nalevo od právě použité nabídky



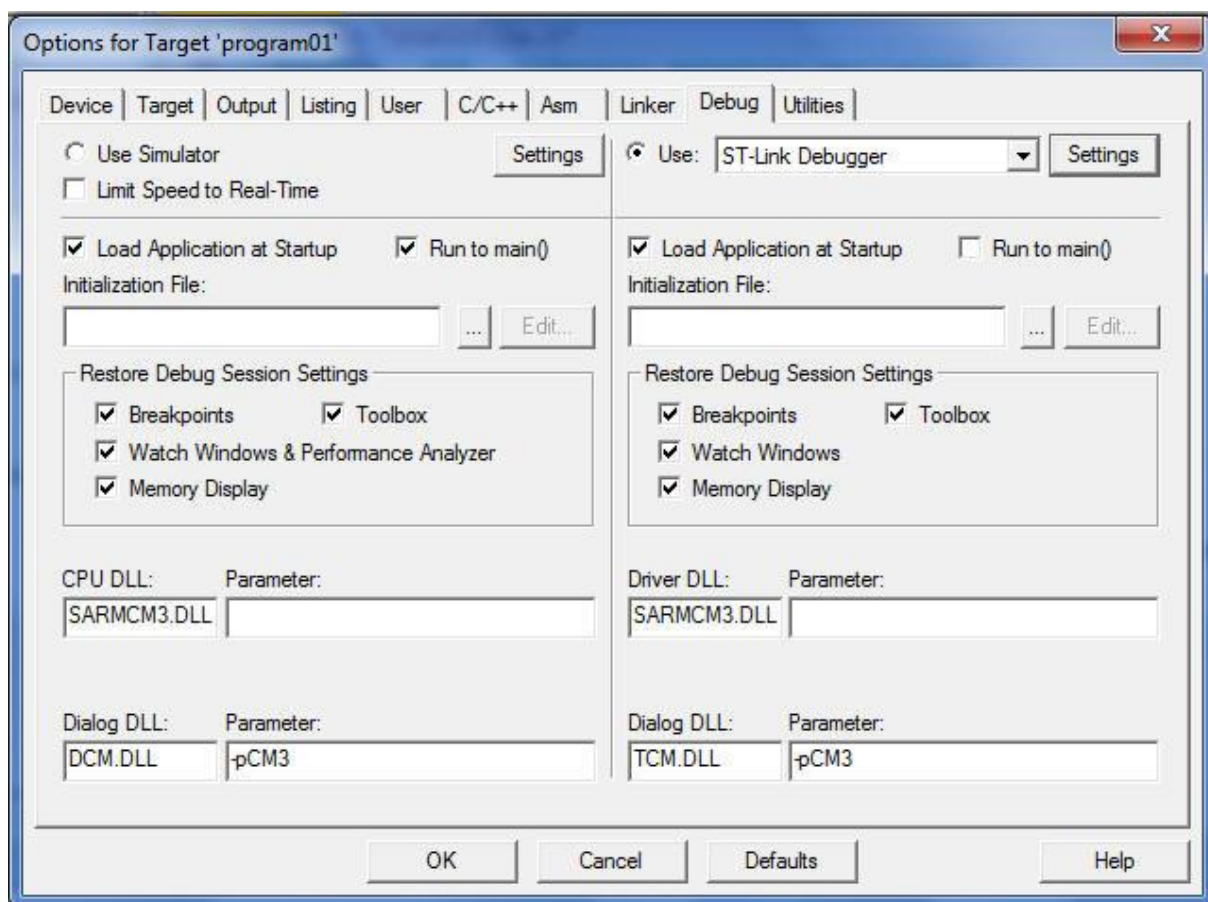
Ještě klikneme na tlačítko **Settings** napravo od této nabídky. Dostaneme



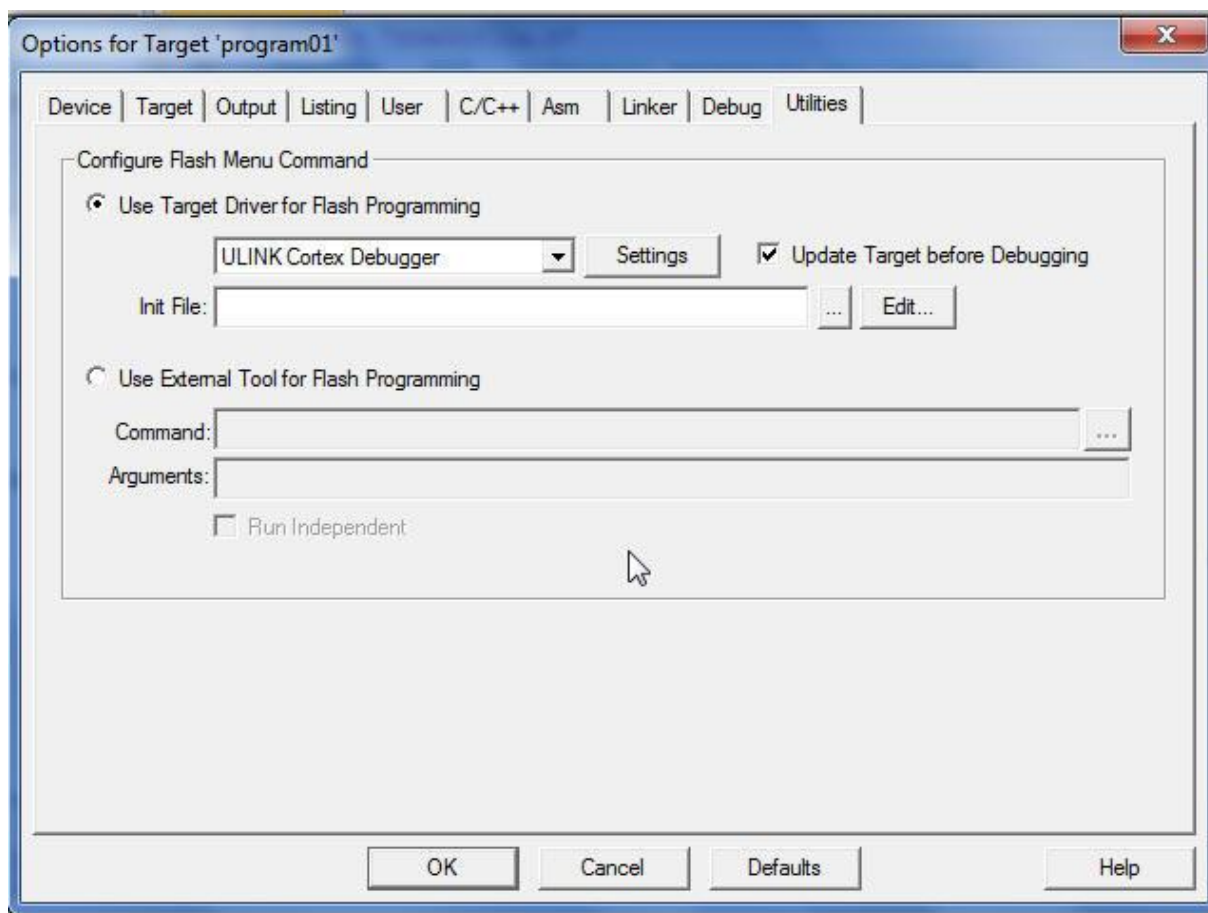
Zvolíme radiobutton **SWD** a potvrdíme **OK**



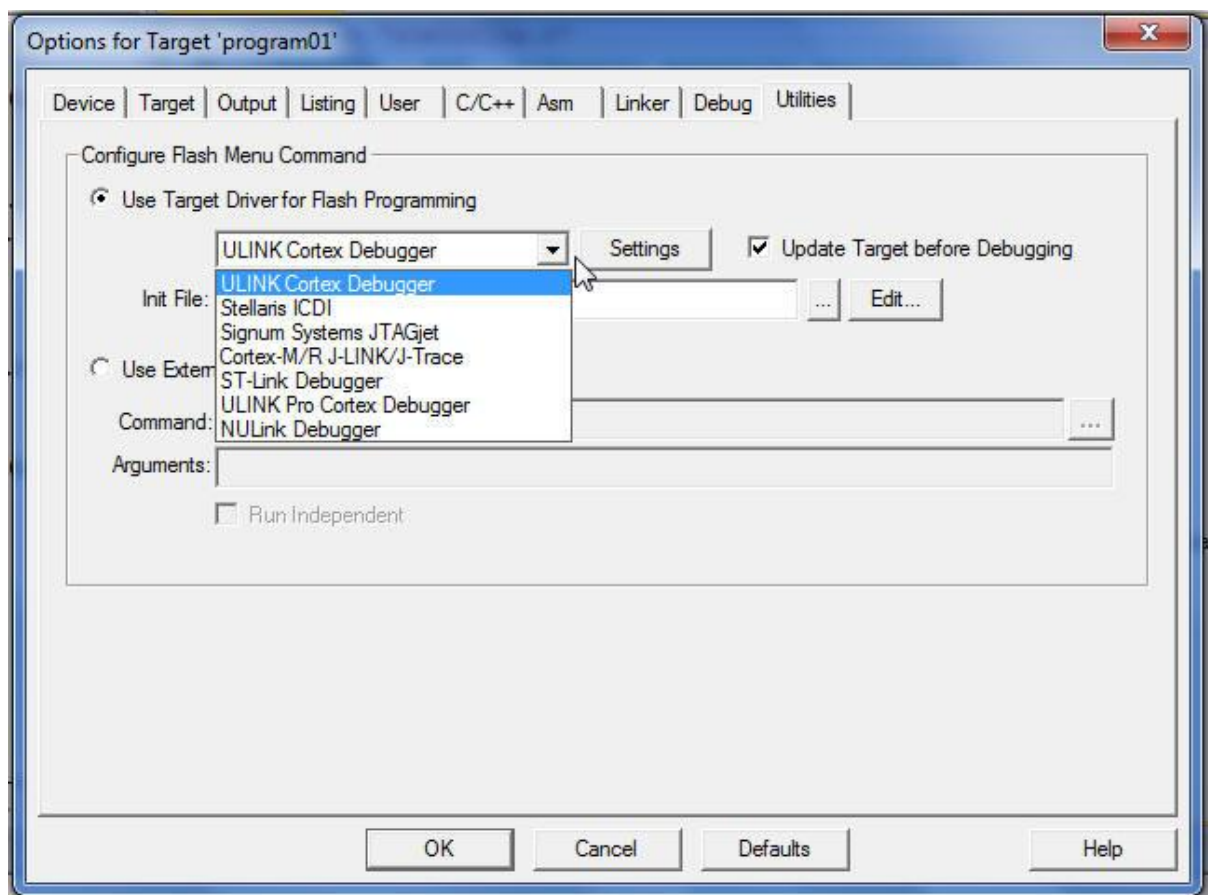
Máme tak



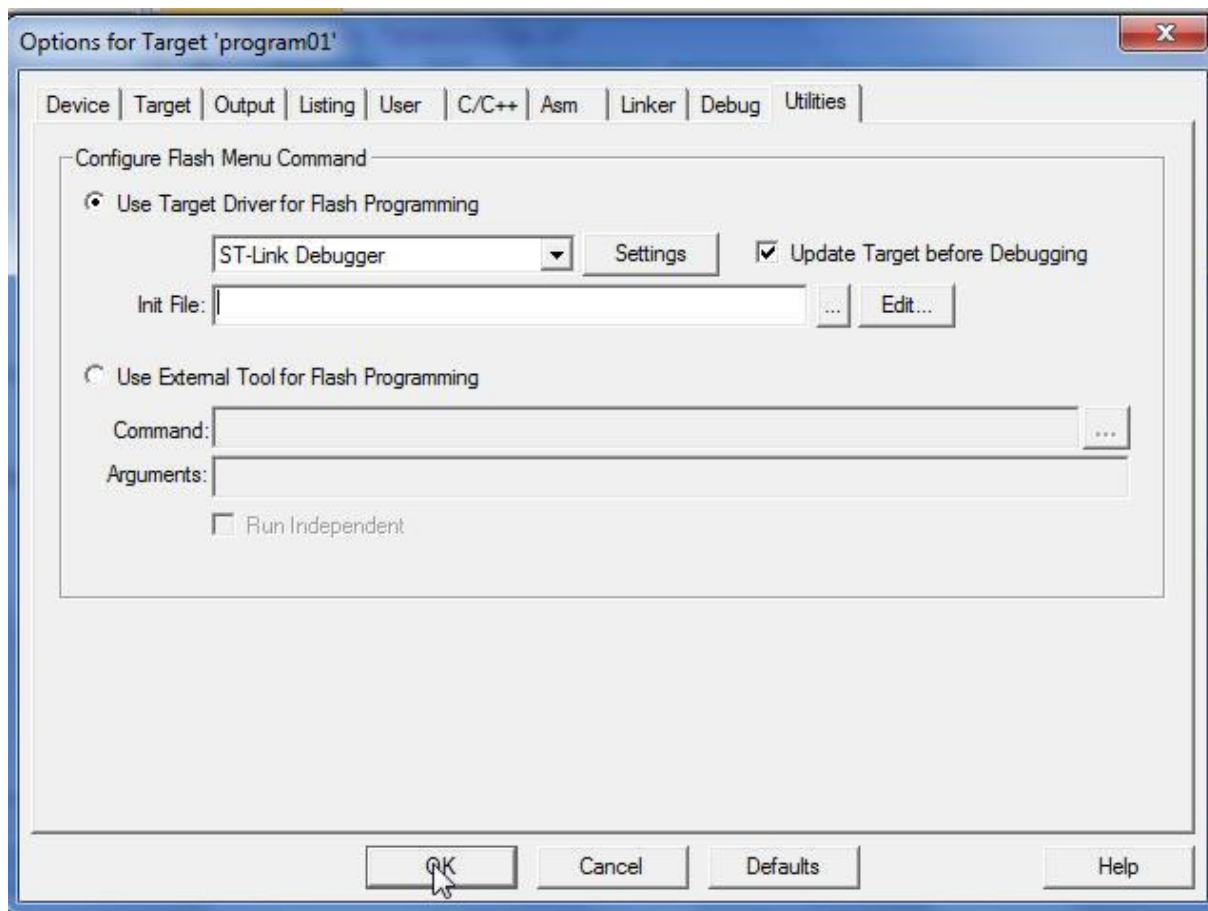
Ještě vybereme záložku **Utilities**



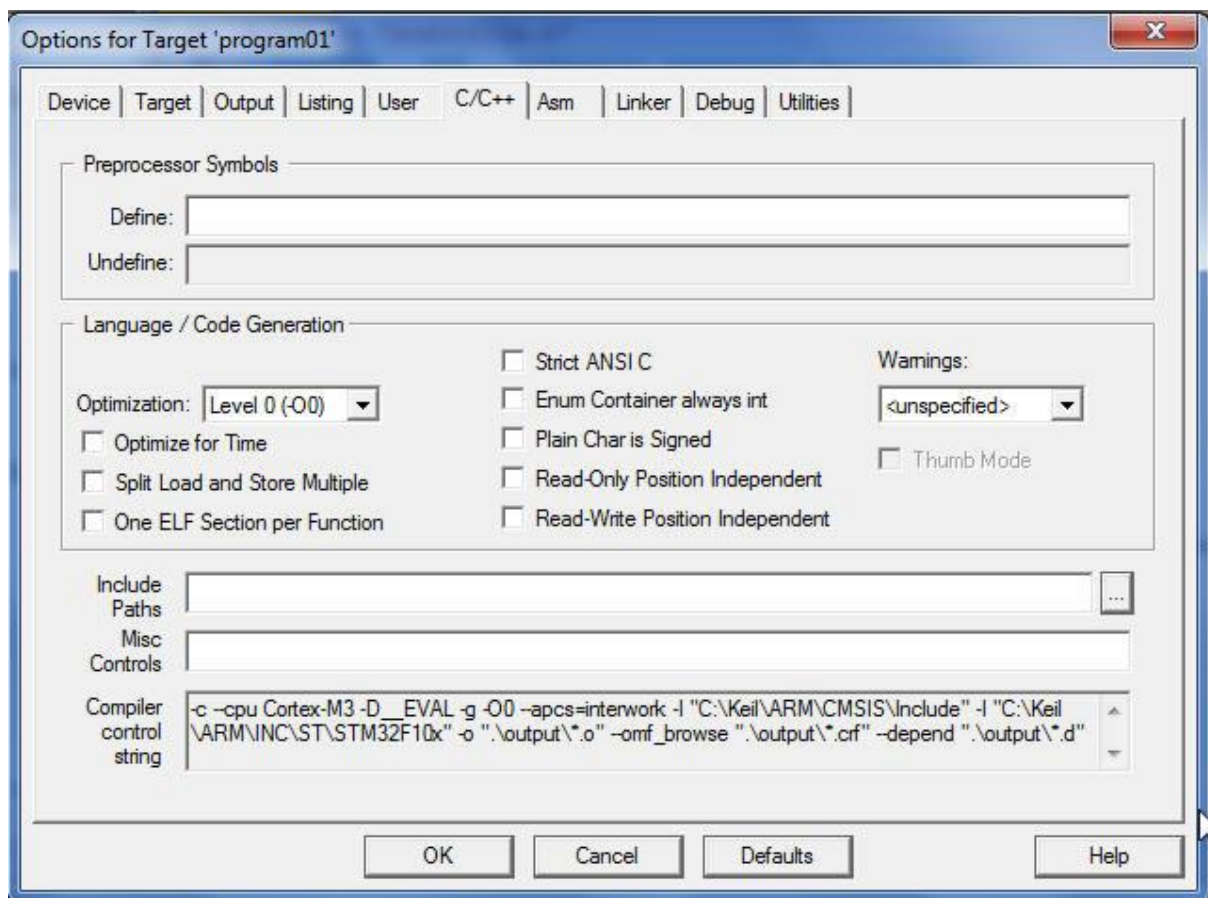
A nastavíme **Target Driver for Flash Programming**



Na **ST-Link Debugger**



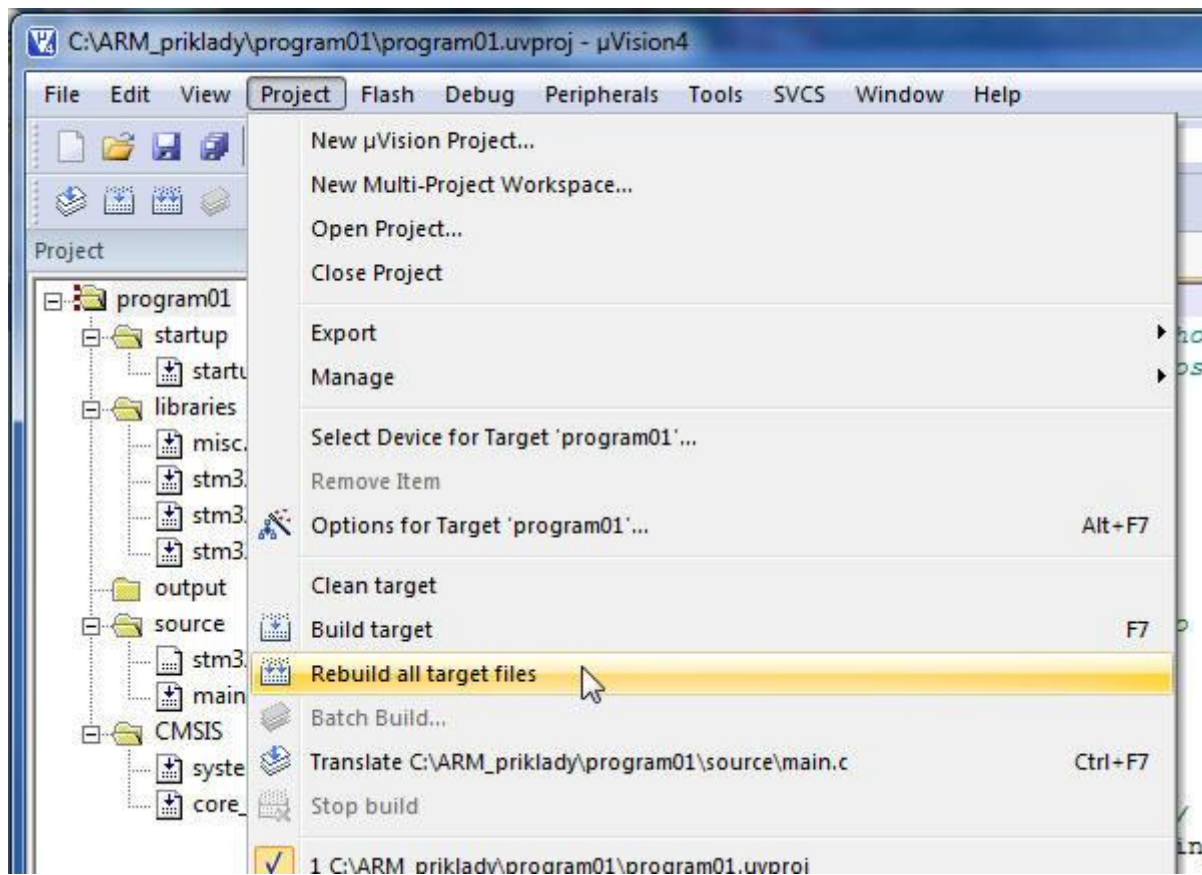
Dále klikneme na záložku **C/C+**



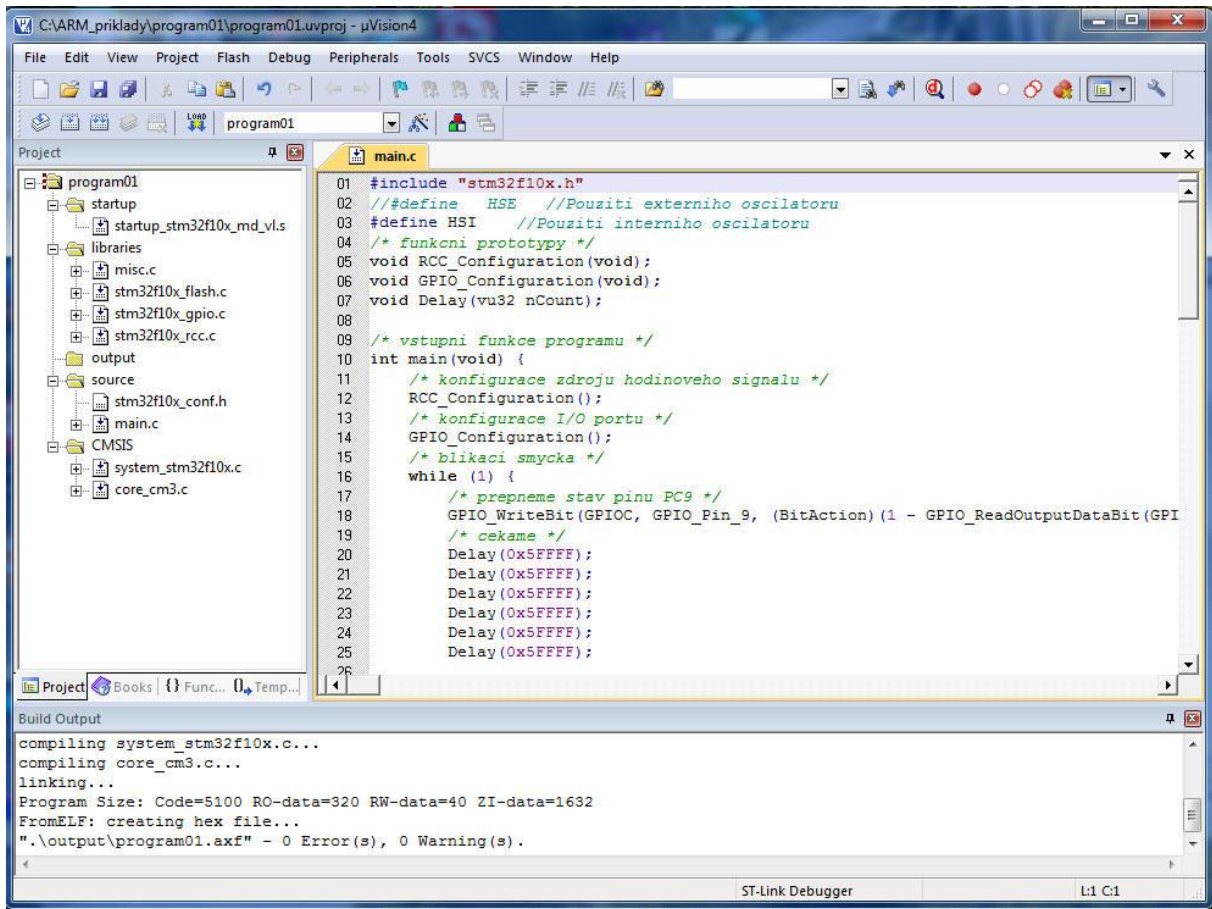
Do textboxu **Define** vložíme řetězec
STM32F10X_MD_VL, USE_STDPERIPH_DRIVER

A do textboxu **Include Path** řetězec
.\Libraries\CMSIS\CM3\CoreSupport;.\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x;.\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\StdPeriph_Driver\inc;.\source
Potvrdíme **OK**.

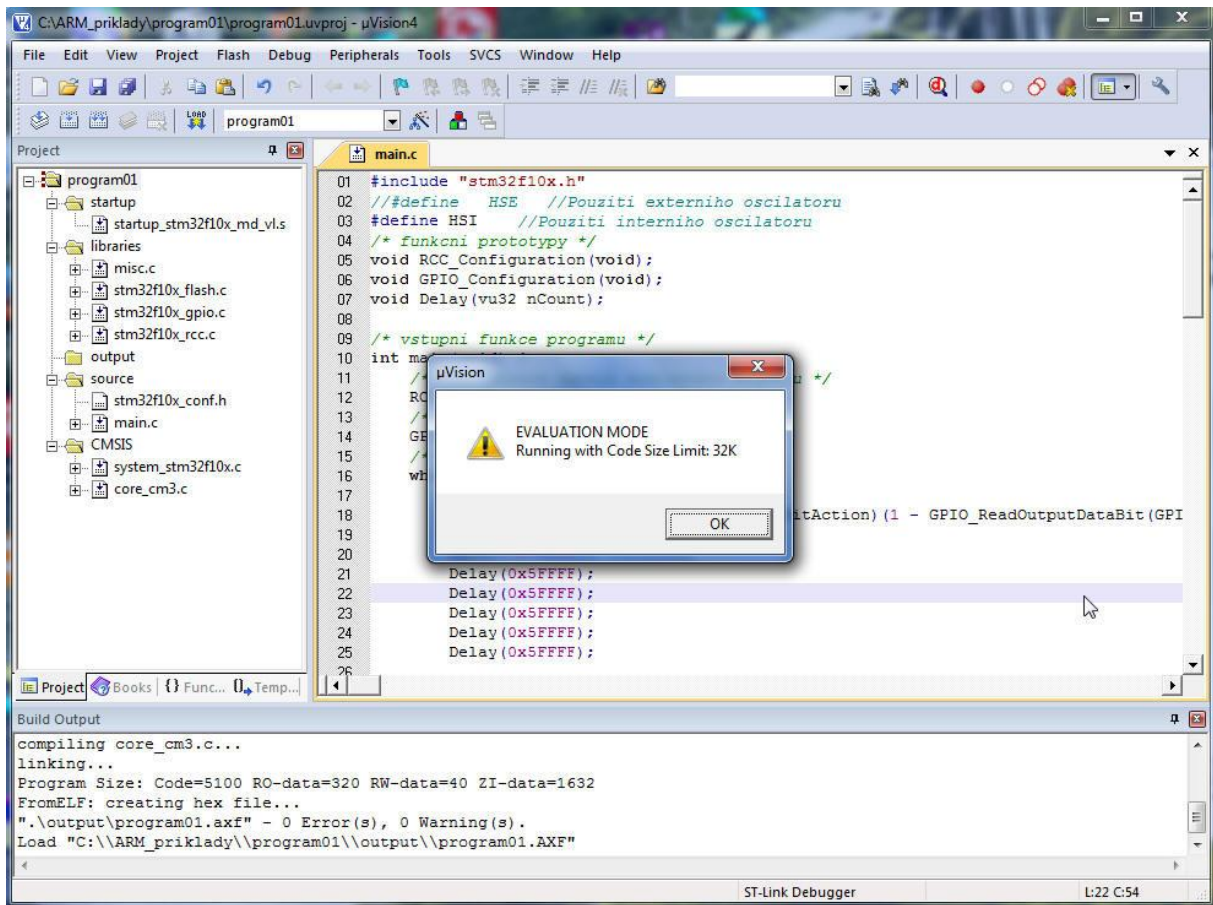
Nyní již provedeme překlad



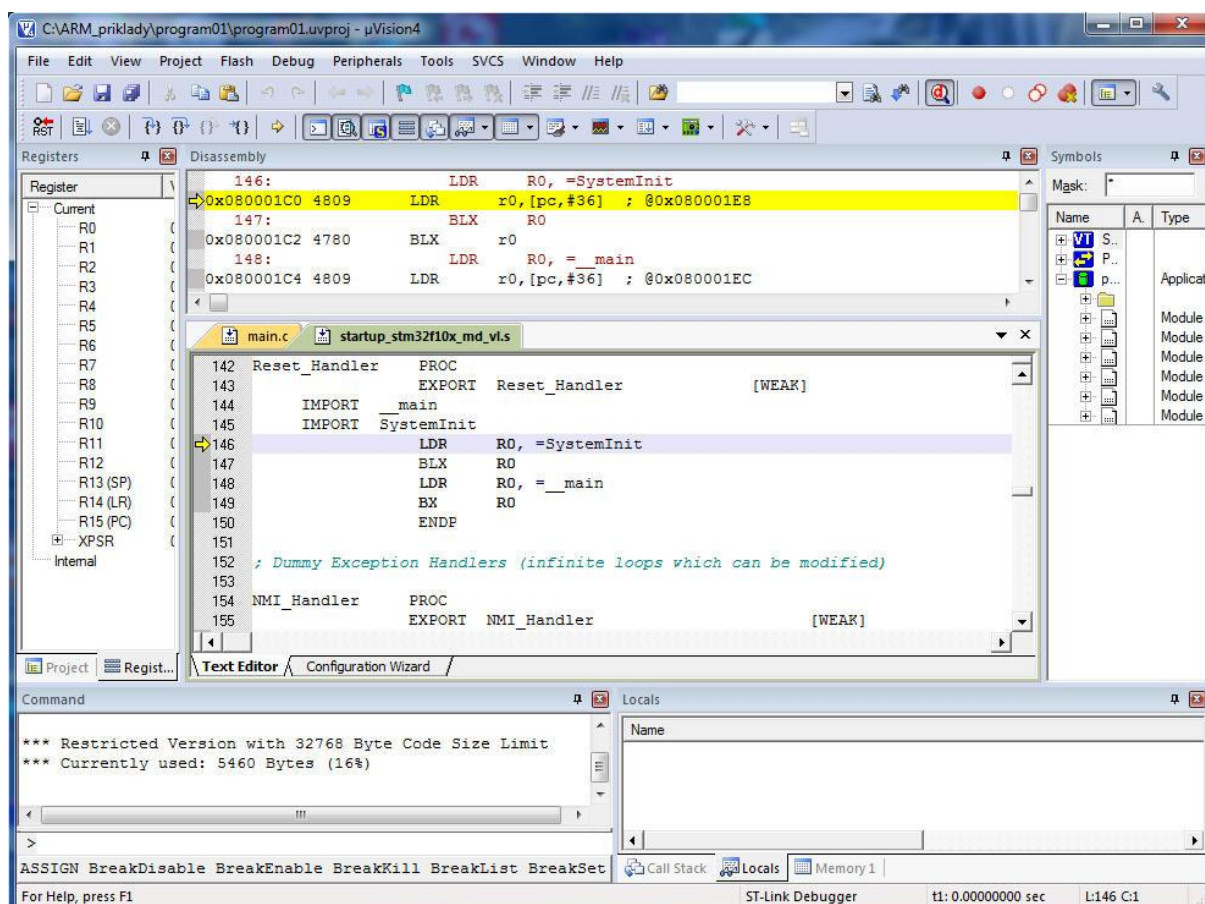
Po úspěšném překladu máme



Nyní naprogramuje vytvořený binární kód do procesoru. Přes USB připojíme **STM32VL DISCOVERY** kit. Zmáčkne **Ctrl + F5**, abychom spustili debugging. Objeví se



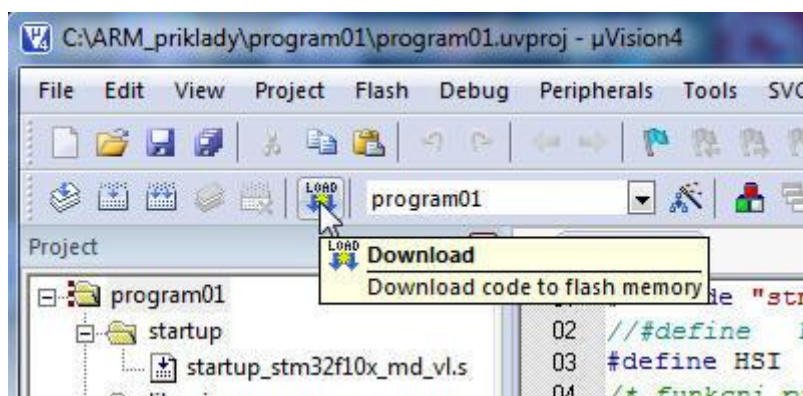
Potvrdíme **OK** a po chvíli dostaneme



A nakonec pomocí **F5** naprogramuje STM32. Na startkitu bliká zelená LEDka. Vývojové prostředí můžeme zavřít.

Při vytváření zdrojového kódu v **main.c** jsme převzali kód z DVD přílohy k tomuto výukovému materiálu. Zbývá tedy vysvětlit si jeho jednotlivé části a také s kódem experimentovat. Předtím ale učiníme několik poznámek k práci se startkitem i vývojovým prostředím.

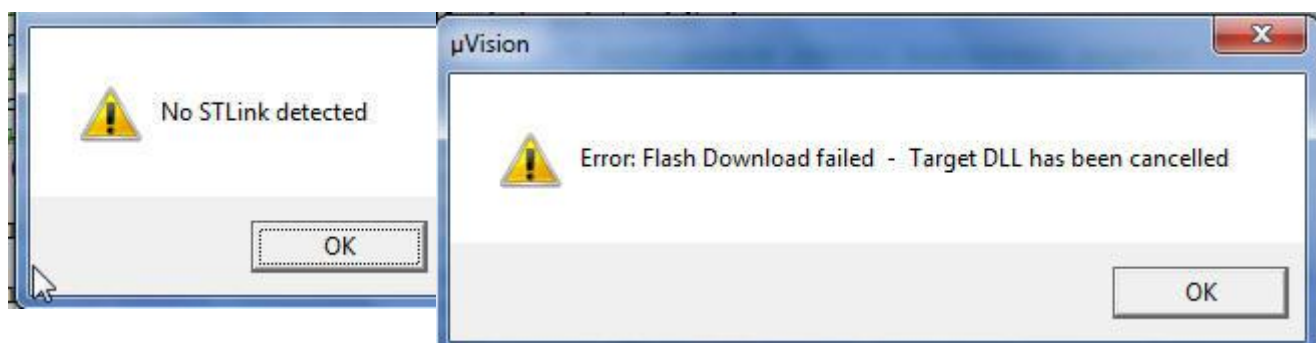
Pro nahrání programu do paměti flash STM32 bych očekával použití volby



Bohužel toto nefunguje, což ostatně konstatuje i materiál z katedry měření ČVUT a proto je programování STM32 provádět oklikou – nejdřív pomocí **Ctrl F5** spustit debugging a potom pomocí **F5** již přenést program do flash STM32.

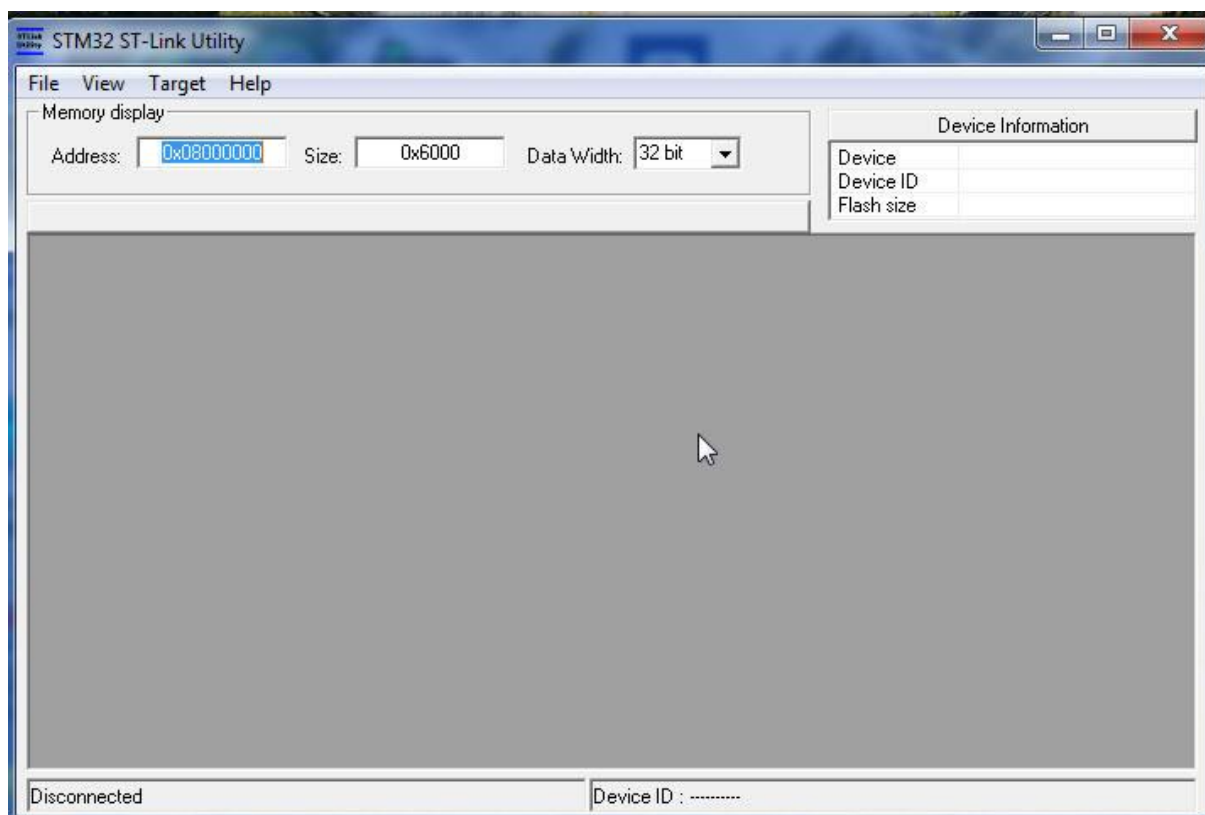
Toto ovšem funguje bez problémů u uVision4 nainstalovaném pod 32bitovými Windows7. Při použití 64 bitových se sice podaří projekt přeložit a získat soubory **axf** a **hex**, ale při pokusu o debugging dojde ke zkolabování uVision4.

K problémům může dojít i na 32 bitových windows, kdy se nepodaří navázat komunikaci s STM32VL Discovery a dostáváme chybová hlášení, např.

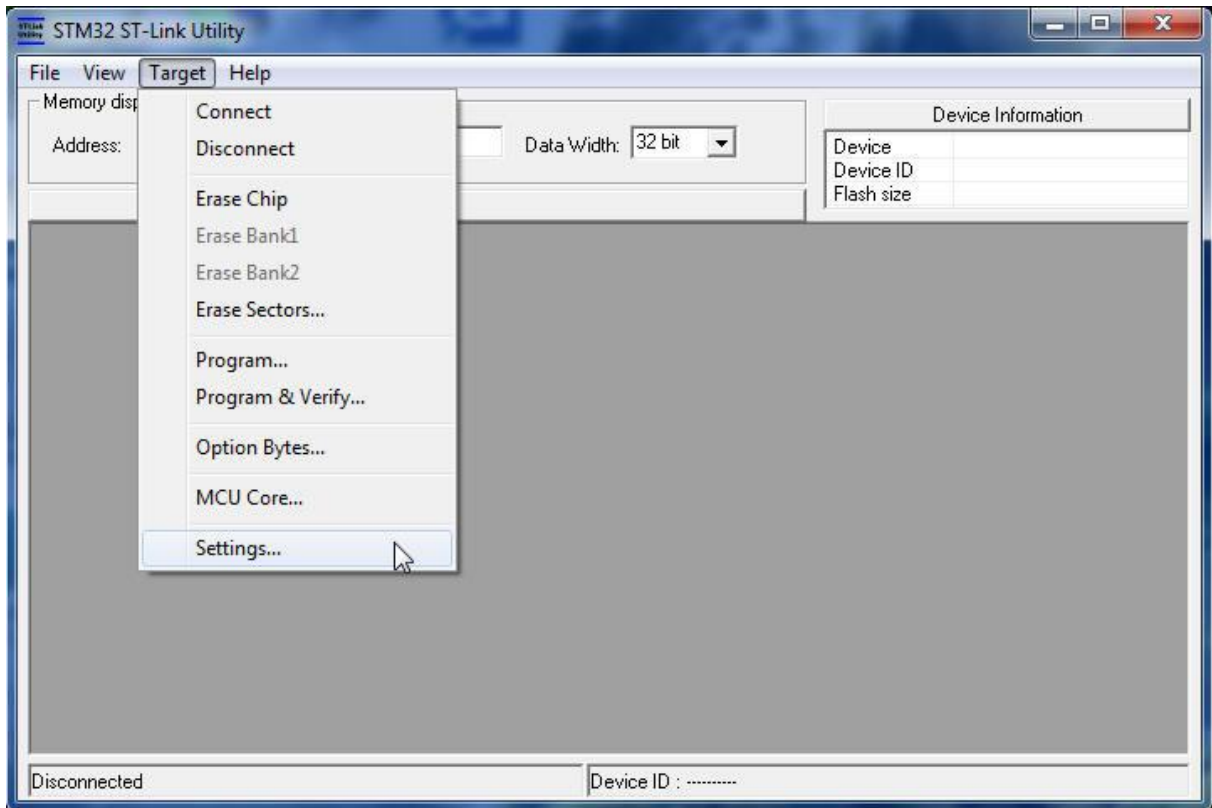


Chyba může být způsobená tím, že máme nižší verzi souboru **STLinkUSBDriver.dll** než 4.0.1.2 . Zde je na místě si říci, co je to **ST Link**. Na startkitu jsou totiž dva MCU – STM32F100RBT6, který programujeme a s kterým pracujeme a dále menší čip STM32F103C8T6, který je hlavní částí programátoru **ST Link** realizujícího protokol **SWD**. Tímto programátorem můžeme náš program přenést do STM32F100RBT6 na startkitu STM32VL Discovery, ale (po překonfigurování propojek) pomocí kablíku i chip umístěný na samostatné destičce, což bude náš případ při použití STM32F100RBT6 jako řídicího počítače letového kusu CANSATu – je zbytečné v CANSATU mít umístěn celý STM32VL Discovery kit – je zbytečně rozměrný, těžký a obsahuje i programátor, který v CANSATu je zbytečný. Proto startkit budeme používat jen pro vývoj software.

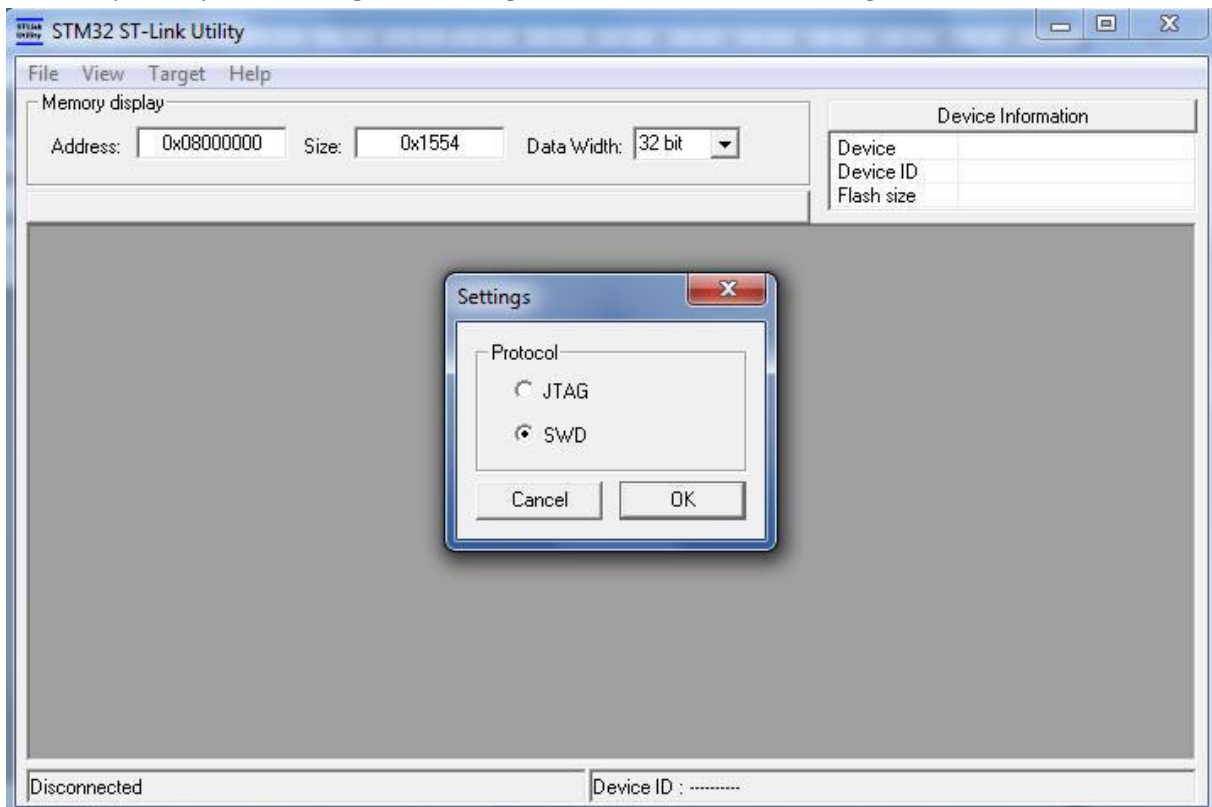
Programový kód můžeme do startkitu přenášet nejen z prostředí **uVision 4**, ale lze použít **STM32 ST-Link Utility**



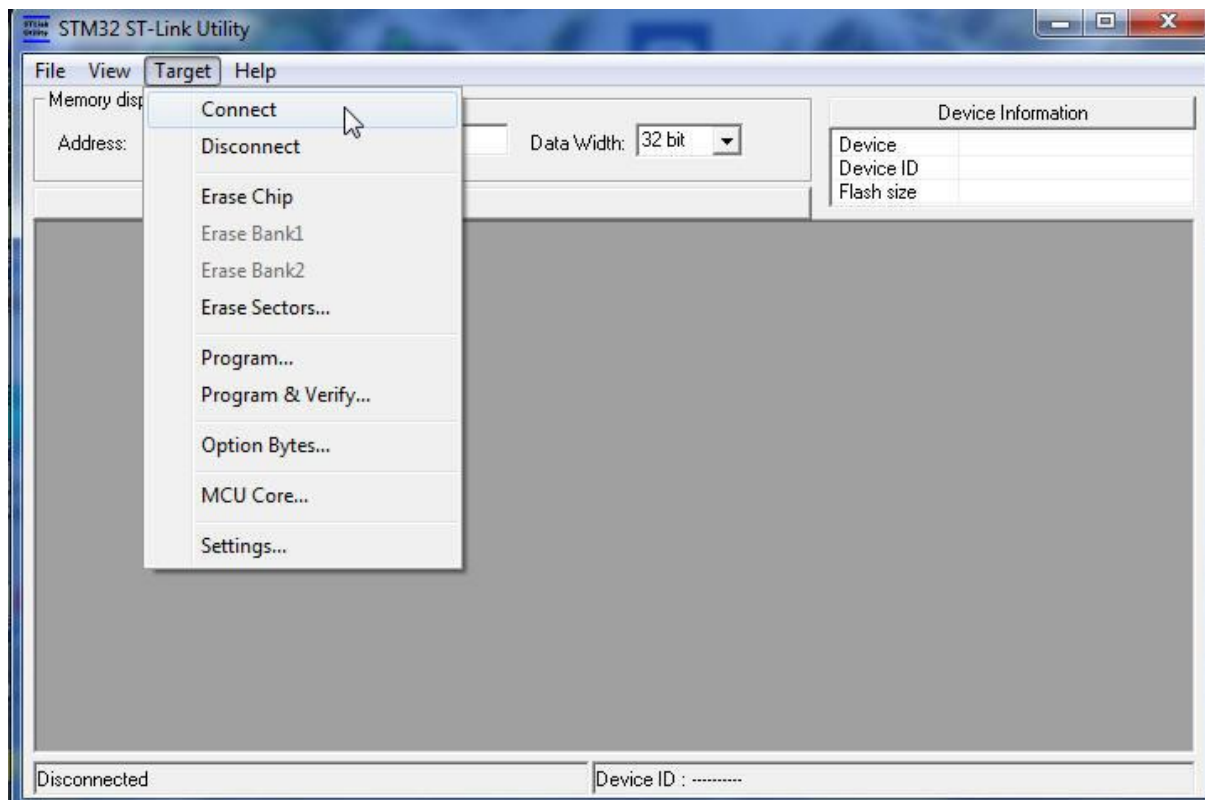
Tento sw nainstalujeme z instalačního souboru **STM32 ST-Link Utility_V1.2.exe**. Pokud máme tento sw již nainstalovaný, připojíme pomocí usb startkit k počítači. Nejprve se přesvědčíme, že máme nastavený protokol **SWD**.



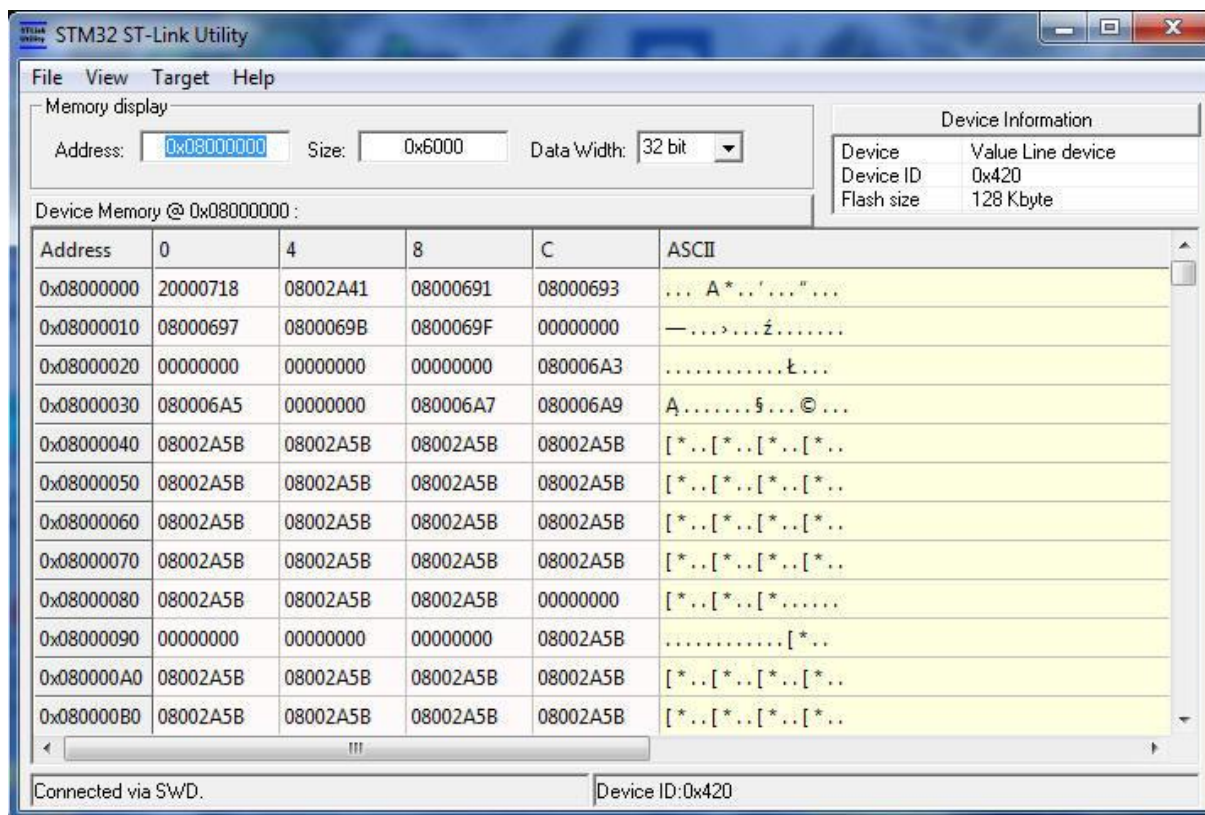
V menu proto vybereme **Target** → **Settings...** a dostaneme okno **Settings**. Radiobuton na **SWD**



Poté vybereme v menu **Target** → **Connect**

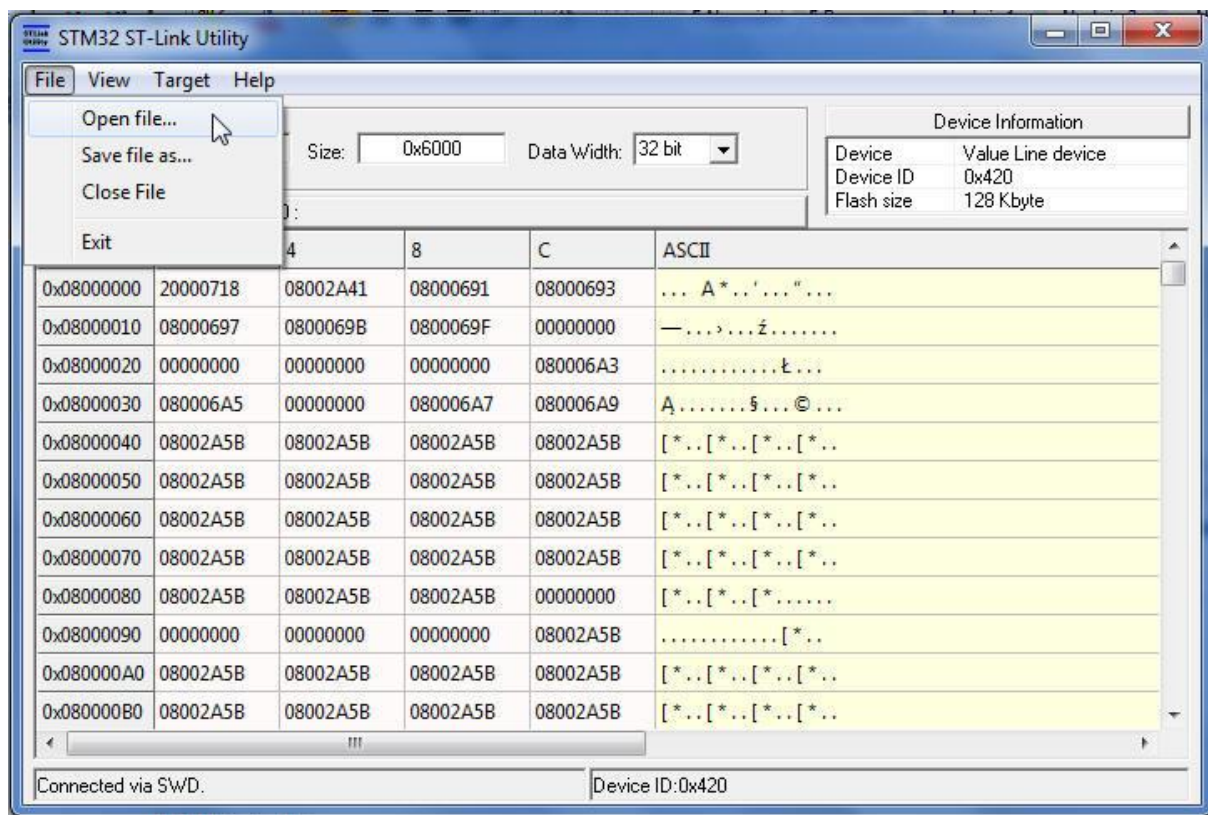


Dostaneme

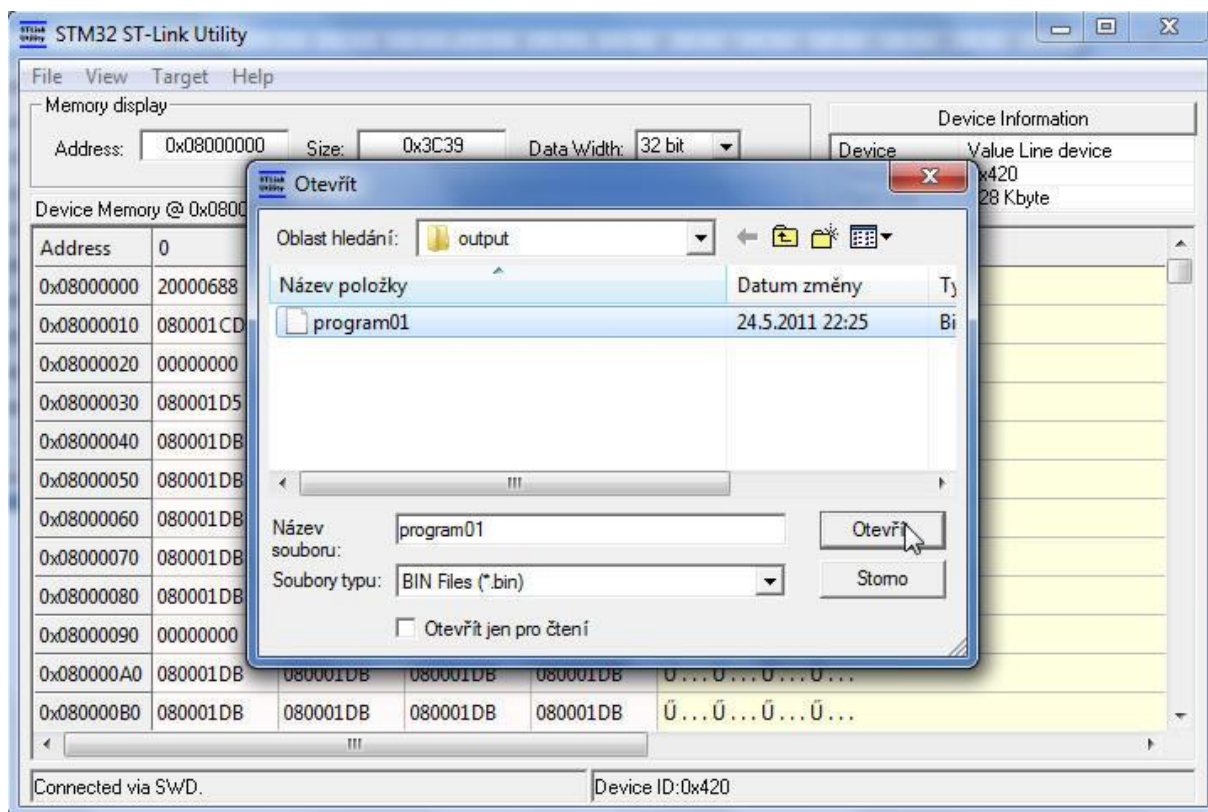


Tím jsme si ověřili, že u našeho startkitu je **ST-Link** funkční. Doporučuji toto ověření provést jako první akci, kterou se startkitem budeme provádět.

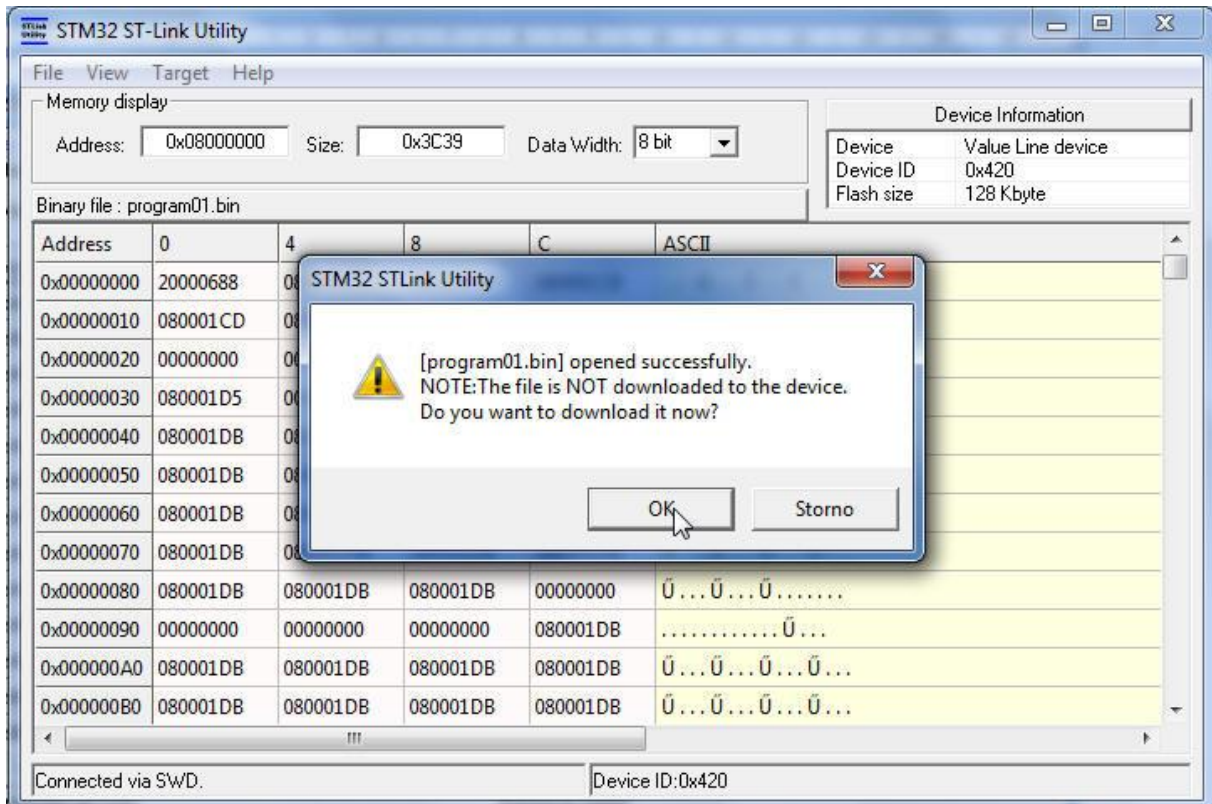
STM32 ST-Link Utility můžeme použít i k naprogramování MCU . V menu vybereme **File – Open File**



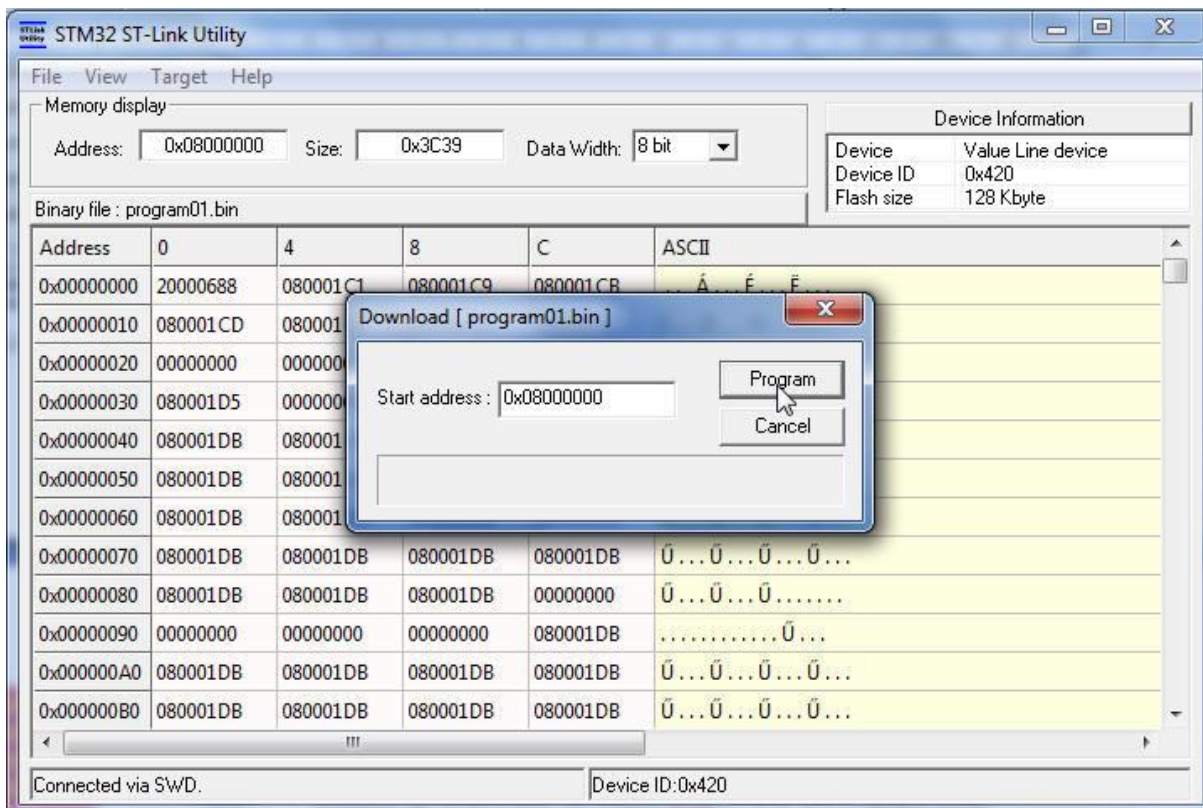
dostaneme



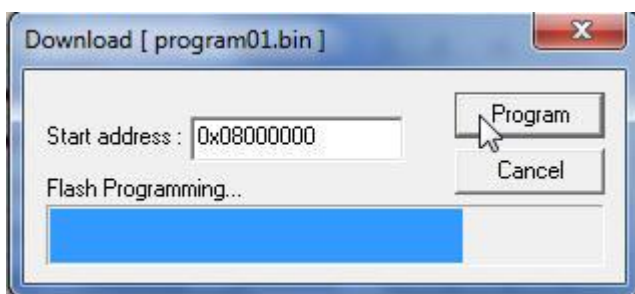
Vybereme **program01.bin** a klikneme na tlačítko **Otevřít**. Soubor program01.bin jsme ovšem předem získali pomocí utility fromelf.exe , kterou najdeme v C:\Keil\ARM\Bin40\ a spustíme s parametry: **fromelf --bin program01.axf --output program01.bin**



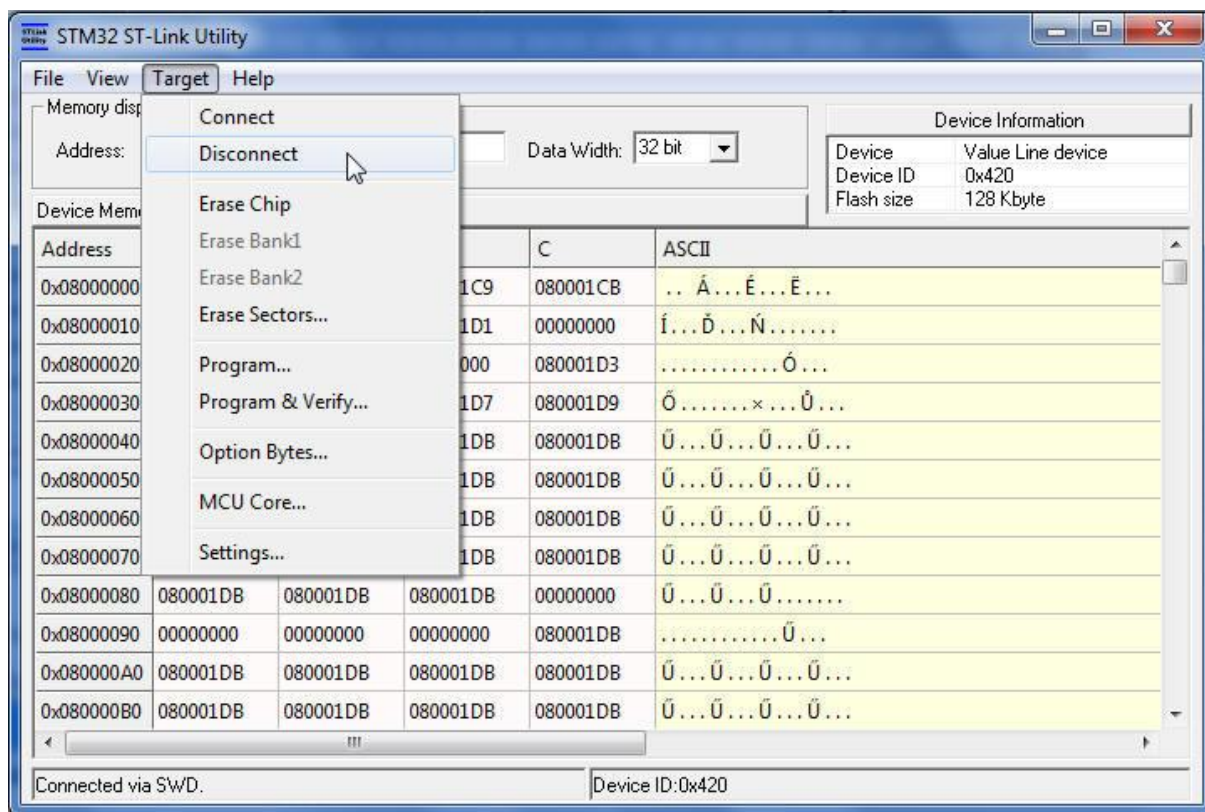
Potvrdíme , dostaneme



Po kliknutí na tlačítko **Program** již začne vlastní programování



Po dokončení programování ještě ukončíme spojení tj v menu vybereme **Target -> Disconet**



Po resetování MCU již běží nový program.

V zbývající části této podkapitoly si konečně vysvětlíme jednotlivé části zdrojového kódu v **main.c**

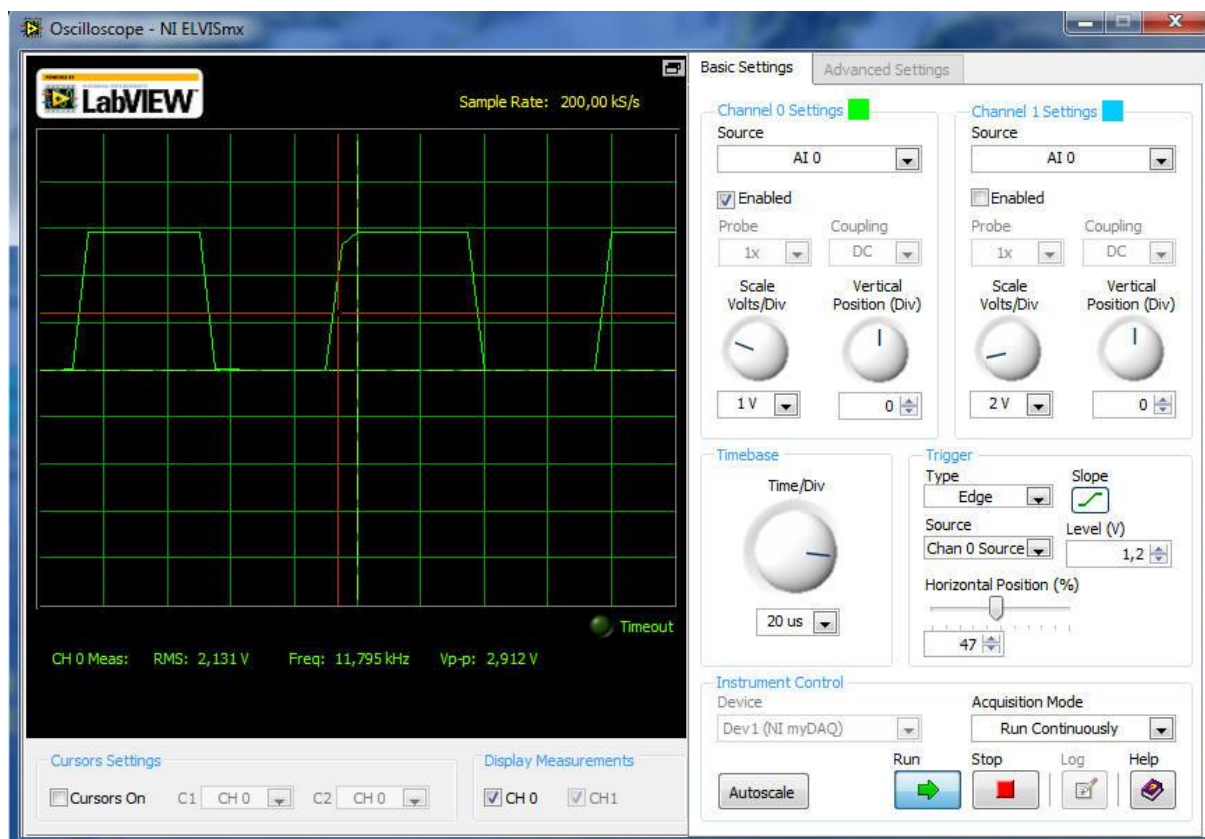
Nakonec uděláme pokus – zrychlíme čekací smyčku

```

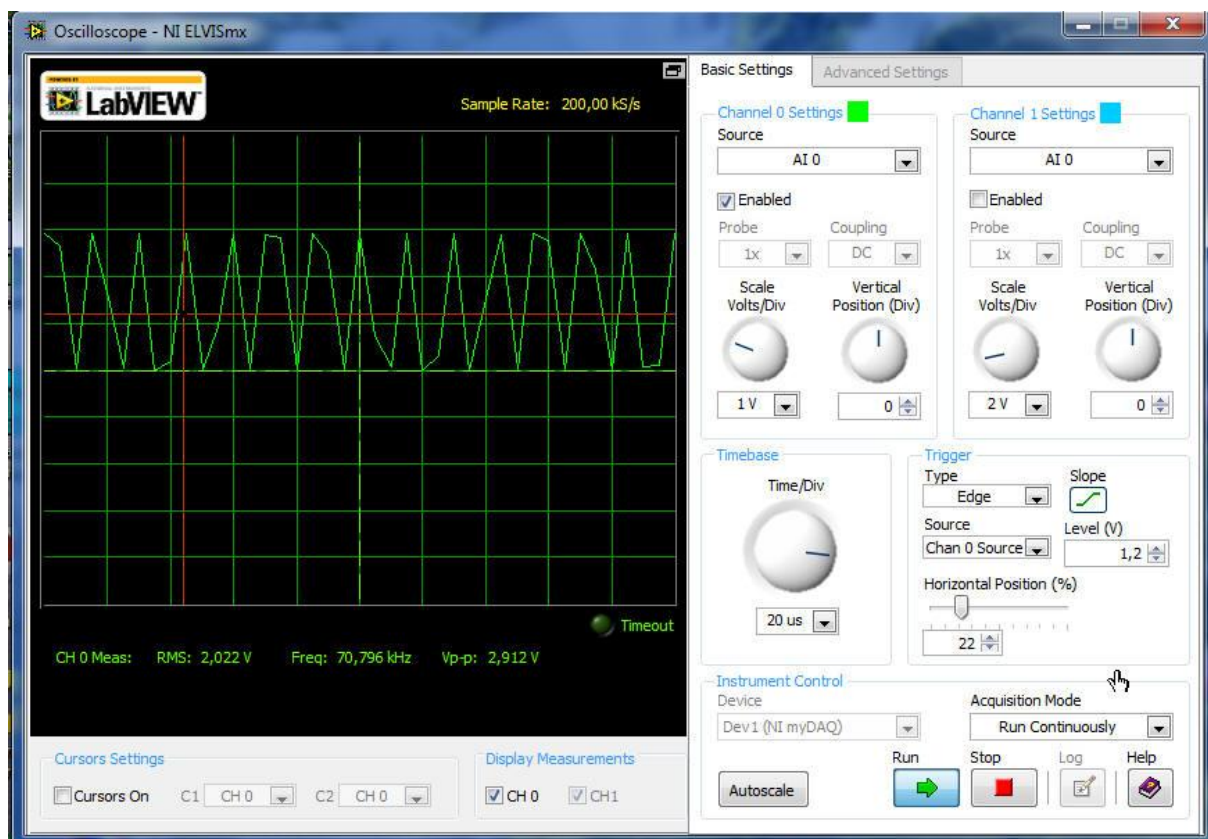
/* blikaci smyčka */
while (1) {
    /* prepneme stav pinu PC9 */
    GPIO_WriteBit(GPIOC, GPIO_Pin_9, (BitAction)(1 -
GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_9)));
    /* cekame */
    Delay(0x000FF);
    //Delay(0x5FFFF);
    //Delay(0x5FFFF);
    //Delay(0x5FFFF);
    //Delay(0x5FFFF);
    //Delay(0x5FFFF);
}

```

A osciloskopem budeme pozorovat výstupní průběh na pinu **PC9**



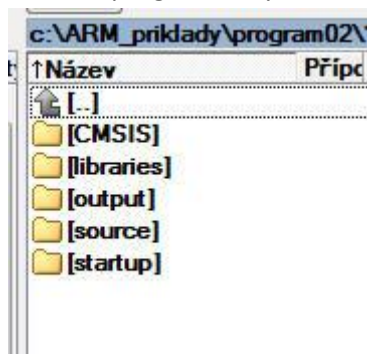
Při použití Delay (0x000FF) ; jsme docílili kmitočtů obdélníků cca 12kHz. Ještě o něco změníme zpoždění Delay (0x0000F) a nyní se průběh změnil na



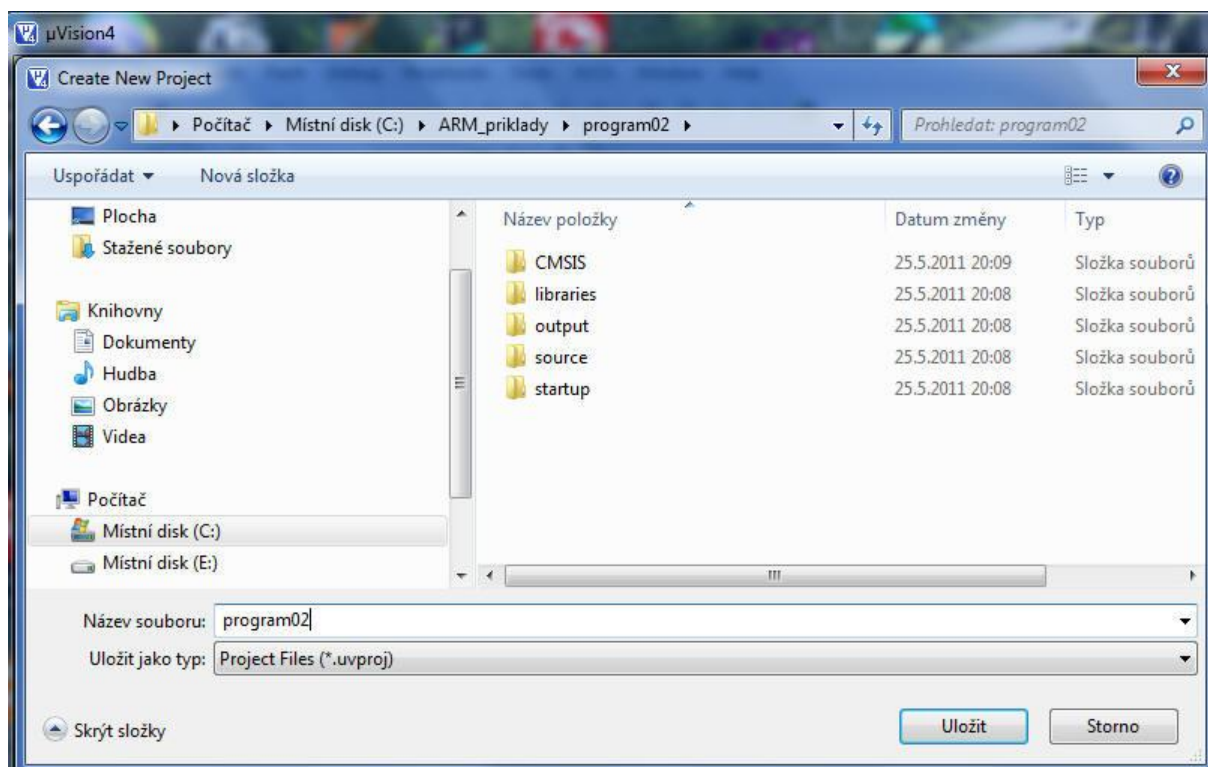
Výstupní kmitočet je cca 70kHz

2.2.2 Program blikáček LEDky podruhé

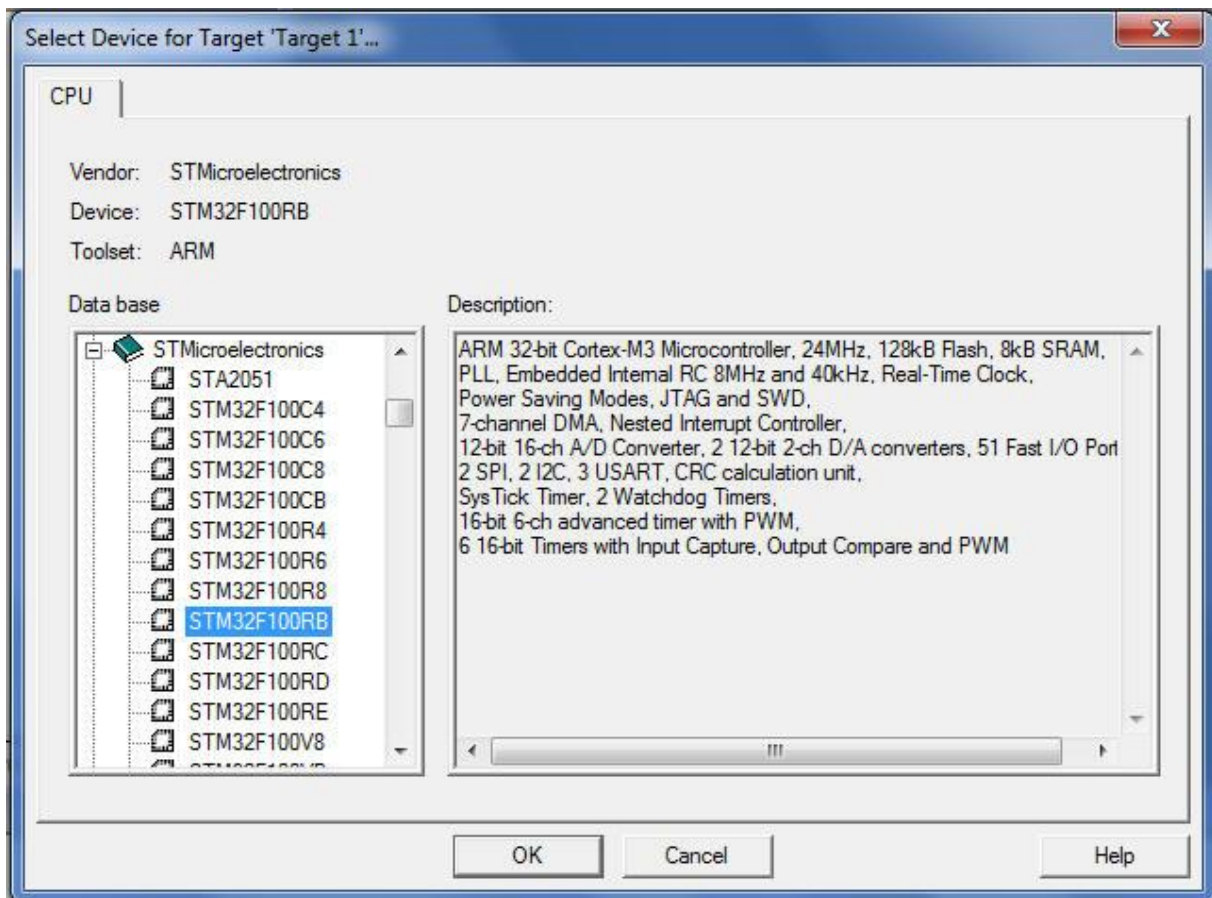
V složce program02 vytvoříme podadresáře



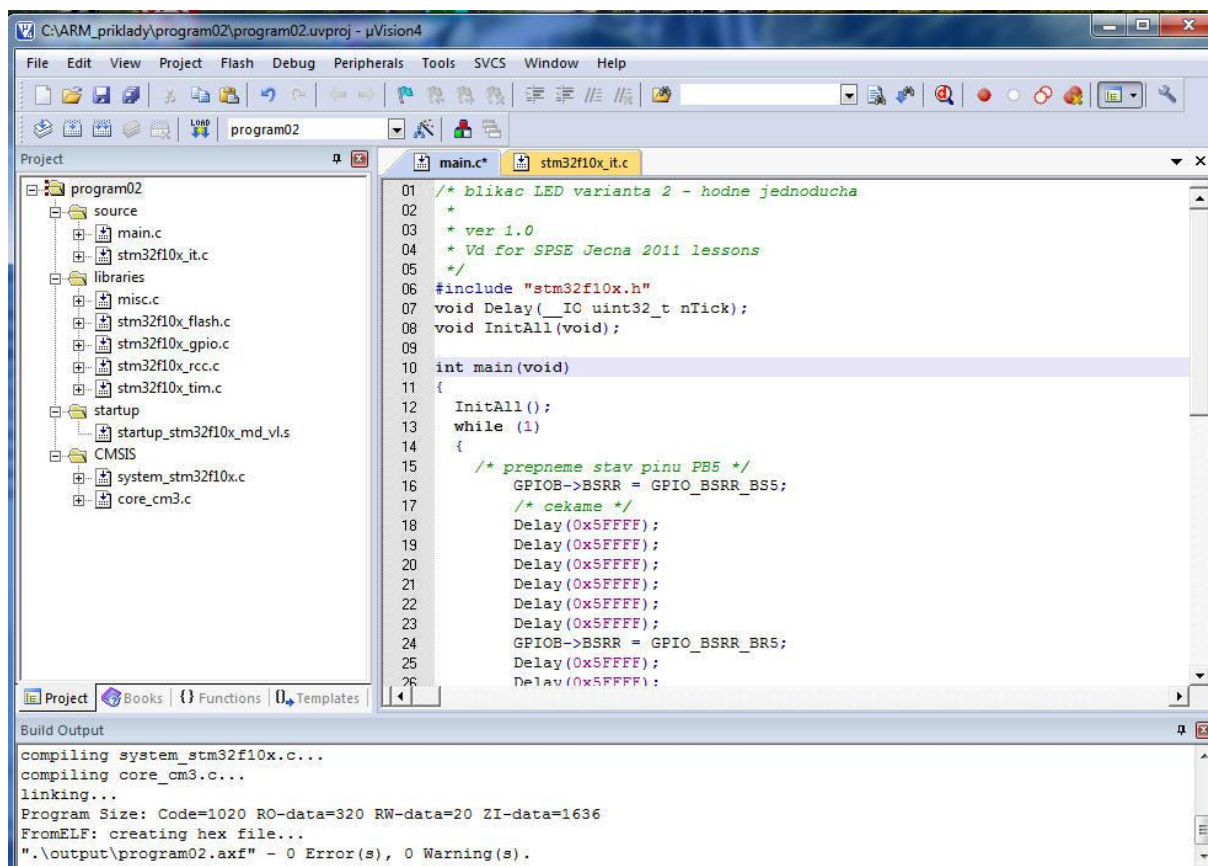
Spustíme Keil uVision4 a v menu vybereme **Project → New μVision Project ...**



Napíšeme **program02**, klikneme na **Uložit** atd. (viz postup u **program01**)



Vybereme CPU STM32F100RB a stiskneme **OK**. Další nastavení stejné jako u **program01**.



Zdrojový kód v souboru **main.c** bude

```

/* blikac LED varianta 2 - hodne jednoduchucha
*
* ver 1.0
* Vd for SPSE Jecna 2011 lessons
*/
#include "stm32f10x.h"
void Delay( IO uint32 t nTick);
void InitAll(void);

int main(void)
{
    InitAll();
    while (1)
    {
        /* prepneme stav pinu PB5 */
        GPIOB->BSRR = GPIO_BSRR_BS5;
        /* cekame */
        Delay(0x5FFFFF);
        Delay(0x5FFFFF);
        Delay(0x5FFFFF);
        Delay(0x5FFFFF);
        Delay(0x5FFFFF);
    }
}

```

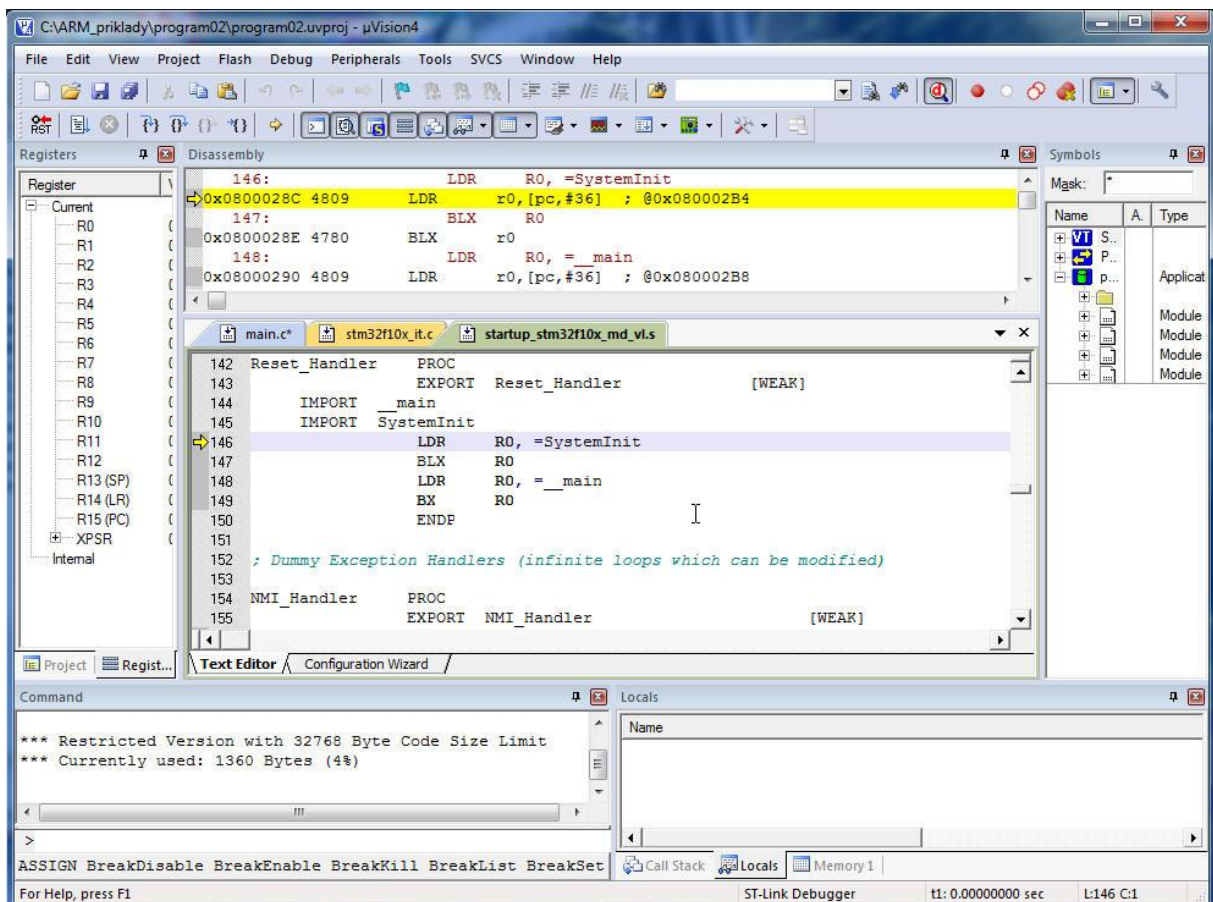
```

Delay(0x5FFFF);
GPIOB->BSRR = GPIO_BSRR_BR5;
Delay(0x5FFFF);
Delay(0x5FFFF);
Delay(0x5FFFF);
Delay(0x5FFFF);
Delay(0x5FFFF);
Delay(0x5FFFF);
}
}

void InitAll(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
    GPIOB->CRL &= ~(GPIO_CRL_MODE5 | GPIO_CRL_CNF5);
    GPIOB->CRL |= GPIO_CRL_MODE5_0;
}

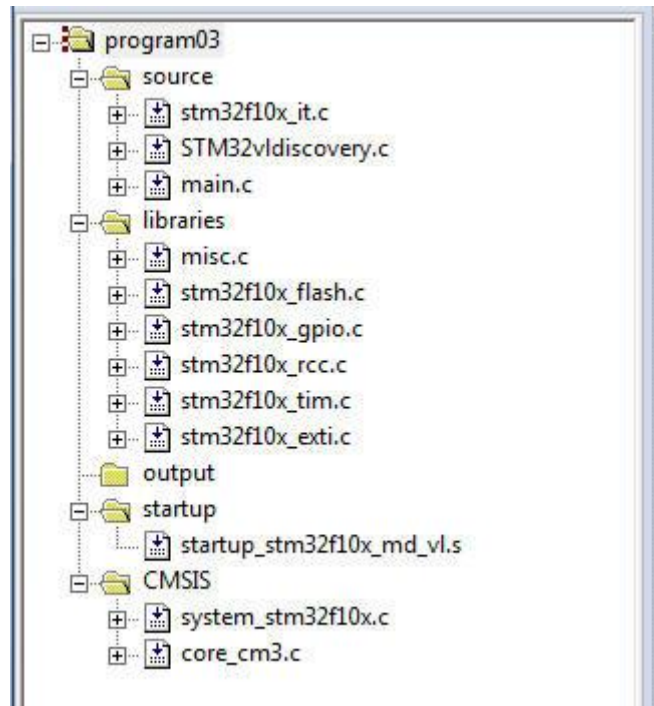
```

LEDku máme připojenou k **PB5** – v ukázce jsem nepoužil LEDky na startkitu, ale připojil jsem LEDku externí.



2.2.3 Blikač s LED potřebí, navíc s tlačítkem

Postup vytváření projektu a odpovídajících souborů bude stejný jako u předchozích programů, proto nám budou stačit následující informace – seznam souborů projektu a zdrojový kód souboru **main.c**. Stejně tomu bude i v dalších programech a tak vždy uvedeme jen obsah **main.c** a strukturu projektu.



A zdrojový kód hlavní funkce **main**

```

/* STM32 VL Discovery Library
 *
 * ver 1.0
 * Vd for SPSE Jecna 2011 lessons
 */

/* Includes -----*/

#include "stm32f10x.h"
#include "STM32vldiscovery.h"

void GPIO_Inicialization(void);
void Delay(__IO uint32_t nTick);

int main(void)
{
    GPIO_Inicialization();

```

```

STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridat jsem
Delay(0xAFFFFFFF);
while (1)
{
    if(STM32vldiscovery_PBGetState(BUTTON_USER) != 0)//kdyz se stiskne
    tlacitko, tak se provede nasledujici:
    {
        STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda
        STM32vldiscovery_LEDOff(LED4); // LED4 - modra dioda

        Delay(0xAFFFFFFF);
        STM32vldiscovery_LEDOff(LED3); // LED3 - zelena dioda
        STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda

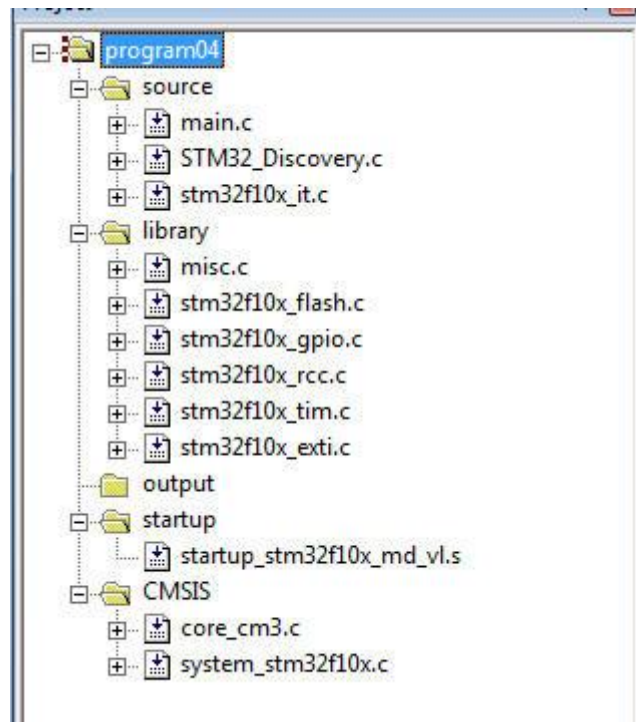
    }
}

void GPIO_Inicialization(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

2.2.4 Blikač s LED počtvrté, s tlačítkem



```

/* STM32 VL Discovery Library
 *
 * ver 1.0
 * Vd for SPSE Jecna 2011 lessons
 */

/* Includes -----*/

#include "stm32f10x.h"
#include "STM32_Discovery.h" //mirne jina knihovna, nez v program03

void GPIO_Inicialization(void);
void Delay(__IO uint32_t nTick);

int main(void)
{
    GPIO_Inicialization();

    STM32_Discovery_LEDOn(LED4); // LED4 - modra dioda
    STM32_Discovery_LEDOn(LED3); // LED3 - zelena dioda pridat jsem
    Delay(0xAFFFFFFF);
    while (1)
    {
        if(STM32_Discovery_PBGetState(BUTTON_USER) != 0) //kdyz se stiskne
            //tlacitko, tak se provede nasledujici:
            {

```

```

STM32_Discovery_LEDOn(LED3); // LED3 - zelena dioda
STM32_Discovery_LEDOff(LED4); // LED4 - modra dioda

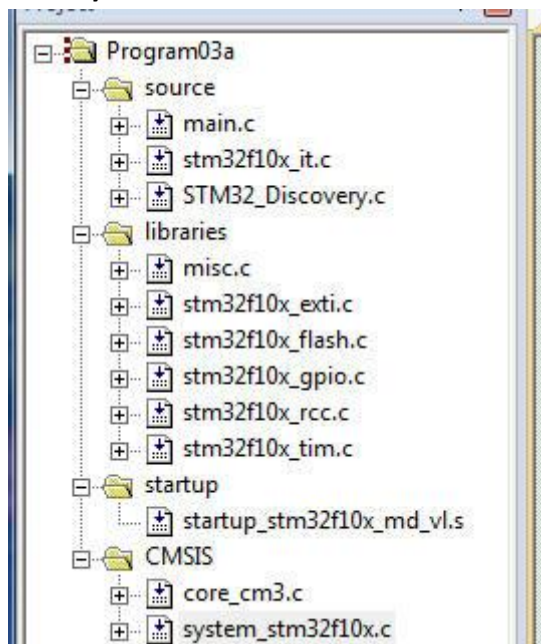
Delay(0xAFFFFFFF);
STM32_Discovery_LEDOff(LED3); // LED3 - zelena dioda
STM32_Discovery_LEDOn(LED4); // LED4 - modra dioda
}
}
}

void GPIO_Inicialization(void)
{
    STM32_Discovery_LEDInit(LED3);
    STM32_Discovery_LEDInit(LED4);
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    STM32_Discovery_LEDOff(LED4);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

2.2.5 SysTick



V main.c

```

/*
 * SysTick main.c
 * Blikani LED pomocí SysTick
 * podle MCU
 * cviceni SPSE
 */

#include <stdint.h>
#include "stm32f10x.h"
#include "STM32_Discovery.h"
#include "main.h"

__IO uint32_t Count;
static __IO uint32_t CasProDelay;

void GPIO_Inicializace(void);
void Delay(__IO uint32_t nCount);
void UbyvaniCasu_Delay(void);

int main(void)
{
    GPIO_Inicializace();

    /* Nastaveni casovace SysTick na 1 msec preruseni */
    if (SysTick_Config(SystemCoreClock / 1000))
    {
        /* Capture error */
        while (1);
    }

    while (1)
    {
        // kontrola stisku USER tlacitka
        if(0 != STM32_Discovery_PBGetState(BUTTON_USER))
        {
            // tj. když je stisknuto tlacitko
            Count = 350; // delsi cas
        }
        else
        {
            // tj. když není stisknuto tlacitko
            Count = 100; // kratssi cas
        }

        STM32_Discovery_LEDOn(LED3); // rozsvitime LED3 - zelena
        Delay(Count);

        STM32_Discovery_LEDOff(LED3); // zhasnem LED3 - zelena
        Delay(Count);
    }
}

void GPIO_Inicializace(void)

```



```

{
    STM32_Discovery_LEDInit(LED3);
    STM32_Discovery_LEDInit(LED4);
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    STM32_Discovery_LEDOff(LED4);
}

/**
 * @brief Inserts a delay time.
 * @param nTime: specifies the delay time length, in milliseconds.
 * @retval : None
 */
void Delay(__IO uint32_t nTick)
{
    CasProDelay = nTick;

    while(CasProDelay != 0); // cas se snizuje pomoci preruseni
}

void UbyvaniCasu_Delay(void)
{
    if (CasProDelay != 0x00)
    {
        CasProDelay--;
    }
}

```

V stm32f10x_it.c

```

#include "stm32f10x_it.h"
#include "main.h"

```

```

void SysTick_Handler(void)
{
    UbyvaniCasu_Delay();
}

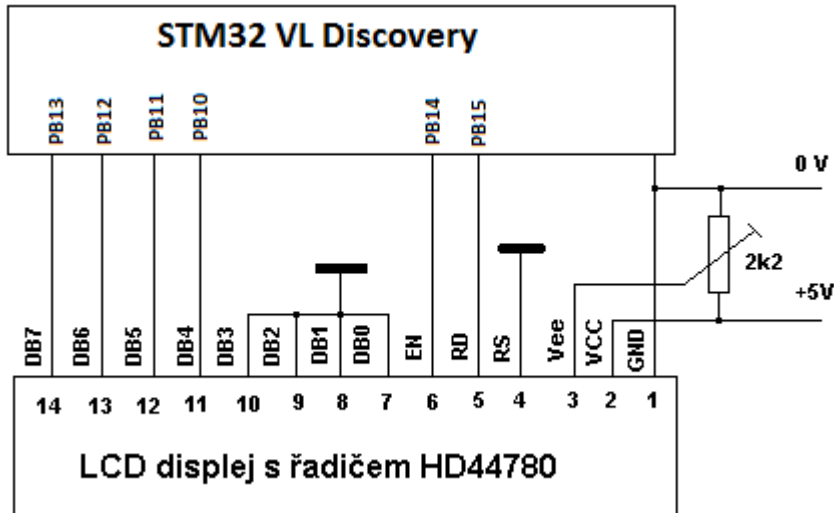
```

2.3 LCD displej

Velkého rozšíření mezi konstruktéry jako zobrazovací alfanumerické zařízení dosáhl inteligentní LCD displej s řadičem **HD44780** nebo jeho klon, takže všechny mají stejnou instrukční sadu a stejné ovládání. Tohoto rozšíření dosáhli díky výhodným vlastnostem. Je to především relativně snadné ovládání, možnost definice až osmi vlastních znaků (např. čeština), různorodost ve výběru velikosti (1 x 16 až 4 x 40 znaků) při zachování jednotného ovládání a v neposlední řadě se jejich specifikace stává defacto průmyslovým standardem. To usnadňuje ladění na HW i SW úrovni a případnou výměnu displeje (např. za větší, podsvícený apod.). Velkou výhodou LCD je malý odběr zobrazovací

matice, malé rozměry a nízká hmotnost ve srovnání s klasickou obrazovkou, lepší geometrie a ostrost zobrazení, delší životnost, stálost obrazu. Nevýhodou, která je však již u některých LCD odstraněna (na úkor ceny), je teplotní závislost, kdy kapalné krystaly při záporných a vysokých kladných teplotách ztrácejí své fyzikální vlastnosti a displej přestává dočasně pracovat.

S displejem lze komunikovat buď osmi nebo čtyřbitově. Náš program předpokládá čtyřbitovou komunikaci a propojení displeje s některým z portů STM32 MCU podle obr (např. k PORTB):

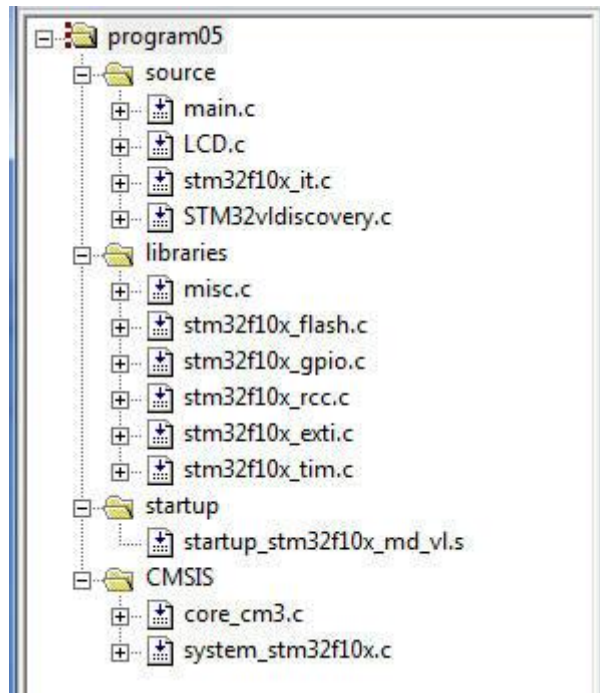


```
#define LCD_E      GPIO_Pin_14
#define LCD_RS     GPIO_Pin_15
#define LCD_D4     GPIO_Pin_10
#define LCD_D5     GPIO_Pin_11
#define LCD_D6     GPIO_Pin_12
#define LCD_D7     GPIO_Pin_13
#define LCD_PORT   GPIOB
```

Při této čtyřbitové komunikaci slouží DB4, DB5, DB6 a DB7 pro přenos instrukcí do řadiče LCD a pro zápis či čtení dat do/z paměti kódů znaků (DDRAM) nebo paměti uživatelských znaků (CGRAM). To, zda se zapisují data či instrukce je dáno signálem na RS (0 ... vstup instrukce, 1 Data), signál RD určuje zápis/čtení do LCD a EN znamená **platná data**. Pomocí HD44780 můžeme na displej zobrazovat pouze znakovou sadou uloženou v paměti ROM na řadiči, která je pro řadič vlastní viz. obrázek

| | | | | | | | | | | | | | |
|----------|----|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | |
| | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| xxxx0000 | | 0 | 0 | P | \ | P | - | 9 | 3 | 0 | P | | |
| xxxx0001 | ! | 1 | A | Q | a | q | . | 7 | 4 | ä | Q | | |
| xxxx0010 | " | 2 | B | R | b | r | 「 | イ | ツ | × | β | θ | |
| xxxx0011 | # | 3 | C | S | c | s | 」 | ウ | テ | モ | ε | ω | |
| xxxx0100 | \$ | 4 | D | T | d | t | 、 | エ | ト | ト | μ | Ω | |
| xxxx0101 | % | 5 | E | U | e | u | ・ | オ | ナ | 1 | 0 | ü | |
| xxxx0110 | & | 6 | F | V | f | v | ヲ | カ | ニ | ヨ | ρ | Σ | |
| xxxx0111 | ' | 7 | G | W | g | w | フ | キ | ヲ | ラ | Q | π | |
| xxxx1000 | (| 8 | H | X | h | x | イ | ク | ネ | リ | 」 | ⌘ | |
| xxxx1001 |) | 9 | I | Y | i | y | ッ | ト | ル | 」 | 」 | 」 | |
| xxxx1010 | * | : | J | Z | j | z | エ | コ | ン | レ | i | 〒 | |
| xxxx1011 | + | ; | K | [| k | [| オ | サ | ヒ | ロ | * | 〒 | |
| xxxx1100 | , | < | L | ¥ | l | ¥ | ハ | シ | フ | ワ | φ | 〒 | |
| xxxx1101 | - | = | M |] | m |] | ユ | ズ | ハ | シ | ト | ÷ | |
| xxxx1110 | . | > | N | ^ | n | ^ | ヨ | セ | ホ | 」 | 」 | 」 | |
| xxxx1111 | / | ? | 0 | _ | o | _ | ウ | ツ | マ | 」 | 」 | 」 | |

Stejně jako v předchozích příkladech, i nyní uvedeme jen nezbytné informace



Soubor LCD.c

```

/* STM32 VL Discovery Library
 * LCD driver LCD.c
 * ver 1.0
 * SPSE Jecna 2011 Vladimir Vana
 */

/* Includes -----
---*/
#include "LCD.h"

/** @defgroup Private_TypesDefinitions */
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructureMili =
{36000, TIM_CounterMode_Up, 0, 0, 0};

void delay_ms(uint16_t time);
void LCD_Inicialization(void);
void SendCommandLCD(uint8_t cmd);
void SendCharLCD(uint8_t character);
void LCD_Clear(void);
void LCD_CursorHome(void);
void printLCD(uint8_t* message);
void SendBitsLCD(uint8_t bits);
void Send4bitLCD(uint8_t cmd);
void FirstRow(void);
void SecondRow(void);

/**
 * @brief Nastaveni LCD a pinu.
 * @retval : None
 */
void LCD_Inicialization(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(LCD_PORT_CLK, ENABLE);

    GPIO_InitStructure.GPIO_Pin = LCD_E | LCD_RS | LCD_D4 | LCD_D5 | LCD_D6 |
LCD_D7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(LCD_PORT, &GPIO_InitStructure);

    GPIO_ResetBits(LCD_PORT, LCD_RS);
    //RS=0:Instruction
    GPIO_ResetBits(LCD_PORT, LCD_E);
    //E=0:Disable
    //delay_us(200000);
    delay_ms(200);
    Send4bitLCD(0x30);
    //delay_us(8000); // dle datasheetu cekat vice nez 4.1 ms
    delay_ms(8);
}

```

```

GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
Send4bitLCD(0x30);
//delay_us(1000);
delay_ms(1);
GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
Send4bitLCD(0x30);
//delay_us(1000);
delay_ms(1);
GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
// delay_us(1000);
delay_ms(1);
// ted by mel byt display v 8-bitovem modu a definitivne ho prepneme do
4-bitoveho modu!
Send4bitLCD(0x20);
//delay_us(1000);
delay_ms(1);
GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
//delay_us(1000);
delay_ms(1);
// nyni jsme ve 4-bitovem rezimu
// Prikaz 0 0 1 DL N F - -
// N = pocet radek 1=2 radky, 0=1 radka
// DL = bitu komunikace 1=8bitu, 0=4bity
// F = Font 1=5x10bodou, 0=5x8 bodu
SendCommandLCD(0x28);
//delay_us(1000);
delay_ms(1);
// zvedej sam adresu, posunuj kurzor
SendCommandLCD(0x0E); //nyni OK
//delay_us(1000);
delay_ms(1);
//PosliPrikazLCD(0b00000110);
LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici prvnioho
radku
}

/**
 * @brief Smaze LCD a nastavi LCD kurzor na home pozici
 * @retval : None
 */
void LCD_Clear(void)
{
    SendCommandLCD(0x01);
}

/**
 * @brief Nastavi LCD kurzor na home pozici
 * @retval : None
 */
void LCD_CursorHome(void)
{
    SendCommandLCD(0x02);
}

```

```

/**
 * @brief Posle prikaz do LCD
 * @param cmd prikaz do LCD
 * @retval : None
 */
void SendCommandLCD(uint8_t cmd) //v promenne cmd mame zaslany prikaz
{
    GPIO_ResetBits(LCD_PORT, LCD_RS);
    //RS=0:Instruction
    SendBitsLCD(cmd); // posle bity jako instrukci
}

/**
 * @brief Posle znak do LCD
 * @param character znak
 * @retval : None
 */
void SendCharLCD(uint8_t character) //v promenne cmd je znak poslany na LCD
{
    GPIO_SetBits(LCD_PORT, LCD_RS); //RS=1:Data

    SendBitsLCD(character); // posle bity jako data
}

/**
 * @brief Posle bity do LCD
 * @param bits znak nebo prikaz
 * @retval : None
 */
void SendBitsLCD(uint8_t bits)
{
    Send4bitLCD(bits); // zaslani bitu 4-7
    bits = bits<<4; // bitovy posun na zaslani bitu 0 az 3
    Send4bitLCD(bits); // zaslani bitu 0-3
}

/**
 * @brief Posle retezec znaku do LCD
 * @param message pointer na retezec
 * @retval : None
 */
void printLCD(uint8_t* message)
{
    uint8_t i=0;
    if (!message) return;
    while(message[i]!='\0')
    {
        if(i%20==0){
            if(i%40==0){ // pokud je na konci druheho radku prejde na
zacatek prvniho
                FirstRow();
                delay_ms(1);
            }else{ // pokud je na konci prvniho radku
prejde na zacatek druheho
                SecondRow();
            }
        }
        SendBitsLCD(message[i]);
        i++;
    }
}

```

```

    }
    }
    SendCharLCD(message[i]);
    i++;
}

/**
 * @brief Prejde na prvni radek
 * @retval : None
 */
void FirstRow(void)
{
    SendCommandLCD(0x80); // prejde na zacatek prvnio radku - na adresu
prvniho znaku
    delay_ms(1);
}

/**
 * @brief Prejde na druhy radek
 * @retval : None
 */
void SecondRow(void)
{
    SendCommandLCD(0xC0); // prejde na zacatek druheho radku - na
adresu prvniho znaku
    delay_ms(1);
}

/**
 * @brief Posle po 4 bitech prikaz nebo znak do LCD
 * @param command
 * @retval : None
 */
void Send4bitLCD(uint8_t cmd)
{
    GPIO_ResetBits(LCD_PORT, LCD_E);
    delay_ms(1);
    if (cmd & 0x10) // vymaskovani 4 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D4);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D4);
    }
    if (cmd & 0x20) // vymaskovani 5 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D5);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D5);
    }
    if (cmd & 0x40) // vymaskovani 6 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D6);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D6);
    }
}

```

```

    if (cmd & 0x80) // vymaskovani 7 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D7);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D7);
    }
    GPIO_SetBits(LCD_PORT, LCD_E);
    //E=1:Enable
    delay_ms(1);
    GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
}

/**
 * @brief Cekat n milisekund
 * @param time kolik ms se ma cekat
 * @retval None
 */
void delay_ms(uint16_t time)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
    TIM_TimeBaseStructureMili.TIM_Period = ((time+1) * 2)-1;
    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructureMili);
    TIM_SelectOnePulseMode(TIM4, TIM_OPMode_Single);
    TIM_SetCounter(TIM4, 2);
    TIM_Cmd(TIM4, ENABLE);
    while (TIM_GetCounter(TIM4)){};
    TIM_Cmd(TIM4, DISABLE);
}

```

Soubor LCD.h

```

/* STM32 VL Discovery Library
 * LCD driver LCD.h
 * ver 1.0
 * Vd for SPSE Jecna 2011 lessons
 */

#ifndef LCD_H_
#define LCD_H_

/* Includes -----
---*/
#include "stm32f10x.h"

/** @defgroup Exported_Types */
/** @defgroup Exported_Functions */

void delay_ms(uint16_t time);
void LCD_Inicialization(void);
void SendCommandLCD(uint8_t cmd);

```



```
void SendCharLCD(uint8_t cmd);
void LCD_Clear(void);
void LCD_CursorHome(void);
void printLCD(uint8_t* message);
void Send4bitLCD(uint8_t cmd);
void FirstRow(void);
void SecondRow(void);

/** @defgroup Defines */
// piny pro ovladani displeje - zapojeni stejne jako na Katedre mereni CVUT
#define LCD_E      GPIO_Pin_14
#define LCD_RS     GPIO_Pin_15
#define LCD_D4     GPIO_Pin_10
#define LCD_D5     GPIO_Pin_11
#define LCD_D6     GPIO_Pin_12
#define LCD_D7     GPIO_Pin_13
#define LCD_PORT   GPIOB
#define LCD_PORT_CLK   RCC_APB2Periph_GPIOB

#endif /* LCD_H_ */
```

Soubor main.c

```
/* STM32 VL Discovery Library
 * LCD driver LCD.c
 * ver 1.0 for SPSE Jecna lectures
 * Created Vd on: Apr 20, 2011
 */

/* Includes -----
---*/
#include <stdint.h>
#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include "LCD.h"

uint8_t LCD_Message1[] = "Test displeje.";
uint8_t LCD_Message2[] = "Stisk tlacitka! ";

/* Private function prototypes -----
---*/

void GPIO Inicialization(void);
void Delay(__IO uint32_t nTick);

int main(void)
{
    LCD_Inicialization();
    GPIO_Inicialization();

    STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
```

```

STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridat jsem

printLCD(LCD_Message1);

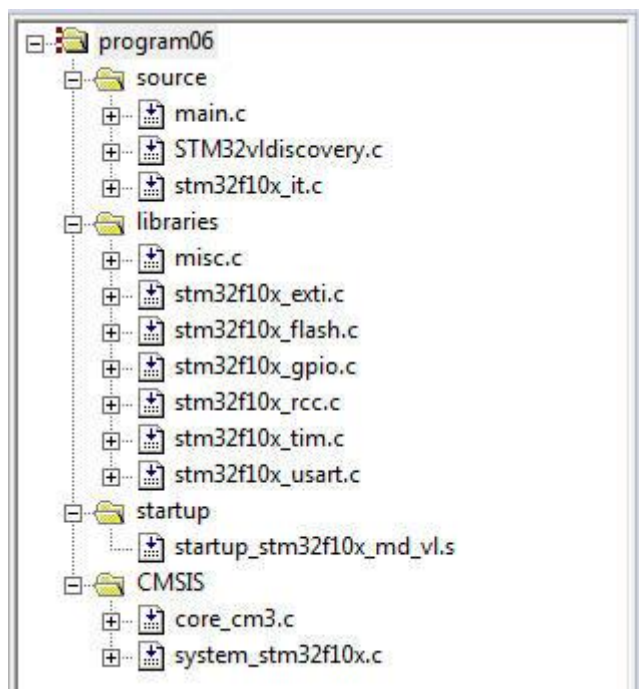
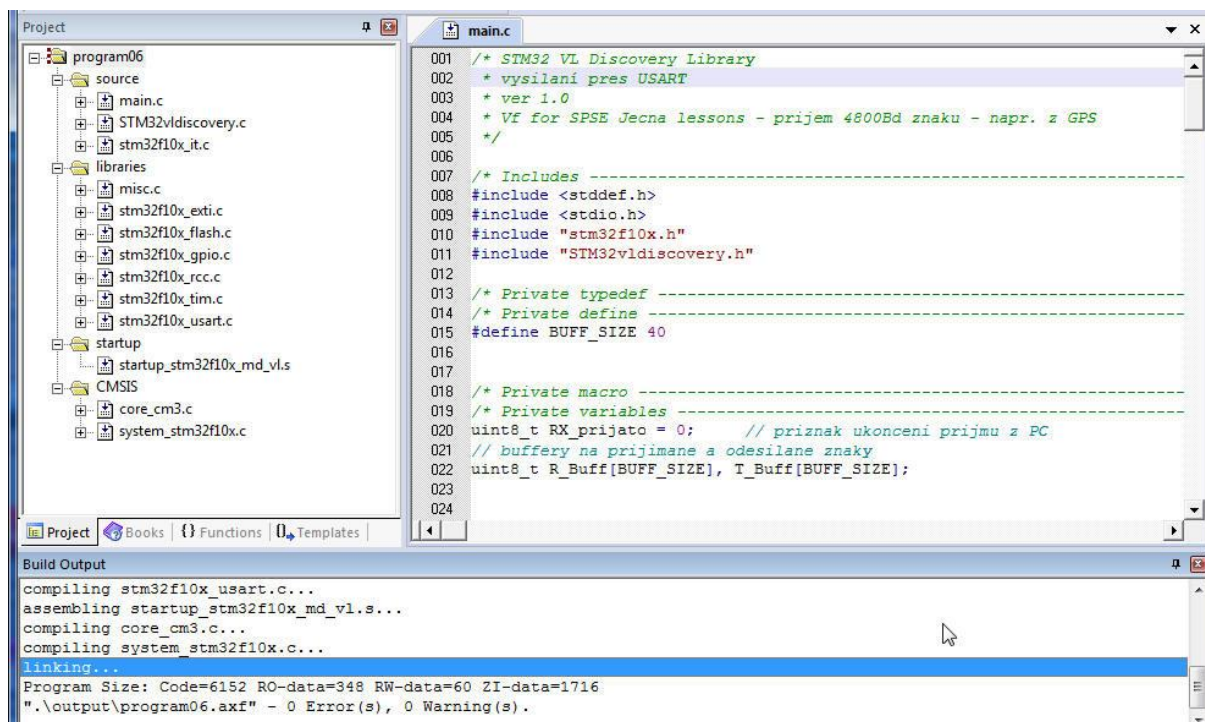
while (1)
{
    if (STM32vldiscovery_PBGetState (BUTTON_USER) != 0) //kdyz se stiskne
    tlacitko, tak se provede nasledujici:
    {
        STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda
        STM32vldiscovery_LEDOff(LED4); // LED4 - modra dioda
        LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici
        prvniho radku
        printLCD(LCD_Message2); // zprava je delsi nez oba radky
        displeje - demonstuje prechod mezi radky v metode printLCD v knihovne
        LCD.c
        Delay(0xAFFFFFFF);
        STM32vldiscovery_LEDOff(LED3); // LED3 - zelena dioda
        STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
        LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici
        prvniho radku
        printLCD(LCD_Message1);
    }
}

void GPIO_Inicialization(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit (BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

2.4 UART TX – vysílání znaků (program06)



Soubor main.c

```

/* STM32 VL Discovery Library
 * vysilání pres USART
    
```

```

* ver 1.0 Program06
* Vf for SPSE Jecna lessons - vysilani 4800Bd znaku - napr. do GPS
*/

#include <stddef.h>
#include <stdio.h>
#include "stm32f10x.h"
#include "STM32vldiscovery.h"

#define BUFF_SIZE 40

uint8_t RX_prijato = 0;          // priznak ukonceni prijmu z PC
// buffery na prijimane a odesilane znaky
uint8_t R_Buff[BUFF_SIZE], T_Buff[BUFF_SIZE];

void USART_Inicializace(void);
void GPIO_Inicialization(void);
int fputc(int ch, FILE * f);

void Delay(__IO uint32_t nTick);
uint8_t Strcmp (const uint8_t * s1, const uint8_t * s2); // porovnani str
void Strcpy(uint8_t *d1, const uint8_t *s1); // kopirovani str

int main(void)
{

    GPIO_Inicialization();
    // aktivujeme USART
    USART_Inicializace();

    STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
    STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridal jsem

    while (1)
    {
        if(STM32vldiscovery_PBGetState(BUTTON_USER) != 0)//kdyz se stiskne
        tlacitko, tak se provede nasledujici:
        {
            STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda
            STM32vldiscovery_LEDOff(LED4); // LED4 - modra dioda
            Strcpy (T_Buff, "> Tlacitko stisknuto - ahoj\n\r"); //
            nastavime co se odesle po USART
            USART_ITConfig(USART1, USART_IT_TXE, ENABLE); // povolime
            preruseni

            Delay(0xAFFFFFFF);
            STM32vldiscovery_LEDOff(LED3); // LED3 - zelena dioda
            STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
            Strcpy (T_Buff, "> stiskni tlacitko - nazdar\n\r"); //
            nastavime co se odesle po USART
            USART_ITConfig(USART1, USART_IT_TXE, ENABLE); // povolime
            preruseni
        }
    }
}

```

```

    }
}

void GPIO_Inicialization(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void USART_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    /* pustime hodiny do periferie (protoze jde o alternativni funkci,
musi
    * jit hodiny i do AFIO periferie! */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
RCC_APB2Periph_AFIO, ENABLE);

    // PA9 je Tx
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // alternate
function!
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // PA10 je Rx
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    USART_InitStructure.USART_BaudRate = 4800;
    /* Pokud mate dobry prevodnik USB-ttl, tak muzete pochopitelne
    * zvyсит rychlost. Hodnota 4800 je pouzivana GPS */
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART1, ENABLE);

    // jeste zbyva konfigurace preruseni

```

```

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

/**
 * @brief Porovnání dvou řetězců
 * @param řetězec1, řetězec2
 * @retval : 0 - nerovná se, 1 - rovná se, typ uint8_t
 */
uint8_t Strcmp (const uint8_t * s1, const uint8_t * s2)
{
    for(; *s1 == *s2; ++s1, ++s2)
        if(*s1 == 0 || *s1 == 0x0d)
            return 0;
    return *(unsigned char *)s1 < *(unsigned char *)s2 ? -1 : 1;
}

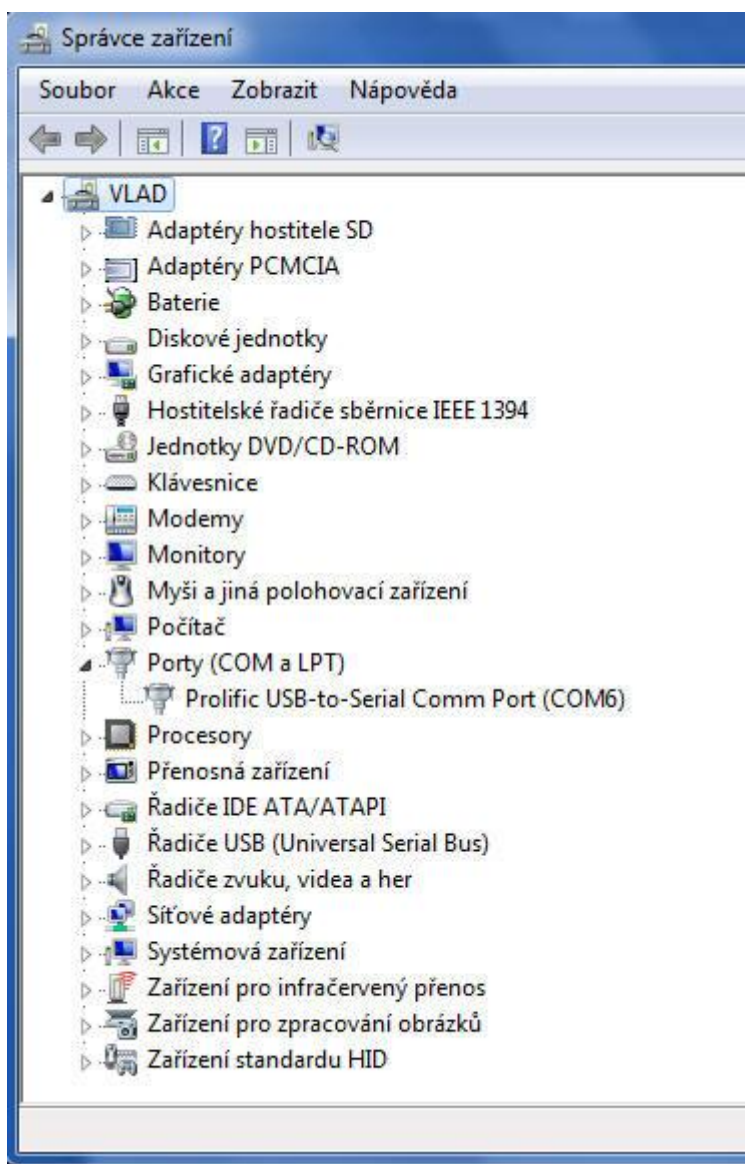
/**
 * @brief Kopírování jednoho řetězce do druhého
 * @param pointer na cílový řetězec, pointer na zdrojový řetězec
 * @retval : none
 */
void Strcpy(uint8_t *d1, const uint8_t *s1)
{
    uint8_t i;
    for (i=0; s1[i] != '\0'; ++i)
        d1[i] = s1[i];
    d1[i] = '\0';
}

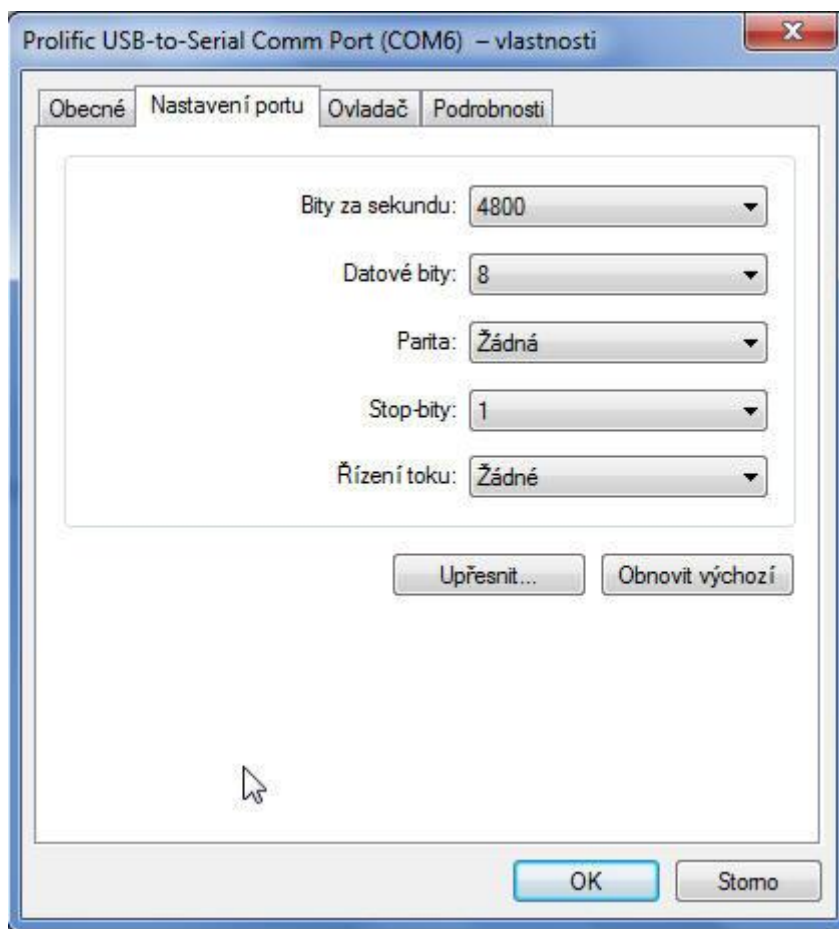
/**
 * @brief Čekání
 * @param nTick: délka čekací smyčky
 * @retval : None
 */
void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

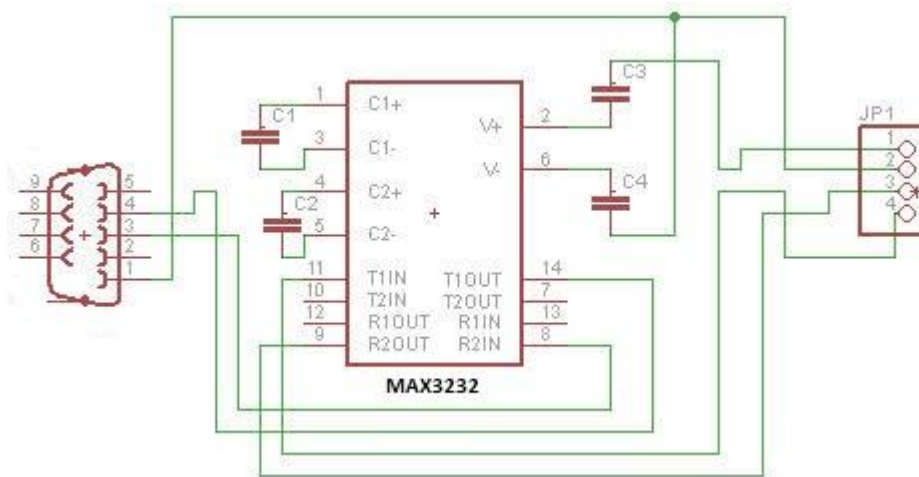
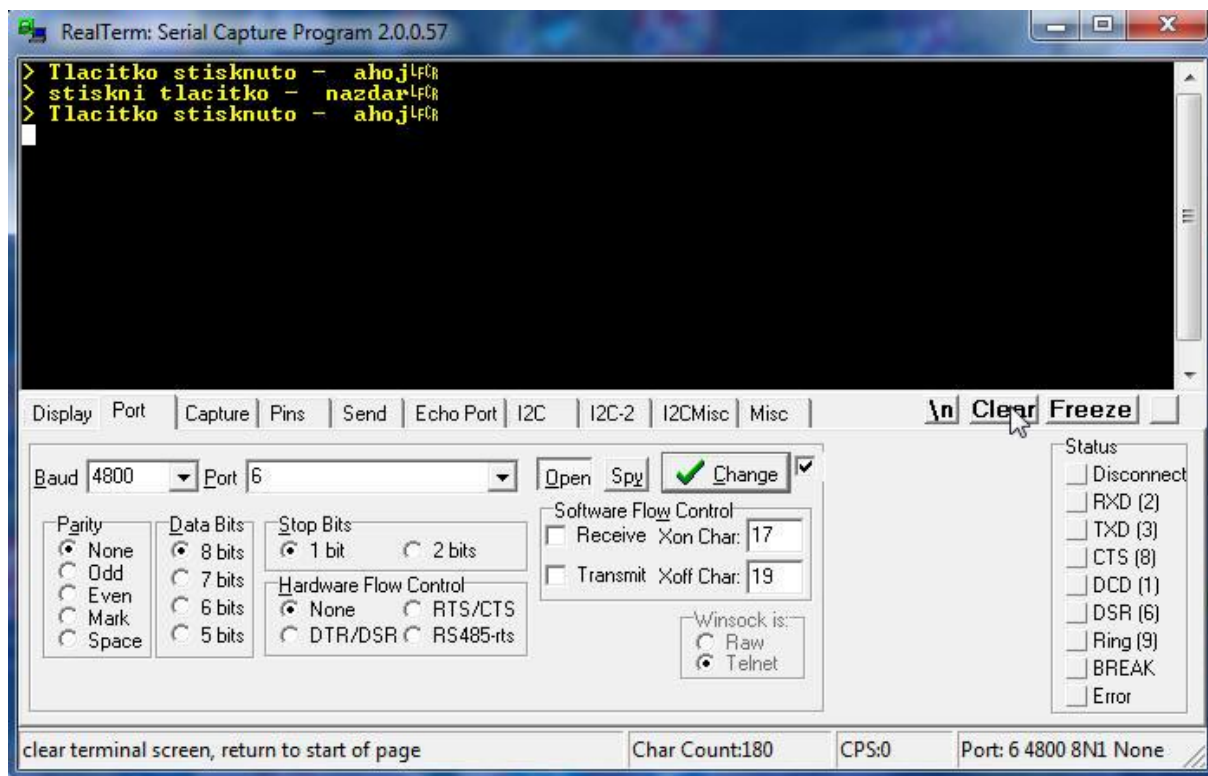
/* Zapsání znaku 'ch' do souboru 'f' */
int fputc(int ch, FILE * f)
{
    /* Předání znaku pro odeslání UARTem */
    USART_SendData(USART1, (u8) ch);
    /* Čekáme, dokud není přenos dokončen */
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    return ch;
}

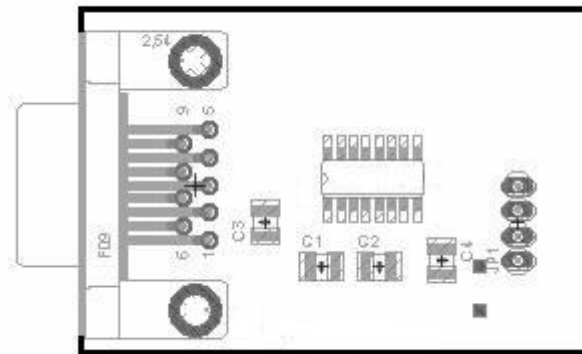
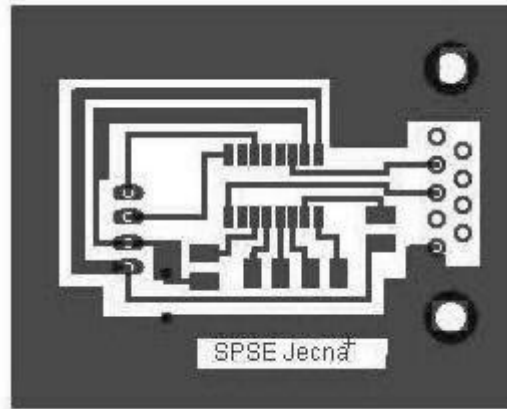
```

PA9 je připojen na pin 11 obvodu MAX3232 a po převedení na úroveň RS232 na pin 2 konektoru CANON9 (poz. určitě znáte dnes již klasický obvod MAX232 pro převod TTL \leftrightarrow RS232 jenž je napájen 5V, obvod MAX3232 je s ním pinově kompatibilní a může být napájen 5V či 3.3V přičemž při napájení 5V se chová jako obvod MAX232, při napájení 3.3V slouží k převodu mezi RS232 a 3V logikou s nímž mj pracuje i startkit **STM32 VL DISCOVERY**)

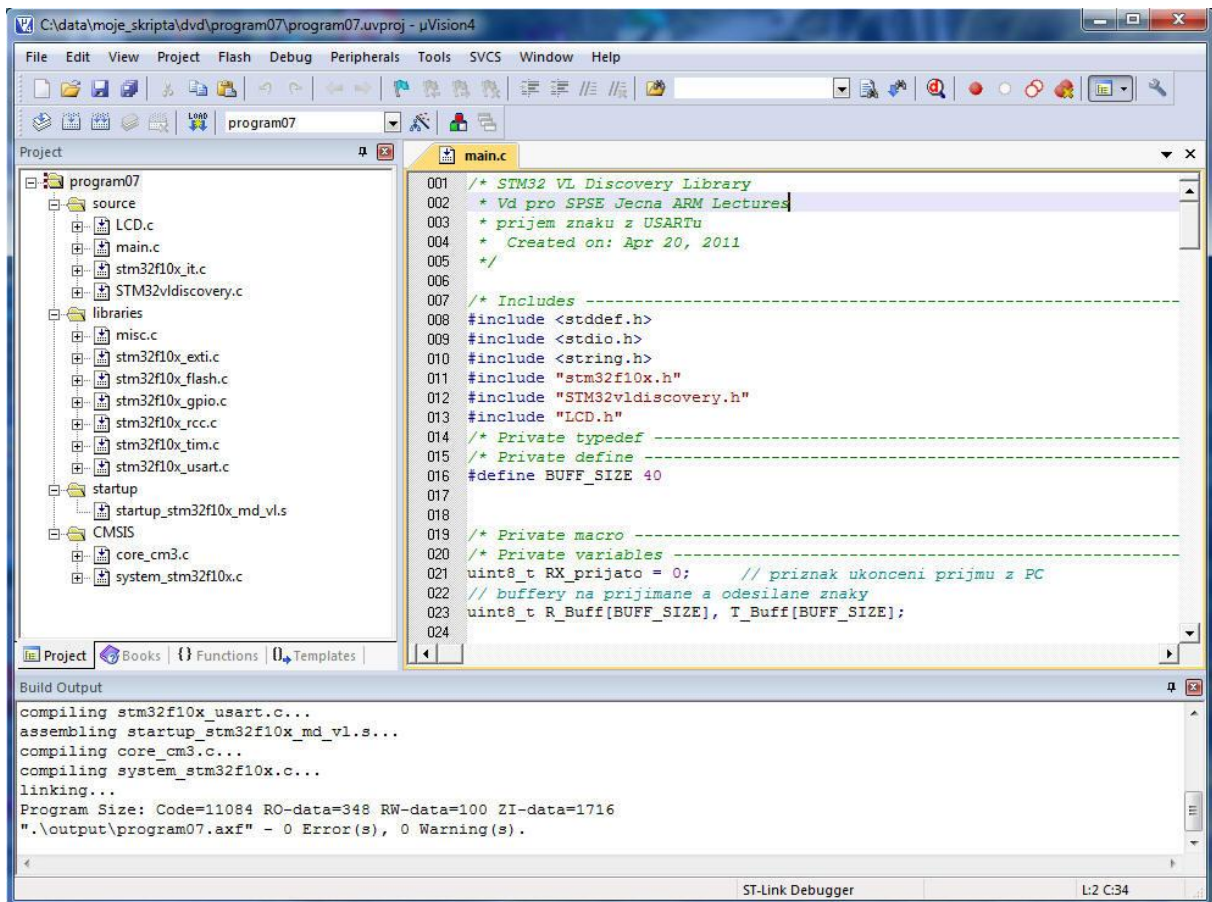








2.5 UART RX – příjem znaků (program07)



The screenshot shows the uVision4 IDE interface. The main window displays the source code for 'main.c' in the 'program07' project. The code includes comments in Czech and C headers. The Build Output window at the bottom shows the compilation and linking process, indicating a successful build with no errors or warnings.

```

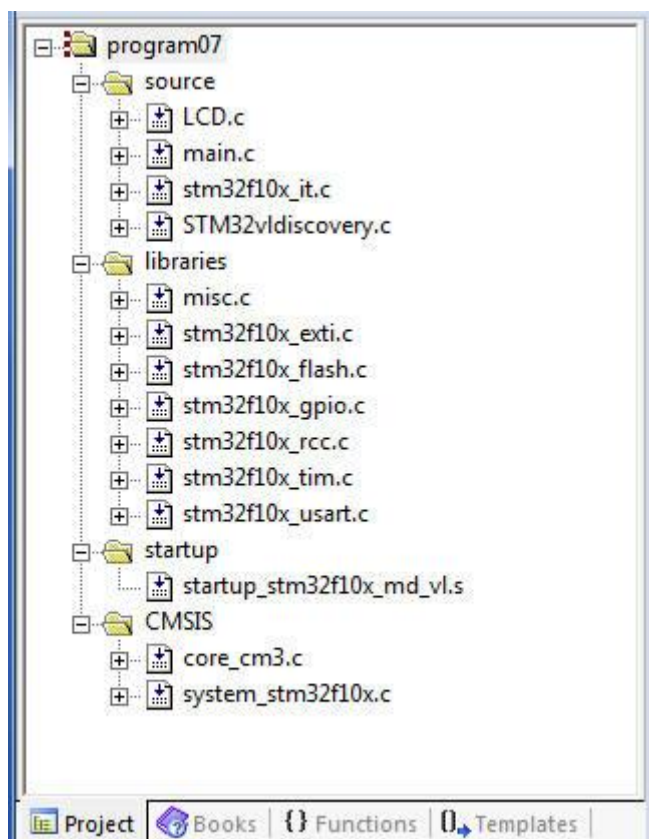
001 /* STM32 VL Discovery Library
002 * Vd pro SPSE Jecna ARM Lectures
003 * prijem znaku z USARTu
004 * Created on: Apr 20, 2011
005 */
006
007 /* Includes -----
008 #include <stddef.h>
009 #include <stdio.h>
010 #include <string.h>
011 #include "stm32f10x.h"
012 #include "STM32vldiscovery.h"
013 #include "LCD.h"
014 /* Private typedef -----
015 /* Private define -----
016 #define BUFF_SIZE 40
017
018
019 /* Private macro -----
020 /* Private variables -----
021 uint8_t RX_prijato = 0; // priznak ukonceni prijmu z PC
022 // buffery na prijimane a odesilane znaky
023 uint8_t R_Buff[BUFF_SIZE], T_Buff[BUFF_SIZE];
024

```

```

compiling stm32f10x_usart.c...
assembling startup_stm32f10x_md_v1.s...
compiling core_cm3.c...
compiling system_stm32f10x.c...
linking...
Program Size: Code=11084 RO-data=348 RW-data=100 ZI-data=1716
".\output\program07.axf" - 0 Error(s), 0 Warning(s).

```



Soubor main.c

```

/* STM32 VL Discovery Library
 * Vd pro SPSE Jecna ARM Lectures
 * příjem znaku z USARTu, zobrazení na LCD
 * Created on: Apr 20, 2011 program07
 */

/* Includes -----
---*/
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include "LCD.h"
/* Private typedef -----
---*/
/* Private define -----
---*/
#define BUFF_SIZE 40

```

```

uint8_t RX_prijato = 0;          // priznak ukonceni prijmu z PC
// buffery na prijimane a odesilane znaky
uint8_t R_Buff[BUFF_SIZE], T_Buff[BUFF_SIZE];

uint8_t LCD_Message1[] = "Prijem GPS 4800Bd";
uint8_t LCD_Message2[] = "Stisk tlacitka! ";

/* Private function prototypes -----
---*/
void USART_Inicializace(void);
void GPIO_Inicialization(void);
int fputc(int ch, FILE * f);

void Delay(__IO uint32_t nTick);
uint8_t Strcmp (const uint8_t * s1, const uint8_t * s2); // porovnani str
void Strcpy(uint8_t *d1, const uint8_t *s1); // kopirovani str
void vynulovaniRXregistru(void);

int main(void)
{
    LCD_Inicialization();
    GPIO_Inicialization();
    // aktivujeme USART
    USART_Inicializace();
    STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
    STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridat jsem
    printLCD(LCD_Message1);
    while (1)
    {
        if (RX_prijato == 1) // neco jsme prijali - zakoncene CR
        {
            LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici prvnioho
radku
            Strcpy (LCD_Message1,R_Buff );
            printLCD(LCD_Message1);
            vynulovaniRXregistru();
            RX_prijato = 0; // vymazeme flag ukonceni prijmu
        }
    }
}

void GPIO_Inicialization(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

```

```

}

void USART_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    /* pustime hodiny do periferie (protoze jde o alternativni funkci,
musi
    * jit hodiny i do AFIO periferie! */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
RCC_APB2Periph_AFIO, ENABLE);

    // PA9 je Tx
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // alternate
function!
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // PA10 je Rx
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    USART_InitStructure.USART_BaudRate = 4800;
    /* Pokud mate dobry prevodnik USB-ttl, tak muzete pochopitelne
    * zvyсит rychlost. Hodnota 4800 je pro prijem z GPS */
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART1, ENABLE);

    // jeste zbyva konfigurace preruseni
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

/**
 * @brief Porovnani dvou retezcu
 * @param retezec1, retezec2
 * @retval : 0 - nerovna se, 1 - rovna se, typ uint8_t
 */

```

```

uint8_t Strcmp (const uint8_t * s1, const uint8_t * s2)
{
    for (; *s1 == *s2; ++s1, ++s2)
        if (*s1 == 0 || *s1 == 0x0d)
            return 0;
    return *(unsigned char *)s1 < *(unsigned char *)s2 ? -1 : 1;
}

/**
 * @brief Prekopirovani jednoho retezce do druheho
 * @param pointer na cilovy retezec, pointer na zdrojovy retezec
 * @retval : none
 */
void Strcpy(uint8_t *d1, const uint8_t *s1)
{
    uint8_t i;
    for (i=0; s1[i] != '\0'; ++i)
        d1[i] = s1[i];
    d1[i] = '\0';
}

void Delay(__IO uint32_t nTick)
{
    for (; nTick != 0; nTick--);
}

/* Zapis znaku 'ch' do souboru 'f' */
int fputc(int ch, FILE * f)
{
    /* Predani znaku pro odeslani UARTem */
    USART_SendData(USART1, (u8) ch);
    /* Cekame, dokud neni prenos dokoncen */
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);

    return ch;
}

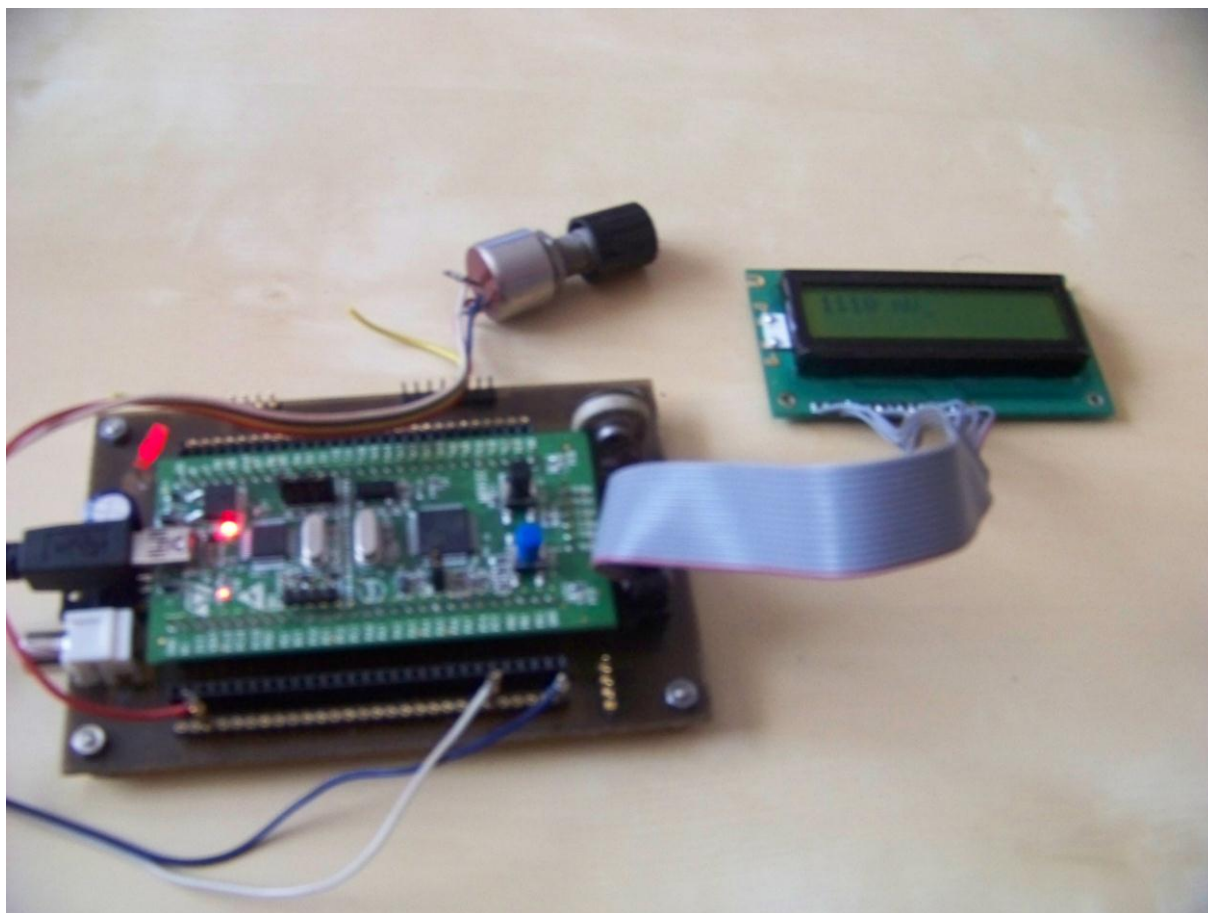
void vynulovaniRXregistru(void)
{
    uint8_t i;
    for(i=0; i<40; i++)
        R_Buff[i]= '\0';
    ;
}

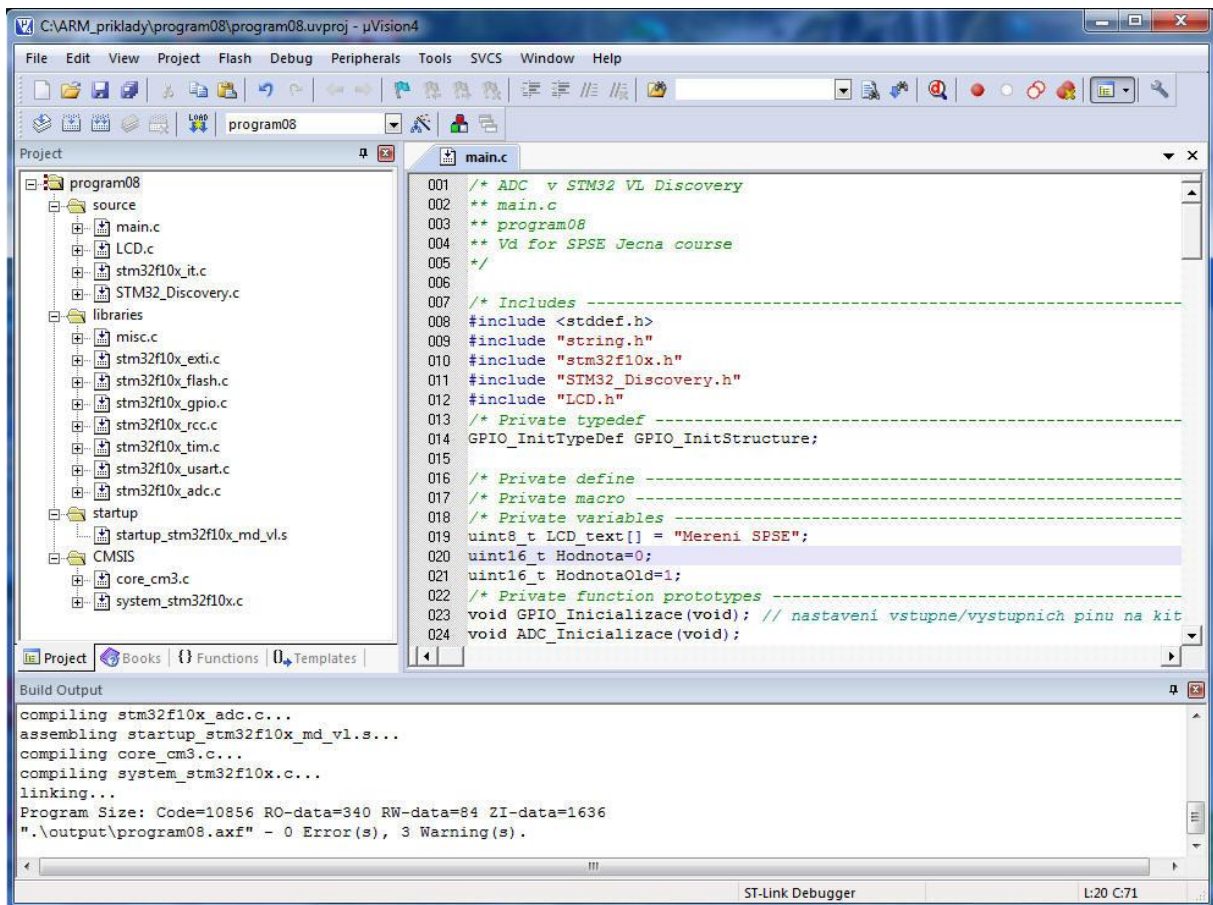
```

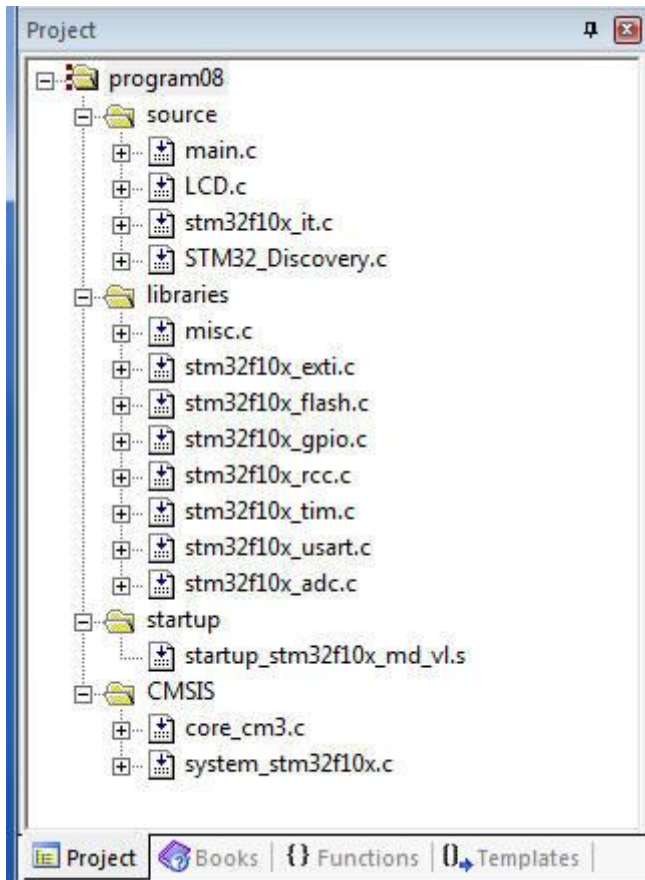
2.6 ADC (program08)

Mezi GND a +3,3V připojíme potenciometr cca 22k, jeho běžec spojíme s **PC4** mající význam

ADC1_INT14







Soubor **main.c**

```

/* ADC v STM32 VL Discovery
** main.c
** program08
** Vd for SPSE Jecna course
*/

#include <stdint.h>
#include "string.h"
#include "stm32f10x.h"
#include "STM32_Discovery.h"
#include "LCD.h"

GPIO_InitTypeDef GPIO_InitStructure;

uint8_t LCD_text[] = "Mereni SPSE";
uint16_t Hodnota=0;
uint16_t HodnotaOld=1;

void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu

```

```

void ADC_Inicializace(void);
void Delay(__IO uint32_t nTick);
uint16_t ADC1_Read(void);
void itoa(uint16_t n, int8_t s[]);
void reverse(int8_t s[]);
void KopirujLCDtext(int8_t s[]);
void barGraph(uint16_t Hodnota);

int main(void)
{
    //GPIO_LCD_Inicializace();
    LCD_Inicialization();
    STM32_Discovery_LEDOn(LED4);          // LED4 - modra
    //Delay(0xAAAA);
    GPIO_Inicializace();
    //LCD_Init();
    LCD_Clear();
    ADC_Inicializace();
    printLCD(LCD_text);
    Delay(0xAAAAA);
    STM32_Discovery_LEDOff(LED4); // LED4 - modra

    while (1)
    {
        Delay(0xAAAAA);
        Delay(0xAAAAA);
        Delay(0xAAAAA);
        LCD_Clear();
        //Hodnota = ADC1_Read(); //Hodnota je cele cislo mezi 0 a 4095
        Hodnota = (ADC1_Read() * 3000)/4095;
        itoa(Hodnota,LCD_text);
        *strcat(LCD_text," mV"); //k s1 prida s2
        printLCD(LCD_text);
    }
}

void GPIO_Inicializace(void)
{
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    //STM32_Discovery_LEDOff(LED4);
}

void ADC_Inicializace(void)
{
    ADC_InitTypeDef  ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    /* ADCCLK = PCLK2/1 = 24/2 = 12MHz*/
    RCC_ADCCLKConfig(RCC_PCLK2_Div2);
    /* povolime hodiny do portu */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4; // tady pripojime merene
    napeti
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
}

```

```

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOC, &GPIO_InitStructure);
/* povolime hodiny do A/D prevodniku */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
//ADC_DeInit(ADC1);
/* ADC1 Configuration -----
---*/
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init( ADC1, &ADC_InitStructure );
/* ADC1 Regular Channel 14 */
ADC_RegularChannelConfig( ADC1, ADC_Channel_14, 1,
ADC_SampleTime_239Cycles5);
//ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1,
ADC_SampleTime_13Cycles5);
/* Enable ADC1 external trigger conversion */
ADC_ExternalTrigConvCmd( ADC1, ENABLE );
ADC_Cmd(ADC1, ENABLE);
/* Provedeme kalibraci prevodniku */
/* Enable ADC1 reset calibration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
}

void barGraph(uint16_t Hodnota)
{
if ((Hodnota > 100) && (Hodnota <= 200)) printLCD("X");
if ((Hodnota > 200) && (Hodnota <= 300)) printLCD("XX");
if ((Hodnota > 300) && (Hodnota <= 400)) printLCD("XXX");
if ((Hodnota > 400) && (Hodnota <= 500)) printLCD("XXXX");
if ((Hodnota > 500) && (Hodnota <= 600)) printLCD("XXXXX");
if ((Hodnota > 600) && (Hodnota <= 700)) printLCD("XXXXXX");
if ((Hodnota > 700) && (Hodnota <= 800)) printLCD("XXXXXXX");
if ((Hodnota > 800) && (Hodnota <= 900)) printLCD("XXXXXXXX");
if ((Hodnota > 900) && (Hodnota <= 1000)) printLCD("XXXXXXXXX");
if ((Hodnota > 1000) && (Hodnota <= 1100)) printLCD("XXXXXXXXXX");
if ((Hodnota > 1100) && (Hodnota <= 1200)) printLCD("XXXXXXXXXXX");
if ((Hodnota > 1200) && (Hodnota <= 1300)) printLCD("XXXXXXXXXXXX");
if ((Hodnota > 1300) && (Hodnota <= 1400)) printLCD("XXXXXXXXXXXXX");
if (Hodnota > 1400) printLCD("XXXXXXXXXXXXXX");
};

uint16_t ADC1_Read(void)
{
// Start the conversion

```

```

ADC_SoftwareStartConvCmd(ADC1, ENABLE);
// Wait until conversion completion
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
// Get the conversion value
return ADC_GetConversionValue(ADC1);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void itoa(uint16_t n, int8_t s[])
{
    int i, sign;
    if ((sign = n) < 0) /* record sign */
        n = -n; /* make n positive */
    i = 0;
    do { /* generate digits in reverse order */
        s[i++] = n % 10 + '0'; /* get next digit */
    } while ((n /= 10) > 0); /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

void reverse(int8_t s[])
{
    int i, j, k;
    char c;
    k = strlen(s)-1;
    for (i = 0, j = k; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

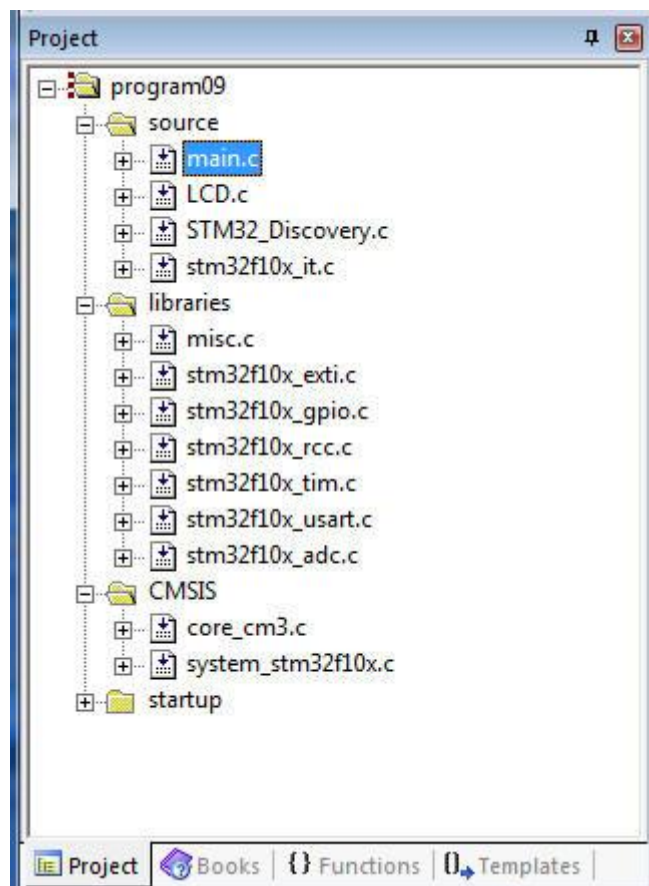
void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}

```



2.7 ADC s výstupem na LCD jako BarGraph – program09



Souboru **main.c**

```

/* ADC s STM32 VL Discovery
** main.c
** vystup na LCD jako BarGraph
** Vd pro SPSE Jecna ARM Courses
*/

#include <stdint.h>
#include <string.h>
#include "stm32f10x.h"
#include "STM32_Discovery.h"
#include "LCD.h"
/* Private typedef -----
---*/
GPIO_InitTypeDef GPIO_InitStructure;

uint8_t LCD_text[] = "Mereni SPSE";
uint16_t Hodnota=0;
uint16_t HodnotaOld=1;
/* Private function prototypes -----
---*/
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu

```

```

void ADC_Inicializace(void);
void Delay(__IO uint32_t nTick);
uint16_t ADC1_Read(void);
void itoa(uint16_t n, int8_t s[]);
void reverse(int8_t s[]);
void barGraph(uint16_t Hodnota);

int main(void)
{
    LCD_Inicialization();
    STM32_Discovery_LEDOn(LED4);          // LED4 - modra
    GPIO_Inicializace();
    LCD_Clear();
    ADC_Inicializace();
    printLCD(LCD_text);
    Delay(0xAAAAA);
    STM32_Discovery_LEDOff(LED4); // LED4 - modra

    while (1)
    {
        Delay(0xAAAAA);
        Hodnota = ADC1_Read()/3 ;
        if (HodnotaOld!= Hodnota)
        {
            LCD_Clear();
            barGraph(Hodnota);
            HodnotaOld = Hodnota ;      //aby se porad neprekreslovalo
        }
    }
}

void GPIO_Inicializace(void)
{
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
}

void ADC_Inicializace(void)
{
    ADC_InitTypeDef  ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    /* ADCCLK = PCLK2/1 = 24/2 = 12MHz*/
    RCC_ADCCLKConfig(RCC_PCLK2_Div2);
    /* povolime hodiny do portu */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;      // tady pripojime merene
    napeti
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    /* povolime hodiny do A/D prevodniku */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
}

```



```

/* ADC1 Configuration -----
---*/
ADC_InitStructure.ADC_Mode           = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode   = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign      = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel   = 1;
ADC_Init( ADC1, &ADC_InitStructure );

/* ADC1 Regular Channel 14 */
ADC_RegularChannelConfig( ADC1, ADC_Channel_14, 1,
ADC_SampleTime_239Cycles5);
/* Enable ADC1 external trigger conversion */
ADC_ExternalTrigConvCmd( ADC1, ENABLE );

ADC_Cmd(ADC1, ENABLE);
/* Provedeme kalibraci prevodniku */
/* Enable ADC1 reset calibration register */
ADC_ResetCalibration(ADC1);
/* Check the end of ADC1 reset calibration register */
while(ADC_GetResetCalibrationStatus(ADC1));
/* Start ADC1 calibration */
ADC_StartCalibration(ADC1);
/* Check the end of ADC1 calibration */
while(ADC_GetCalibrationStatus(ADC1));
}

void barGraph(uint16_t Hodnota)
{
if ((Hodnota > 100)&&(Hodnota<=200)) printLCD("X");
if ((Hodnota > 200)&&(Hodnota<=300)) printLCD("XX");
if ((Hodnota > 300)&&(Hodnota<=400)) printLCD("XXX");
if ((Hodnota > 400)&&(Hodnota<=500)) printLCD("XXXX");
if ((Hodnota > 500)&&(Hodnota<=600)) printLCD("XXXXX");
if ((Hodnota > 600)&&(Hodnota<=700)) printLCD("XXXXXX");
if ((Hodnota > 700)&&(Hodnota<=800)) printLCD("XXXXXXX");
if ((Hodnota > 800)&&(Hodnota<=900)) printLCD("XXXXXXXX");
if ((Hodnota > 900)&&(Hodnota<=1000)) printLCD("XXXXXXXXX");
if ((Hodnota > 1000)&&(Hodnota<=1100)) printLCD("XXXXXXXXXX");
if ((Hodnota > 1100)&&(Hodnota<=1200)) printLCD("XXXXXXXXXXX");
if ((Hodnota > 1200)&&(Hodnota<=1300)) printLCD("XXXXXXXXXXXX");
if ((Hodnota > 1300)&&(Hodnota<=1400)) printLCD("XXXXXXXXXXXXX");
if (Hodnota >1400) printLCD("XXXXXXXXXXXXXX");
};

uint16_t ADC1_Read(void)
{
// Start the conversion
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
// Wait until conversion completion
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
// Get the conversion value

```

```

    return ADC_GetConversionValue(ADC1);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void itoa(uint16_t n, int8_t s[])
{
    int i, sign;
    if ((sign = n) < 0) /* record sign */
        n = -n;      /* make n positive */
    i = 0;
    do { /* generate digits in reverse order */
        s[i++] = n % 10 + '0'; /* get next digit */
    } while ((n /= 10) > 0); /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

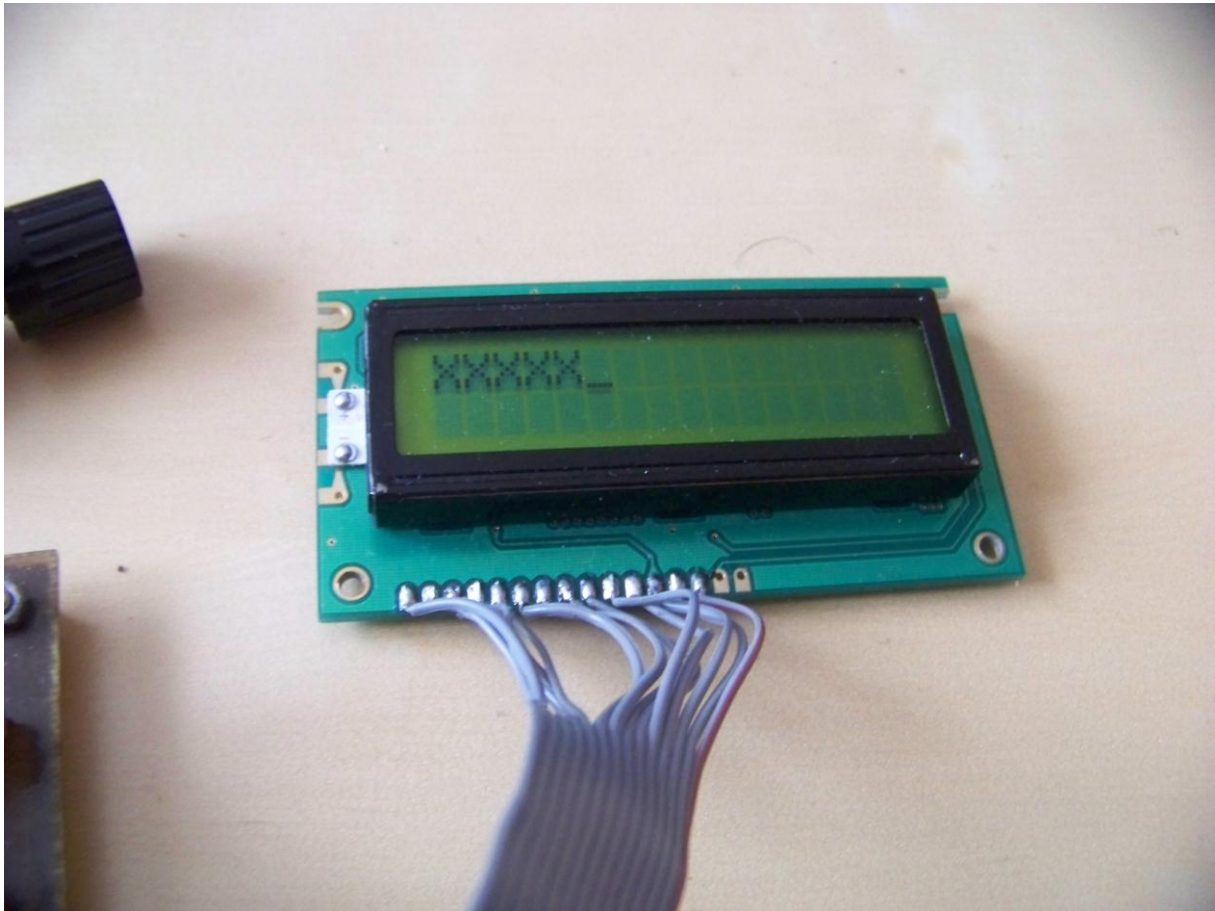
void reverse(int8_t s[])
{
    int i, j;
    char c;

    for (i = 0, j = strlen(s)-1; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

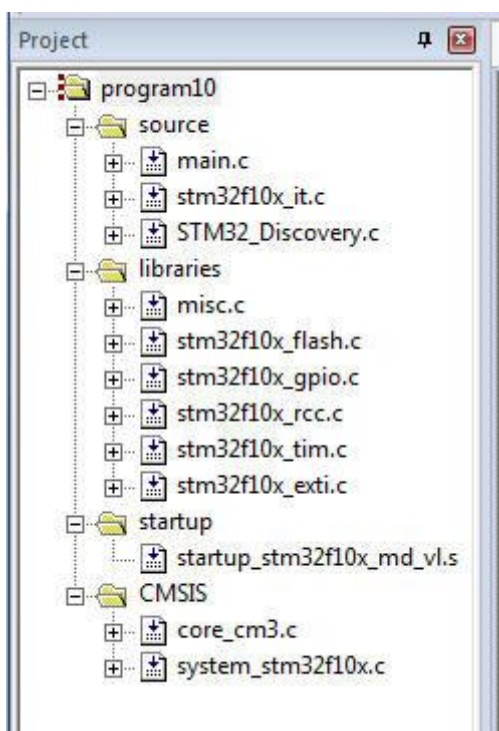
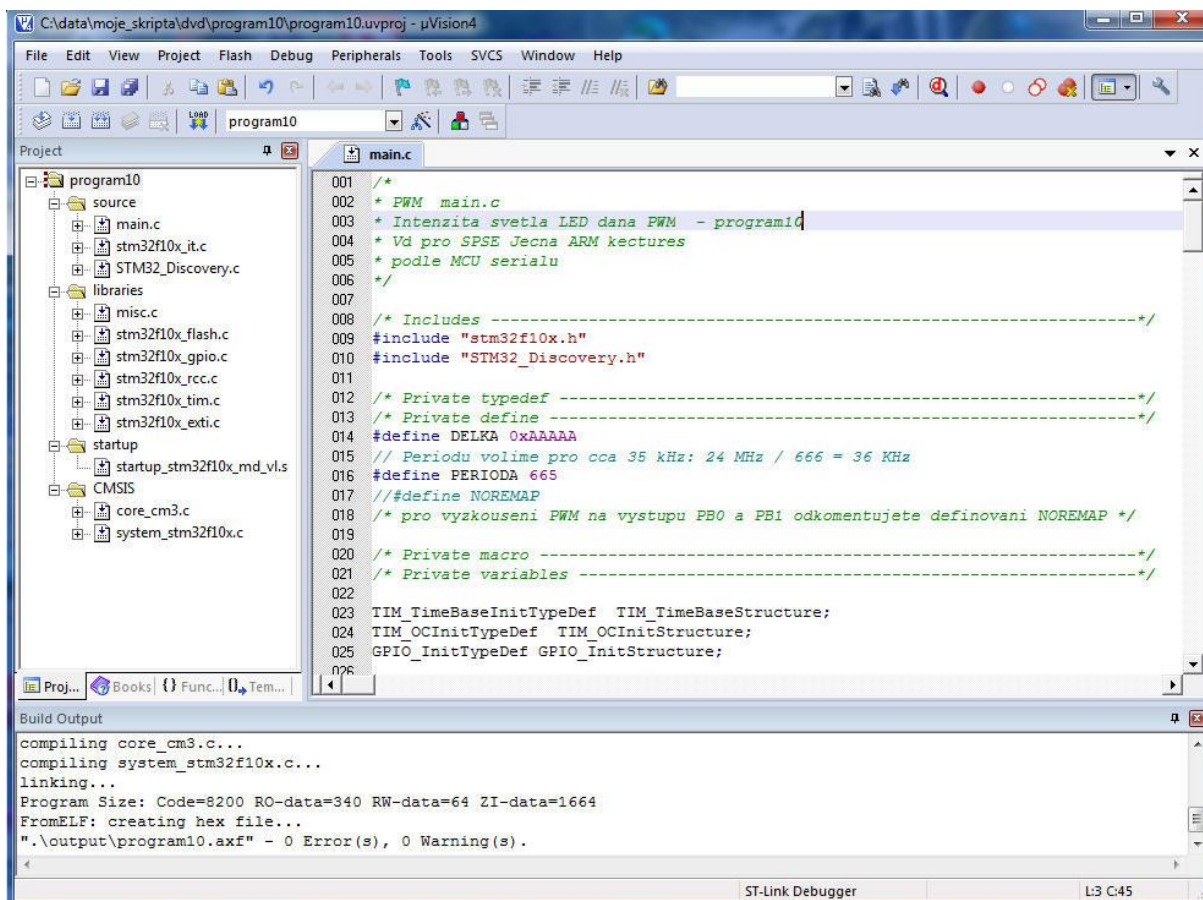
void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}

```



2.8 PWM (program10)



```

/*
 * PWM main.c
 * Intenzita svetla LED dana PWM      - program10
 * Vd pro SPSE Jecna ARM lectures
 * podle MCU serialu
 */

/* Includes -----
---*/
#include "stm32f10x.h"
#include "STM32_Discovery.h"

/* Private typedef -----
---*/
/* Private define -----
---*/
#define DELKA 0xAAAAA
// Periodu volime pro cca 35 kHz: 24 MHz / 666 = 36 KHz
#define PERIODA 665
// #define NOREMAP
/* pro vyzkouseni PWM na vystupu PB0 a PB1 odkomentujete definovani NOREMAP
*/

/* Private macro -----
---*/
/* Private variables -----
---*/

TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
TIM_OCInitTypeDef  TIM_OCInitStructure;
GPIO_InitTypeDef GPIO_InitStructure;

int ZmenaIntenzity = 1;
uint16_t Intenzita = 0;
uint16_t PrescalerValue = 0;

/* Private function prototypes -----
---*/

void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu
void TIM3_Inicializace(void); // nastaveni Timeru3
void Delay(__IO uint32_t nTick); // cekaci smycka
void My_Led3_Off(void); // zhasnuti zelene LED
void My_Led3_On(void); // rozsviceni zelene LED

/* Private functions -----
---*/

/**
 * @brief Main program.
 * @param None
 * @retval : None
 */

```

```
int main(void)
{
    /* inicializace timeru3 - povolime mu hodinovy takt */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3 , ENABLE);
    /* GPIOA a GPIOB spuštění hodin */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
        RCC_APB2Periph_GPIOC |RCC_APB2Periph_AFIO, ENABLE);

    GPIO_Inicializace();

#ifdef NOREMAP
    /* pro nepremapovany vystup TIM3 chan 3 a 4 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
#else
    /* pro premapovany vystup TIM3 chan 3 a 4 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    /* musime remapovat funkci PC8,PC9 na timer3 chanel 3 a 4 output */
    GPIO_PinRemapConfig(GPIO_FullRemap_TIM3, ENABLE);
#endif

    /* musime nastaveit Timer3 */
    TIM3_Inicializace();

    while (1)
    {
        TIM_OCInitStructure.TIM_Pulse = Intenzita;
        TIM_OC3Init(TIM3, &TIM_OCInitStructure);
        TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);
        TIM_ARRPreloadConfig(TIM3, ENABLE);

        if ((ZmenaIntenzity>0 && Intenzita<PERIODA) || (ZmenaIntenzity<0 &&
            Intenzita>1) )
        {
            Intenzita = Intenzita + ZmenaIntenzity;
        }
        else
        {
            Intenzita = Intenzita - ZmenaIntenzity;
        }

#ifdef NOREMAP
        STM32_Discovery_LEDOn(LED4); // zapnem LED4 - modra
#endif
        ZmenaIntenzity = - ZmenaIntenzity; // obratime smer zmeny svetla
        Delay(0xFFFFF); // chvili pockame -
    }
    osetreni kmitani kontaktu
#ifdef NOREMAP

```

```

    STM32_Discovery_LEDOff(LED4); // zapnem LED4 - modra
    /* tedy pri zmene typu zhasinani/rozsvectovani klikneme modrou led */
#endif
}

    Delay(0xFFFF); Delay(0xFFFF);
    Delay(0xFFFF); // kratka pauza aby
zmena intenzity nebyla moc rychla

    if(0 != STM32_Discovery_PBGetState(BUTTON_USER))
    {
        // tj. když je stisknuto tlacitko
#ifdef NOREMAP
        STM32_Discovery_LEDOn(LED3); // zapnem LED3 - zelena
#endif
        // nastavime 100% PWM pro CH4, tj. remapovana LED3 zelena bude svitit
        naplno
        My_Led3_On();
        Delay(0xFFFFF); // chvili pockame -
osetreni kmitani kontaktu
#ifdef NOREMAP
        STM32_Discovery_LEDOff(LED3); // vypnem LED3 - zelena
#endif
        // nastavime 0% PWM pro CH4, tj. LED3 zelena nebude svitit
        My_Led3_Off();
    }
}

/**
 * @brief Zhasnuti zelene LED.
 * @param None
 * @retval : None
 */
void My_Led3_Off(void)
{
    /* PWM1 Mode configuration: Channel4 */
    TIM_OCInitStructure.TIM_Pulse = 0;
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);
    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Enable);
    TIM_ARRPreloadConfig(TIM3, ENABLE);
}

/**
 * @brief Rozsviceni zelene LED.
 * @param None
 * @retval : None
 */
void My_Led3_On(void)
{
    /* PWM1 Mode configuration: Channel4 */
    TIM_OCInitStructure.TIM_Pulse = PERIODA;
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);
    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Enable);
    TIM_ARRPreloadConfig(TIM3, ENABLE);
}

```

```

}

/**
 * @brief Configure the GPIO Pins.
 * @param None
 * @retval : None
 */
void GPIO_Inicializace(void)
{
    STM32_Discovery_LEDInit(LED3);
    STM32_Discovery_LEDInit(LED4);
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    STM32_Discovery_LEDOff(LED4);
}

/**
 * @brief Configure the Tim3.
 * @param None
 * @retval : None
 */
void TIM3_Inicializace(void)
{
    /* Compute the prescaler value */
    PrescalerValue = (uint16_t) (SystemCoreClock / 24000000) - 1;
    /* Uvedených 24000000 platí pro nezmenený System Clock!! */

    /* Time base configuration */
    TIM_TimeBaseStructure.TIM_Period = PERIODA;
    TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

    /* PWM1 Mode configuration: Channel3 */
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 100;
    TIM_OC3Init(TIM3, &TIM_OCInitStructure);

    TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);

    /* PWM1 Mode configuration: Channel4 */
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 1;
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);

    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Enable);

    TIM_ARRPreloadConfig(TIM3, ENABLE);

    /* TIM3 enable counter */

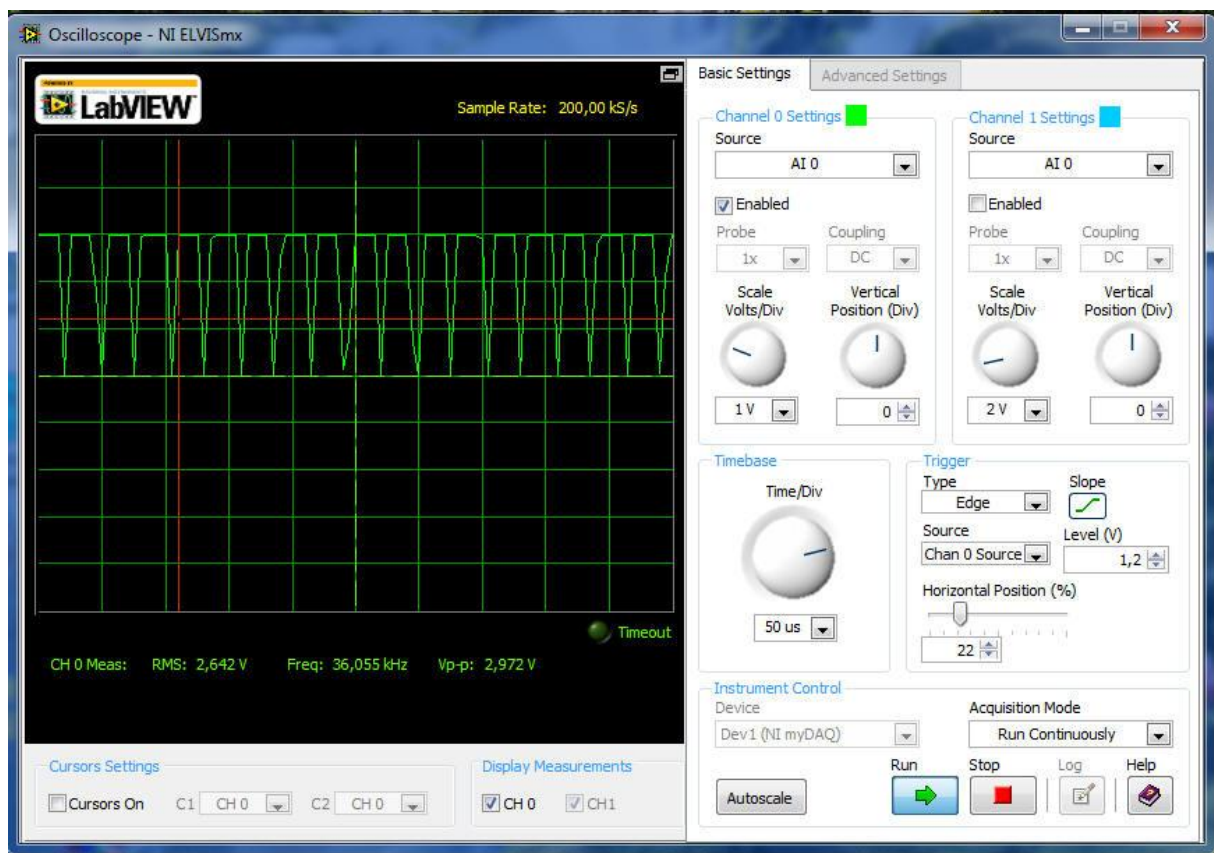
```



```
TIM_Cmd(TIM3, ENABLE);

}

void Delay( IO uint32 t nTick)
{
    for(; nTick != 0; nTick--);
}
```



Všimněme si, že platí naše poznámka na začátku zdrojového kódu

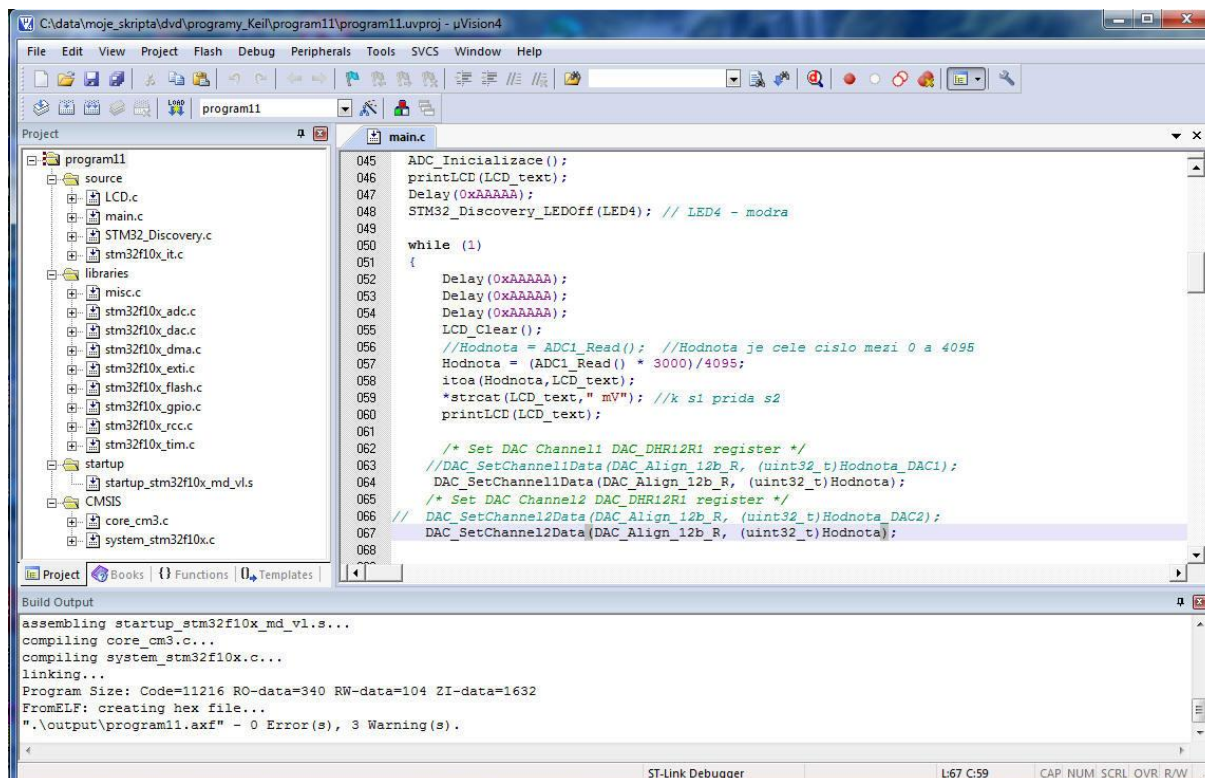
```
// Periodu volime pro cca 35 kHz: 24 MHz / 666 = 36 KHz
#define PERIODA 665
```

2.9 DAC (program11)

Ve většině programů ukazujících převod DAC se obvykle generuje nějaký signál mající pilový, sinusový apod. průběh. Programujeme v nich čítače, plníme paměť hodnotami funkce sinus apod. a mj. i převod DAC. Takže kód pro převod je ztracen v mnoha dalším kódu.

Aby vynikla jednoduchost naprogramování DAC, budeme postupovat trochu jinak. O převod DAC doplníme náš **program08** ukazující převod ADC, kdy jsme mezi GND a +3V připojili potenciometr 22k a jeho běžec na **PC4**. Po převodu ADC jsme získanou hodnotu po vynásobení konstantou zobrazili na LCD, který tak ukazoval napětí na běžci potenciometru v mV.

Nyní tuto digitální hodnotu ještě převody DAC převedeme na analogovou. Použijeme přitom dva DAC kanály a budeme měřit napětí na výstupech převodníků tj. **PA4** a **PA5**.



```

045 ADC_Inicializace();
046 printLCD(LCD_text);
047 Delay(0xAAAAA);
048 STM32_Discovery_LEDOff(LED4); // LED4 - modra
049
050 while (1)
051 {
052     Delay(0xAAAAA);
053     Delay(0xAAAAA);
054     Delay(0xAAAAA);
055     LCD_Clear();
056     //Hodnota = ADC1_Read(); //Hodnota je cele cislo mezi 0 a 4095
057     Hodnota = (ADC1_Read() * 3000)/4095;
058     itoa(Hodnota,LCD_text);
059     *strcat(LCD_text," mV"); //k s1 prida s2
060     printLCD(LCD_text);
061
062     /* Set DAC Channel1 DAC_DHR12R1 register */
063     //DAC_SetChannel1Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC1);
064     DAC_SetChannel1Data(DAC_Align_12b_R, (uint32_t)Hodnota);
065     /* Set DAC Channel2 DAC_DHR12R1 register */
066     // DAC_SetChannel2Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC2);
067     DAC_SetChannel2Data(DAC_Align_12b_R, (uint32_t)Hodnota);
068

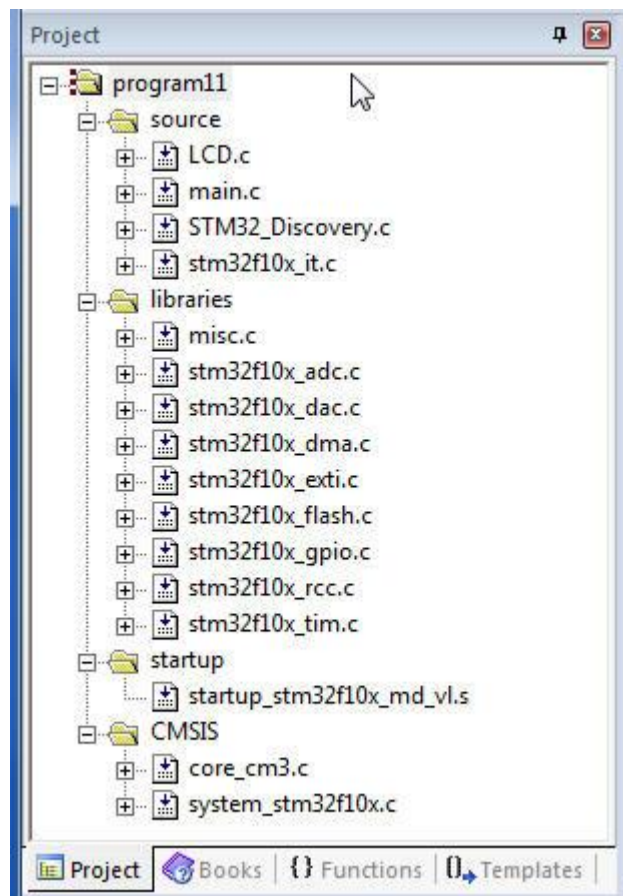
```

Build Output

```

assembling startup_stm32f10x_md_v1.s...
compiling core_cm3.c...
compiling system_stm32f10x.c...
linking...
Program Size: Code=11216 RO-data=340 RW-data=104 ZI-data=1632
FromELF: creating hex file...
".\output\program11.axf" - 0 Error(s), 3 Warning(s).

```



Soubor main.c

```

/* ADC a DAC v STM32 VL Discovery
** main.c
** program11
** Vd for SPSE Jecna course
*/

#include <stdint.h>
#include "string.h"
#include "stm32f10x.h"
#include "STM32_Discovery.h"
#include "LCD.h"

GPIO_InitTypeDef GPIO_InitStructure;

uint8_t LCD_text[] = "Testovani ADC a DAC Mereni SPSE";
uint16_t Hodnota=0;
uint16_t HodnotaOld=1;

```

```

/* Private function prototypes -----*/
---*/
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu
void ADC_Inicializace(void);
void Delay(__IO uint32_t nTick);
uint16_t ADC1_Read(void);
void itoa(uint16_t n, int8_t s[]);
void reverse(int8_t s[]);
void KopirujLCDtext(int8_t s[]);
void DAC_Inicializace(void);

int main(void)
{
    //GPIO_LCD_Inicializace();
    LCD_Inicialization();
    STM32_Discovery_LEDOn(LED4); // LED4 - modra
    //Delay(0xAAAA);
    GPIO_Inicializace();
    DAC_Inicializace();

    //LCD_Init();
    LCD_Clear();
    ADC_Inicializace();
    printLCD(LCD_text);
    Delay(0xAAAAA);
    STM32_Discovery_LEDOff(LED4); // LED4 - modra

    while (1)
    {
        Delay(0xAAAAA);
        Delay(0xAAAAA);
        Delay(0xAAAAA);
        LCD_Clear();
        Hodnota = ADC1_Read(); //Hodnota je cele cislo mezi 0 a 4095
        /* Set DAC Channel1 DAC_DHR12R1 register */
        //DAC_SetChannel1Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC1);
        DAC_SetChannel1Data(DAC_Align_12b_R, (uint32_t)Hodnota);
        /* Set DAC Channel2 DAC_DHR12R1 register */
        //DAC_SetChannel2Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC2);
        DAC_SetChannel2Data(DAC_Align_12b_R, (uint32_t)Hodnota);

        Hodnota = (Hodnota * 3000)/4095;
        itoa(Hodnota, LCD_text);
        *strcat(LCD_text, " mV"); //k s1 prida s2
        printLCD(LCD_text);
    }
}

void GPIO_Inicializace(void)
{
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
}

```

```

void ADC_Inicializace(void)
{
    ADC_InitTypeDef  ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    /* ADCCLK = PCLK2/1 = 24/2 = 12MHz*/
    RCC_ADCCLKConfig(RCC_PCLK2_Div2);
    /* povolime hodiny do portu */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;    // tady pripojime merene
napeti
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    /* povolime hodiny do A/D prevodniku */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    //ADC_DeInit(ADC1);
    /* ADC1 Configuration -----
---*/
    ADC_InitStructure.ADC_Mode                = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode        = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv    = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign           = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel        = 1;
    ADC_Init( ADC1, &ADC_InitStructure );
    /* ADC1 Regular Channel 14 */
    ADC_RegularChannelConfig( ADC1, ADC_Channel_14, 1,
ADC_SampleTime_239Cycles5);
    //ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1,
ADC_SampleTime_13Cycles5);
    /* Enable ADC1 external trigger conversion */
    ADC_ExternalTrigConvCmd( ADC1, ENABLE );
    ADC_Cmd(ADC1, ENABLE);
    /* Provedeme kalibraci prevodniku */
    /* Enable ADC1 reset calibration register */
    ADC_ResetCalibration(ADC1);
    /* Check the end of ADC1 reset calibration register */
    while(ADC_GetResetCalibrationStatus(ADC1));
    /* Start ADC1 calibration */
    ADC_StartCalibration(ADC1);
    /* Check the end of ADC1 calibration */
    while(ADC_GetCalibrationStatus(ADC1));
}

void DAC_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    DAC_InitTypeDef   DAC_InitStructure;
    /* GPIOA Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    /* DAC Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
}

```

```

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* DAC channell Configuration */
DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
    // zadny trigger, sami si budeme poustet signal na vystup
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
    // viz text k tomuto dilu - nepozadujeme sutomatickou
generaci!
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    // kdyz nepovol9te output buffer a zatizite vystup, tak
napeti bude klesat!!
DAC_Init(DAC_Channel_1, &DAC_InitStructure);
DAC_Init(DAC_Channel_2, &DAC_InitStructure);

/* Enable DAC Channels */
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_Cmd(DAC_Channel_2, ENABLE);
}

uint16_t ADC1_Read(void)
{
    // Start the conversion
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    // Wait until conversion completion
    while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
    // Get the conversion value
    return ADC_GetConversionValue(ADC1);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void itoa(uint16_t n, int8_t s[])
{
    int i, sign;
    if ((sign = n) < 0) /* record sign */
        n = -n; /* make n positive */
    i = 0;
    do { /* generate digits in reverse order */
        s[i++] = n % 10 + '0'; /* get next digit */
    } while ((n /= 10) > 0); /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

void reverse(int8_t s[])
{
    int i, j, k;

```

```

char c;
    k = strlen(s)-1;
    for (i = 0, j = k; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

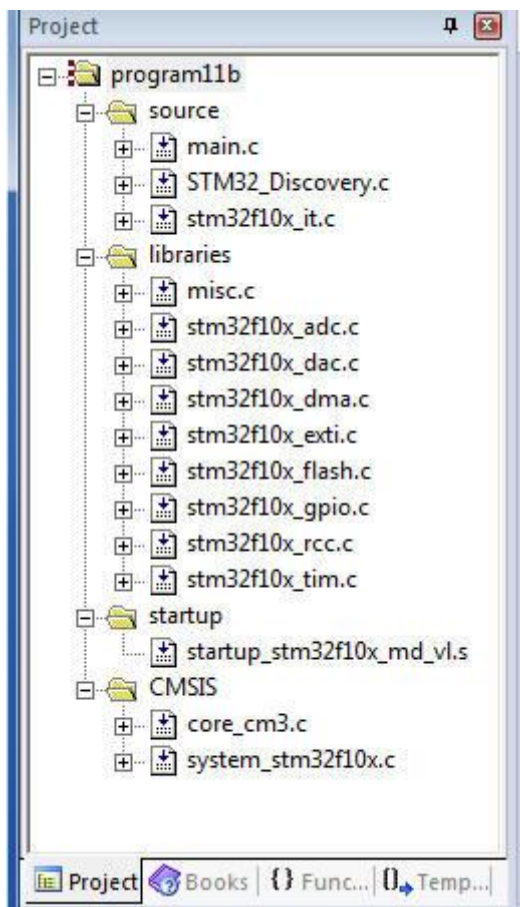
void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}

```

2.10.1 DAC (program11B) – generátor trojúhelníkového průběhu

Úpravou programu **program11** dostaneme **program11b** v němž číselné hodnoty, které budeme převádět na analogové získáme výpočtem z hodnoty sw čítače **Counter**, která se mění v rozsahu 0 až 100 tak, že při každém průchodu hlavní nekonečnou smyčkou se zvětší o jedničku, popř. se po dosažení hodnoty 100 vynuluje. Vypočítané hodnoty pro D/A konverzi jsou v rozsahu 0 až 4095.



A zdrojový kód main.c

```

/* DAC v STM32 VL Discovery - výstupní generované signaly na PA4 a PA5
** velikost ciselne hodnoty, kterou prevadime na analogovou hodnotu
** pocitame z hodnoty citace
** main.c
** program11b - upraveno podle MCU serialu - díl 19
** Vd for SPSE Jecna course
*/

#include <stddef.h>
#include "string.h"
#include "stm32f10x.h"
#include "STM32_Discovery.h"

TIM_TimeBaseInitTypeDef TIM_TimeBaseStructureMili =
{36000,TIM_CounterMode_Up,0,0,0}; //musi byt !!!!!
GPIO_InitTypeDef GPIO_InitStructure;

uint16_t Hodnota=0;
__IO uint8_t c;
__IO uint16_t Perioda = 100; // budeme mit celkem 100 dilku na periodu

```



```

__IO uint16_t Counter = 0;          // interni citac generatoru probehu
__IO uint16_t Hodnota_DAC1 = 0;
__IO uint16_t Hodnota_DAC2 = 0;
/* Private function prototypes -----*/
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu

void Delay( __IO uint32_t nTick);
//void delay_ms(uint16_t time);
void DAC_Inicializace(void);
void Generuj_DAC(void);
void BliknutiLEDZelena(void);

int main(void)
{
    //delay_ms(2);

    GPIO_Inicializace();
    STM32_Discovery_LEDOn(LED4);    // LED4 - modra
    DAC_Inicializace();
    Delay(0xAAAAA);
    STM32_Discovery_LEDOff(LED4); // LED4 - modra

    while (1)
    {
        Generuj_DAC();
        BliknutiLEDZelena(); //dva krat Delay(0x1FFFF);
    }
}

void GPIO_Inicializace(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE); //musi byt !!!!!
    STM32_Discovery_LEDInit(LED3);
    STM32_Discovery_LEDInit(LED4);
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    STM32_Discovery_LEDOff(LED4);
}

void BliknutiLEDZelena(void)
{
    STM32_Discovery_LEDOn(LED3);
    Delay(0x1FFFF);
    STM32_Discovery_LEDOff(LED3);
    Delay(0x1FFFF);
}

void DAC_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    DAC_InitTypeDef DAC_InitStructure;
    /* GPIOA Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    /* DAC Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
}

```

```

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* DAC channel1 Configuration */
DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
// zadny trigger, sami si budeme poustet signal na vystup
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
// viz text v MCU serialu - nepozadujeme automatickou generaci!
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
//kdyz nepovolite output buffer a zatizite vystup, tak napeti bude klesat!!
DAC_Init(DAC_Channel_1, &DAC_InitStructure);
DAC_Init(DAC_Channel_2, &DAC_InitStructure);

/* Enable DAC Channels */
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_Cmd(DAC_Channel_2, ENABLE);
}

void Generuj_DAC(void)
{
    STM32_Discovery_LEDOff(LED4); // Vypneme LED4 - modrou
    if (Counter <= (Perioda / 2))
    {
        Hodnota_DAC1 = ((uint32_t)Counter)*2 * 0xFFF / Perioda;
//vysledek je v rozmezi 0 az 4095
        Hodnota_DAC2 = 0xFFF - (((uint32_t)Counter)* 2 * 0xFFF /
Perioda);
    }
    else
    {
        Hodnota_DAC1 = 0xFFF - (((uint32_t)Counter - (Perioda / 2))*2 *
0xFFF / Perioda); //vysledek je v rozmezi 0 az 4095
        Hodnota_DAC2 = (((uint32_t)Counter) - (Perioda / 2))*2 * 0xFFF /
Perioda;
    }

    /* Set DAC Channel1 DAC_DHR12R1 register */
    DAC_SetChannel1Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC1);

    /* Set DAC Channel2 DAC_DHR12R1 register */
    DAC_SetChannel2Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC2);

    Counter++;
    if (Counter >= Perioda)
    {
        Counter = 0;
        STM32_Discovery_LEDOn(LED4);
// zapneme LED4 - modra - signal zacatku periody
    }
}

void Delay(__IO uint32_t nTick)

```

```

{
    for(; nTick != 0; nTick--);
}

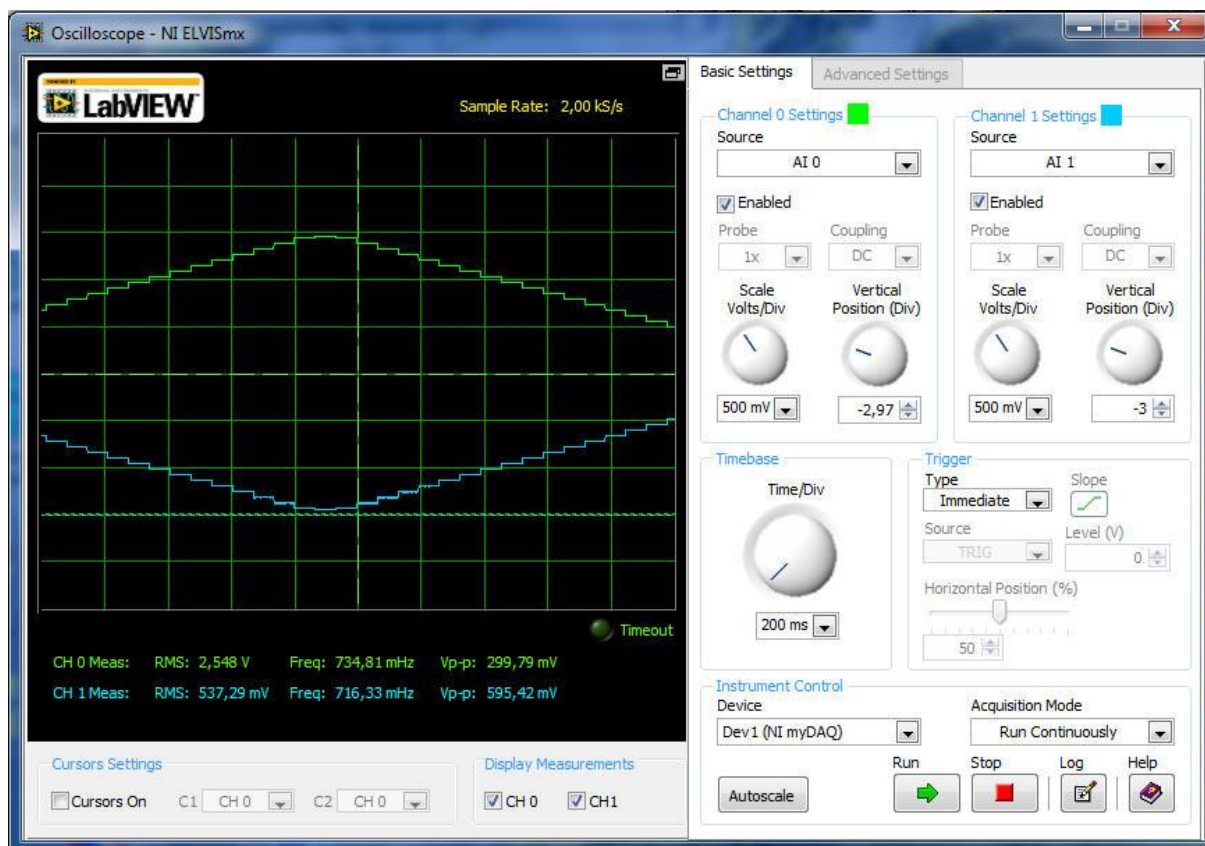
/*
void delay_ms(uint16_t time)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
    TIM_TimeBaseStructureMili.TIM_Period = ((time+1) * 2)-1;
    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructureMili);
    TIM_SelectOnePulseMode(TIM4, TIM_OPMode_Single);
    TIM_SetCounter(TIM4,2);
    TIM_Cmd(TIM4, ENABLE);
    while (TIM_GetCounter(TIM4)){};
    TIM_Cmd(TIM4, DISABLE);
}
*/

void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}

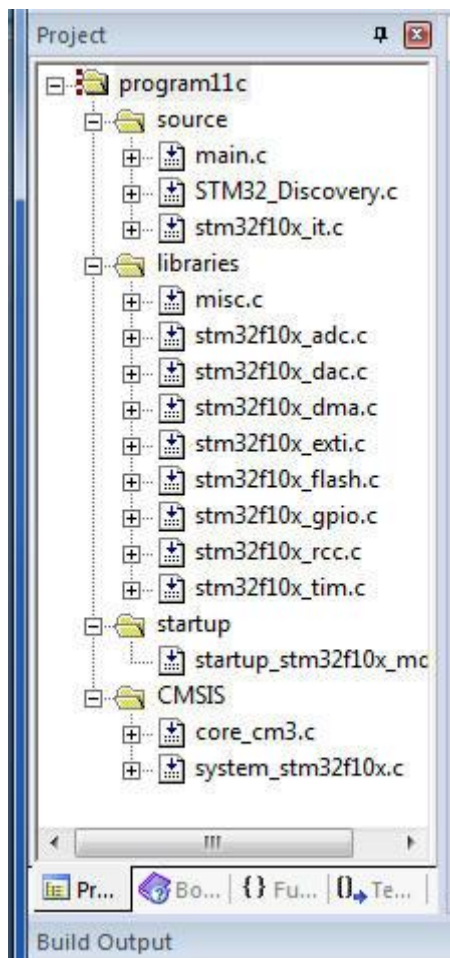
```

Průběhy výstupních napětí na **PA4** a **PA5**



2.10.2 DAC (program11c) – generátor trojúhelníkového průběhu 5Hz s využitím přerušení

Program byl upraven podle 20.dílu MCU seriálu.



Dále main.c

```

/* DAC v STM32 VL Discovery - výstupní generované signaly na PA4 a PA5
** velikost ciselne hodnoty, kterou prevadime na analogovou hodnotu
** vyuzivane preruseni, jeho obsluhu jsme dali do stm32f10x.c !!!!!!!
** main.c
** program11c - upraveno podle MCU serialu - díl 20
** Vd for SPSE Jecna course
*/

#include <stdint.h>
#include "string.h"
#include "stm32f10x.h"
#include "STM32_Discovery.h"

GPIO_InitTypeDef GPIO_InitStructure;

__IO uint8_t c;

/* Private function prototypes -----
---*/

```

```

void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu

void Delay(__IO uint32_t nTick);
void DAC_Inicializace(void);
void Generuj_DAC(void);
void BliknutiLEDZelena(void);

void NVIC_Inicializace(void);
void TIM3_Inicializace(void);
void TIM3_IRQHandler(void);

int main(void)
{
    GPIO_Inicializace();
    DAC_Inicializace();
    NVIC_Inicializace();
    TIM3_Inicializace();

    STM32_Discovery_LEDOn(LED4); // LED4 - modra
    Delay(0xAAAAA);
    STM32_Discovery_LEDOff(LED4); // LED4 - modra

    while (1)
    {
        BliknutiLEDZelena(); //dva krat Delay(0x1FFFF);
    }
}

void GPIO_Inicializace(void)
{
    //RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE); //musi byt !!!!!
    STM32_Discovery_LEDInit(LED3);
    STM32_Discovery_LEDInit(LED4);
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    STM32_Discovery_LEDOff(LED4);
}

void BliknutiLEDZelena(void)
{
    STM32_Discovery_LEDOn(LED3);
    Delay(0x1FFFF);
    STM32_Discovery_LEDOff(LED3);
    Delay(0x1FFFF);
}

void DAC_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    DAC_InitTypeDef DAC_InitStructure;
    /* GPIOA Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    /* DAC Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
}

```

```

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* DAC channel1 Configuration */
DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
    // zadny trigger, sami si budeme poustet signal na vystup
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
    // viz text k tomuto dilu - nepozadujeme sutomatickou
generaci!
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    // kdyz nepovol9te output buffer a zatizite vystup, tak
napeti bude klesat!!
DAC_Init(DAC_Channel_1, &DAC_InitStructure);
DAC_Init(DAC_Channel_2, &DAC_InitStructure);

/* Enable DAC Channels */
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_Cmd(DAC_Channel_2, ENABLE);
}

void NVIC_Inicializace(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    /* Configure the NVIC Preemption Priority Bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    /* Clear the SC_EXTI IRQ Pendingbit Bit */
    NVIC_ClearPendingIRQ(TIM3_IRQn);

    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

void TIM3_Inicializace(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3 , ENABLE);           // pustime
hodiny
    /* -----
    -----
    TIM3 Configuration: Output Compare Toggle Mode:
    TIM3CLK = 24 MHz, Prescaler = 24000, TIM3 counter clock = 1kHz
    -----
    ---*/
    /* Time base configuration */
    TIM_TimeBaseStructure.TIM_Period = 1;           //nas pilovy prubeh by mel mit
f=25Hz
    TIM_TimeBaseStructure.TIM_Prescaler = 24000;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Down;
    TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;

```

```
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
/* TIM IT enable */
TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
/* TIM enable counter */
TIM_Cmd(TIM3, ENABLE);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}
```

A ještě **stm32f10x_it.c** který obsahuje obsluhu přerušení provádějící D/A převod.

```
/**
 * @file    Project/STM32F10x_StdPeriph_Template/stm32f10x_it.c
 * @version V3.3.0      doplnene o DA prevod s prerusenim
 */

#include "stm32f10x_it.h"

void NMI_Handler(void)
{
}

void HardFault_Handler(void)
{
    /* Go to infinite loop when Hard Fault exception occurs */
    while (1)
    {
    }
}

void MemManage_Handler(void)
{
    /* Go to infinite loop when Memory Manage exception occurs */
    while (1)
    {
    }
}
```



```

}

void BusFault_Handler(void)
{
    /* Go to infinite loop when Bus Fault exception occurs */
    while (1)
    {
    }
}

void UsageFault_Handler(void)
{
    /* Go to infinite loop when Usage Fault exception occurs */
    while (1)
    {
    }
}

void SVC_Handler(void)
{
}

void DebugMon_Handler(void)
{
}

void PendSV_Handler(void)
{
}

void SysTick_Handler(void)
{
}

/*****
*****/
/*          STM32F10x Peripherals Interrupt Handlers
*/
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for
the */
/*  available peripheral interrupt handler's name please refer to the
startup */
/*  file (startup_stm32f10x_xx.s).
*/
/*****
*****/

void TIM3_IRQHandler(void)
// původní funkce void Generuj_DAC(void)

```

```

{
    static uint16_t Counter = 0;           // interni citac generatoru
    probehu
    __IO uint16_t Hodnota_DAC1 = 0;
    __IO uint16_t Hodnota_DAC2 = 0;

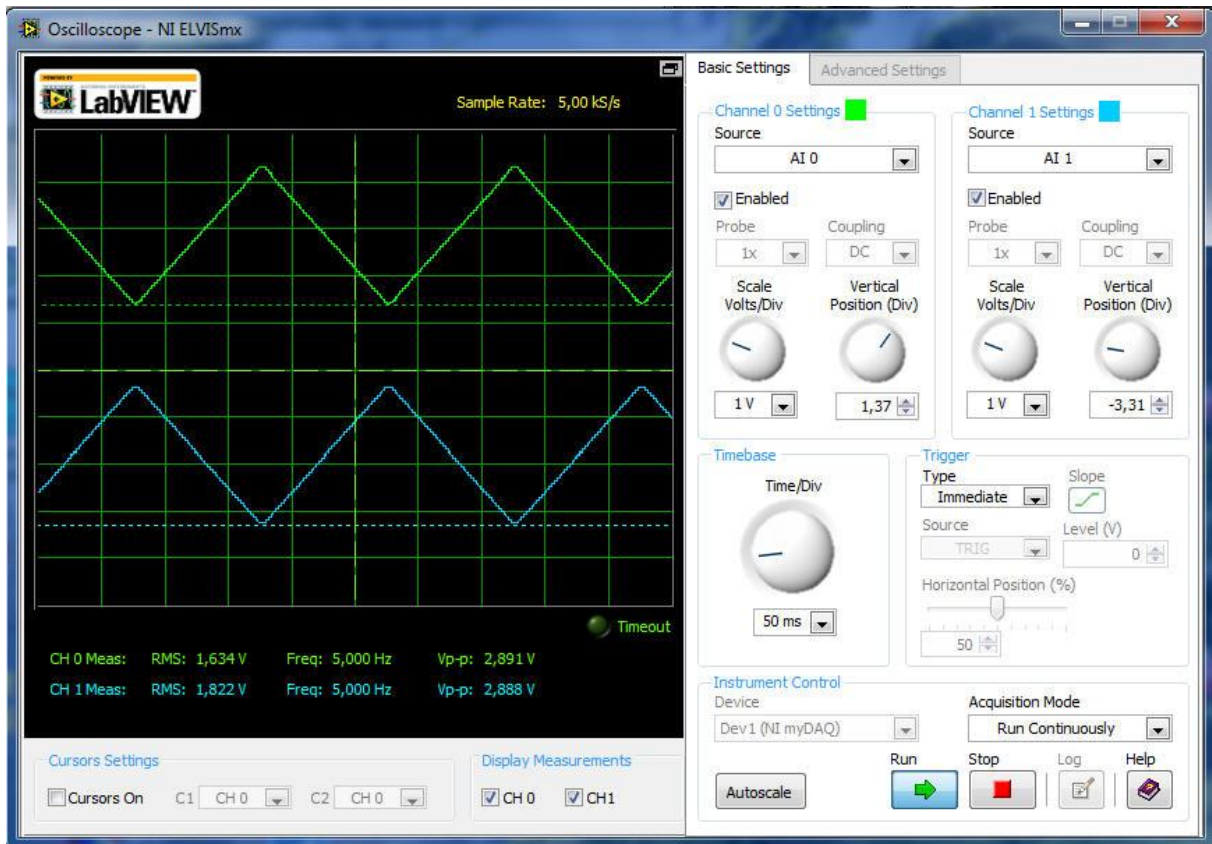
    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
    {
        /* Clear TIM3 update interrupt pending bit*/
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
        // ----- zde zacina puvodni fce Generuj_DAC
        STM32_Discovery_LEDOff(LED4); // Vypneme LED4 - modrou
        if (Counter <= (Perioda / 2))
        {
            Hodnota_DAC1 = ((uint32_t)Counter)*2 * 0xFFFF / Perioda;
            Hodnota_DAC2 = 0xFFFF - (((uint32_t)Counter)* 2 * 0xFFFF /
Perioda);
        }
        else
        {
            Hodnota_DAC1 = 0xFFFF - (((uint32_t)Counter - (Perioda / 2))*2 *
0xFFFF / Perioda);
            Hodnota_DAC2 = (((uint32_t)Counter) - (Perioda / 2))*2 * 0xFFFF /
Perioda;
        }

        /* Set DAC Channel1 DAC_DHR12R1 register */
        DAC_SetChannel1Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC1);

        /* Set DAC Channel2 DAC_DHR12R1 register */
        DAC_SetChannel2Data(DAC_Align_12b_R, (uint32_t)Hodnota_DAC2);

        Counter++;
        if (Counter >= Perioda)
        {
            Counter = 0;
            STM32_Discovery_LEDOn(LED4); // zapneme LED4 - modra - signal
zacatku periody
        }
        // ----- zde konci puvodni fce Generuj_DAC
    }
}

```



2.10.3 DAC (program11d) – generátor trojúhelníkového a sinusového signálu s využitím DMA

Program byl upraven podle 22.dílu MCU seriálu.

```

/* DAC v STM32 VL Discovery - výstupní generované signály na PA4 a PA5
** velikost císle hodnoty, kterou prevadime na analogovou hodnotu
** vyuzivame DMA
** main.c
** program11d - upraveno podle MCU serialu - díl 22
** Vd for SPSE Jecna course
*/

#include <stdint.h>
#include "string.h"
#include "stm32f10x.h"
#include "STM32_Discovery.h"

GPIO_InitTypeDef GPIO_InitStructure;
DMA_InitTypeDef DMA_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
DAC_InitTypeDef DAC_InitStructure;

#define DAC_DHR12RD_Address 0x40007420

```

```

const uint16_t Pila[16] = { 0, 273, 546, 819, 1092, 1365, 1638, 1911,
2184, 2457, 2730, 3003, 3276, 3549, 3822, 4095};

const uint16_t Sinus[16] = {2048, 2831, 3495, 3939, 4095, 3939, 3495, 2831,
2048, 1264, 600, 156, 0, 156, 600, 1264};

uint32_t Dva_kanaly[16]; // do této proměnné nastrkáme data pro DAC

__IO uint8_t Pozice, Stisk = 0;
__IO int16_t Zmena=0;
__IO uint32_t Perioda=100;

/* Private function prototypes -----*/
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu

void Delay(__IO uint32_t nTick);

void BliknutiLEDZelena(void);
void TIM2_Inicializace(void);

int main(void)
{
    GPIO_Inicializace();
    TIM2_Inicializace();

    /* Enable DAC clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

    /* DAC channel1 Configuration */
    DAC_InitStructure.DAC_Trigger = DAC_Trigger_T2_TRGO;
    // tedy bude trigger, využijeme signál od Timeru
    DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
    // nepozadujeme automatickou generaci!
    DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Disable;
    // když nepovolíte output buffer a zatízíte výstup, tak napětí bude
    klesat!!
    DAC_Init(DAC_Channel_1, &DAC_InitStructure);

    /* DAC channel2 Configuration */
    DAC_Init(DAC_Channel_2, &DAC_InitStructure);

    STM32_Discovery_LEDOn(LED4); // LED4 - modra
    Delay(0xAAAAA);
    STM32_Discovery_LEDOff(LED4); // LED4 - modra

    /* naplníme buffer daty */
    for (Pozice = 0; Pozice < 16; Pozice++)
    {
        Dva_kanaly[Pozice] = ( Sinus[Pozice] << 16) + (Pila[Pozice]);
    }
    /* v horním halfwordu je Sinus = PA5, ve spodním je Pila = PA4 */

```

```

/* Enable DMA clock */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

/* DMA1 channel4 configuration */
DMA_DeInit(DMA1_Channel4);
DMA_InitStructure.DMA_PeripheralBaseAddr = DAC_DHR12RD_Address;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&Dva_kanaly;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize = 16;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel4, &DMA_InitStructure);

// uz tam máme data, muzeme pustit prenos
/* Enable DMA1 Channel4 */
DMA_Cmd(DMA1_Channel4, ENABLE);

/* Enable DAC Channels */
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_Cmd(DAC_Channel_2, ENABLE);

/* Enable DMA for DAC Channel2 */
DAC_DMAMCmd(DAC_Channel_2, ENABLE);

/* TIM2 enable counter */
TIM_Cmd(TIM2, ENABLE);

Zmena = -5;

while (1)
{
    //BliknutiLEDZelena(); //dva krat Delay(0x1FFFF);
    if(0 != STM32_Discovery_PBGetState(BUTTON_USER))
    { // je stisknuto tlacitko
        Stisk = 1; // nastavime priznak stisku tlacitka
        if (Perioda <= 150)
        {
            Perioda += Zmena;
        } else
        {
            Perioda += 10*Zmena; // pro nizsi frekvence urychlime
zmenu
        }

        if (Perioda < 1)
        {
            STM32_Discovery_LEDOn(LED4); // rozsvitime modrou kdyz
jsme na konci
            Perioda = 1;
        }
    }
}

```

```

        if (Perioda > 1000)
        {
            STM32_Discovery_LEDOn(LED4); // rozsvitime modrou kdyz
jsme na konci
            Perioda = 1000;
        }
        STM32_Discovery_LEDOn(LED3); // rozsvitime zelenou ze je
stlacene tlacitko
        /* zmenime nastaveni frekvence */
        TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) 1200; //
pro 20Hz-20kHz!!
        TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
        TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Down;
        TIM_TimeBaseStructure.TIM_Period = (uint16_t) Perioda;
        TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
        Delay(0x5FFFF);
    } else
    {
        // neni stisknuto tlacitko
        if (Stisk ==1)
        {
            // prave doslo k uvolneni tlacitka, zhasneme ledky
            Stisk = 0;
            Zmena = - Zmena;
            STM32_Discovery_LEDOff(LED3);
            STM32_Discovery_LEDOff(LED4);
        }
    }
}

void GPIO_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    STM32_Discovery_LEDInit(LED3);
    STM32_Discovery_LEDInit(LED4);
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    STM32_Discovery_LEDOff(LED4);

    /* GPIOA Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    /* Once the DAC channel is enabled, the corresponding GPIO pin is
automatically connected to the DAC converter. In order to avoid parasitic
consumption, the GPIO pin should be configured in analog */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void BliknutiLEDZelena(void)
{

```

```

STM32_Discovery_LEDOn(LED3);
Delay(0x1FFFF);
STM32_Discovery_LEDOff(LED3);
Delay(0x1FFFF);
}

void TIM2_Inicializace(void)
{
    /* TIM2 Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    /* TIM2 Configuration */
    TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
    TIM_TimeBaseStructure.TIM_Period = 0x500; // 0.5s!! Pro velmi pomaly
prubeh
    TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) 0x24000;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

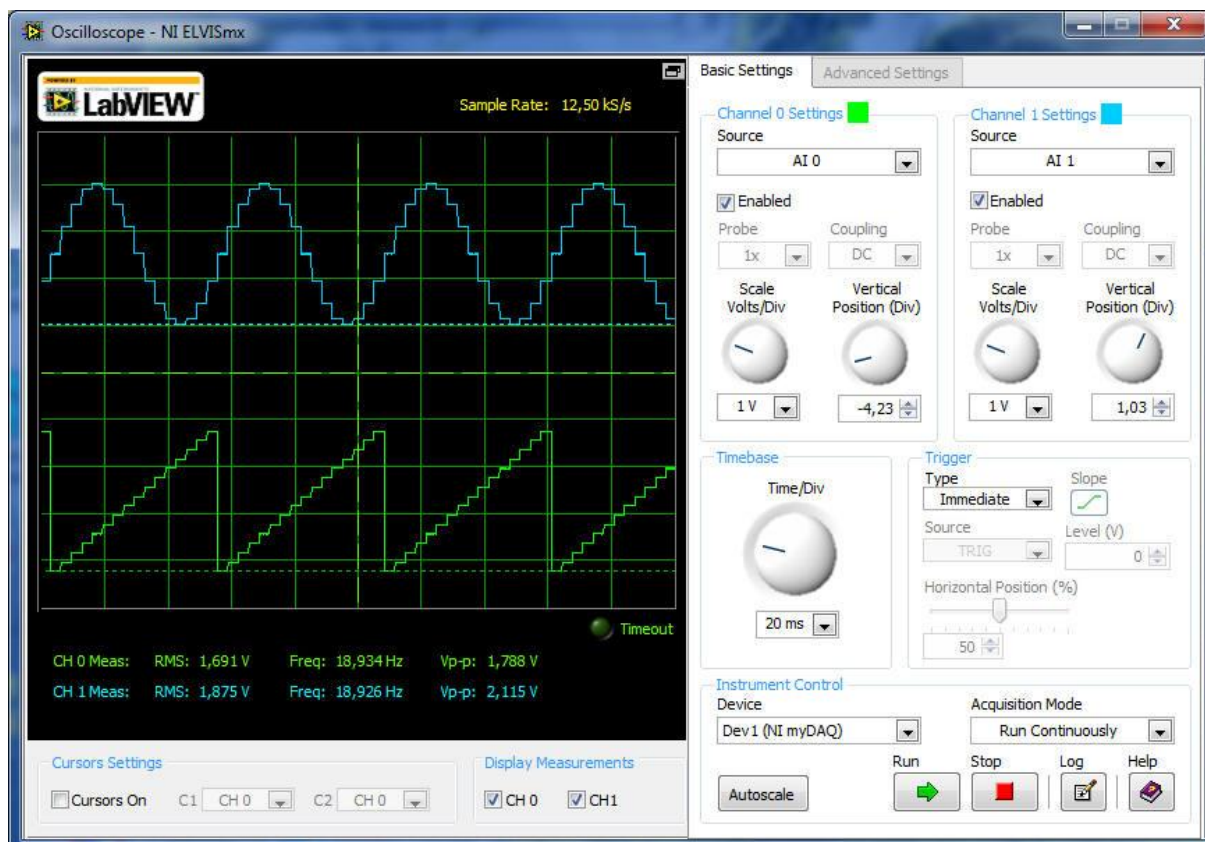
    /* TIM2 TRGO selection */
    TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}

```



2.10.4 DAC (program11e) – generátor trojúhelníkového a sinusového signálu pevného kmitočtu s využitím DMA

Jedná se o vlastně o předchozí program, který se od předchozího liší jen jednodušším obsahem nekonečné hlavní smyčky. Oproti předchozímu programu jsem zjednodušil tento obsah odstraněním kódu na změnu hodnoty proměnné Perioda. Tím generujeme sinusový a trojúhelníkový signál o stálém kmitočtu. Můžeme tak otestovat jak závisí kmitočet výstupního signálu na hodnotě Perioda a na dělicím poměru `TIM_TimeBaseStructure.TIM_Prescaler`.

```
while (1)
{
/* zmenime nastaveni frekvence */
/* 12Hz     pri TIM_Prescaler = 1200 */ //Perioda = 100;
/* 113Hz    pri TIM_Prescaler = 1200 */ //Perioda = 10;
/* 224z     pri TIM_Prescaler = 1200 */ //Perioda = 1;

/* 126Hz    pri TIM_Prescaler = 120 */ //Perioda = 100;
/* 1.13kHz  pri TIM_Prescaler = 120 */ //Perioda = 10;
/* 6.24 kHz pri TIM_Prescaler = 120*/ //Perioda = 1;

/* 243Hz    pri TIM_Prescaler = 60 */ //Perioda = 100;
/* 2.2kHz   pri TIM_Prescaler = 60 */ //Perioda = 10;
/* 12.2 kHz pri TIM_Prescaler = 60 */ //Perioda = 1;
```

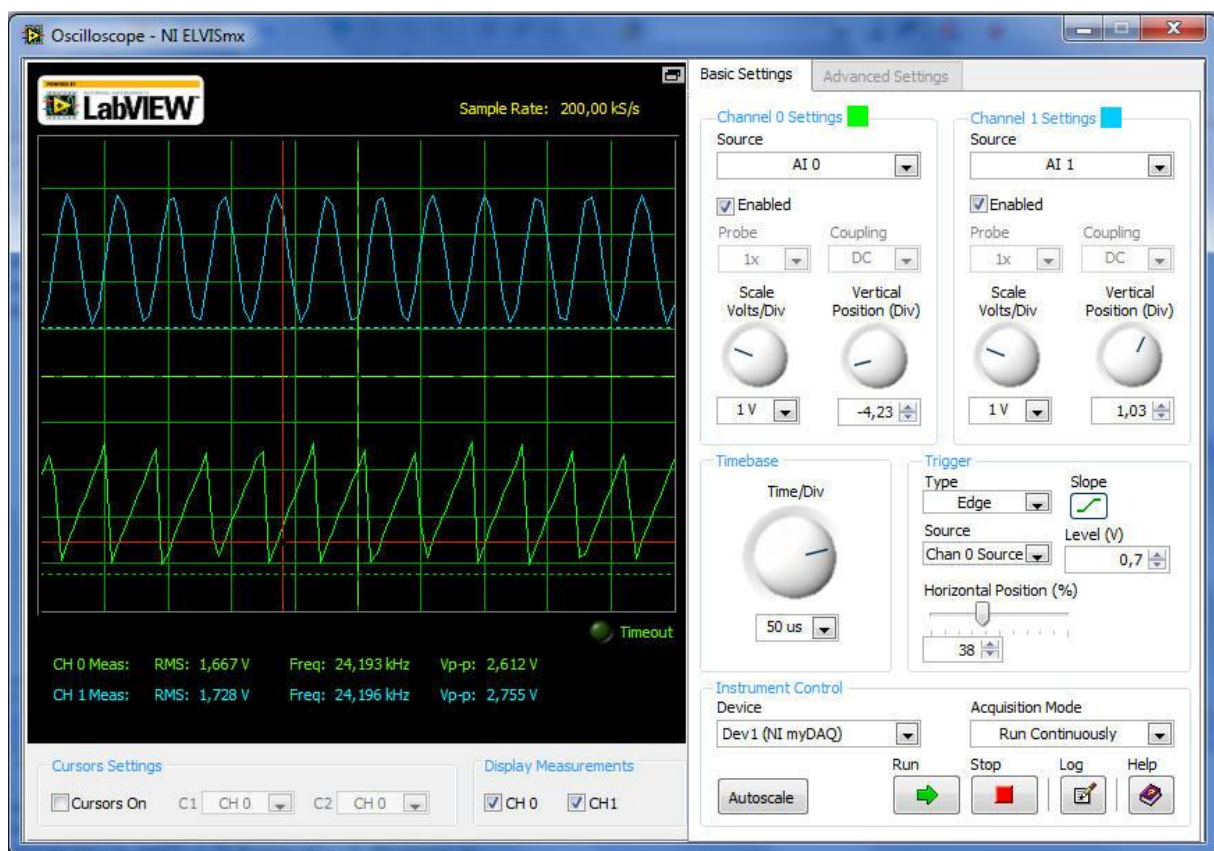


```

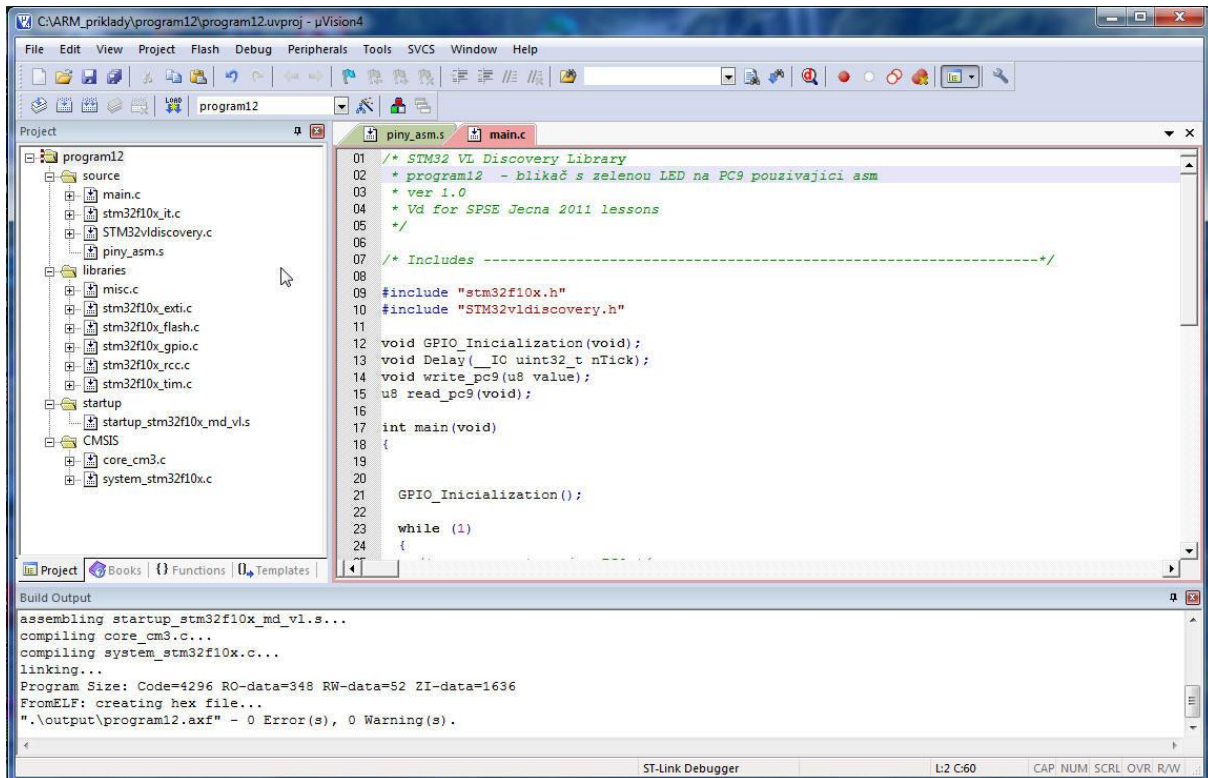
/* 24.2 kHz pri TIM_Prescaler = 30 */          Perioda = 1;

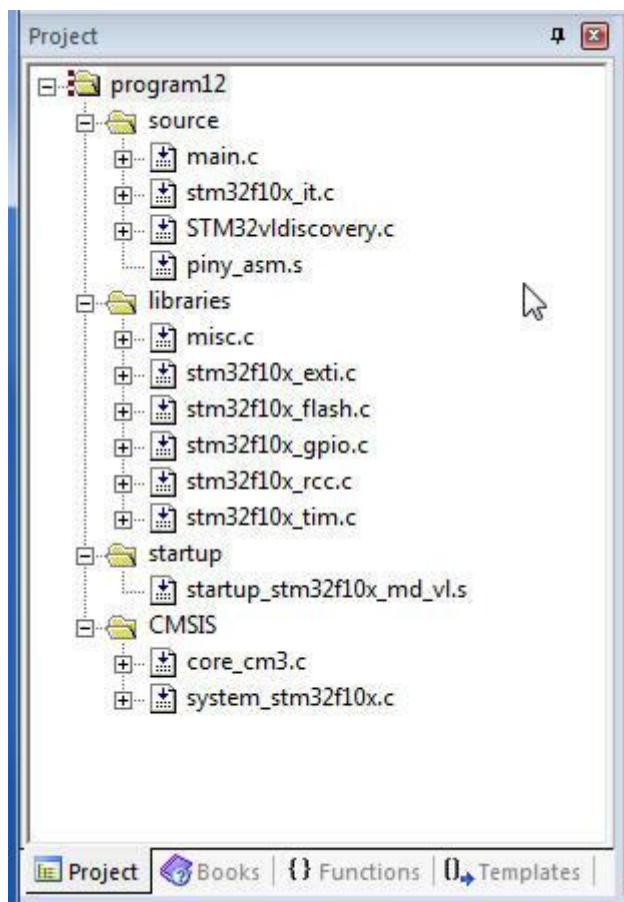
TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) 30;
TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Down;
TIM_TimeBaseStructure.TIM_Period = (uint16_t) Perioda;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
Delay(0x0FFFF);
}

```



2.11 Assembler – blikáme zelenou LED na PC9 (program12)





Obsah souboru **piny_asm.s**

```

; specifikace oblasti
AREA |.text|, CODE, READONLY
; zapis hodnoty bitu PC9
write_pc9    PROC
                EXPORT write_pc9 ;
; konstanty
                LDR r1, =0x40011010
                MOV r2, #0x0200
; podle predaneho argumentu nastavim adresu SET nebo RESET registru
                CMP r0, #0
                ADDEQ r1, r1, #0x04
; zapisu data do ovladacih registru portu C
                STR r2, [r1]
; odchod z funkce

```

```

        BX lr
        ENDP

; cteni hodnoty bitu PC9
read_pc9 PROC

        EXPORT read_pc9;
; nactu obsah vystupniho registru portu C
        LDR r2, =0x4001100c
        LDR r1, [r2]
; vyctu stav pinu 9
        ANDS r1, r1, #0x0200
        MOVEQ r0,#0x00
        MOVNE r0, #0x01
        BX lr
        ENDP

END

```

Pozn. V výše uvedeném asm kódu jsou (občas) důležité i mezery, což je rozdíl od většiny vyšších jazyků

Hodnoty konstant pro jednotlivé piny najdeme např. v souboru **stm32f10x_gpio.h**

```

#define GPIO_Pin_0      ((uint16_t)0x0001) /*!< Pin 0 selected */
#define GPIO_Pin_1      ((uint16_t)0x0002) /*!< Pin 1 selected */
#define GPIO_Pin_2      ((uint16_t)0x0004) /*!< Pin 2 selected */
#define GPIO_Pin_3      ((uint16_t)0x0008) /*!< Pin 3 selected */
#define GPIO_Pin_4      ((uint16_t)0x0010) /*!< Pin 4 selected */
#define GPIO_Pin_5      ((uint16_t)0x0020) /*!< Pin 5 selected */
#define GPIO_Pin_6      ((uint16_t)0x0040) /*!< Pin 6 selected */
#define GPIO_Pin_7      ((uint16_t)0x0080) /*!< Pin 7 selected */
#define GPIO_Pin_8      ((uint16_t)0x0100) /*!< Pin 8 selected */
#define GPIO_Pin_9      ((uint16_t)0x0200) /*!< Pin 9 selected */
#define GPIO_Pin_10     ((uint16_t)0x0400) /*!< Pin 10 selected */
#define GPIO_Pin_11     ((uint16_t)0x0800) /*!< Pin 11 selected */
#define GPIO_Pin_12     ((uint16_t)0x1000) /*!< Pin 12 selected */
#define GPIO_Pin_13     ((uint16_t)0x2000) /*!< Pin 13 selected */
#define GPIO_Pin_14     ((uint16_t)0x4000) /*!< Pin 14 selected */
#define GPIO_Pin_15     ((uint16_t)0x8000) /*!< Pin 15 selected */

```

Obsah souboru main.c

```

/* STM32 VL Discovery Library
 * program12 - blikač s zelenou LED na PC9 pouzivajici asm
 * ver 1.0
 * Vd for SPSE Jecna 2011 lessons
 */

#include "stm32f10x.h"
#include "STM32vldiscovery.h"

void GPIO_Inicialization(void);
void Delay(__IO uint32_t nTick);
void write_pc9(u8 value);
u8 read_pc9(void);

int main(void)
{

    GPIO_Inicialization();

    while (1)
    {
        /* prepneme stav pinu PC9 */
        write_pc9(1 - read_pc9());
        /* cekame */
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
    }
}

void GPIO_Inicialization(void)
{

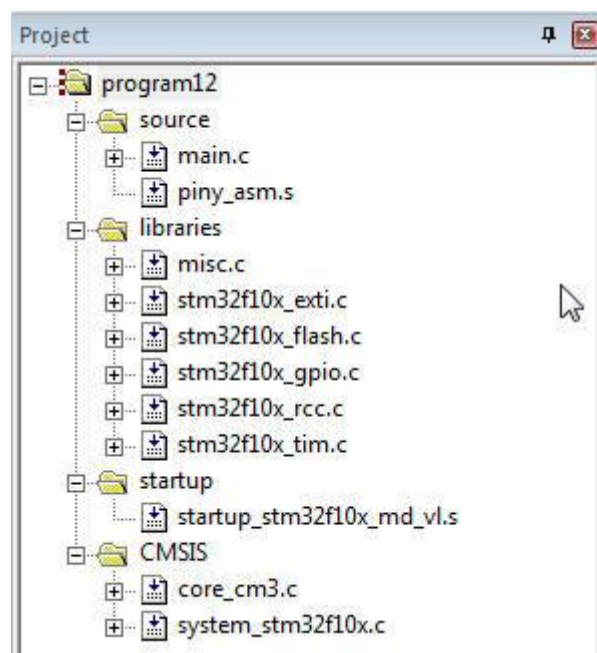
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

}

2.12.1 Blikač s LED na PC7 (program12A)



Obsah souboru **piny_asm.s**

```

; specifikace oblasti
AREA |.text|, CODE, READONLY
; zapis hodnoty bitu PC7
write_pc7    PROC
                EXPORT write_pc7 ;
; konstanty
                LDR r1, =0x40011010
                MOV r2, #0x0080
; podle predaneho argumentu nastavim adresu SET nebo RESET registru
                CMP r0, #0
                ADDEQ r1, r1, #0x04
; zapisu data do ovladaciho registru portu C
                STR r2, [r1]
; odchod z funkce
    
```

```

        BX lr
        ENDP

; cteni hodnoty bitu PC7
read_pc7 PROC

        EXPORT read_pc7;
; nactu obsah vystupniho registru portu C
        LDR r2, =0x4001100c
        LDR r1, [r2]
; vyctu stav pinu 7
        ANDS r1, r1, #0x0080
        MOVEQ r0, #0x00
        MOVNE r0, #0x01
        BX lr
        ENDP

END

```

Obsah souboru **main.c**

```

/* STM32 VL Discovery Library
 * program12a - ukazka pouziti assembleru
 * ver 1.0
 * Vd for SPSE Jecna 2011 lessons
 */

#include "stm32f10x.h"
//#define HSE //Pouziti externiho oscilatoru
#define HSI //Pouziti interniho oscilatoru
/* funkčni prototypy */
void RCC_Configuration(void);
void GPIO_Configuration(void);
void Delay(__IO uint32_t nTick);
void write_pc7(u8 value);
u8 read_pc7(void);

int main(void)
{
    /* konfigurace zdroju hodinoveho signalu */
    RCC_Configuration();
    /* konfigurace I/O portu */
    GPIO_Configuration();
    /* blikaci smycka */
    while (1)
    {
        /* prepneme stav pinu PC7 */

```

```

        write_pc7(1 - read_pc7());
        /* cekame */
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
        Delay(0x5ffff);
    }
}

/* nastaveni zdroju hodinoveho signalu (HSE) */
void RCC_Configuration(void) {
#ifdef HSE
    ErrorStatus HSEStartUpStatus;
    /* reset nastaveni */
    RCC_DeInit();
    /* aktivace signalu HSE (External High Speed oscillator) */
    RCC_HSEConfig(RCC_HSE_ON);
    /* kontrola stability oscilatoru */
    HSEStartUpStatus = RCC_WaitForHSEStartUp();
    if (HSEStartUpStatus == SUCCESS) {
        /*Nastaveni delicek HS*/
        RCC_HCLKConfig(RCC_SYSCLK_Div1); //HCLK = 24 MHz, AHB
        RCC_PCLK2Config(RCC_HCLK_Div1); //APB1 = 24 MHz
        RCC_PCLK1Config(RCC_HCLK_Div1); //APB2 = 24 MHz
        /* nastaveni PLL */
        RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_4); //PLLCLK =
24 MHz
    }
#endif
#ifdef HSI
    /* reset nastaveni */
    RCC_DeInit();
    /* aktivace signalu HSI (Internal High Speed oscillator) */
    RCC_HSIcmd(ENABLE);
    while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET);
    /* nastaveni delicek HS */
    RCC_HCLKConfig(RCC_SYSCLK_Div1); //HCLK = 24 MHz, AHB
    RCC_PCLK1Config(RCC_HCLK_Div1); //APB1 = 24 MHz
    RCC_PCLK2Config(RCC_HCLK_Div1); //APB2 = 24 MHz
    /* nastaveni PLL */
    RCC_PLLConfig(RCC_PLLSource_HSI_Div2, RCC_PLLMul_8); //PLLCLK = 24
MHz
#endif
    /* nastaveni FLASH */
    FLASH_SetLatency(FLASH_Latency_2);
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
    RCC_PLLCmd(ENABLE);
    while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
    /*Zhroj hodin pro AHB*/
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
    while (RCC_GetSYSCLKSource() != 0x08);
}

```



```

/* aktivace zdroje HS pro periferie */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC , ENABLE);
}

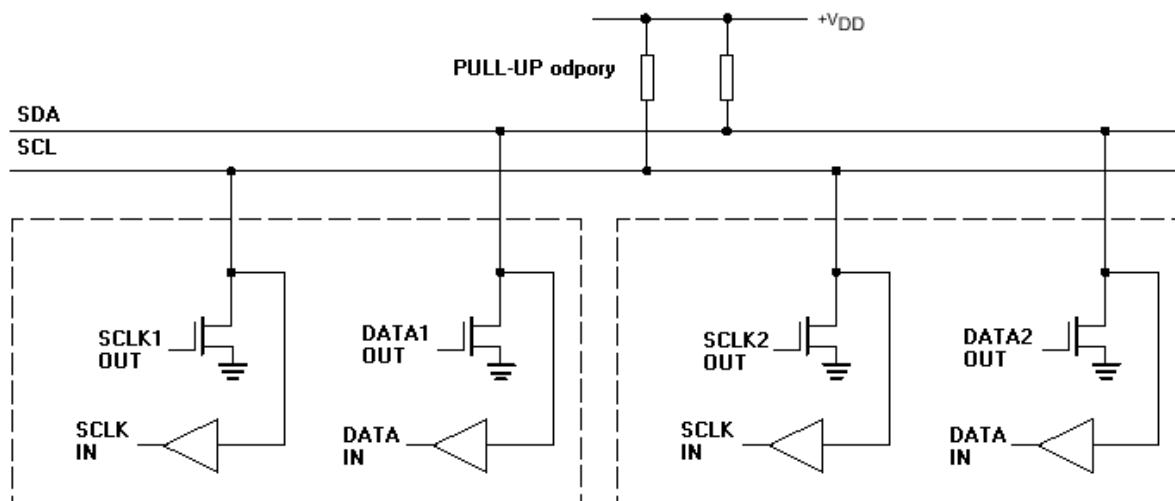
/* konfigurace I/O portu */
void GPIO_Configuration(void) {
    /* struktura s informacemi o konfiguraci pinu */
    GPIO_InitTypeDef GPIO_InitStructure;
    /* konfigurujeme pin PC7 jako output push-pull */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

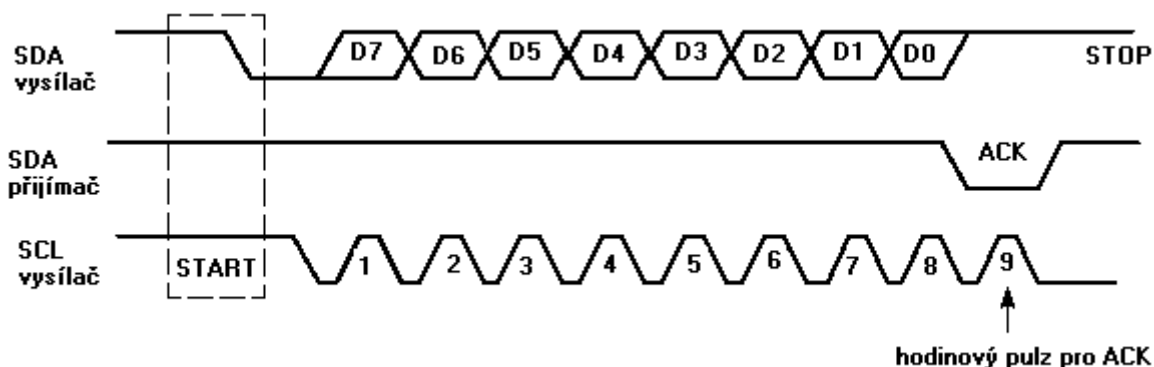
```

2.13.1 I2C EEPROM AT24C02 (program13)

Z hlediska univerzálnosti použití má velký praktický význam sběrnice I2C (Inter-Integrated-Circuit Bus) vytvořená firmou Philips. Podporu této sběrnice však nalezneme i u řady integrovaných obvodů jiných výrobců. Sběrnice je tvořena dvojicí vodičů, na kterých je v klidovém stavu přes pull up odpory nastavena logická jednička, viz.obr.:



Jeden z vodičů, značený SCL , přenáší hodinový signál. Druhý, značený SDA, slouží k synchronnímu přenosu dat:



Budiče s otevřeným kolektorem umožňují, aby sběrnice byla využívána skupinou rovnoprávných řadičů sběrnice I2C (multimaster konfigurace). Odpovídající arbitrážní protokol, který řeší přidělení sběrnice při současném požadavku více řadičů sběrnice I2C je popsán v dokumentaci k tomuto protokolu. V následujících dvou programech budeme uvažovat nejčastější případ, kdy sběrnici bude řídit jen jeden řadič sběrnice. Bude jím mikrořadič STM32 řízený našim programem a využívající jeho knihovní funkce pro I2C

Komunikaci zahajuje řadič prvkem START – sestupná hrana signálu SDA při jedničce signálu SCL. Dále je vyslán osmibitový znak po vodiči SDA počínajíc nejvýznamějším bitem. Vyslaný znak je potvrzován přijmačem stažením signálu SCL na úroveň nula – signál ACK (acknowledge).

Norma I2C definuje i formát přenášených dat, jejich potvrzování a předávání řízení mezi účastníky komunikace. Prvním znakem, vyslaným řadičem, je vždy řídicí slovo. Jeho sedm nejvýznamějších bitů A7, A6 ... A1 slouží k specifikaci podřízeného obvodu, s nímž bude probíhat komunikace. Nejméně významný bit A0 řídicího slova pak určuje směr následující komunikace, tj. směr druhého, popř. dalších slov. A0 = 0 znamená směr přenosu z řadiče do podřízeného obvodu, A0 = 1 obrácený směr přenosu. První čtyři (popř. další) bity řídicího slova A7, A6, A5, A4 jsou určeny typem obvodu, Další bity A3, A2, A1 jsou nastaveny adresacími vývody podřízeného obvodu. Těchto podřízených obvodů tedy může být ke sběrnici I2C připojeno až $2^3 = 8$.

Pro aplikace v oblasti spotřební elektroniky a telekomunikací vyrábějí firmy Philips a Siemens mnoho obvodů řízených sběrnici I2C. Pro ilustraci si některé uvedeme:

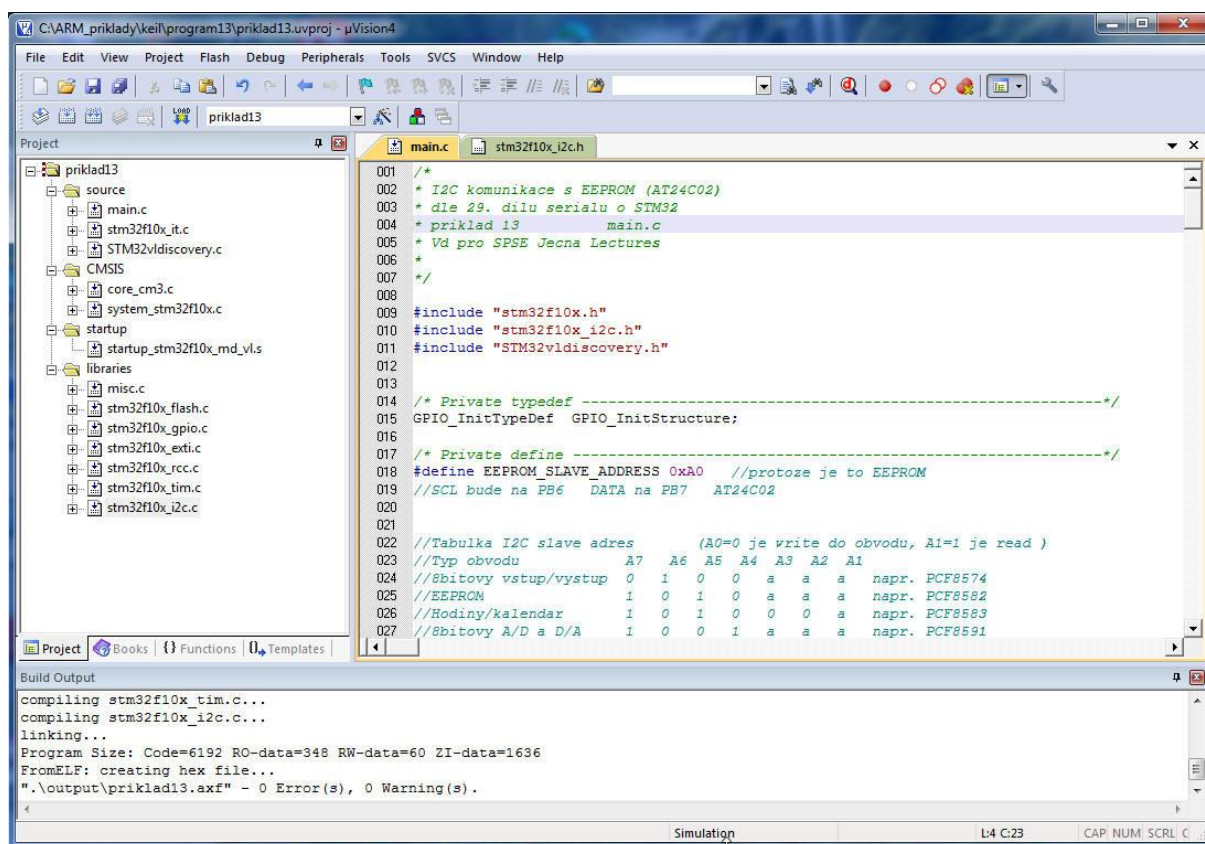
| | | A7 | A6 | A5 | A4 | A3 | A2 | A1 |
|----------|--------------------------------------|----|----|----|----|----|----|----|
| PCF8574 | Osmibitový vstup/výstup | 0 | 1 | 0 | 0 | a | a | a |
| PCF8577 | 64-segmentový řadič LCD | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| PCF8578 | Řadič bodového LCD | 0 | 1 | 1 | 1 | 1 | 0 | a |
| PCF8582A | Paměť EEPROM 256 x 8 | 1 | 0 | 1 | 0 | a | a | a |
| PCF8583 | Hodiny/kalendář a paměť RAM | 1 | 0 | 1 | 0 | 0 | 0 | a |
| PCF8591 | Osmibitový A/D, D/A převodník | 1 | 0 | 0 | 1 | a | a | a |
| SAA3028 | Infra přijímač kódu RD5 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| TDA8444 | Osminásobný D/A převodník | 0 | 1 | 0 | 0 | a | a | a |
| TSA5055 | Syntezátor kmitočtu – PLL do 2,6 GHz | 1 | 1 | 0 | 0 | 0 | a | a |

Protože obvodů řízených I2C je velký počet, není možné, aby každý takový obvod měl v této tabulce svoji řádku a proto některé obvody mají stejné hodnoty bitů A7, A6, A5, A4. Jde však přitom obvody s obdobnou funkcí. Např. všechny paměti EEPROM mají hodnoty těchto bitů 1 0 1 0, takže např. AT24C01 či 24C02 jsou řízeny a programovány stejně, jako PCF8582A. Tato paměť nám také poslouží v následujícím příkladě k podrobnějšímu vysvětlení funkce protokolu I2C i snadnosti naprogramování

mikrořadiče STM32 jako řadiče I2C. Nejprve si ukážeme jednoduchý program umožňující naprogramovat několik byte v paměti EEPROM 24C01 či 24C02, Ta bude s MCU STM32 propojena dvěma vodiči , např.:

```
// I2C EEPROM připojíme k PB
//SCL bude na PB6 DATA na PB7 AT24C02
```

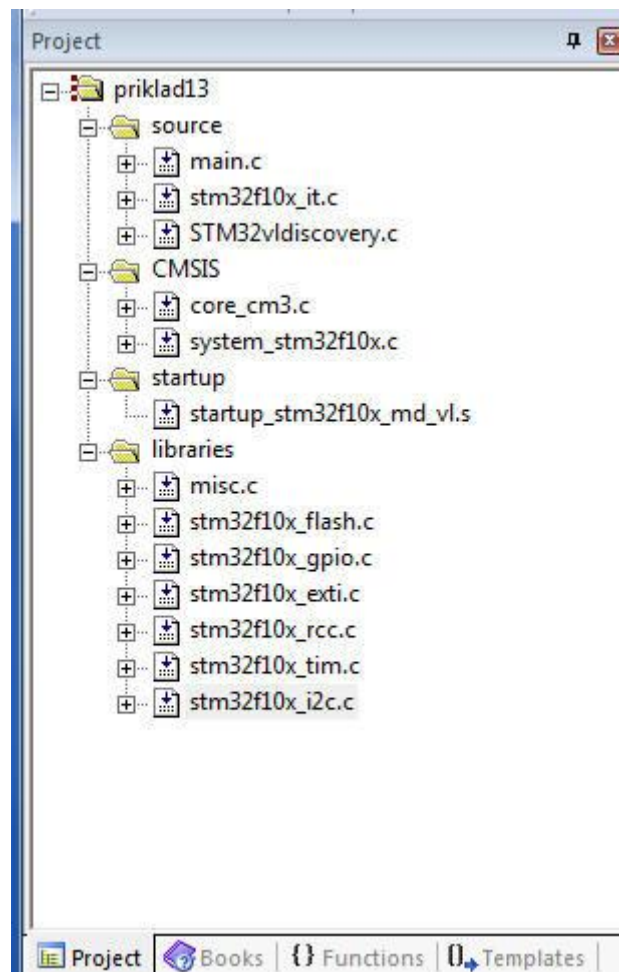
Tak jako v předchozích příkladech uvedeme jen základní informace, tj. zdrojový kód **main.c** a strukturu projektu:



```
001 /*
002 * I2C komunikace s EEPROM (AT24C02)
003 * dle 29. dílu seriálu o STM32
004 * příklad 13 main.c
005 * Vd pro SPSE Jecna Lectures
006 *
007 */
008
009 #include "stm32f10x.h"
010 #include "stm32f10x_i2c.h"
011 #include "STM32vldiscovery.h"
012
013
014 /* Private typedef -----*/
015 GPIO_InitTypeDef GPIO_InitStructure;
016
017 /* Private defines -----*/
018 #define EEPROM_SLAVE_ADDRESS 0xA0 //protoze je to EEPROM
019 //SCL bude na PB6 DATA na PB7 AT24C02
020
021
022 //Tabulka I2C slave adres (A0=0 je write do obvodu, A1=1 je read )
023 //Typ obvodu A7 A6 A5 A4 A3 A2 A1
024 //8bitovy vstup/vystup 0 1 0 0 a a a napr. PCF8574
025 //EEPROM 1 0 1 0 a a a napr. PCF8582
026 //Hodiny/kalendar 1 0 1 0 0 0 a napr. PCF8583
027 //8bitovy A/D a D/A 1 0 0 1 a a a napr. PCF8591
```

Build Output

```
compiling stm32f10x_tim.c...
compiling stm32f10x_i2c.c...
linking...
Program Size: Code=6192 RO-data=348 RW-data=60 ZI-data=1636
FromELF: creating hex file...
".\output\priklad13.axf" - 0 Error(s), 0 Warning(s).
```



```

/*
 * I2C komunikace s EEPROM (AT24C02)
 * dle 29. dilu serialu o STM32
 * priklad 13          main.c
 * Vd pro SPSE Jecna Lectures
 *
 */

#include "stm32f10x.h"
#include "stm32f10x_i2c.h"
#include "STM32vldiscovery.h"

/* Private typedef -----*/
GPIO_InitTypeDef  GPIO_InitStructure;

/* Private define -----*/
#define EEPROM_SLAVE_ADDRESS 0xA0    //protoze je to EEPROM
//SCL bude na PB6   DATA na PB7   AT24C02

```

```
//Tabulka I2C slave adres          (A0=0 je write do obvodu, A1=1 je read )
//Typ obvodu          A7      A6      A5      A4      A3      A2      A1
//8bitovy vstup/vystup 0      1      0      0      a      a      a      napr.
PCF8574
//EEPROM              1      0      1      0      a      a      a      napr.
PCF8582
//Hodiny/kalendar    1      0      1      0      0      0      a      napr.
PCF8583
//8bitovy A/D a D/A  1      0      0      1      a      a      a      napr.
PCF8591
//PLL                  1      1      0      0      0      a      a      napr.
TSA5055

/* Private variables -----*/
uint8_t hodnota=0;

void Delay(__IO uint32_t nTick);
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu

void I2C_Inicializace(void); // nastavení I2C
uint8_t Read_EEPROM(uint8_t adresa); // cteni
void Write_EEPROM(uint8_t adresa, uint8_t data); // zapis
void Ready_EEPROM(void); // polling potvrzeni
void bliknutiLEDMODRA(void); //
void bliknutiLEDZELENA(void); //

int main(void)
{
    GPIO_Inicializace();
    I2C_Inicializace();
    // blikneme modrou jako priznak startu
    bliknutiLEDMODRA();
    /* pouzita EEPROM AT24C02 - 256 bajtu x 8 bitu */

    hodnota = Read_EEPROM(0x00);
    hodnota = Read_EEPROM(0x01);
    hodnota = Read_EEPROM(0x02);
    hodnota = Read_EEPROM(0x03);
    hodnota = Read_EEPROM(0x04);

    Write_EEPROM(0x00, 0x01);
    Ready_EEPROM();
    bliknutiLEDZELENA();
    Write_EEPROM(0x01, 0x23);
    Ready_EEPROM();
    bliknutiLEDMODRA();
    Write_EEPROM(0x02, 0x45);
    Ready_EEPROM();
    bliknutiLEDZELENA();
    Write_EEPROM(0x03, 0x67);
    Ready_EEPROM();
    bliknutiLEDMODRA();
    hodnota = Read_EEPROM(0x00);
}
```

```

hodnota = Read_EEPROM(0x01);
hodnota = Read_EEPROM(0x02);
hodnota = Read_EEPROM(0x03);
hodnota = Read_EEPROM(0x04);

// blikneme zelenou jako priznak konce
bliknutiLEDZELENA();

while (1)
{
    bliknutiLEDMODRA();
}

void GPIO_Inicializace(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void bliknutiLEDMODRA(void)
{
    STM32vldiscovery_LEDOn(LED4); // zapnem LED4 - modra
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED4); // vypnem LED4 - modra
    Delay(0x5FFFF);
}

void bliknutiLEDZELENA(void)
{
    STM32vldiscovery_LEDOn(LED3); // zapnem LED3 - zelena
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED3); // vypnem LED3 - zelena
    Delay(0x5FFFF);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void I2C_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    I2C_InitTypeDef I2C_InitStructure;
    I2C_DeInit(I2C1);

```

```

/* I2C Periph clock enable */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
/* GPIO Periph clock enable */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

/* Configure I2C pins: SCL and SDA */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD; //GPIO_Mode_AF_PP
GPIO_Init(GPIOB, &GPIO_InitStructure); //I2C EEPROM pripojime k PB
//SCL bude na PB6 DATA na PB7 AT24C02

/* I2C configuration */
I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStructure.I2C_OwnAddress1 = 0xA0; //EEPROM
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_InitStructure.I2C_ClockSpeed = 10000;

/* I2C Peripheral Enable */
I2C_Cmd(I2C1, ENABLE);
/* Apply I2C configuration after enabling it */
I2C_Init(I2C1, &I2C_InitStructure);
}

uint8_t Read_EEPROM(uint8_t adresa)
{
    uint8_t Data;
    Data = 0;
    /* While the bus is busy */
    while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

    /* Send START condition */
    I2C_GenerateSTART(I2C1, ENABLE);

    /* Test on EV5 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

    /* Send EEPROM address for read */
    I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);

    /* Test on EV6 and clear it */
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    /* Clear EV6 by setting again the PE bit */
    I2C_Cmd(I2C1, ENABLE);

    /* Send the EEPROM's internal address to read from: MSB of the
address first */
    I2C_SendData(I2C1, adresa);
}

```

```

/* Test on EV8 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

/* Send START condition a second time */
I2C_GenerateSTART(I2C1, ENABLE);

/* Test on EV5 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

/* Send EEPROM address for read */
I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Receiver);

/* Test on EV6 and clear it */
while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

/* Disable Acknowledgement */
I2C_AcknowledgeConfig(I2C1, DISABLE);

/* Send STOP Condition */
I2C_GenerateSTOP(I2C1, ENABLE);
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)) {}

/* Read a byte from the EEPROM */
Data = I2C_ReceiveData(I2C1);

/* Enable Acknowledgement to be ready for another reception */
I2C_AcknowledgeConfig(I2C1, ENABLE);

return(Data);
}

void Write_EEPROM(uint8_t adresa, uint8_t data)
{
/* Send START condition */
I2C_GenerateSTART(I2C1, ENABLE);

/* Test on EV5 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

/* Send EEPROM address for write */
I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);

/* Test on EV6 and clear it */
while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

/* Send the EEPROM's internal address to write to : MSB of the
address first */
I2C_SendData(I2C1, adresa);

/* Test on EV8 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

```



```

/* Send the byte to be written */
I2C_SendData(I2C1, data);

/* Test on EV8 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

/* Send STOP condition */
I2C_GenerateSTOP(I2C1, ENABLE);
}

void Ready_EEPROM(void)
{
    __IO uint16_t temp = 0;

    do
    {
        /* Send START condition */
        I2C_GenerateSTART(I2C1, ENABLE);

        /* Read I2C_EE SR1 register to clear pending flags */
        temp = I2C_ReadRegister(I2C1, I2C_Register_SR1);

        /* Send EEPROM address */
        I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);

        }while(!(I2C_ReadRegister(I2C1, I2C_Register_SR1) & 0x0002));

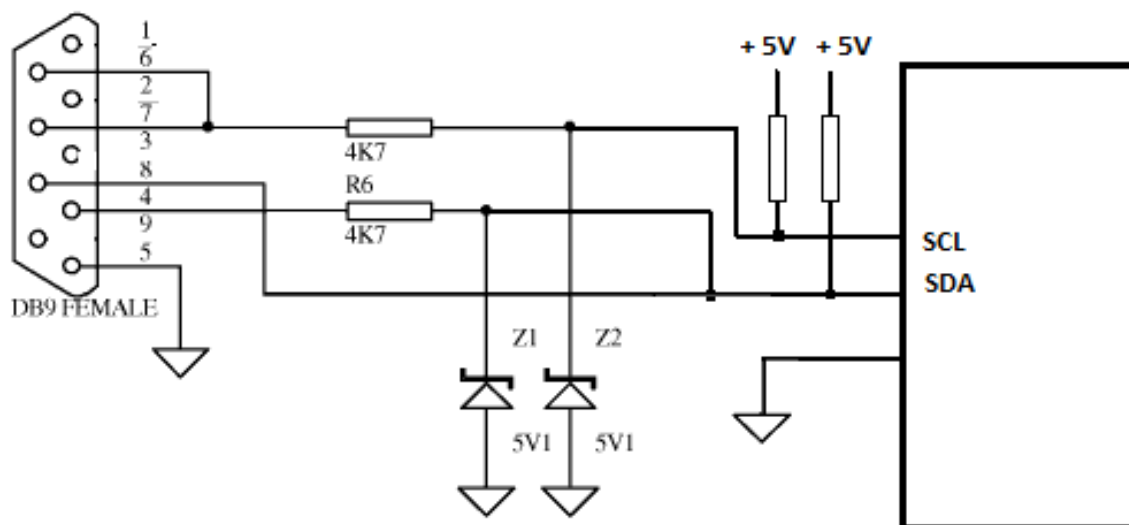
        /* Clear AF flag */
        I2C_ClearFlag(I2C1, I2C_FLAG_AF);

        /* STOP condition */
        I2C_GenerateSTOP(I2C1, ENABLE);
    }
}

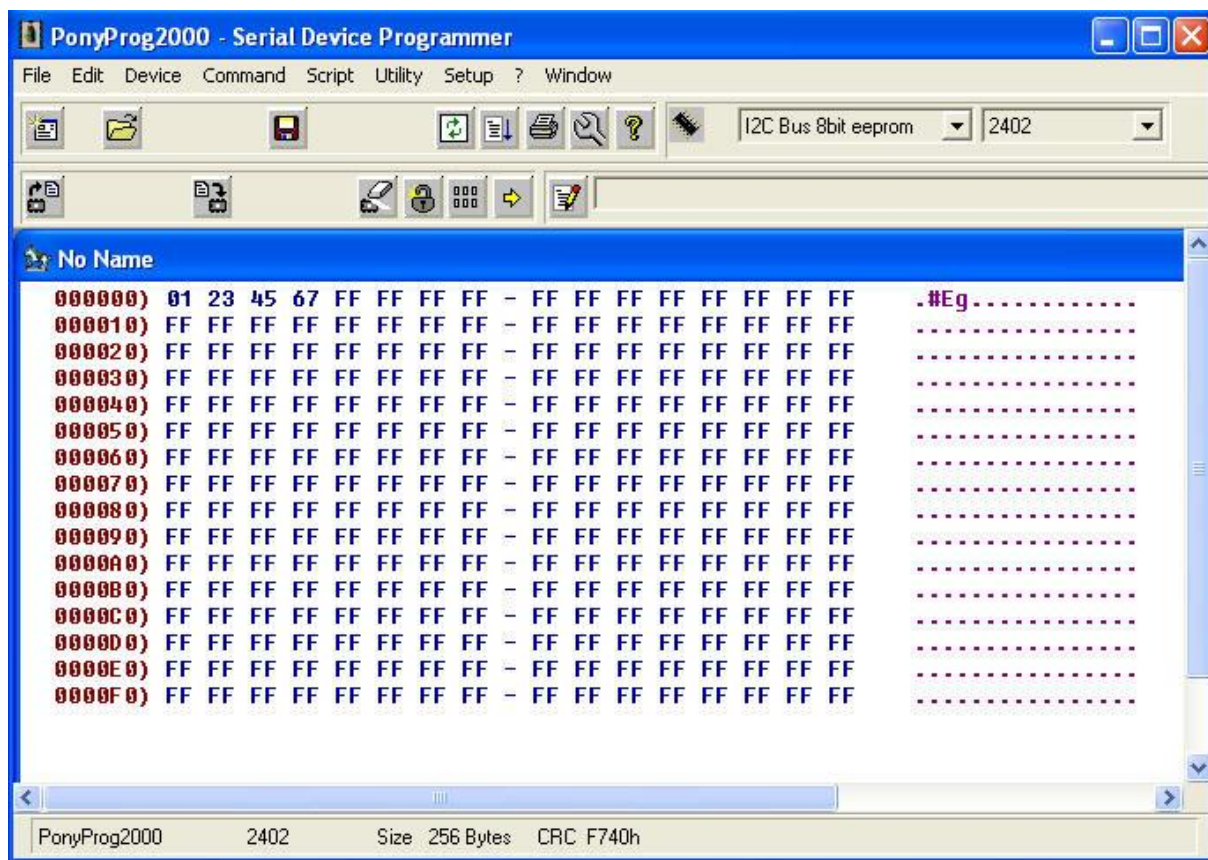
```

Zkontrolovat obsah EEPROM můžeme např. free programem **PonyProg2000**

<http://www.lancos.com/prog.html> a přípravkem připojeným k sériovému portu **COM1** či **COM2**.

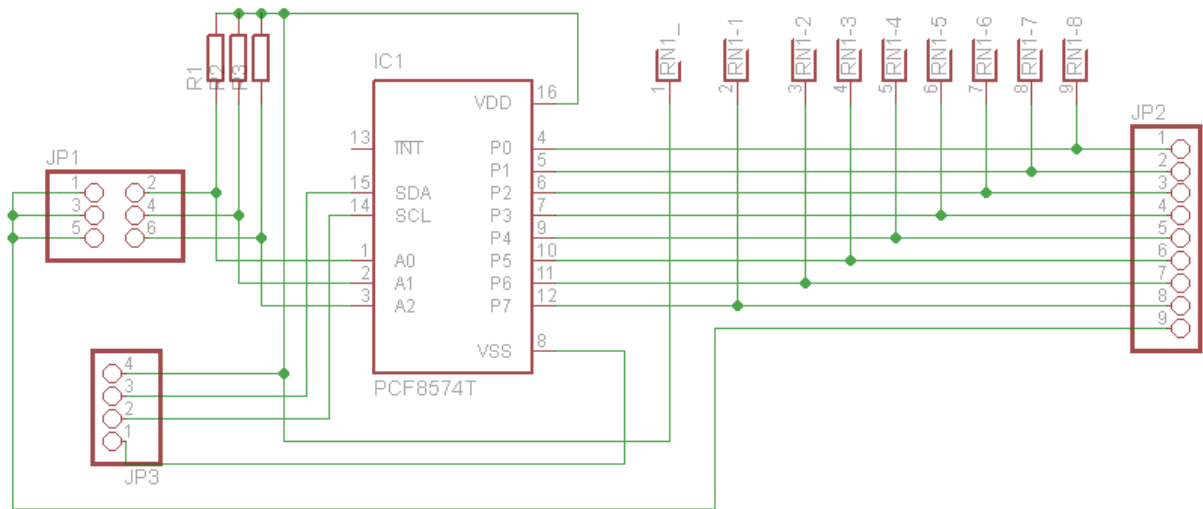


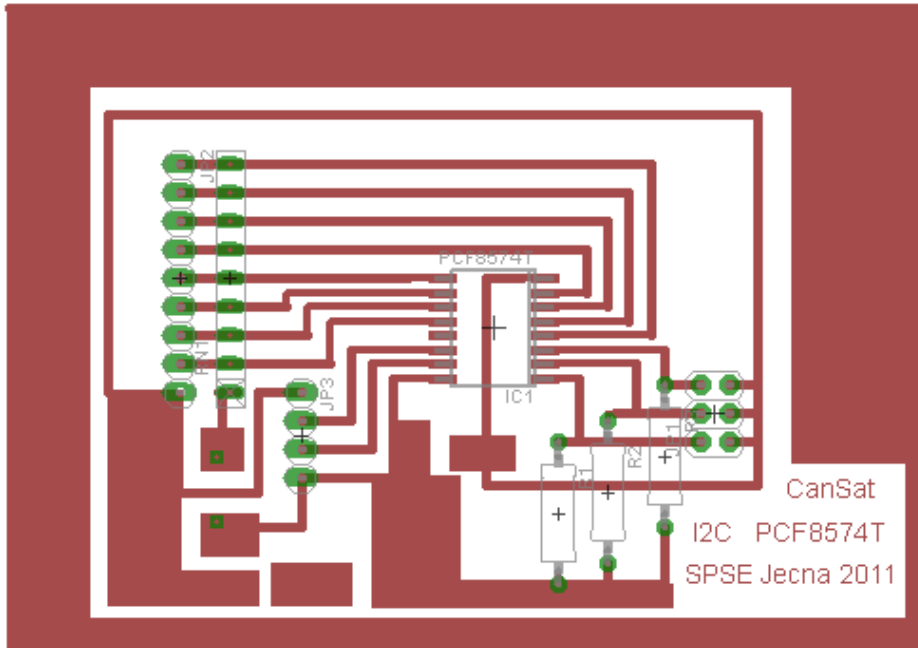
Nové počítače už obvykle port COM nemají a tak můžeme použít převodník USB/COM a jeho driver, čímž získáme virtuální sériový port.

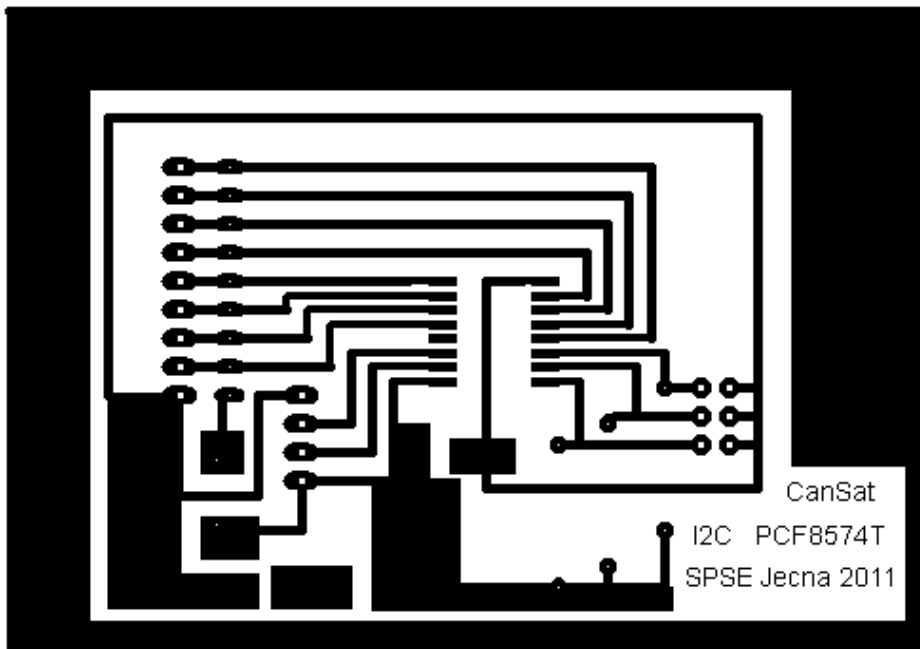


2.13.2 I2C – paralelní port PCF8574 (program14 a program14b)

Schéma a PCB v free EAGLE :

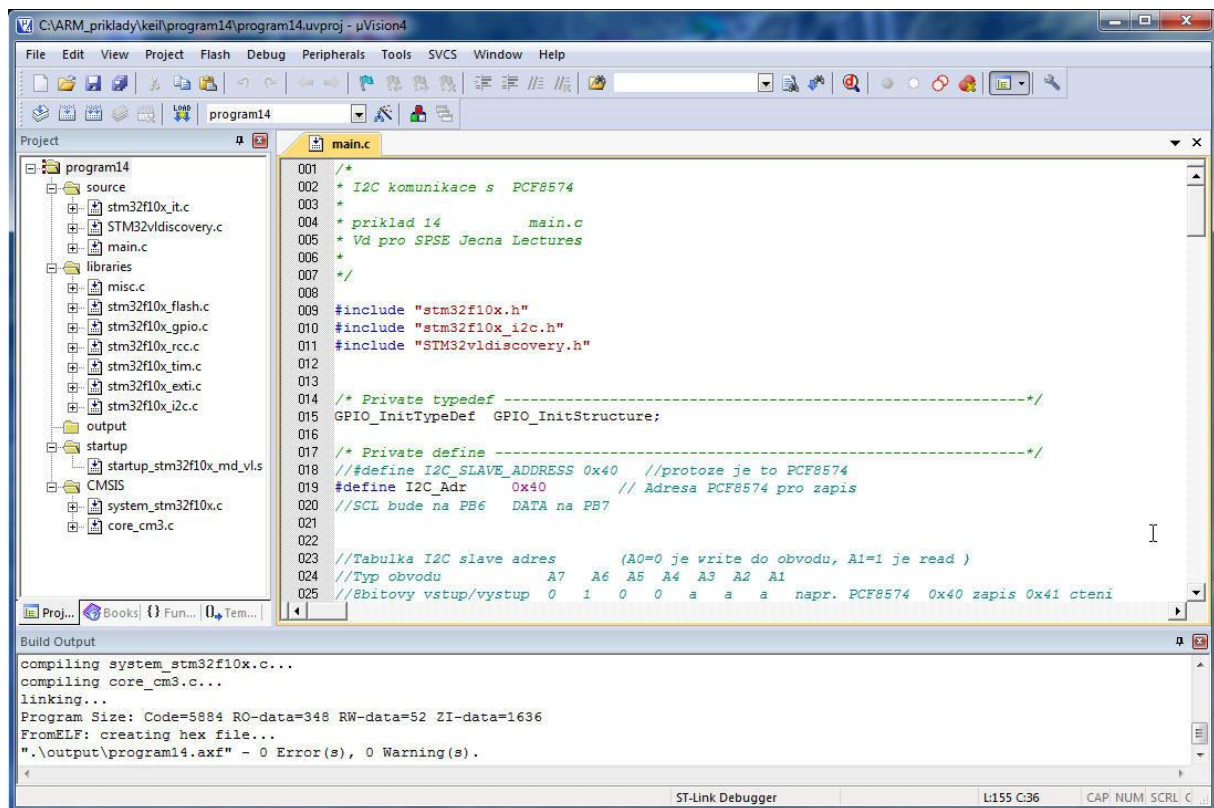






Nyní se již budeme věnovat programování:

Nejprve vytvoříme program pro zápis tj. pomocí I2C nastavíme logické úrovně na 8 I/O obvodu PCF8574



```

001  /*
002  * I2C komunikace s PCF8574
003  *
004  * priklad 14      main.c
005  * Vd pro SPSE Jecna Lectures
006  *
007  */
008
009  #include "stm32f10x.h"
010  #include "stm32f10x_i2c.h"
011  #include "STM32vldiscovery.h"
012
013
014  /* Private typedef -----*/
015  GPIO_InitTypeDef  GPIO_InitStructure;
016
017  /* Private define -----*/
018  //#define I2C_SLAVE_ADDRESS 0x40 //protoze je to PCF8574
019  #define I2C_Adr      0x40 // Adresa PCF8574 pro zapis
020  //SCL bude na PB6  DATA na PB7
021
022
023  //Tabulka I2C slave adres      (A0=0 je write do obvodu, A1=1 je read )
024  //Typ obvodu      A7  A6  A5  A4  A3  A2  A1
025  //8bitovy vstup/vystup  0  1  0  0  a  a  a  napr. PCF8574  0x40 zapis 0x41 cteni

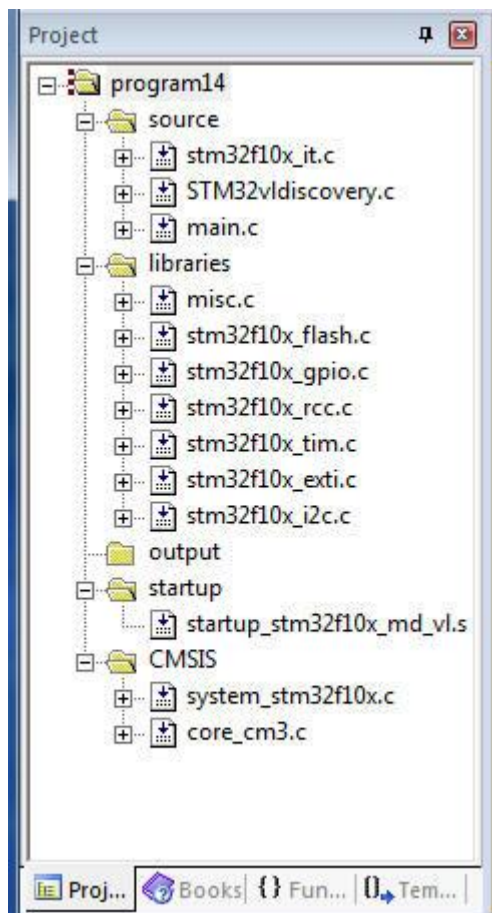
```

Build Output

```

compiling system_stm32f10x.c...
compiling core_cm3.c...
linking...
Program Size: Code=5884 RO-data=348 RW-data=52 ZI-data=1636
FromELF: creating hex file...
".\output\program14.axf" - 0 Error(s), 0 Warning(s).

```



```

/*
 * I2C komunikace s      PCF8574
 *
 * priklad 14          main.c
 * Vd pro SPSE Jecna Lectures
 *
 */

#include "stm32f10x.h"
#include "stm32f10x_i2c.h"
#include "STM32vldiscovery.h"

/* Private typedef -----
---*/
GPIO_InitTypeDef  GPIO_InitStructure;

/* Private define -----
---*/
// #define I2C_SLAVE_ADDRESS 0x40 // protoze je to PCF8574
#define I2C_Adr      0x40 // Adresa PCF8574 pro zapis

```

```
//SCL bude na PB6   DATA na PB7

//Tabulka I2C slave adres          (A0=0 je write do obvodu, A1=1 je read )
//Typ obvodu                       A7    A6    A5    A4    A3    A2    A1
//8bitovy vstup/vystup              0    1    0    0    a    a    a
//napr. PCF8574  0x40 zapis 0x41 cteni
//EEPROM
//napr. PCF8582
//Hodiny/kalendar                   1    0    1    0    0    0    a
//napr. PCF8583
//8bitovy A/D a D/A                 1    0    0    1    a    a    a
//napr. PCF8591
//PLL                                1    1    0    0    0    a    a
//napr. TSA5055

/* Private variables -----
---*/
uint8_t hodnota=0;
void Wait_ms(u8 Time);
void Wait_us(void);
void Delay(__IO uint32_t nTick);
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu
void I2C_Inicializace(void); // nastavení I2C
void PCF8574_Init(void);
void bliknutiLEDMODRA(void); //
void bliknutiLEDZELENA(void); //

int main(void)
{
    GPIO_Inicializace();
    I2C_Inicializace();
    // blikneme modrou jako priznak startu
    bliknutiLEDMODRA();
    PCF8574_Init();
    bliknutiLEDZELENA();
    //zapis 0x55
    I2C_SendData(I2C1, 0x55);
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_BYTE_TRANSMITTED));
    Wait_ms(1);
    bliknutiLEDMODRA();
    Delay(0xAFFFFFFF);
    Delay(0xAFFFFFFF);
    Delay(0xAFFFFFFF);
    Delay(0xAFFFFFFF);
    // Send STOP condition
    I2C_GenerateSTOP(I2C1, ENABLE);
    // blikneme zelenou jako priznak konce
    bliknutiLEDZELENA();
}
```



```

while (1)
{
    PCF8574_Init();
    I2C_SendData(I2C1, 0x55);
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_BYTE_TRANSMITTED));
    Wait_ms(1);
    bliknutiLEDZELENA();
    I2C_SendData(I2C1, 0xAA);
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_BYTE_TRANSMITTED));
    Wait_ms(1);
    bliknutiLEDMODRA();
    I2C_GenerateSTOP(I2C1, ENABLE);
}
}

void GPIO_Inicializace(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void bliknutiLEDMODRA(void)
{
    STM32vldiscovery_LEDOn(LED4); // zapnem LED4 - modra
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED4); // vypnem LED4 - modra
    Delay(0x5FFFF);
}

void bliknutiLEDZELENA(void)
{
    STM32vldiscovery_LEDOn(LED3); // zapnem LED3 - zelena
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED3); // vypnem LED3 - zelena
    Delay(0x5FFFF);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

```

void I2C_Inicializace(void) //pro zapis
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    I2C_InitTypeDef  I2C_InitStructure;
    I2C_DeInit(I2C1);

    /* I2C Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* GPIO Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    /* Configure I2C pins: SCL and SDA */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD; //GPIO_Mode_AF_PP
    GPIO_Init(GPIOB, &GPIO_InitStructure); //I2C PCF8574 pripojime k PB
    //SCL bude na PB6  DATA na PB7

    /* I2C configuration */
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = 0x40; //PCF8574 zapis
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = 10000;

    /* I2C Peripheral Enable */
    I2C_Cmd(I2C1, ENABLE);
    /* Apply I2C configuration after enabling it */
    I2C_Init(I2C1, &I2C_InitStructure);
}

void PCF8574_Init(void) //pro zapis
{
    Wait_ms(200); // Pockat na nabeh
napeti

//Start komunikace s PCF8574
    I2C_GenerateSTART(I2C1, ENABLE); // Generovat START

// Test on EV5 and clear it
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

// Send I2C address for write
    I2C_Send7bitAddress(I2C1, I2C_Adr, I2C_Direction_Transmitter);

// Test on EV6 and clear it
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
}

void Wait_us(void)

```

```
{
vu32 nTick = 0xFFF;
  for(; nTick != 0; nTick--);
}

void Wait_ms(u8 Time)
{
  if (Time > 0 )
    for(; Time != 0; Time--)
    {
      Wait_us();
    }
}
```

Program pro čtení: tj pomocí I2C přečteme logické úrovně na 8 I/O obvodu PCF8574 a přeneseme do STM32.

Bude to vyžadovat následující změny:

Ve funkci I2C_Inicializace (void) bude

```
I2C_InitStructure.I2C_OwnAddress1 = 0x40; //PCF8574 zapis
```

nahraženo

```
I2C_InitStructure.I2C_OwnAddress1 = 0x41; //PCF8574 ctení
```

Změní se i funkce

```
void PCF8574_Init(void) //pro ctení
{
  Wait_ms(200);          // Pockat na nabeh napeti

  //Start komunikace s PCF8574
  I2C_GenerateSTART(I2C1, ENABLE);    // Generovat START

  // Test on EV5 and clear it
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

  // Send I2C address for write
  I2C_Send7bitAddress(I2C1, I2C_Adr,I2C_Direction_Receiver);

  // Test on EV6 and clear it
  while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
}
```

Posloupnost příkazů pro čtení byte dat z PCF8574 bude

```
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
    Wait_ms(1);
    hodnota = I2C_ReceiveData(I2C1);
```

Následující výpis **main.c** je z programu **příklad14b**, ve kterém se na LCD zobrazuje číslo 0 až 255 odpovídající nastavení hodnot na vstupech obvodu PCF8574

```
/*
 * I2C komunikace s      PCF8574
 * cteni dat z PCF8574
 * priklad 14b          main.c
 * Vd pro SPSE Jecna Lectures
 *
 */
#include <stdint.h>
#include "stm32f10x.h"
#include "stm32f10x_i2c.h"
#include "STM32vldiscovery.h"
#include "LCD.h"

uint8_t LCD_text[] = "Test PCF8574.";

/* Private typedef -----*/
GPIO_InitTypeDef  GPIO_InitStructure;

/* Private define -----*/
// #define I2C_SLAVE_ADDRESS 0x40 // protoze je to PCF8574
#define I2C_Adr      0x41 // Adresa PCF8574 pro cteni
// SCL bude na PB6   DATA na PB7

// Tabulka I2C slave adres (A0=0 je write do obvodu, A1=1 je read )
// Typ obvodu           A7    A6    A5    A4    A3    A2    A1
// 8bitovy vstup/vystup  0    1    0    0    a    a    a
//   napr. PCF8574 0x40 zapis 0x41 cteni
// EEPROM              1    0    1    0    a    a    a
//   napr. PCF8582
// Hodiny/kalendar     1    0    1    0    0    0    a
//   napr. PCF8583
// 8bitovy A/D a D/A   1    0    0    1    a    a    a
//   napr. PCF8591
// PLL                  1    1    0    0    0    a    a
//   napr. TSA5055

/* Private variables -----
---*/
```

```

uint8_t hodnota;
void itoa(uint16_t n, int8_t s[]);
void reverse(int8_t s[]);
void Wait_ms(u8 Time);
void Wait_us(void);
void Delay(__IO uint32_t nTick);
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu
void I2C_Inicializace(void); // nastavení I2C
void PCF8574_Init(void);
void bliknutiLEDMODRA(void); //
void bliknutiLEDZELENA(void); //

int main(void)
{
    LCD_Inicialization();
    GPIO_Inicializace();
    I2C_Inicializace();
    // blikneme modrou jako priznak startu
    bliknutiLEDMODRA();
    printLCD(LCD_text);
    PCF8574_Init();
    bliknutiLEDZELENA();
    //cteni
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
    Wait_ms(1);
    hodnota = I2C_ReceiveData(I2C1);
    bliknutiLEDMODRA();
    LCD_Clear();
    itoa(hodnota,LCD_text);
    printLCD(LCD_text);
    Wait_ms(100);
    // Send STOP condition
    I2C_GenerateSTOP(I2C1, ENABLE);
    // blikneme zelenou jako priznak konce
    bliknutiLEDMODRA();

    while (1)
    {
        bliknutiLEDZELENA();
        PCF8574_Init();
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
        Wait_ms(1);
        hodnota = I2C_ReceiveData(I2C1);
        LCD_Clear();
        itoa(hodnota,LCD_text);
        printLCD(LCD_text);
        Wait_ms(100);
        I2C_GenerateSTOP(I2C1, ENABLE);
    }
}

void GPIO_Inicializace(void)
{
    STM32vldiscovery_LEDInit(LED3);
}

```

```

STM32vldiscovery_LEDInit(LED4);
STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
STM32vldiscovery_LEDOff(LED3);
STM32vldiscovery_LEDOff(LED4);
}

void bliknutiLEDMODRA(void)
{
    STM32vldiscovery_LEDOn(LED4); // zapnem LED4 - modra
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED4); // vypnem LED4 - modra
    Delay(0x5FFFF);
}

void bliknutiLEDZELENA(void)
{
    STM32vldiscovery_LEDOn(LED3); // zapnem LED3 - zelena
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED3); // vypnem LED3 - zelena
    Delay(0x5FFFF);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void I2C_Inicializace(void) //pro cteni
{
    GPIO_InitTypeDef GPIO_InitStructure;
    I2C_InitTypeDef I2C_InitStructure;
    I2C_DeInit(I2C1);

    /* I2C Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* GPIO Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    /* Configure I2C pins: SCL and SDA */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD; //GPIO_Mode_AF_PP
    GPIO_Init(GPIOB, &GPIO_InitStructure); //I2C PCF8574 pripojime k PB
    //SCL bude na PB6 DATA na PB7

    /* I2C configuration */
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = 0x41; //PCF8574 cteni
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = 10000;

    /* I2C Peripheral Enable */
    I2C_Cmd(I2C1, ENABLE);

```

```

/* Apply I2C configuration after enabling it */
I2C_Init(I2C1, &I2C_InitStructure);
}

void PCF8574_Init(void) //pro cteni
{
    Wait_ms(200);                // Pockat na nabež napeti

//Start komunikace s PCF8574
    I2C_GenerateSTART(I2C1, ENABLE);    // Generovat START

// Test on EV5 and clear it
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

// Send I2C address for write
    I2C_Send7bitAddress(I2C1, I2C_Adr,I2C_Direction_Receiver);

// Test on EV6 and clear it
    while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
}

void Wait_us(void)
{
vu32 nTick = 0xFFFF;
    for(; nTick != 0; nTick--);
}

void Wait_ms(u8 Time)
{
    if (Time > 0 )
        for(; Time != 0; Time--)
        {
            Wait_us();
        }
}

void itoa(uint16_t n, int8_t s[])
{
    int i, sign;
    if ((sign = n) < 0) /* record sign */
        n = -n;        /* make n positive */
    i = 0;
    do {                /* generate digits in reverse order */
        s[i++] = n % 10 + '0'; /* get next digit */
    } while ((n /= 10) > 0); /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

void reverse(int8_t s[])
{
    int i, j, k;
    char c;

```

```

    k = strlen(s)-1;
    for (i = 0, j = k; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}

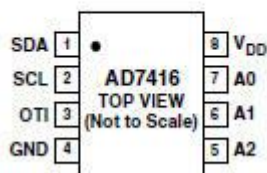
```

2.13.3 I2C – čidlo teploty TI AD7416 (program15)

Předchozí programy ve kterých jsme zapisovali do EEPROM AT24C02 a zapisovali a četli data z PCF8574 nám ukázaly, jak psát kód pro I2C komunikaci STM32 s obvody I2C. Z kódu pochopitelně odstraníme kód pro blikání zelené a modré LED, který nám pomáhal při odladování programu a sledování jeho chodu. Pro jiné obvody I2C pak ještě použijeme informace z datasheetu tohoto obvodu (staženého z webu výrobce) a dále změníme příslušně básovou adresu obvodu I2C. Při odladování můžeme opět použít kódy pro blikání LED. Po odladění programu pro obvod I2C pak kód pro blikání LED odstraníme, musíme mít ovšem na zřeteli, že pro některé obvody bude po odstranění kódu pro blikání nutné do těchto míst vložit nějaké zpoždění, aby obvod I2C správně fungoval.

Nicméně ukážeme si ještě kód pro práci s čidlem 10bitovým čidlem teploty AD7416. Jde o obvod od Analog Devices. Tyto obvody lze od AD získat jako free samples. Obvod v 8pinovém pouzdru je přímo předurčený pro konstrukce CanSATů, neboť měření teploty je při soutěžích CanSATů často požadováno (ESA CanSAT Competition). Popis tohoto obvodu najdeme v firemní dokumentaci. Pinově a funkčně kompatibilní teplotní čidla vyrábí i STMicroelectronics a to STTS75 a STDS75 s rozlišením 9bitů (default) až 12bitů (rozlišení můžeme volit, lepší rozlišení je však vykoupeno delší dobou měření teploty), STLM75 a STCM75 s rozlišením 9bitů v pouzdrech SO8. Do CanSATů doporučuji používat právě obvody od STMicroelectronics.

AD7416 PIN CONFIGURATION SOIC/MSOP



Pokud na A0,A1,A2 připojíme úroveň 0, bude bázová adresa tohoto obvodu pro zápis 0x90, pro čtení 0x91. Tento obvod, má stejnou sadu příkazů jako obvody AD7417 a AD7418, které mohou pracovat nejen jako čidla teploty, ale i jako 10bitové A/D převodníky. Proto nejprve musíme do těchto obvodů zapsat tzv. *Adress Pointer Register*. V případě, kdy požadujeme na obvodu, aby pracoval jako čidlo teploty bude tato hodnota 00000000.

Proto nejprve musí být na I2C sběrnici posloupnost následujících akcí: START, poslání 0x90 (protože budeme zapisovat), poslání 0x00 (Adres Pointer Register) a STOP. Poté již může následovat přečtení hodnoty naměřené teploty a to ve dvou bytech dat, tj. následující akce na sběrnici I2C: START, poslání 0x91 (protože budeme číst), přečtení horního byte dat (více významnější byte), přečtení dolního byte dat (méně významnější byte) a STOP. Tím jsme získali obsah Temperature Value Register

Table III. Temperature Value Register

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 |
|-----|-----|-----|-----|-----|-----|----|----|----|-----|
| MSB | B8 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | LSB |

Informaci o teplotě s přesností 0,25 ° C nese horních 10 bitů. Obsah dolních bitů D5, D4, ... D0 nemá žádný význam. Následující tabulka pak ukazuje převod mezi 10bitovým datovým údajem a hodnotou teploty v ° C.

Table IV. Temperature Data Format

| Temperature | Digital Output |
|-------------|----------------|
| -128°C | 10 0000 0000 |
| -125°C | 10 0000 1100 |
| -100°C | 10 0111 0000 |
| -75°C | 10 1101 0100 |
| -50°C | 11 0011 1000 |
| -25°C | 11 1001 1100 |
| -0.25°C | 11 1111 1111 |
| 0°C | 00 0000 0000 |
| +0.25°C | 00 0000 0001 |
| +10°C | 00 0010 1000 |
| +25°C | 00 0110 0100 |
| +50°C | 00 1100 1000 |
| +75°C | 01 0010 1100 |
| +100°C | 01 1001 0000 |
| +125°C | 01 1111 0100 |
| +127°C | 01 1111 1100 |

Nejvíce významný bit je znaménkový – 0 je nezáporná čísla, 1 pro záporná. Pro záporná čísla je další bity v doplňkovém kódu.

Pozn. Pinově i funkčně kompatibilní (včetně I2C komunikace) s obvodem AD7416 je obvod LM75 firmy National Semiconductor. Jediný rozdíl je v tom, že LM75 měří teplotu s přesností 0,5 ° C a informaci o teplotě nese 9 horních bitů Temperature Value Register.

1.12 TEMPERATURE REGISTER

(Read Only):

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-----|-------|-------|-------|-------|-------|-------|-------|-----|----|----|----|----|----|----|----|
| MSB | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | LSB | X | X | X | X | X | X | X |

D0–D6: Undefined

D7–D15: Temperature Data. One LSB = 0.5°C. Two's complement format.

Jak jsme se již zmínili, kompatibilní jsou i obvody STMicroelectronics STTS75, STDS75, STLM75 a STCM75.

Algoritmus přechtění dat o teplotě z AD7416 tedy bude

```
float ad7416_calculate()
{
int t1,t2;
float t;
i2c_start();
i2c_write( 0x90 ); // Address
i2c_write( 0x00 ); // Address pointer register
i2c_stop();

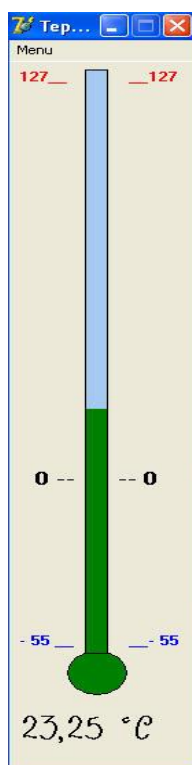
i2c_start();
i2c_write( 0x91 );
t1 = (int)i2c_read(0); // horních osm bitů Temperature registru
t2 = (int)i2c_read(1); // dolních osm bitů Temperature registru
i2c_stop();
t = (float)( ( t2>>6)+(t1<<2) ) * 0.25 );
return t;
}
```

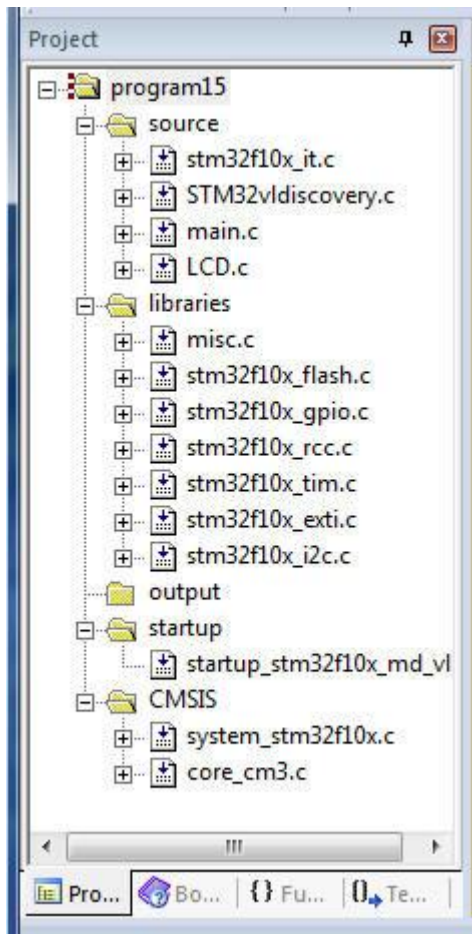
Neboli horních 8bitů posuneme o dva bity doleva, čímž dostaneme 10bitové číslo s dolními dvěma bity nulovými. Dolních 8 bitů posuneme o 6 bitů doprava. Tím přijdeme o 6 dolních (nevýznamných) bitů z t2. Zůstanou nám tak 2 významné bity z t2. Po součtu již dostaneme v 10ti bitový údaj o teplotě, kde jeden bit odpovídá 0,25 ° C, tak že po vynásobení 0,25 dostaneme v t hodnotu teploty ve stupních C.

Pozn.: Pro teploty větší nebo rovné nule obsahuje t1 rovnou údaj o teplotě s přesností 1 °C.

K otestování AD7426 a dalších obdobných obvodů (STMicroelectronics) můžeme použít PC se sériovým portem či s USB a převodníkem USB/COM (a tudíž s virtuálním sériovým portem) a zapojit ho pomocí stejného přípravku, který jsme použili pro I2C čtení obsahu EEPROM programem PonyProg2000, nebo můžeme použít přípravek z <http://www.amaterskaelektronika.cz/file/hw-prevodnik-i2c-rs232>

V obou případech můžeme na PC spustit free program z <http://www.amaterskaelektronika.cz/file/teplomer-lm75a-delphi-i2c>





```

/*
 * I2C komunikace s      AD7416 a obdobnými čidly teploty
 * ctení dat z AD7416
 * příklad 15          main.c
 * Vd pro SPSE Jecna Lectures
 *
 */
#include <stdint.h>
#include "stm32f10x.h"
#include "stm32f10x_i2c.h"
#include "STM32vldiscovery.h"
#include "LCD.h"

uint8_t LCD_text[] = "Test AD7416.";

/* Private typedef -----*/
GPIO_InitTypeDef  GPIO_InitStructure;
I2C_InitTypeDef  I2C_InitStructure;
/* Private define -----*/

#define I2C_Adr      0x91          // Adresa AD7416

```

```
//SCL bude na PB6   DATA na PB7

//Tabulka I2C slave adres          (A0=0 je write do obvodu, A1=1 je read )
//Typ obvodu                       A7      A6      A5      A4      A3      A2      A1
//8bitovy vstup/vystup             0      1      0      0      a      a      a      napr.
PCF8574  0x40 zapis 0x41 cteni
//EEPROM                           1      0      1      0      a      a      a      napr.
PCF8582
//Hodiny/kalendar                  1      0      1      0      0      0      a      napr.
PCF8583
//8bitovy A/D a D/A                1      0      0      1      a      a      a      napr.
PCF8591  nebo AD7416
//PLL                              1      1      0      0      0      a      a      napr.
TSA5055

/* Private variables -----*/
uint8_t hodnota1;
uint8_t hodnota2;

void itoa(uint16_t n, int8_t s[]);
void reverse(int8_t s[]);

void Wait_ms(u8 Time);
void Wait_us(void);
void Delay(__IO uint32_t nTick);
void GPIO_Inicializace(void);           // nastavení vstupne/vystupnich
pinu na kitu
void I2C_Inicializace_R(void);         // nastavení I2C pro cteni
void AD7416_Init_R(void);
void I2C_Inicializace_W(void);         // nastavení I2C pro zapis
void AD7416_Init_W(void);
void bliknutiLEDMODRA(void);          //
void bliknutiLEDZELENA(void);        //

int main(void)
{
    LCD_Inicialization();
    GPIO_Inicializace();
    printLCD(LCD_text);

    // blikneme modrou jako priznak startu
    bliknutiLEDMODRA();
    //zapis do registru AD7416

    I2C_Inicializace_W();
    Wait_ms(2);
    bliknutiLEDZELENA();
    AD7416_Init_W();
    Wait_ms(2);
    bliknutiLEDZELENA();
    I2C_SendData(I2C1, 0x00);
    bliknutiLEDZELENA();
}
```

```

while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
Wait_ms(2);
bliknutiLEDZELENA();
I2C_GenerateSTOP(I2C1, ENABLE);
bliknutiLEDZELENA();
// Delay(0xAFFFFFFF);

//cteni
I2C_Inicializace_R();
Wait_ms(2);
    bliknutiLEDZELENA();
AD7416_Init_R();
Wait_ms(2);
    bliknutiLEDZELENA();
//cteni data0
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
Wait_ms(2);
    bliknutiLEDZELENA();
hodnota1 = I2C_ReceiveData(I2C1);
    bliknutiLEDZELENA();
    LCD_Clear();

    itoa(hodnota1,LCD_text);
    bliknutiLEDMODRA();
    printLCD(LCD_text);
    Wait_ms(5000);

    //cteni data1
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
Wait_ms(1);
    bliknutiLEDZELENA();
hodnota2 = I2C_ReceiveData(I2C1);
    bliknutiLEDZELENA();
    LCD_Clear();
    itoa(hodnota2,LCD_text);
    printLCD(LCD_text);
    Wait_ms(1);
// Send STOP condition
I2C_GenerateSTOP(I2C1, ENABLE);
// blikneme zelenou jako priznak konce
    bliknutiLEDMODRA();

while (1)
{
    bliknutiLEDZELENA();
    //cteni
    I2C_Inicializace_R();
    bliknutiLEDZELENA();
    Wait_ms(2);
    AD7416_Init_R();
        bliknutiLEDZELENA();
    Wait_ms(2);
//cteni data0
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
        Wait_ms(2);

```

```

    bliknutiLEDZELENA();
    hodnota1 = I2C_ReceiveData(I2C1);
    LCD_Clear();
    itoa(hodnota1, LCD_text);
    printLCD(LCD_text);
    Wait_ms(500);

    //cteni data1
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
    Wait_ms(1);
    bliknutiLEDZELENA();
    hodnota2 = I2C_ReceiveData(I2C1);
    bliknutiLEDZELENA();
    Wait_ms(1);
    // Send STOP condition
    I2C_GenerateSTOP(I2C1, ENABLE);
    bliknutiLEDZELENA();
}
}

void GPIO_Inicializace(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void bliknutiLEDMODRA(void)
{
    STM32vldiscovery_LEDOn(LED4); // zapnem LED4 - modra
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED4); // vypnem LED4 - modra
    Delay(0x5FFFF);
}

void bliknutiLEDZELENA(void)
{
    STM32vldiscovery_LEDOn(LED3); // zapnem LED3 - zelena
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED3); // vypnem LED3 - zelena
    Delay(0x5FFFF);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

```

void I2C_Inicializace_R(void) //pro cteni
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    //I2C_InitTypeDef  I2C_InitStructure;
    I2C_DeInit(I2C1);

    /* I2C Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* GPIO Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    /* Configure I2C pins: SCL and SDA */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD; //GPIO_Mode_AF_PP
    GPIO_Init(GPIOB, &GPIO_InitStructure); //I2C PCF8574 pripojime k PB
    //SCL bude na PB6  DATA na PB7

    /* I2C configuration */
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = 0x91; //AD7416 cteni
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = 10000;

    /* I2C Peripheral Enable */
    I2C_Cmd(I2C1, ENABLE);
    /* Apply I2C configuration after enabling it */
    I2C_Init(I2C1, &I2C_InitStructure);
}

void I2C_Inicializace_W(void) //pro zapis
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    //I2C_InitTypeDef  I2C_InitStructure;
    I2C_DeInit(I2C1);

    /* I2C Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* GPIO Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    /* Configure I2C pins: SCL and SDA */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD; //GPIO_Mode_AF_PP
    GPIO_Init(GPIOB, &GPIO_InitStructure); //I2C AD7416 pripojime k PB
    //SCL bude na PB6  DATA na PB7

    /* I2C configuration */

```



```

I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStructure.I2C_OwnAddress1 = 0x90; //AD7416 zapis
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_InitStructure.I2C_ClockSpeed = 10000;

/* I2C Peripheral Enable */
I2C_Cmd(I2C1, ENABLE);
/* Apply I2C configuration after enabling it */
I2C_Init(I2C1, &I2C_InitStructure);
}

void AD7416_Init_W(void) //pro zapis
{
    Wait_ms(200); // Pockat na nabež napeti

//Start komunikace s PCF8574
    I2C_GenerateSTART(I2C1, ENABLE); // Generovat START

// Test on EV5 and clear it
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

// Send I2C address for write
    I2C_Send7bitAddress(I2C1, I2C_Adr, I2C_Direction_Transmitter);

// Test on EV6 and clear it
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
}

void AD7416_Init_R(void) //pro cteni
{
    Wait_ms(200); // Pockat na nabež napeti

//Start komunikace s AD7416
    I2C_GenerateSTART(I2C1, ENABLE); // Generovat START

// Test on EV5 and clear it
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

// Send I2C address for write
    I2C_Send7bitAddress(I2C1, I2C_Adr, I2C_Direction_Receiver);

// Test on EV6 and clear it
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
}

void Wait_us(void)
{
    vu32 nTick = 0xFFFF;
    for(; nTick != 0; nTick--);
}

```

```

/*****
* Function Name      : Wait_ms
* Description        : Program pro zpozdění
* Input              : Time = číslo, které se odcítá k NULĚ
* Output             : None
* Return             : None
*****/
void Wait_ms(u8 Time)
{
    if (Time > 0 )
        for(; Time != 0; Time--)
        {
            Wait_us();
        }
}

void itoa(uint16_t n, int8_t s[])
{
    int i, sign;
    if ((sign = n) < 0) /* record sign */
        n = -n;        /* make n positive */
    i = 0;
    do {                /* generate digits in reverse order */
        s[i++] = n % 10 + '0'; /* get next digit */
    } while ((n /= 10) > 0); /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

void reverse(int8_t s[])
{
    int i, j, k;
    char c;
    k = strlen(s)-1;
    for (i = 0, j = k; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

void __assert_func(const char *file, int line, const char *func, const char
*failedexpr)
{
    while(1)
    {}
}

void __assert(const char *file, int line, const char *failedexpr)
{
    __assert_func (file, line, NULL, failedexpr);
}

```

}

Nejprve se zobrazily čísla 24 a 192, poté ve smyčce občas 24, občas 23 – ve smyčce zobrazujeme jen s přesností na 1 ° C. V programu pro jednoduchost neprovádíme výpočet teploty a neuvažujeme teploty pod nulou (pro CanSAT v Norsku budeme muset uvažovat i nízké teploty). Před smyčkou jsme zobrazili i hodnotu2 což bylo 192 tj 11000000. Těch dolních 6 nul je nevýznamných, neboli význam má jen 11 což odpovídá 0,75 °C, takže teplota naměřená AD7416 byla 24,75 °C.

Program ukazuje jen principy, tj není v něm prováděn výpočet teploty, obsahuje kontrolní místa s blikajícími LED. Při použití STM obvodů popř. bude třeba odprogramovat nastavení na větší přesnost měření teploty. Podle použitých obvodů popř. po odstranění kódů pro blikání LED umístí potřebné Delay.

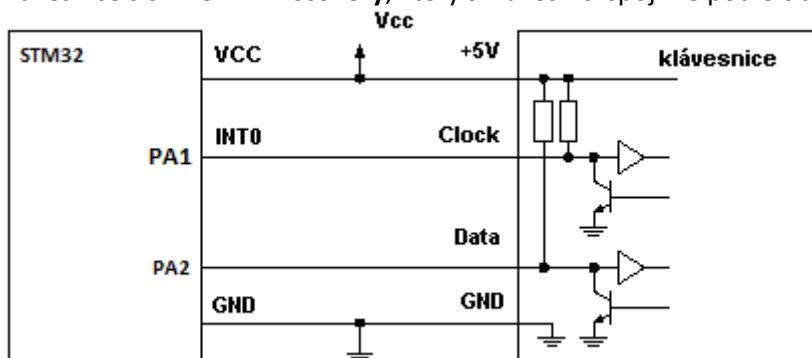
Dále bude třeba upravit kódy typu

```
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
```



neboť v této smyčce může program uvíznout např. v případě poruchy obvodu či I2C komunikaci. Stačí např. „studený spoj“ u pull odporu 4k7, kdy bez pull odporů na SDA a SCL nefunguje I2C komunikace.

2.14 Klávesnice PS2 (program16)

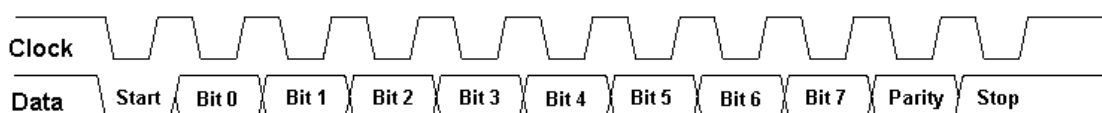
V případě, kdy potřebujeme klávesnici s větším počtem kláves může být výhodné použít běžnou klávesnici používanou u počítačů PC. Důsledkem jejího rozšíření je nízká cena a snadnost jejího připojení k mikropočítači či mikrořadiči. PC AT klávesnice tvoří samostatný konstrukční celek, obsahující obvykle 102 kláves a mikrořadič, který vykonává řadu funkcí. Tou hlavní je neustálé sledování stavu matice spínačů. V případě jakékoli změny zjistí mikrořadič souřadnice spínače, který ji způsobil, vybere v této situaci odpovídající SCAN kód, a ten po sériové lince DATA vyšle směrem k zařízení, s nímž klávesnice komunikuje. Je-li některá klávesa stisknuta déle než 0,5 s, vysílá její kód automaticky znovu. Kromě vlastních dat vysílá klávesnice při stisku klávesy i hodinový signál CLK o frekvenci 10 až 20 kHz.. Při sestupné hraně hodinového signálu CLK jsou vysílána DATA platná. Toho můžeme využít při komunikaci klávesnice s nějakým systémem jako PC, mikrořadič STM32 apod. Tuto komunikaci včetně obslužného programu napsaného v jazyce C si ukážeme na komunikaci PC klávesnice s **STM32VL Discovery**, který s klávesnicí spojíme podle obrázku:



Signál CLK z klávesnice přitom bude vyvolávat vnější přerušení mikrořadiče STM32. Přerušení vyvolané sestupnou hranou hodinového signálu CLK bude obsluhováno funkcí, která která bude snímat data vysílaná klávesnicí. Klávesnice bude k mikrořadiči připojena přes její konektor. Zapojení konektoru PC AT klávesnice popisuje následující tabulka:

| klávesnice PC AT |  |  |
|------------------|---|---|
| Signál | DIN41524 ,zásuvka 5-pin | p-pin Mini DIN PS2 |
| Clock | 1 | 5 |
| Data | 2 | 1 |
| nezapojen | 3 | 2 , 6 |
| GND | 4 | 3 |
| +5V | 5 | 4 |
| stínění | stínění | stínění |

Klávesnice vysílá data sériově vždy po 11 bitech. Nejprve je vysílán Start bit (logická nula), poté 8 datových bitů (první je LSB, poslední MSB), dále parita a nakonec Stop bit (logická jednička) :



Při obsluze přerušení se při prvním přerušení vynuluje proměnná nesoucí informaci o počtu přijatých bitů, při každém dalším přerušení se tato proměnná inkrementuje. Obsah bitů 1 až 8 (bit 0 byl Start bit) datového vodiče se načítá do jednoho znaku, bit 9 a 10 se ignorují. Poté se provádí dekódování přijatého znaku. Toto řešení s použitím přerušení se dá použít pro jakékoli jednočipové procesory a jeho výhodou je minimální režie, kterou si obsluha klávesnice z celého systému zabere.

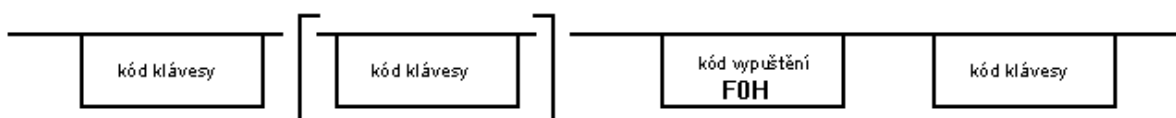
Minimální mezera mezi dvěma kódy vyslanými po sobě je 1,2 ms.

Tato synchronní sériová komunikace je obousměrná, my však budeme klávesnici používat jen ve směru klávesnice – mikrořadič. Kromě kódů kláves vysílá klávesnice i řídicí signály:

- FFh přetečení bufferu, klávesnice detekuje chybu
- FEh žádost o zaslání posledního znaku, špatně přijatý znak, parita
- FAh potvrzení – ACK
- F0h kód uvolnění klávesy
- AAh úspěšný power-on test

EEh echo – klávesnice odpoví zpět také EEh jako echo – pro test
 00h přetečení bufferu, klávesnice detekuje chybu

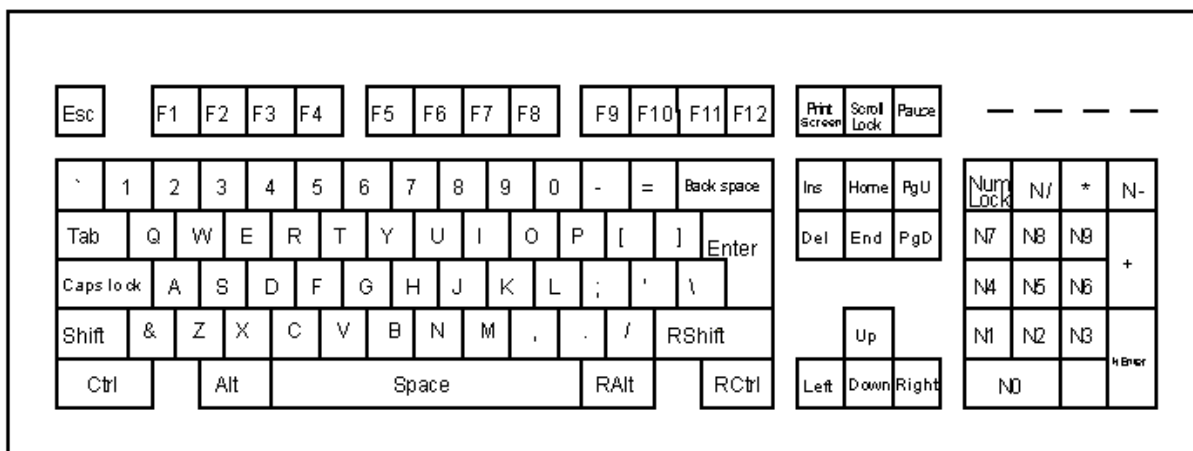
Z hlediska kódování kláves klávesnicí je možné rozdělit klávesy do tří skupin. Jsou to skupina základní, skupina rozšířená a skupina speciální. Do *základní skupiny* patří 83 kláves. Po stisku některé z těchto kláves je vyslán kód této klávesy a po uvolnění této klávesy je vyslán kód uvolnění klávesy a znovu kód uvolněné klávesy. Při delším stisku je kód neustále vyslán až do uvolnění. Kód klávesy v hranatých závorkách (na následujícím obrázku) se vysílá dokud je klávesa stisknuta:



Rozšířená skupina zahrnuje 14 kláves. Tyto klávesy mají kódy shodné s klávesami numerické klávesnice, patřících do základní skupiny, ale před tímto kódem je předřazen kód F8, který je od těchto kláves odlišuje. Dva kódy v hranatých závorkách (na následujícím obrázku) se vysílají dokud je klávesa stisknuta:



Speciální skupina zahrnuje všechny ostatní klávesy. Jedná se o klávesy Pause a Print Screen. Při současném stisku dvou a více ostatních kláves (základní a rozšířená skupina) jsou kódy vysílány v odpovídajícím pořadí v jakém se jednotlivé akce stisku a uvolnění stanou a tyto kódy se vzájemně neovlivňují, kromě případu kdy je přerušeno cyklické vysílání kódu klávesy – autorepeat. Rozmístění všech kláves a názvy jejich kódů jsou zřejmé z obr:



Kódy kláves jsou uvedeny v následujících tabulkách:

CSAN kódy kláves základní skupiny:

| Klávesa | kód | Klávesa | kód | Klávesa | kód |
|-----------|-----|---------|-----|-------------|-----|
| ` | 8F | S | 27 | F1 | 5F |
| 1 | 97 | D | 3B | F2 | 9F |
| 2 | 87 | F | 2B | F3 | DF |
| 3 | 9B | G | D3 | F4 | CF |
| 4 | 5B | H | 33 | F5 | 3F |
| 5 | 8B | J | 23 | F6 | 2F |
| 6 | 93 | K | BD | F7 | 3E |
| 7 | 43 | L | 2D | F8 | AF |
| 8 | 83 | ; | CD | F9 | 7F |
| 9 | 9D | , | B5 | F10 | 6F |
| 0 | 5D | Shift | B7 | F11 | E1 |
| - | 8D | Z | A7 | F12 | 1F |
| = | 55 | X | BB | Scroll lock | 81 |
| Backspace | 99 | C | 7B | Num lock | 11 |
| Tab | 4F | V | AB | * | C1 |
| Q | 57 | B | B3 | N- | 21 |
| W | 47 | N | 73 | + | 61 |
| E | DB | M | A3 | N0 | F1 |
| R | 4B | , | 7D | N1 | 69 |
| T | CB | . | 6D | N2 | B1 |
| Y | 53 | / | AD | N3 | A1 |
| U | C3 | P Shift | 65 | N4 | 29 |
| I | 3D | \ | 79 | N5 | 31 |

| | | | | | |
|-----------|----|-------|----|----|----|
| O | DD | Ctrl | D7 | N6 | D1 |
| P | 4D | Alt | 77 | N7 | C9 |
| [| D5 | Space | 6B | N8 | 51 |
|] | 25 | Enter | A5 | N9 | 41 |
| Caps lock | E5 | Esc | 91 | N. | 71 |
| A | C7 | | | | |

SCAN kódy kláves rozšířené skupiny:

| Klávesa | kód | zkratka |
|---------|-----|---------|
| R alt | 77 | Alt |
| R ctrl | D7 | Ctrl |
| Ins | F1 | N0 |
| Home | C9 | N7 |
| PgUp | 41 | N9 |
| Del | 71 | N5 |
| End | 69 | N1 |
| PgDw | A1 | N3 |
| Left | 29 | N4 |
| Right | D1 | N6 |
| Up | 51 | N8 |
| Down | B1 | N2 |
| / | AD | /? |
| enter | A5 | enter |

Speciální skupina:

| Klávesa | kód |
|--------------|----------------------|
| Print Screen | F8 B7 F8 C1 |
| Break | 78 D7 11 F0 D7 F0 11 |

Náš program provádí na LCD výpis znaků odpovídajících stisknuté klávese na běžné PS2 klávesnici připojené svým hodinovým výstupem na PA1 a datovým výstupem na PA2. PS2 klávesnici nebudeme pochopitelně používat jako periférii letícího CanSATu. Můžeme ji však používat jako vstup dat při pozemním testování CanSATu. Hlavním důvodem uvedení tohoto programu je to, že je ideální ukázkou použití vnějšího přerušení.

Klávesnice totiž vysílá dva sériové signály – datový a hodinový. Hodinový signál máme přiveden na **PA1** a náběžnou hranou tohoto signálu se vyvolá externí přerušení, jehož obsluhou je funkce EXT11_IRQHandler. Větší část kódu PS2.c a PS2.h jsem převzal z 17. dílu seriálu z MCU serveru. Bylo přitom nutné přejmenovat jméno funkce obsluhy přerušení a místo **void Preruseni_PS2(void)** použít **void EXT11_IRQHandler(void)**.

Zdrojové kódy:

```

/*
 * ps2.h
 *
 *
 * Author: User
 */

#ifndef PS2_H_
#define PS2_H_

/* Includes -----*/
#include "stm32f10x.h"

/** Exported Types */
/** Exported Functions */
void PS2_Inicializace(void);
//void Preruseni_PS2(void);
void EXT11_IRQHandler(void);
uint8_t keyboard_getchar(void);
void dekoduj_klavesu(uint8_t scan);
void put_to_buff(uint8_t ch);
void Kontrola_hodin(void);

/** Exported Defines */
#define PS2_PORT GPIOA
#define PS2_CLOCK_PIN GPIO_Pin_1
#define PS2_DATA_PIN GPIO_Pin_2
#define PS2_BUFSIZE 16
#endif /* PS2_H_ */

```



```

/*
 * main.h
 *
 *
 *
 */

#ifndef MAIN_H
#define MAIN_H

/* Includes -----
---*/
#include "stm32f10x.h"
#include "STM32vldiscovery.h"

void ZmenaCasu(void);
void Kontrola_hodin(void);
uint32_t Cas(void);

#endif /* MAIN_H */

```

```

/*
 * ps2.c
 * STM32 VL Discovery kit Keyborad interface
 * Vd pro SPSE Jecna lectures
 *
 *
 */

/* Includes -----*/
#include "ps2.h"
#include "main.h"

static uint8_t cislo_bitu = 11; // cislo prijimaneho bitu z klavesnice
static uint32_t casPS2; // okamzik posledniho preruseni hodinami
z klavesnice
static uint8_t znak; // bude obsahovat scan kod

volatile uint8_t buffer_pointer = 0;
volatile uint8_t buffer[PS2_BUFSIZE];
volatile uint16_t status = 0;

// Lookup tabulka pro scankody normalniho stisku
const uint8_t obyc[] = {
    0x0d, 9,
    0x0e, '|',

```

```

0x15, 'q',
0x16, '1',
0x1a, 'z',
0x1b, 's',
0x1c, 'a',
0x1d, 'w',
0x1e, '2',
0x21, 'c',
0x22, 'x',
0x23, 'd',
0x24, 'e',
0x25, '4',
0x26, '3',
0x29, ' ',
0x2a, 'v',
0x2b, 'f',
0x2c, 't',
0x2d, 'r',
0x2e, '5',
0x31, 'n',
0x32, 'b',
0x33, 'h',
0x34, 'g',
0x35, 'y',
0x36, '6',
0x39, ', ',
0x3a, 'm',
0x3b, 'j',
0x3c, 'u',
0x3d, '7',
0x3e, '8',
0x41, ', ',
0x42, 'k',
0x43, 'i',
0x44, 'o',
0x45, '0',
0x46, '9',
0x49, '.',
0x4a, '-',
0x4b, 'l',
0x4c, 'ř',
0x4d, 'p',
0x4e, '+',
0x55, '\\ ',
0x5a, 13,
0x5b, '"',
0x5d, '\\ ',
0x61, '<',
0x66, 8,
0x69, '1',
0x6b, '4',
0x6c, '7',
0x70, '0',
0x71, ', ',
0x72, '2',

```

```

0x73, '5',
0x74, '6',
0x75, '8',
0x79, '+',
0x7a, '3',
0x7b, '-',
0x7c, '*',
0x7d, '9',
0, 0
};

// Lookup tabulka pro scankody stisku se shiftem
const uint8_t seshiftem[] = {
    0x0d, 9,
    0x0e, 'Š',
    0x15, 'Q',
    0x16, '!',
    0x1a, 'Z',
    0x1b, 'S',
    0x1c, 'A',
    0x1d, 'W',
    0x1e, '"',
    0x21, 'C',
    0x22, 'X',
    0x23, 'D',
    0x24, 'E',
    0x25, '¤',
    0x26, '#',
    0x29, ' ',
    0x2a, 'V',
    0x2b, 'F',
    0x2c, 'T',
    0x2d, 'R',
    0x2e, '%',
    0x31, 'N',
    0x32, 'B',
    0x33, 'H',
    0x34, 'G',
    0x35, 'Y',
    0x36, '&',
    0x39, 'L',
    0x3a, 'M',
    0x3b, 'J',
    0x3c, 'U',
    0x3d, '/',
    0x3e, '(',
    0x41, ';',
    0x42, 'K',
    0x43, 'I',
    0x44, 'O',
    0x45, '=',
    0x46, ')',
    0x49, ':',
    0x4a, '_',
    0x4b, 'L',

```

```

0x4d, 'P',
0x4e, '?',
0x55, `',
0x5a, 13,
0x5b, '^',
0x5d, '*',
0x61, '>',
0x66, 8,
0x69, '1',
0x6b, '4',
0x6c, '7',
0x70, '0',
0x71, ',',
0x72, '2',
0x73, '5',
0x74, '6',
0x75, '8',
0x79, '+',
0x7a, '3',
0x7b, '-',
0x7c, '*',
0x7d, '9',
0, 0
};

void PS2_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    GPIO_InitStructure.GPIO_Pin = PS2_CLOCK_PIN | PS2_DATA_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(PS2_PORT, &GPIO_InitStructure);

    EXTI_StructInit(&EXTI_InitStructure);
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_Line = EXTI_Line1;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Clear SC EXTI Line Pending Bit */
    EXTI_ClearITPendingBit(EXTI1_IRQn);

    /* Configure the NVIC Preemption Priority Bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    /* Clear the SC_EXTI IRQ Pending Bit */
    NVIC_ClearPendingIRQ(EXTI1_IRQn);

    NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn; //preruseni bude
    // od hrany CLK z PS2 na PA1
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;

```

```

NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
}

void EXTI1_IRQHandler(void)           //obsluha preruseni od PA1
//void Preruseni_PS2(void)
{

uint8_t pom;

if(EXTI_GetITStatus(EXTI_Line1) != RESET)
{
    casPS2 = Cas();    // aktualizujeme cas posledniho preruseni

    pom = GPIO_ReadInputDataBit(PS2_PORT, PS2_DATA_PIN);
    if (cislo_bitu==11)           // prvni musi byt start bit
    {
        znak = 0x00;
        if (0 == pom)
        {
            --cislo_bitu;
        }
    } else
    if (cislo_bitu<11 && cislo_bitu>2)
    { // jde o datovy bit, musime ho zpracovat
        znak = (znak >> 1);    // ochranime si predchozi hodnotu
        if (0 != pom)
            znak = znak + 0x80;    // vlozime 8. bit
rovny 1!
        --cislo_bitu;
    } else
    { // muse jit o bit paritni nebo stop, ty nezpracovavame
        if (--cislo_bitu == 0)
        {
            // mame vsechno nactene, musime dekodovat klavesu
            dekoduj_klavesu(znak);
            cislo_bitu = 11;    // nyni prijde start bit
        }
    }
    /* Clear the Key Button EXTI line pending bit */
    EXTI_ClearITPendingBit(EXTI_Line1);
}
}

/**
 * @brief Dekodovani scan kodu klavesy na prislusny znak
 * @param Scan kod klavesy
 * @retval : None
 */
void dekoduj_klavesu(uint8_t scan)
{
    static uint8_t is_up=0, shift = 0;

```

```

uint8_t i;

if (!is_up) // posledni prijaty scan byl indetifikator up-key
{
    switch (scan)
    {
        case 0xF0 : // up-key
            is_up = 1;
            break;

        case 0x12 : // Left SHIFT
            shift = 1;
            break;

        case 0x59 : // Right SHIFT
            shift = 1;
            break;

        /* na Caps lock, Levy Atl, Pravy Alt muzete doplnit kód
        * podle programu na http://eugenemcu.ru/publ/13-1-0-75 */

        default:
            if(!shift) // kdyz neni shift
            {
                i = 0;
                for(i = 0; obyc[i]!=scan && obyc[i]!=0; i+=2);
                if (obyc[i] == scan) {
                    put_to_buff(obyc[i+1]);
                }
            }
            } else { // se shiftem
                for(i = 0; seshiftem[i]!=scan && seshiftem[i]!=0;
i+=2);

                if (seshiftem[i] == scan) {
                    put_to_buff(seshiftem[i+1]);
                }
            }
            break;
        }
    }
} else {
    is_up = 0;
    // Manual pravi ze: Two 0xF0 in a row not allowed!!

    switch (scan)
    {
        case 0x12 : // Left SHIFT
            shift = 0;
            break;

        case 0x59 : // Right SHIFT
            shift = 0;
            break;
    }
}
}

```

```

/**
 * @brief Vložení dekodovaného znaku do bufferu
 * @param Scan kod klavesy
 * @retval : None
 */
void put_to_buff(uint8_t ch)
{
    if (buffer_pointer < PS2_BUFSIZE) // jeste tam mame misto
    {
        *buffer = ch; // soupneme to tam
        buffer_pointer++; // posuneme ukazatel
    }
}

/**
 * @brief Cteni z klavesnice
 * @param None
 * @retval : nacteny znak nebo 0!
 */
uint8_t keyboard_getchar(void)
{
    uint8_t byte = 0;

    if (buffer_pointer > 0) // neco tam mame
    {
        byte = *buffer; // vytahneme to
        buffer_pointer--; // posuneme ukazatel
    }

    return byte;
}

/**
 * @brief Zkontrolujeme zda se nezasekla klavesnice
 * @param None
 * @retval : None
 */
void kontrola_hodin(void)
{
    __IO uint32_t delta;
    if (cislo_bitu != 11)
    {
        delta = Cas(); // zatim jen nacteme cas
        if (delta < casPS2)
        { // citac pretekl
            delta = (0xFFFFFFFF - casPS2) + delta;
        } else
        { // citac nepretekl
            delta = delta - casPS2;
        }
        if (delta > 2) // uz 2 ms se neposlal dalsi bit!!
        {
            cislo_bitu = 11; // zkusime cekat na nový znak
        }
    }
}

```

```

    }
}
}

```

A testovací main.c

```

/* STM32 VL Discovery Library
 * priklad 16 - PS2 klavesnice ... PS2_clk PA1      PS2_data PA2
 * ver 1.0 for SPSE Jecna lectures - ukazka obsluhy preruseni
 * Created Vd on: Jul 20, 2011
 */
#define PS2_PORT          GPIOA
#define PS2_CLOCK_PIN    GPIO_Pin_1
#define PS2_DATA_PIN     GPIO_Pin_2

/* Includes -----*/
#include <stdint.h>
#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include "LCD.h"
#include "PS2.h"

__IO uint8_t c;
uint8_t LCD_Text[] = "Test PS2 klavesnice.";
static uint32_t CasTick = 0;      // centralni casove jednotky

/* Private function prototypes -----*/

void GPIO_Inicialization(void);
void Delay(__IO uint32_t nTick);
void bliknutiLEDMODRA(void);    //
void bliknutiLEDZELENA(void);  //

//void PinInputZ_b (u32 *adr, u8 bit);

int main(void)
{
    /* Nastaveni casovace SysTick na 1 msec preruseni */
    if (SysTick_Config(SystemCoreClock / 1000))
    {
        /* Capture error */
        while (1);
    }

    LCD_Inicialization();
    GPIO_Inicialization();
    PS2_Inicializace();
    bliknutiLEDMODRA();
    printLCD(LCD_Text);
    Delay(0xAFFFF);
    LCD_Clear();
    while (1)

```



```

{
    bliknutiLEDZELENA();
    c = keyboard_getchar();
    if(c != 0 )
    {
        if(c != 13 )
        {
            SendCharLCD(c);
        } else
            LCD_Clear();           // po entru vymazeme LCD
        }
        Kontrola_hodin();         // kontrola na chybu klavesnice
    }
}

void GPIO_Inicialization(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void bliknutiLEDMODRA(void)
{
    STM32vldiscovery_LEDOn(LED4); // zapnem LED4 - modra
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED4); // vypnem LED4 - modra
    Delay(0x5FFFF);
}

void bliknutiLEDZELENA(void)
{
    STM32vldiscovery_LEDOn(LED3); // zapnem LED3 - zelena
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED3); // vypnem LED3 - zelena
    Delay(0x5FFFF);
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void ZmenaCasu(void)
{
    CasTick++;
}

uint32_t Cas(void)
{
    return CasTick;
}

```

}

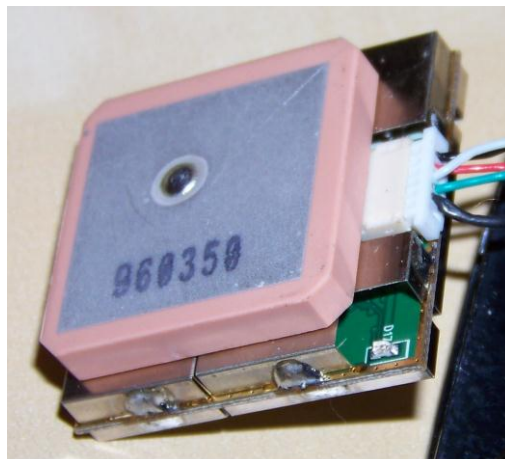
2.15 GPS

GPS moduly, které lze v současné době zakoupit jsou převážně moduly, které se připojují k PC pomocí USB. Před několika lety bylo možné zakoupit takové moduly připojitelné k PC přes COM1 či COM2 se signály dle RS232. Vzhledem k tomu, že nové PC již obvykle nejsou rozhraním COM vybaveny, je obtížné získat GPS modul připojitelný k PC přes sériový port. Bohužel právě takový GPS potřebujeme pro náš CanSAT.

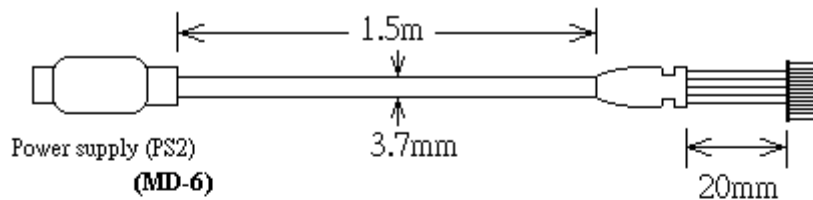
Je tu ještě jedna možnost a to GPS modul připojitelný k PDA. Tuto možnost jsme využili. Následující příklady budou využívat právě takový GPS modul, v našem případě jsem použil Navilock NL-303P GPS PDA Receiver SiRF III.



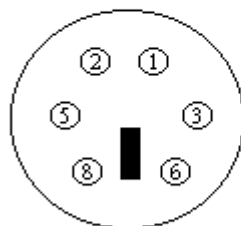
Pro použití v CanSatu ovšem nebudeme potřebovat ani kryt, ani těžký magnet.



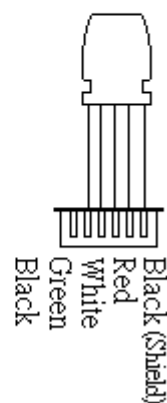
Modul je napájen 5V a zapojení přívodního kabelu s konektory ukazuje obr.



(MD-6) Male-type



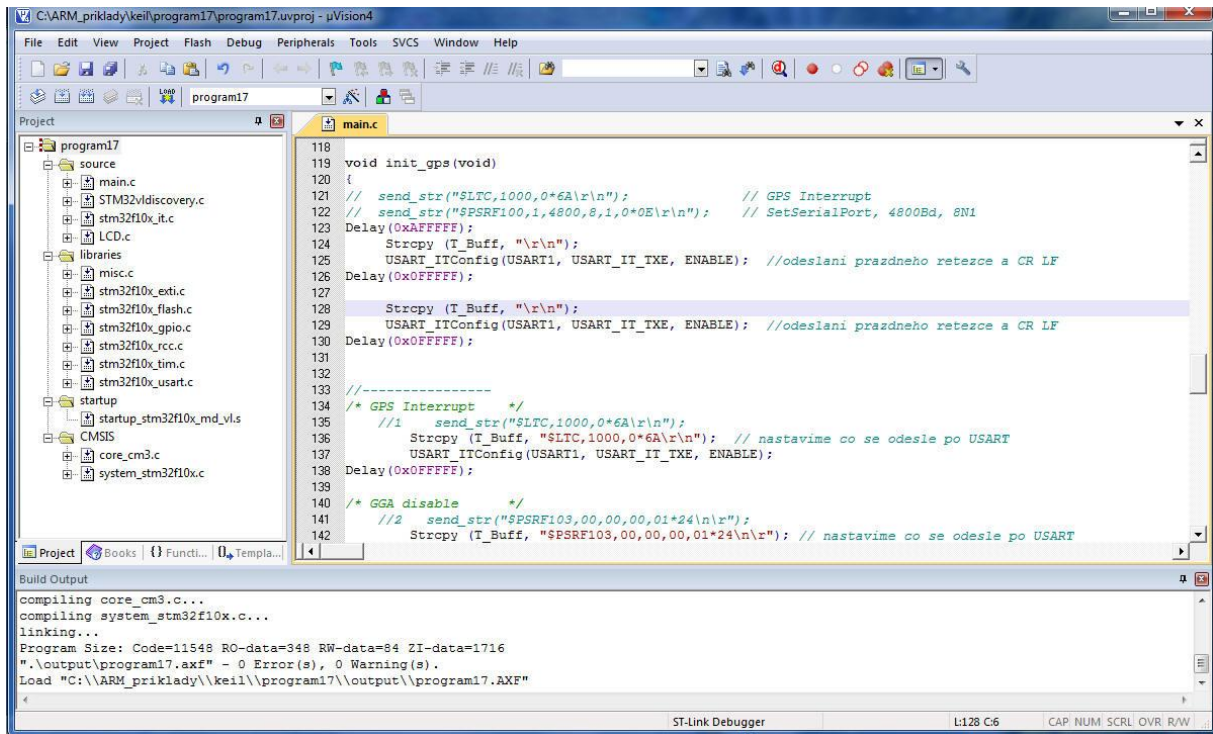
PIN 1 : Black (GND)
PIN 2 : Red (VCC)
PIN 5 : White (RX)
PIN 6 : Green (TX)



Pro účely testování a vývoje zatím ponecháme modul v původním stavu a připojíme se k němu pomocí PS2 konektoru. Jak vidíme, je tento konektor zapojen jinak, než PS2 konektory v PC a nesmíme ho do PC přes PS2 připojovat. Má navíc jiné úrovně signálů, než klávesnice či myš připojované přes PS2 k PC.

Signály RX a TX mají úroveň RS232 a k našemu STM32 ho připojíme přes MAX3232. Můžeme ho připojit i k PC, pokud má náš počítač ještě COM rozhraní. K tomuto účelu si zhotovíme redukci a nezapomeneme připojit 5V napájení pro GPS modul. Pak si jeho funkci můžeme ověřit např. programem SiRFDemo.

Pozn. Při realizaci výše uvedených propojení však nic nefungovalo. Teprve zobrazením signálu TX z GPS modulu se ukázalo, že se tento signál po připojení k MAX3232 či COM portu PC zkrusí natolik, že přijímaný signál je interpretován jako odlišné znaky. Pomohlo vřazení odporu 220 Ω mezi TX GPS modulu a Rx D COM portu PC či STM32 (PA10). Tx D u STM32 je PA9.



```

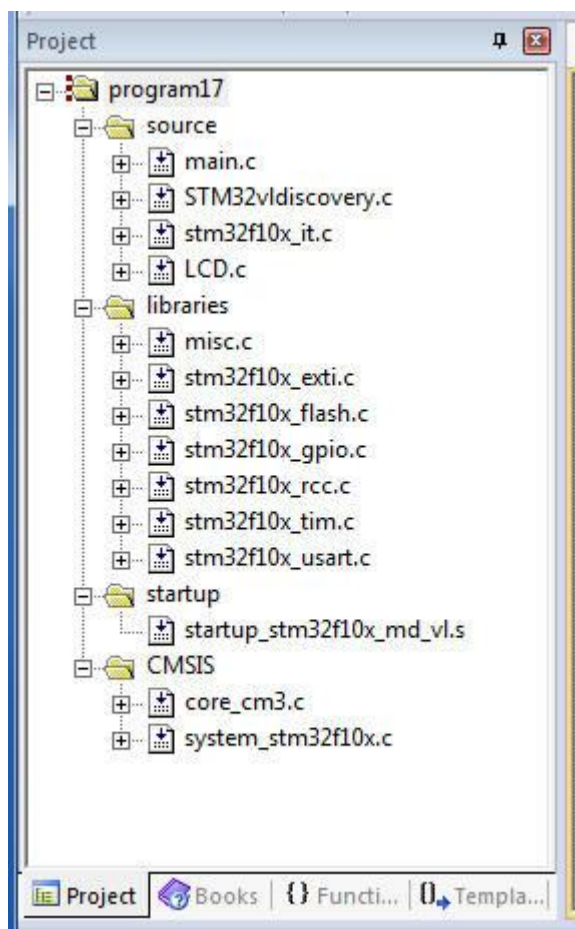
118
119 void init_gps(void)
120 {
121 // send_str("$SLTC,1000,0*6A\r\n"); // GPS Interrupt
122 // send_str("$PSRF100,1,4800,8,1,0*0E\r\n"); // SetSerialPort, 4800Bd, 8N1
123 Delay(0xAFFFFFFF);
124 strcpy (T_Buff, "\r\n");
125 USART_ITConfig(USART1, USART_IT_TXE, ENABLE); //odeslani prazdneho retezce a CR LF
126 Delay(0x0FFFFFFF);
127
128 strcpy (T_Buff, "\r\n");
129 USART_ITConfig(USART1, USART_IT_TXE, ENABLE); //odeslani prazdneho retezce a CR LF
130 Delay(0x0FFFFFFF);
131
132
133 //-----
134 /* GPS Interrupt */
135 //1 send_str("$SLTC,1000,0*6A\r\n");
136 strcpy (T_Buff, "$SLTC,1000,0*6A\r\n"); // nastavime co se odesle po USART
137 USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
138 Delay(0x0FFFFFFF);
139
140 /* GGA disable */
141 //2 send_str("$PSRF103,00,00,00,01*24\r\n");
142 strcpy (T_Buff, "$PSRF103,00,00,00,01*24\r\n"); // nastavime co se odesle po USART

```

```

Build Output
compiling core_cm3.c...
compiling system_stm32f10x.c...
linking...
Program Size: Code=11548 RO-data=348 RW-data=84 ZI-data=1716
".\output\program17.axf" - 0 Error(s), 0 Warning(s).
Load "C:\ARM_prikklady\keil\program17\output\program17.AXF"

```



Inicializace GPS

```
void init_gps(void)
{
    Delay(0x2FFFFFF);
    Strcpy (T_Buff, "\r\n");
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE); //odeslani prazdneho
//retezce a CR LF
    Delay(0x0FFFFFF);
    Strcpy (T_Buff, "\r\n");
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE); //odeslani prazdneho
//retezce a CR LF
    Delay(0x0FFFFFF);
// send_str("$LTC,1000,0*6A\r\n"); // GPS Interrupt
// Strcpy (T_Buff, "$LTC,1000,0*6A\r\n"); // nastavime co se odesle po
//USART
// USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);
}
```

```
// send_str("$PSRF100,1,4800,8,1,0*0E\r\n"); // SetSerialPort,
//4800Bd, 8N1
    Strcpy (T_Buff, "$PSRF100,1,4800,8,1,0*0E\r\n"); // nastavime co
//se odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
Delay(0x0FFFFFF);

/* SiRF nastaveni s*/
/* GGA enable rate=1 s*/
    Strcpy (T_Buff, "$PSRF103,00,00,01,01*25\r\n"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* GLL disable */
    Strcpy (T_Buff, "$PSRF103,01,00,00,01*25\r\n"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);

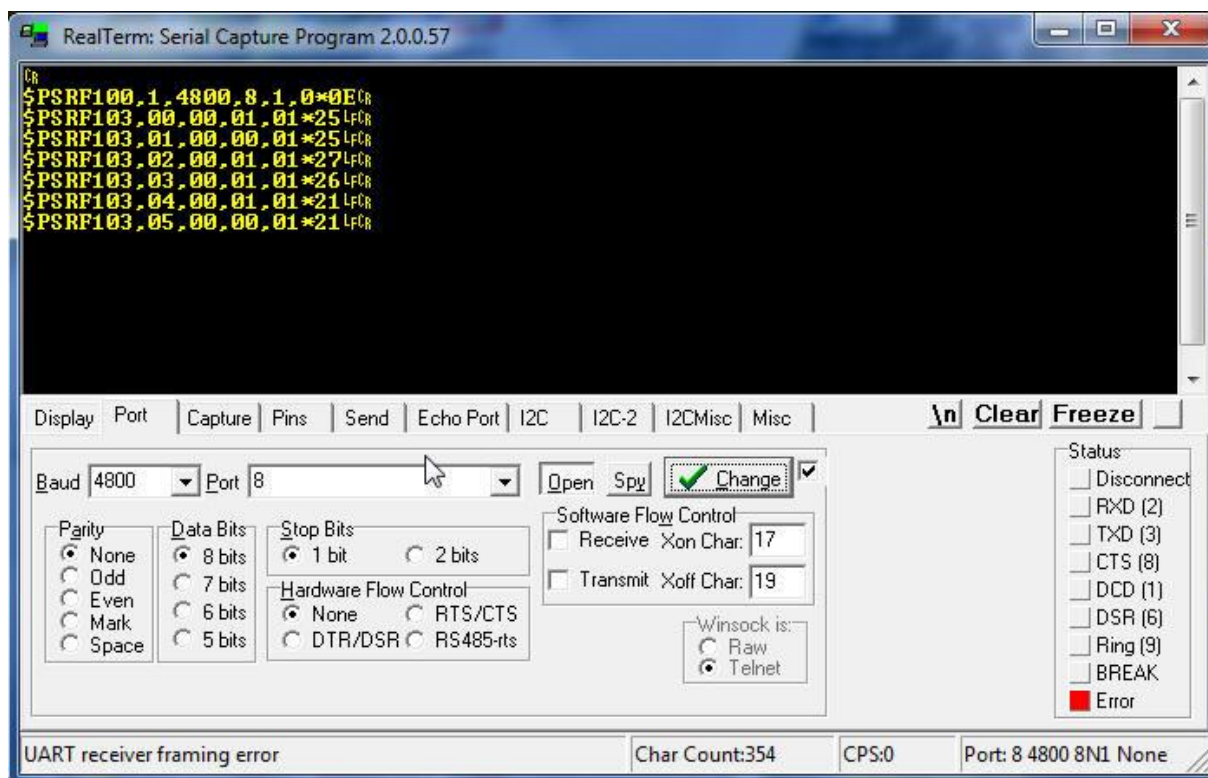
Delay(0x0FFFFFF);

/* GSA enable rate = 1s */
    Strcpy (T_Buff, "$PSRF103,02,00,01,01*27\r\n"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* GSV enable rate = 1s */
    Strcpy (T_Buff, "$PSRF103,03,00,01,01*26\r\n"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* RMC enable rate 1 sec */
    Strcpy (T_Buff, "$PSRF103,04,00,01,01*21\r\n"); // RMC enable rate 1
//sec - nastavime co se odesle po USART
    //Strcpy (T_Buff, "$PSRF103,04,00,00,01*20\r\n"); // RMC disble
//- nastavime co se odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* VTG disable */
    Strcpy (T_Buff, "$PSRF103,05,00,00,01*21\r\n"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x40FFFF);
}
```



Pozn.:

Posláním odpovídajících řetězců do GPS modulu se povolí/zakáže posílání daných typů zpráv SiRF tvaru \$PSRF103,05,00,01,01*20 kde

```
$PSRF103
05      00=GGA
        01=GLL
        02=GSA
        03=GSV
        04=RMC
        05=VTG

00      mode, 0=set rate, 1=query
01      rate in seconds, 0-255
01      checksum 0=no, 1=yes
*20     checksum
```

Takže máme následující možnosti:

GGA povolit (rate = 1s): "\$PSRF103,00,00,01,01*25\n"
GGA zakázat (rate = 0s): "\$PSRF103,00,00,00,01*24\n"

GLL povolit (rate = 1s): "\$PSRF103,01,00,01,01*26\n"
GLL zakázat (rate = 0s): "\$PSRF103,01,00,00,01*25\n"

GSA povolit (rate = 1s): "\$PSRF103,02,00,01,01*27\n"
GSA zakázat (rate = 0s): "\$PSRF103,02,00,00,01*26\n"

GSV povolit (rate = 1s): "\$PSRF103,03,00,01,01*26\n"

GSV zakázat (rate = 0s): "\$PSRF103,03,00,00,01*27\n"

RMC povolit (rate = 1s): "\$PSRF103,04,00,01,01*21\n"

RMC zakázat (rate = 0s): "\$PSRF103,04,00,00,01*20\n"

V našem příkladu jsem GGA, GSA, GSV a RMC povolil, GLL a VTG zakázal.

Pozn. GPS defaultně je na 4800Bd 8N1

Po připojení GPS modulu, tento vysílá např.

```

$GPGGA,143909.000,5007.8173,N,01439.6383,E,1,04,2.3,240.5,M,45.3,M,0.000*50\r\n
$GPGSA,A,3,12,25,21,29,2.5,2.3,1.0*39\r\n
$GPGSU,3,1,11,25,78,092,28,29,72,232,23,31,45,297,,12,40,104,33*73\r\n
$GPGSU,3,2,11,02,35,060,21,14,09,235,22,10,07,049,,21,06,188,41*7B\r\n
$GPGSU,3,3,11,05,04,097,25,04,01,030,,23,00,351,,*41\r\n
$GPRMC,143909.000,A,5007.8173,N,01439.6383,E,0.24,15.39,280711,,*3E\r\n
$GPGGA,143910.000,5007.8176,N,01439.6390,E,1,04,2.3,239.6,M,45.3,M,0.000*52\r\n
$GPGSA,A,3,12,25,21,29,2.5,2.3,1.0*39\r\n
$GPRMC,143910.000,A,5007.8176,N,01439.6390,E,0.41,351.44,280711,,*0B\r\n
$GPGGA,143911.000,5007.8179,N,01439.6399,E,1,04,2.3,238.6,M,45.3,M,0.000*54\r\n
$GPGSA,A,3,12,25,21,29,2.5,2.3,1.0*39\r\n
$GPRMC,143911.000,A,5007.8179,N,01439.6399,E,0.41,350.04,280711,,*09\r\n
$GPGGA,143912.000,5007.8181,N,01439.6400,E,1,04,2.3,238.4,M,45.3,M,0.000*55\r\n
$GPGSA,A,3,12,25,21,29,2.5,2.3,1.0*39\r\n
$GPRMC,143912.000,A,5007.8181,N,01439.6400,E,0.45,341.53,280711,,*0C\r\n

```

\$GPGGA,145058.000,5007.8020,N,01439.6458,E,1,05,3.4,278.0,M,45.3,M,,0000*54

zemepisná šířka: 5007.8020 severní šířky => 50°7,8020'N = 50°7'48,12"N

zemepisná délka: 01439.6458 východní délky => 14°39,6458'E = 14°39'38,74"E

nadmorská výška: 278.0m

\$GPRMC,151952.000,A,5007.7931,N,01439.6520,E,0.19,27.22,280711,,*38

Čas: 15:19:52 UTC tj. 17:19:52 letního času v CZ

Datum: 28.7.2011

Program main.c zobrazující na LCD veškerá přijatá data (obdobně jako např. RealTerm na PC)


```

/* STM32 VL Discovery Library
 * vysilani a prijem pres USART PA9 je Tx          PA10 je Rx
 * ver 1.0 datum a cas z GPS družice
 * Vd for SPSE Jecna lessons - prijem 4800Bd znaku - do/z GPS
 */

/* Includes -----
---*/
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include "LCD.h"

/* Private typedef -----
---*/
/* Private define -----
---*/
#define BUFF_SIZE 100

/* Private macro -----
---*/
/* Private variables -----
---*/
uint8_t RX_prijato = 0;          // priznak ukonceni prijmu z PC
// buffery na prijimane a odesilane znaky
uint8_t R_Buff[BUFF_SIZE]="";
uint8_t T_Buff[BUFF_SIZE]="";
uint8_t LCD_Message1[] = "Prijem GPS 4800";
/* Private function prototypes -----
---*/
void USART_Inicializace(void);
void GPIO_Inicialization(void);
void init_gps(void);

int fputc(int ch, FILE * f);

void Delay(__IO uint32_t nTick);
uint8_t Strcmp (const uint8_t * s1, const uint8_t * s2); // porovnani str
void Strcpy(uint8_t *d1, const uint8_t *s1); // kopirovani str
void vynulovaniRXregistru(void);

/* Private functions -----
---*/

int main(void)
{
    GPIO_Inicialization();
    // aktivujeme USART
    USART_Inicializace();

```

```

LCD_Inicialization();
STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
printLCD(LCD_Message1);
init_gps();
Delay(0xAFFFFFFF);
STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridat jsem
LCD_Clear();
while (1)
{
    if (RX_prijato == 1) // neco jsme prijali - zakoncene CR
    {
        LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici prvnio
radku
        Strcpy (LCD_Message1,R_Buff );
        printLCD(LCD_Message1);
        vynulovaniRXregistru();
        RX_prijato = 0; // vymazeme flag ukonceni prijmu
    }
}

void GPIO_Inicialization(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

void USART_Inicializace(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    /* pustime hodiny do periferie (protoze jde o alternativni funkci,
musi
    * jit hodiny i do AFIO periferie! */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
RCC_APB2Periph_AFIO, ENABLE);

    // PA9 je Tx
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // alternate
function!
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // PA10 je Rx
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

```

```

USART_InitStructure.USART_BaudRate = 4800;
/* Hodnota 4800 je defaultne pouzivana GPS */
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
//8N1

USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART1, &USART_InitStructure);
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
USART_Cmd(USART1, ENABLE);

// konfigurace preruseni
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
}

void init_gps(void)
{
    Delay(0x2FFFFFF);
    Strcpy (T_Buff, "\r\n");
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE); //odeslani prazdneho
//retezce a CR LF
    Delay(0x0FFFFFF);
    Strcpy (T_Buff, "\r\n");
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE); //odeslani prazdneho
//retezce a CR LF
    Delay(0x0FFFFFF);
// send_str("$LTC,1000,0*6A\r\n"); // GPS Interrupt
// Strcpy (T_Buff, "$LTC,1000,0*6A\r\n"); // nastavime co se odesle po
//USART
// USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

// send_str("$PSRF100,1,4800,8,1,0*0E\r\n"); // SetSerialPort,
//4800Bd, 8N1
    Strcpy (T_Buff, "$PSRF100,1,4800,8,1,0*0E\r\n"); // nastavime co
//se odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* SiRF nastaveni s*/
/* GGA enable rate=1 s*/
    Strcpy (T_Buff, "$PSRF103,00,00,01,01*25\n\r"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* GLL disable */

```

```

    Strcpy (T_Buff, "$PSRF103,01,00,00,01*25\n\r"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);

Delay(0x0FFFFFF);

/* GSA enable rate = 1s */
    Strcpy (T_Buff, "$PSRF103,02,00,01,01*27\n\r"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* GSV enable rate = 1s */
    Strcpy (T_Buff, "$PSRF103,03,00,01,01*26\n\r"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* RMC enable rate 1 sec */
    Strcpy (T_Buff, "$PSRF103,04,00,01,01*21\n\r"); // RMC enable rate 1
//sec - nastavime co se odesle po USART
    //Strcpy (T_Buff, "$PSRF103,04,00,00,01*20\n\r"); // RMC disable
//- nastavime co se odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x0FFFFFF);

/* VTG disable */
    Strcpy (T_Buff, "$PSRF103,05,00,00,01*21\n\r"); // nastavime co se
//odesle po USART
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
    Delay(0x40FFFF);
}

uint8_t Strcmp (const uint8_t * s1, const uint8_t * s2)
{
    for(; *s1 == *s2; ++s1, ++s2)
        if(*s1 == 0 || *s1 == 0x0d)
            return 0;
    return *(unsigned char *)s1 < *(unsigned char *)s2 ? -1 : 1;
}

void Strcpy(uint8_t *d1, const uint8_t *s1)
{
    uint8_t i;
    for (i=0; s1[i] != '\0'; ++i)
        d1[i] = s1[i];
    d1[i] = '\0';
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

```

/* Zapis znaku 'ch' do souboru 'f' */
int fputc(int ch, FILE * f)
{
    /* Predani znaku pro odeslani UARTem */
    USART_SendData(USART1, (u8) ch);
    /* Cekame, dokud neni prenos dokoncen */
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    return ch;
}

void vynulovaniRXregistru(void)
{
    uint8_t i;
    for(i=0; i<40; i++)
        R_Buff[i]= '\0';
    ;
}

```

Protože např.

\$GPGGA,155153.000,5007.7974,N,01439.6481,E,1,05,1.4,290.9,M,45.3,M,,0000*51 má délku 75 znaků, proto jsem zvolil

```
#define BUFF_SIZE 100
```

K získání a zobrazení dat jako je čas, datum, souřadnice musíme ovšem přijaté textové řetězce dále zpracovat. **Program17b** např. zobrazuje jen zprávy GGA. Tento příklad je stejný jako **Program17**, liší se jen funkcí **main(void)**

```

int main(void)
{
    GPIO_Inicialization();
    // aktivujeme USART
    USART_Inicializace();
    LCD_Inicialization();
    STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
    printLCD(LCD_Message1);
    init_gps();
    Delay(0x0FFFFFF);

    STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridal jsem
    LCD_Clear();
    while (1)
    {
        if (RX_prijato == 1) // neco jsme prijali - zakoncene CR
        {
            LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici prvnioho
            radku RMC

            if ((R_Buff[0] == '$') & (R_Buff[1] == 'G') & (R_Buff[2] == 'P'))
            {

```

```

    if ((R_Buff[3] == 'R') & (R_Buff[4] == 'M') & (R_Buff[5] == 'C')) //RMC
    {
        Strcpy (LCD_Message1, R_Buff );
        printLCD(LCD_Message1);
    }

    if ((R_Buff[3] == 'G') & (R_Buff[4] == 'G') & (R_Buff[5] == 'A')) //GGA
    {
        Strcpy (LCD_Message1, R_Buff );
        printLCD(LCD_Message1);
    }

}
vynulovaniRXregistru();
RX_prijato = 0; // vymazeme flag ukonceni prijmu
}
}
}

```

Další **program17c** ukazuje hodiny řízené GPS. Od předchozích **program17** a **program17b** se opět liší jen funkcí **main(void)**

```

int main(void)
{
    GPIO_Inicialization();
    // aktivujeme USART
    USART_Inicializace();
    LCD_Inicialization();
    STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
    printLCD(LCD_Message1);
    init_gps();
    Delay(0x0FFFFFF);

    STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridal jsem
    LCD_Clear();
    while (1)
    {
        if (RX_prijato == 1) // neco jsme prijali - zakoncene CR
        {
            //LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici
            //prvniho radku RMC

            if ((R_Buff[0] == '$') & (R_Buff[1] == 'G') & (R_Buff[2] == 'P'))
            {

                if ((R_Buff[3] == 'R') & (R_Buff[4] == 'M') & (R_Buff[5] == 'C')) //RMC
                {
                    LCD_Message1[0] = R_Buff[7];
                }
            }
        }
    }
}

```

```

LCD_Message1[1] = R_Buff[8];
LCD_Message1[2] = ':' ;
LCD_Message1[3] = R_Buff[9];
LCD_Message1[4] = R_Buff[10];
LCD_Message1[5] = ':' ;
LCD_Message1[6] = R_Buff[11];
LCD_Message1[7] = R_Buff[12];
LCD_Message1[8] = ' ' ;
LCD_Message1[9] = 'U' ;
LCD_Message1[10] = 'T' ;
LCD_Message1[11] = 'C' ;
LCD_Message1[12] = 0 ;
LCD_CursorHome();
printLCD(LCD_Message1);
}

    if ((R_Buff[3] == 'G') & (R_Buff[4] == 'G') & (R_Buff[5] == 'A'))
//GGA
    {
        LCD_Message1[0] = R_Buff[7];
        LCD_Message1[1] = R_Buff[8];
        LCD_Message1[2] = ':' ;
        LCD_Message1[3] = R_Buff[9];
        LCD_Message1[4] = R_Buff[10];
        LCD_Message1[5] = ':' ;
        LCD_Message1[6] = R_Buff[11];
        LCD_Message1[7] = R_Buff[12];
        LCD_Message1[8] = ' ' ;
        LCD_Message1[9] = 'U' ;
        LCD_Message1[10] = 'T' ;
        LCD_Message1[11] = 'C' ;
        LCD_Message1[12] = 0 ;
        LCD_CursorHome();
        printLCD(LCD_Message1);
    }

}
vynulovaniRXregistru();
RX_prijato = 0; // vymazeme flag ukonceni prijmu
}

}
}

```

LCD pak ukazuje čas ve formátu 14:58:12 UTC

Na dvouřádkovém LCD bychom ještě na druhém řádku mohli zobrazovat datum, který je rovněž zprávou RMC přenášen, museli bychom ovšem `R_Buff` procházet a „odpočítat“ příslušný počet oddělovacích čárek a poté zkopírovat a zobrazit údaj o aktuálním datumu. Ze zpráv RMC a GGA můžeme též určit zeměpisné souřadnice.



3 Práce v prostředí IAR Embedded Workbench

3.1 Stažení instalačního souboru

Na stránkách firmy IAR Systems přejdeme na download a dále vybereme produkt **Kick Start edition of IAR Embedded Workbench for ARM**:

The screenshot shows a web browser window with the address bar containing `http://supp.iar.com/Download/SW/?item=EWARM-KS32`. The browser's address bar shows "ems - Product Download". The main content area of the page features the IAR Systems logo and a title: "KickStart edition of IAR Embedded Workbench for ARM". Below the title, there is a paragraph of text: "The kickstart edition of IAR Embedded Workbench for ARM is completely free of charge and you may use it for as long as you want. The kickstart tools are ideal for creating small applications or for getting started fast on a new project. The only requirement is that you register to get a license key." This is followed by three sections: "Contents" (describing the code size limitation and included tools), "Limitations" (listing exceptions like 32 kbyte code size limitation and no source code for runtime libraries), and "Upgrade path" (stating that a complete upgrade path is available from IAR Systems). At the bottom of the content area, there is a yellow button labeled "Continue...".

Klikneme na **Continue**, dostaneme registrační formulář

Product Registration and Download

IAR Embedded Workbench for ARM, v. 6.20, 32K Kickstart Edition

When you have registered below, you will receive an e-mail containing information that is required to install the software. Please make sure to spell your e-mail address correctly!

First name *

Last name *


Title

Please specify:



Email *

Phone *

Fax

 Internet | Chráněný režim: Zapnuto

Po vyplnění a odeslání formuláře se objeví

Thank you!

An email has now been sent to the address you specified (vladvana@volny.cz), asking you to confirm the registration. Follow the instructions in that email to receive information on how to download and install the product.

No email received?

If the email mentioned above is not delivered to you within a reasonable time, please check:

- Is the spelling of your email address – vladvana@volny.cz – correct?
If not, use the Back button of your browser to return to the registration form, correct the spelling, and resubmit the form.
- If your email application holds a **Junk E-mail** folder for suspected spam emails, please check that the Product Registration email has not been placed there by mistake.
- If you, or your network administrator, can control the setup for the junk mail filter, add a rule saying that messages from **noreply.www@iar.com** are safe, and should always be delivered, then retry the registration. (If you still have this page open then, you can simply back to the form, and resubmit.)

[IAR Systems website](#)

Poté obdržíme e-mail s licenčním číslem a linky pro stažení sw.

Registration Confirmed

Thank you for your registration of the product:

IAR Embedded Workbench for ARM, v. 6.20, 32K Kickstart Edition

Download software

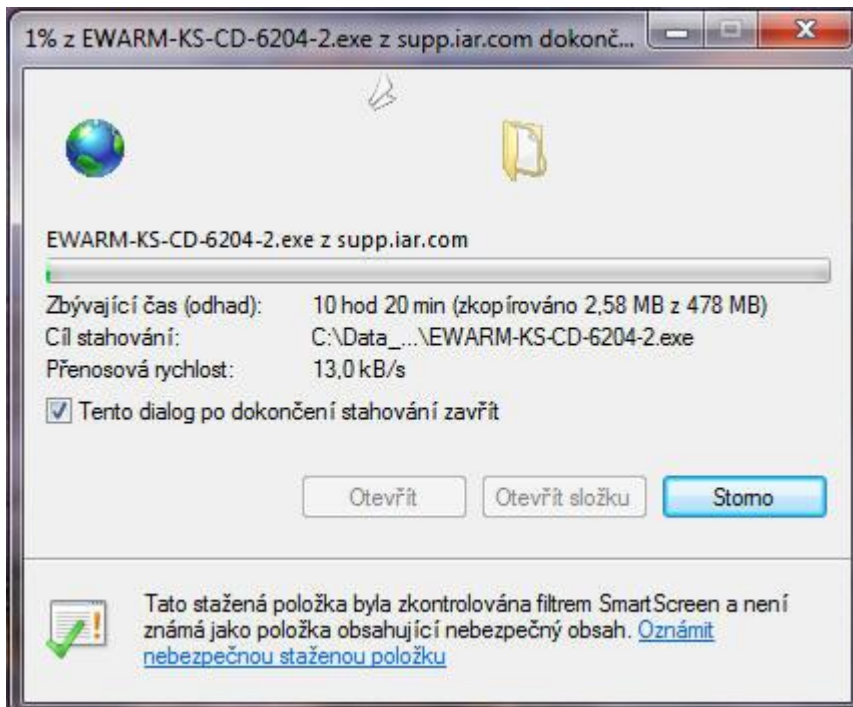
- **English version** (EXE, 478 MB)
 Download from:
 - [Europe \(HTTP site\)](#)
 - [Europe \(FTP site\)](#)
 - [Japan](#)
- **Japanese version** (EXE, 478 MB)
 Download from:
 - [Europe \(HTTP site\)](#)
 - [Europe \(FTP site\)](#)
 - [Japan](#)

When installing the software, you will be asked for a License Number and a License Key. Please use these values:

License Number:

License Key:

Stáhneme instalační program (např. EWARM-KS-CD-6204-2.exe)



Tím jsme získali instalační soubor EWARM-KS-CD-6204-2.exe Jeho instalací získáme (popis v eng. z IAR webu)

IAR Embedded Workbench® for ARM - KickStart edition

IAR Embedded Workbench is an Integrated Development Environment with a complete and easy-to-use set of C/C++ cross compiler and debugger tools for professional embedded applications.

- KickStart 32KB IAR C/C++ Compiler for ARM
- Project manager
- Editor
- Linker and librarian tools
- C-SPY® debugger
- Full integration with IAR J-Link

Example projects for STM32 include:

- GettingStarted

This example project shows basic use of I/O, timer and the interrupt controllers. Displays running lights on the board LED's.

- LCD_Demo

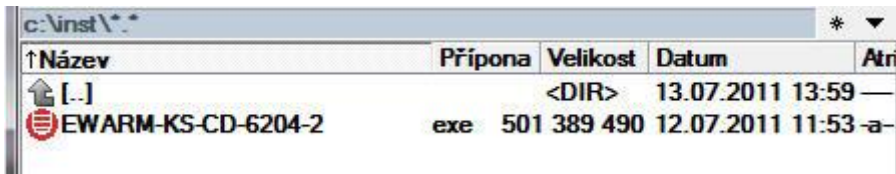
This example project shows basic use of parallel I/O, timer, interrupt controller, ADC and interface to a LCD HD44780 compatible module. The LCD will display the position of the potentiometer in percents.

- USBMouse

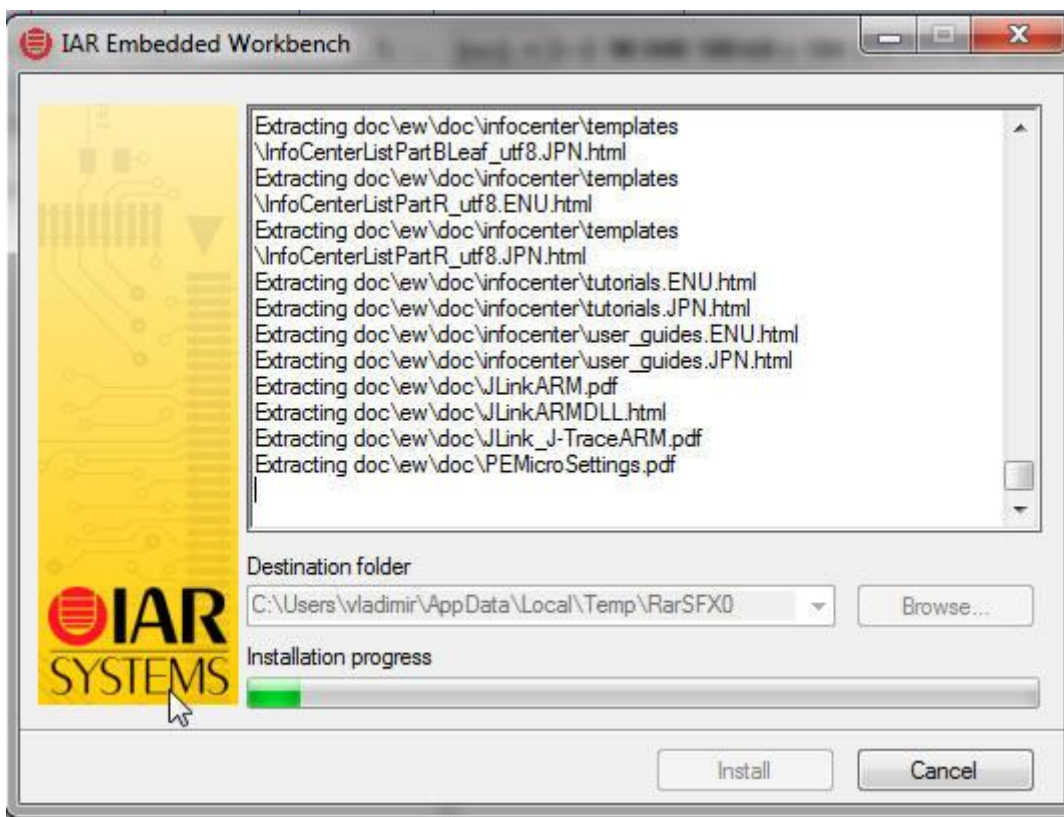
This example project shows how to implement a USB HID mouse. When host install needed driver a mouse's cursor begin to move in a rectangle shape. The WAKE-UP button is used for USB resume when the device is suspended.

The KickStart edition of IAR Embedded Workbench is fully functional, but the code size is limited.

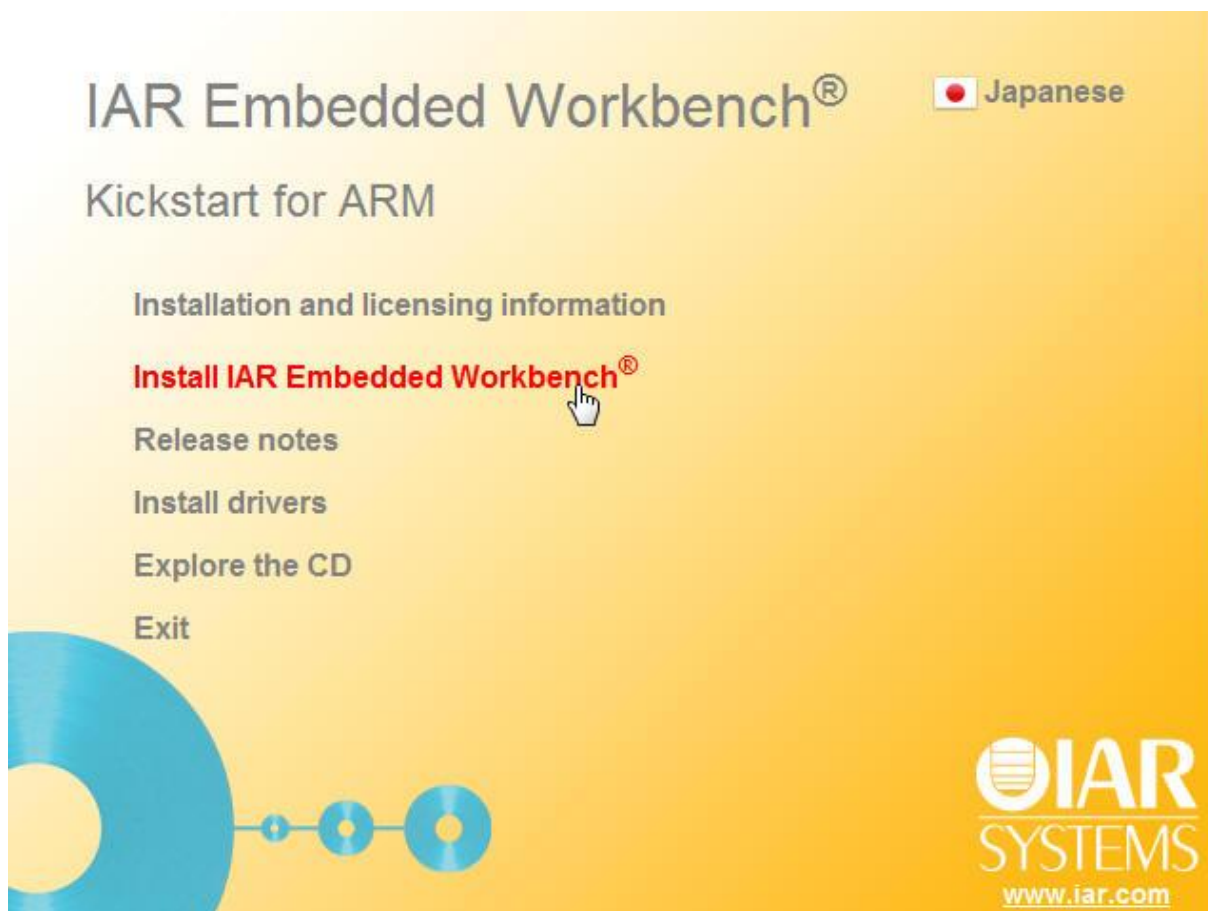
3.2 Instalace IAR



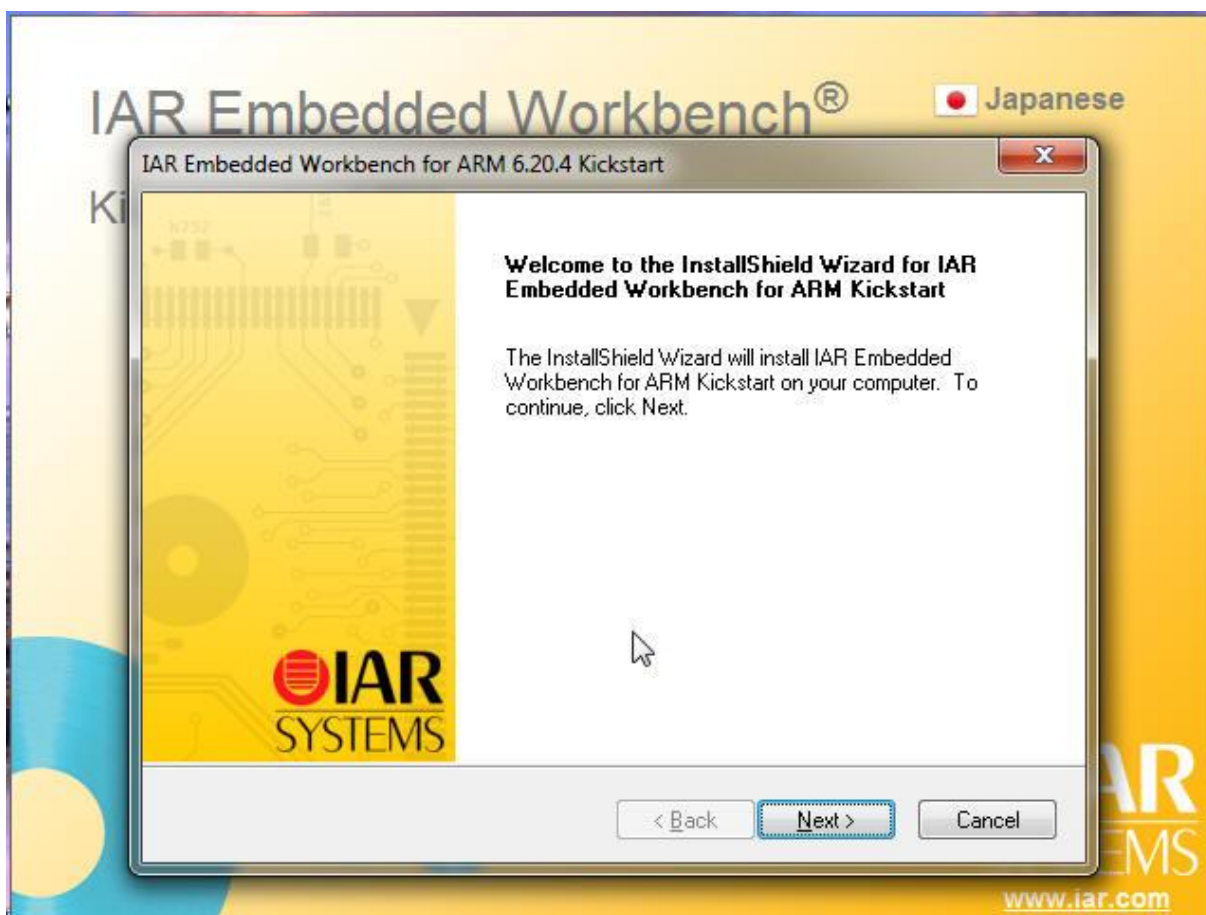
Spustíme instalační program



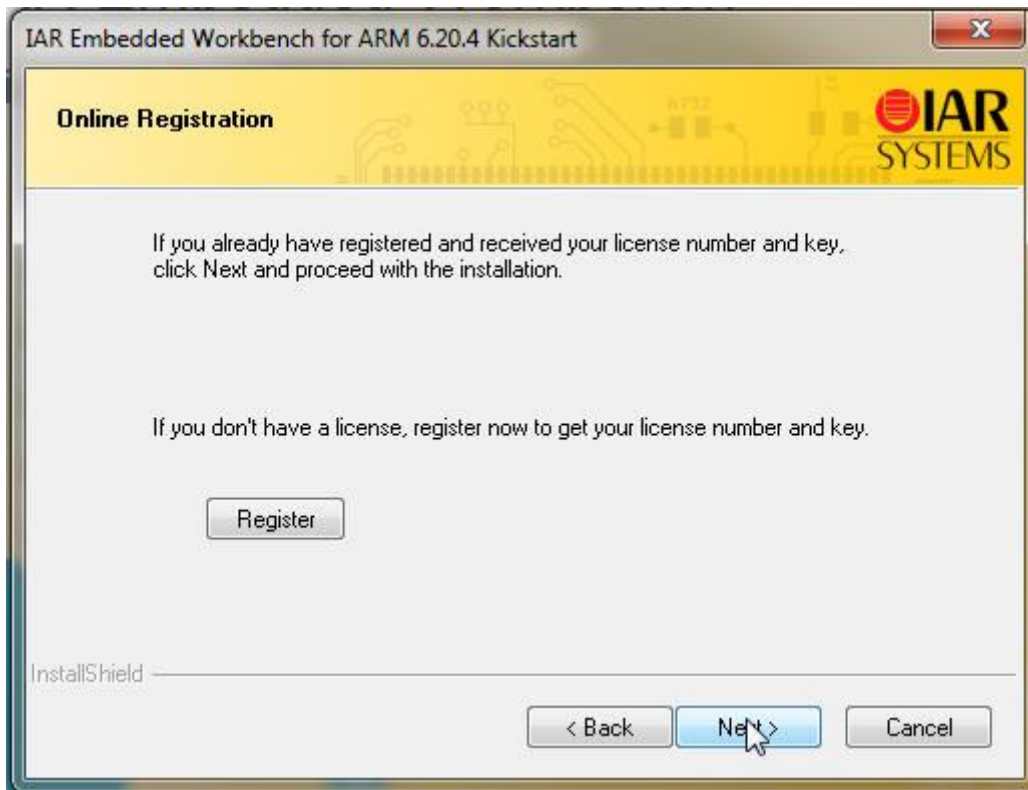
Probíhá extrakce potřebných souborů



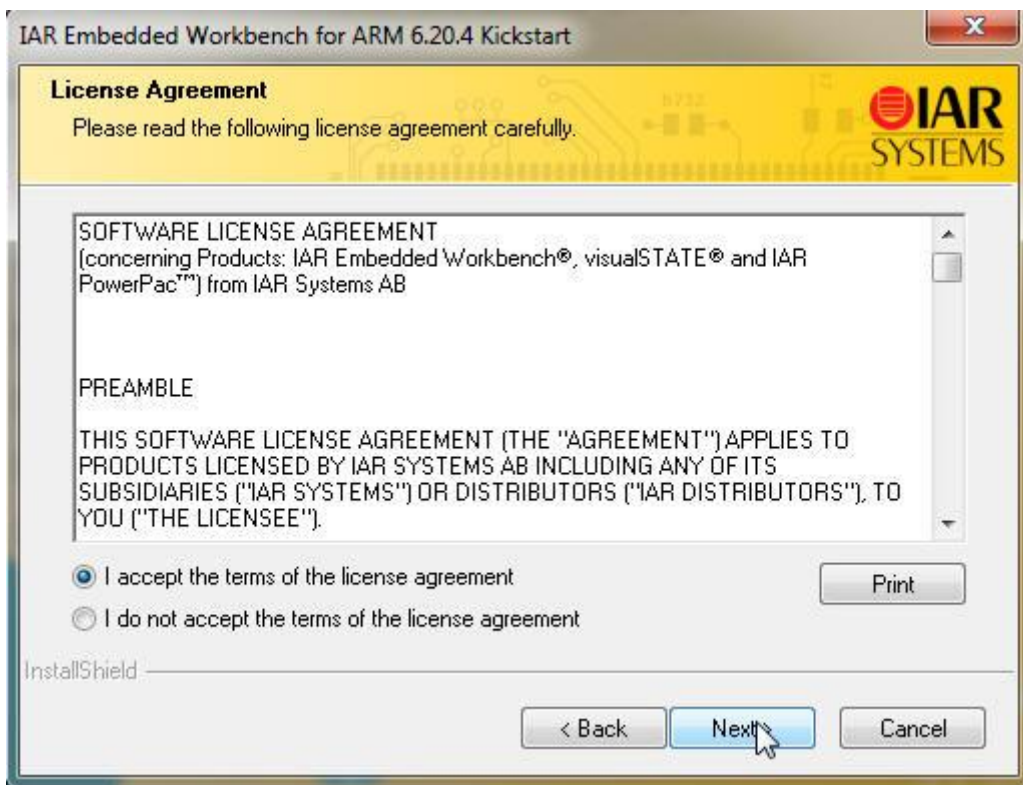
V výše uvedené nabídce již zvolíme vlastní instalace



Klikneme na **Next**



Opět Next



Odsouhlasíme licenční podmínky a opět klikneme na **Next**

IAR Embedded Workbench for ARM 6.20.4 Kickstart

Enter User Information

Enter your name, the name of your company and your IAR Embedded Workbench for ARM Kickstart license number.

Name:

Company:

Can be found on the CD cover, or via e-mail registration

License#:

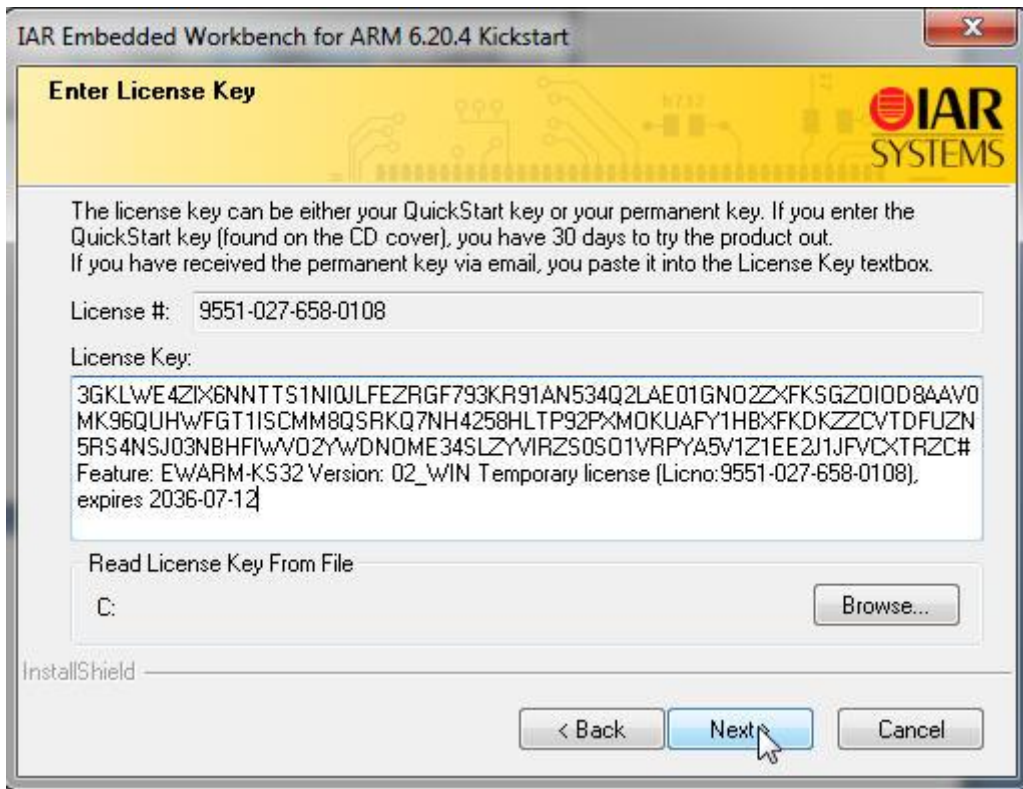
InstallShield

< Back Next > Cancel

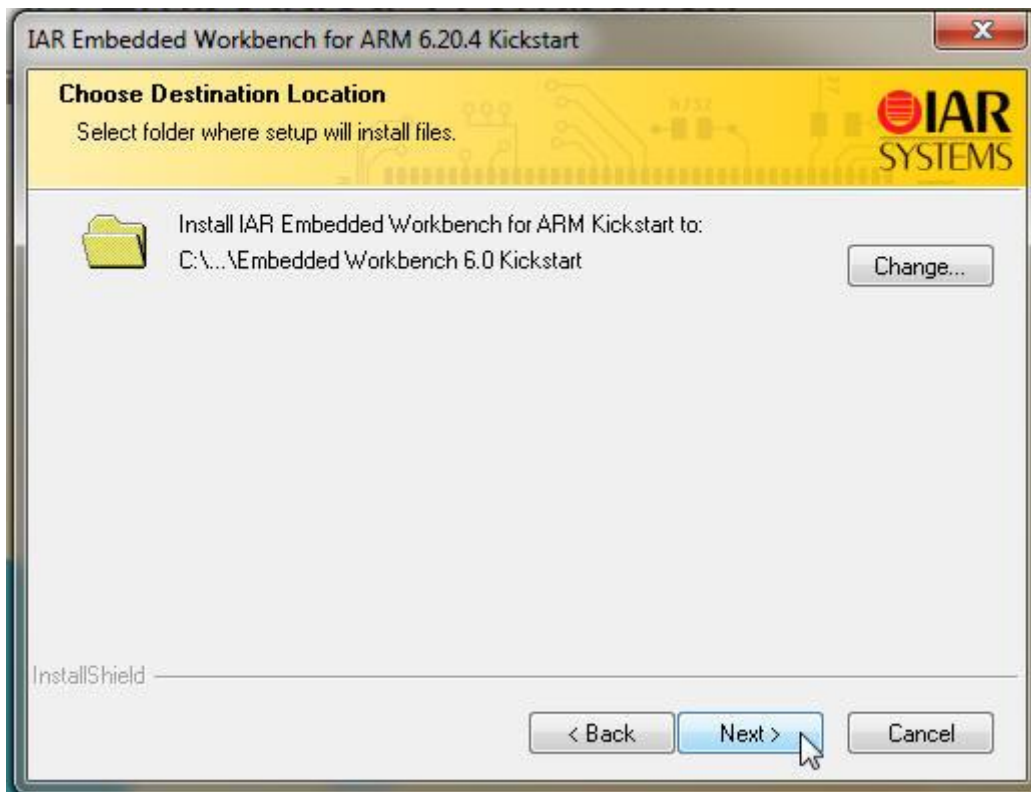
Vyplníme uživatelské informace a licenční číslo

Klikneme **Next**

Vložíme klíč



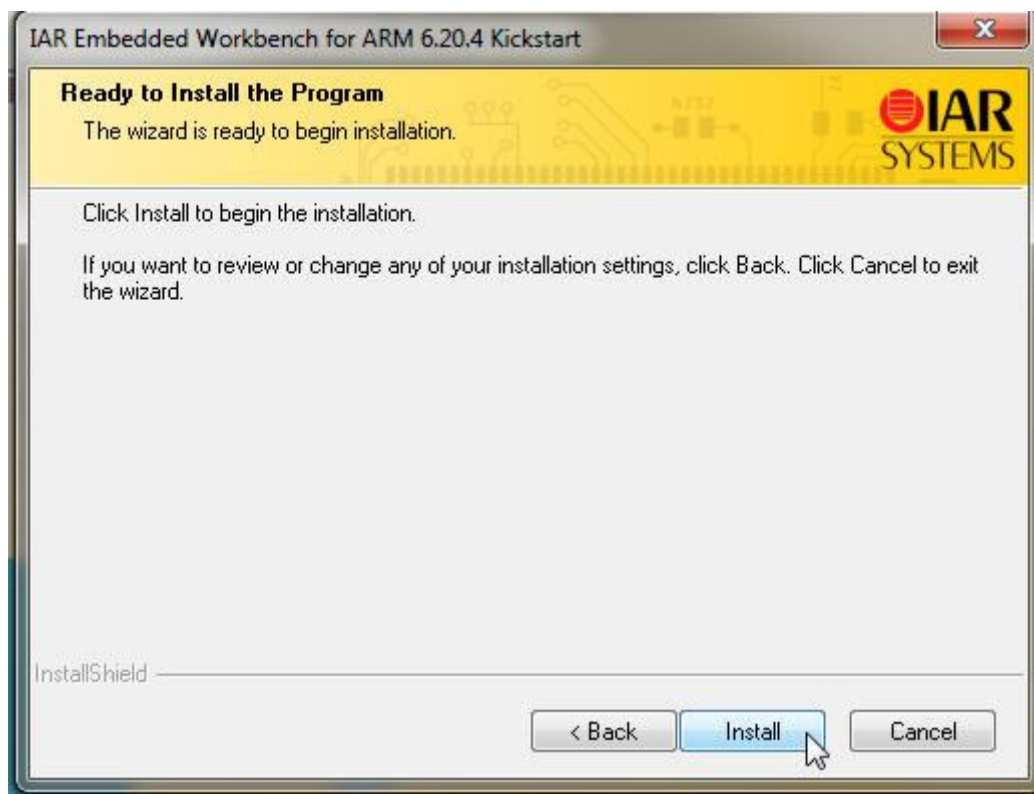
Stiskneme **Next**



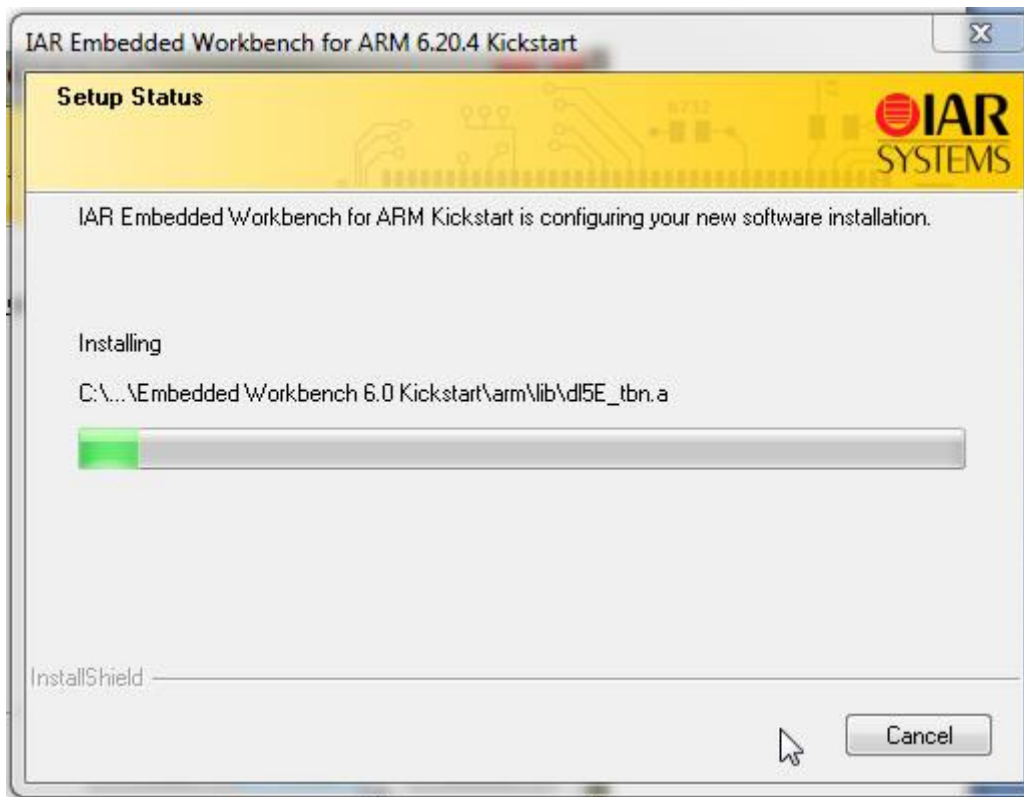
Znovu **Next**



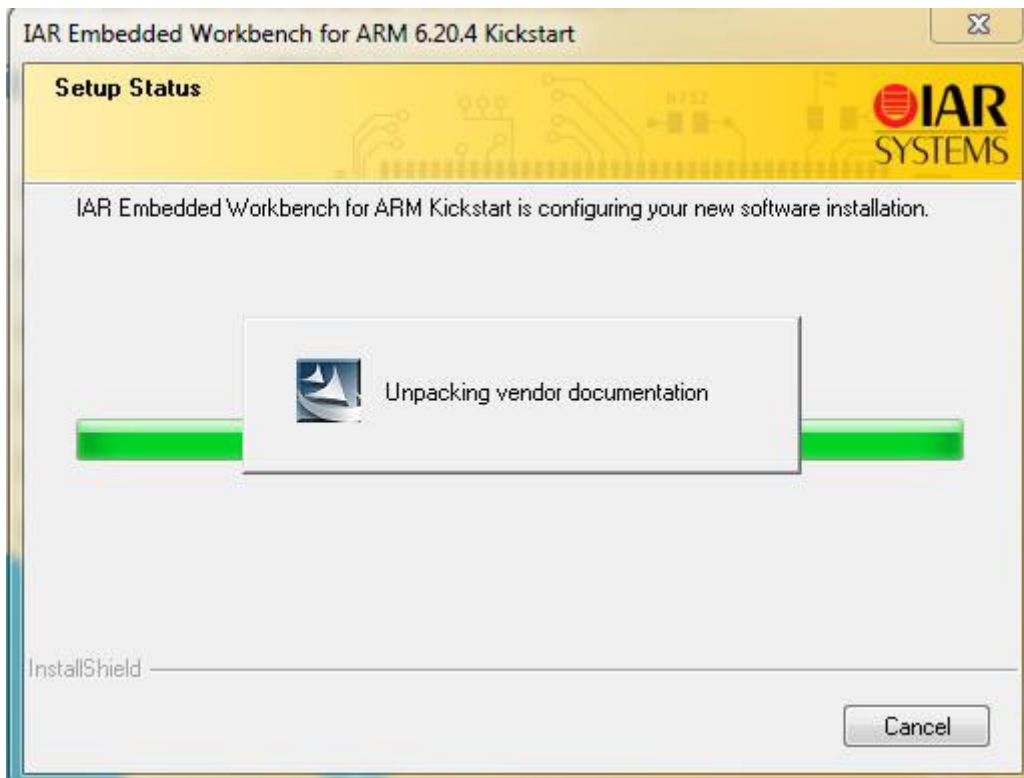
Opět **Next**

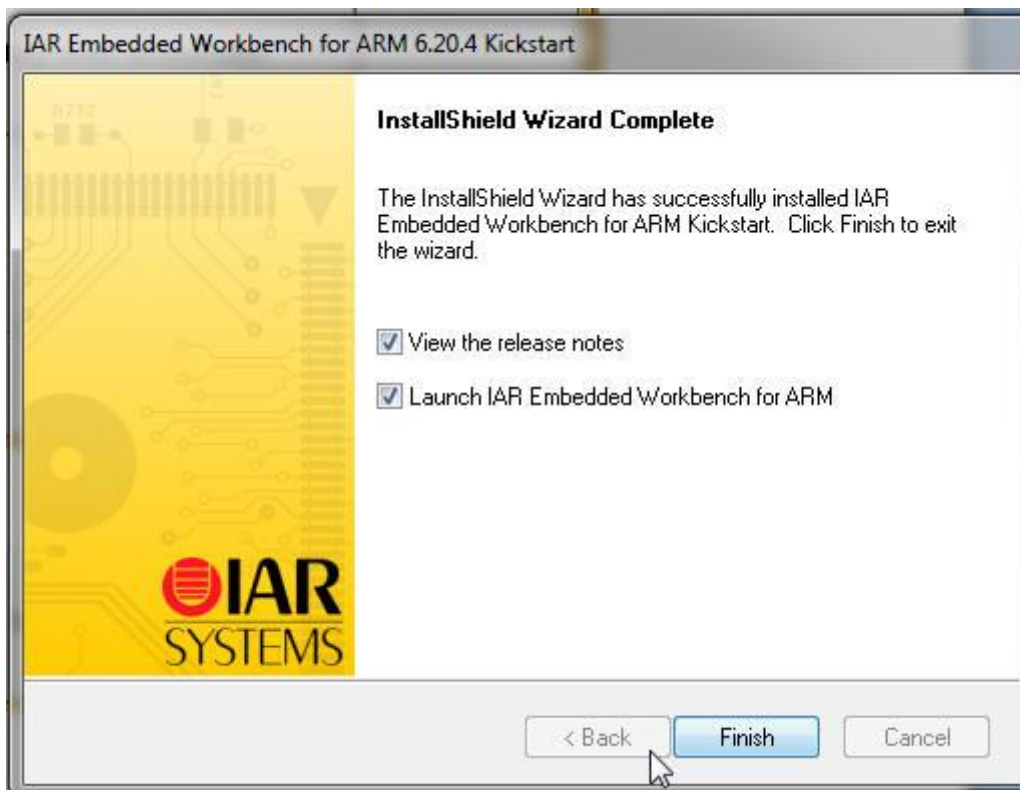


Konečně můžeme tlačítkem **Install** spustit vlastní instalaci

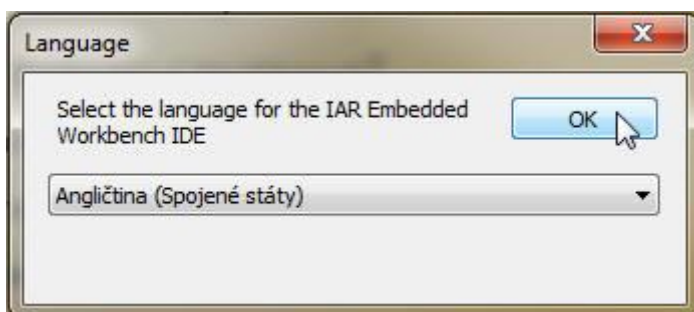


Instalace probíhá

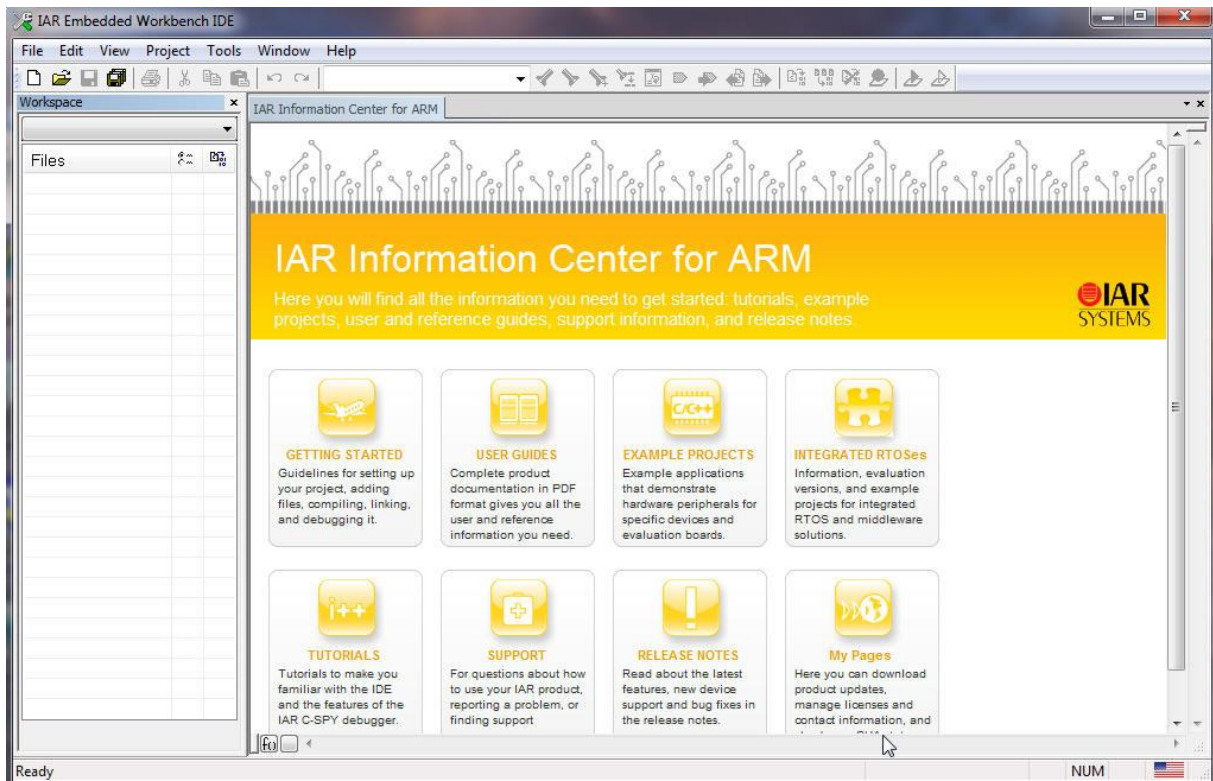




Klikneme na **Finish**



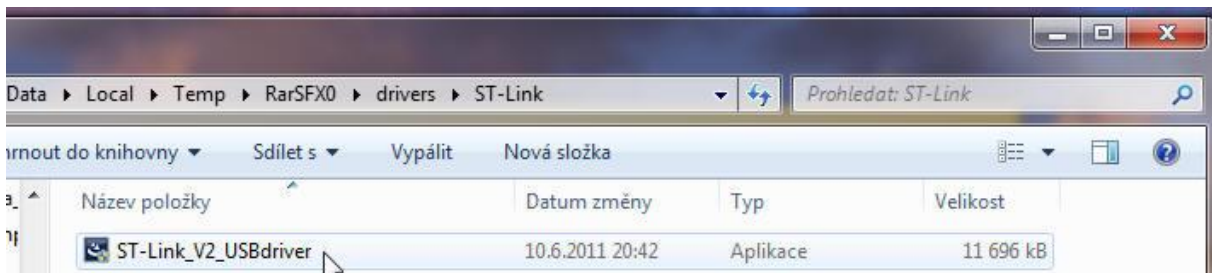
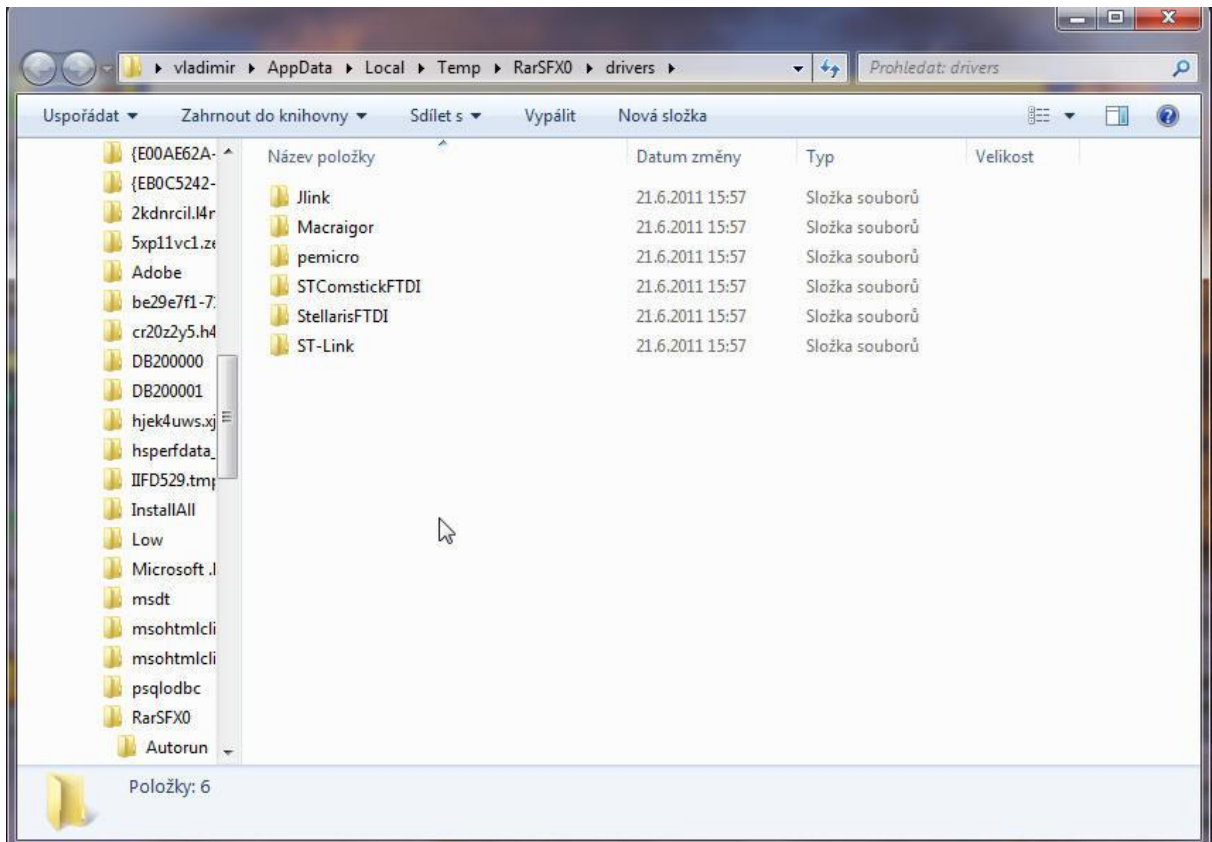
Vybereme angličtinu pro prostředí IDE

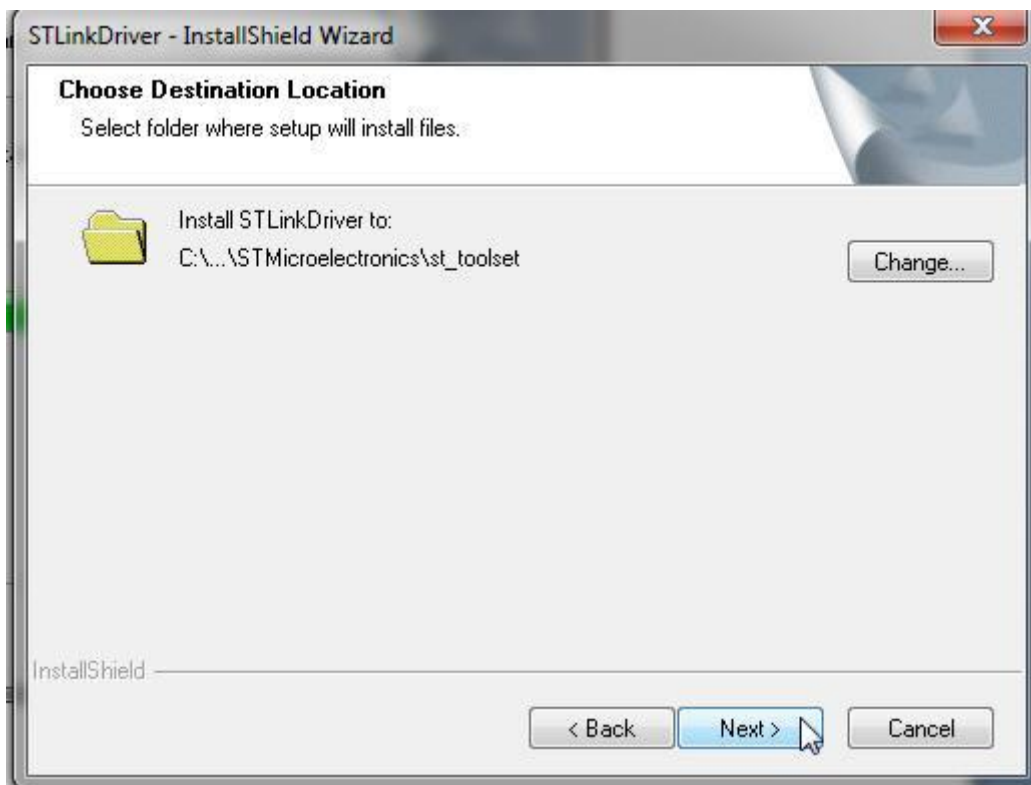
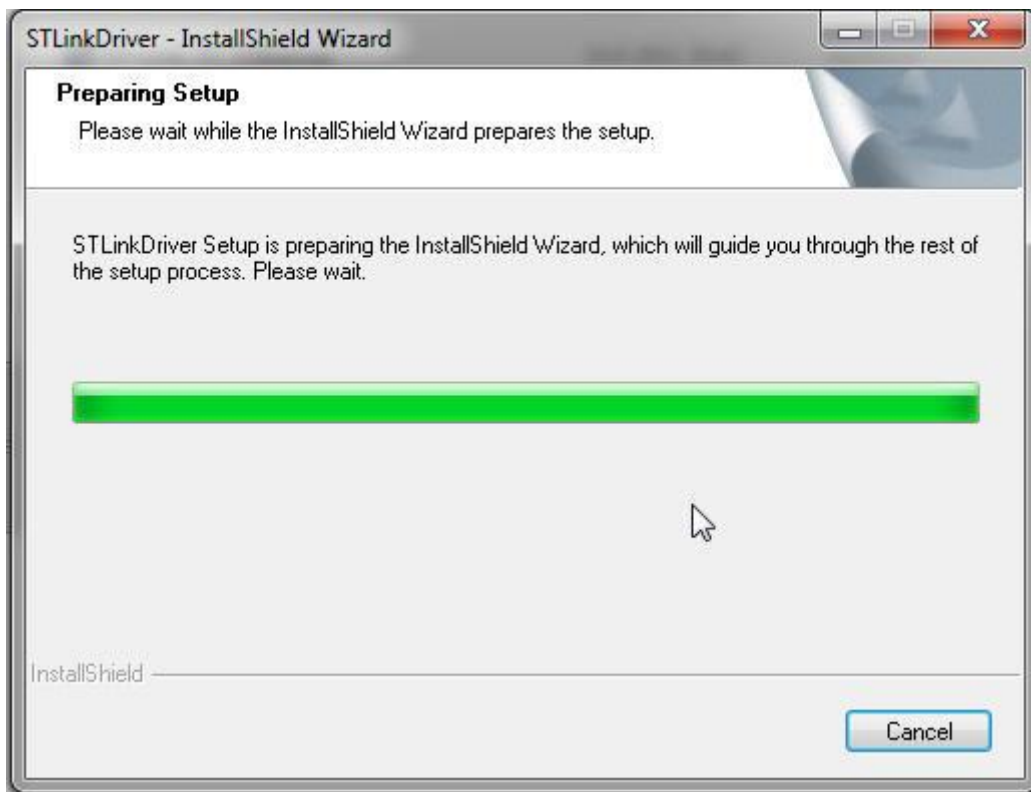


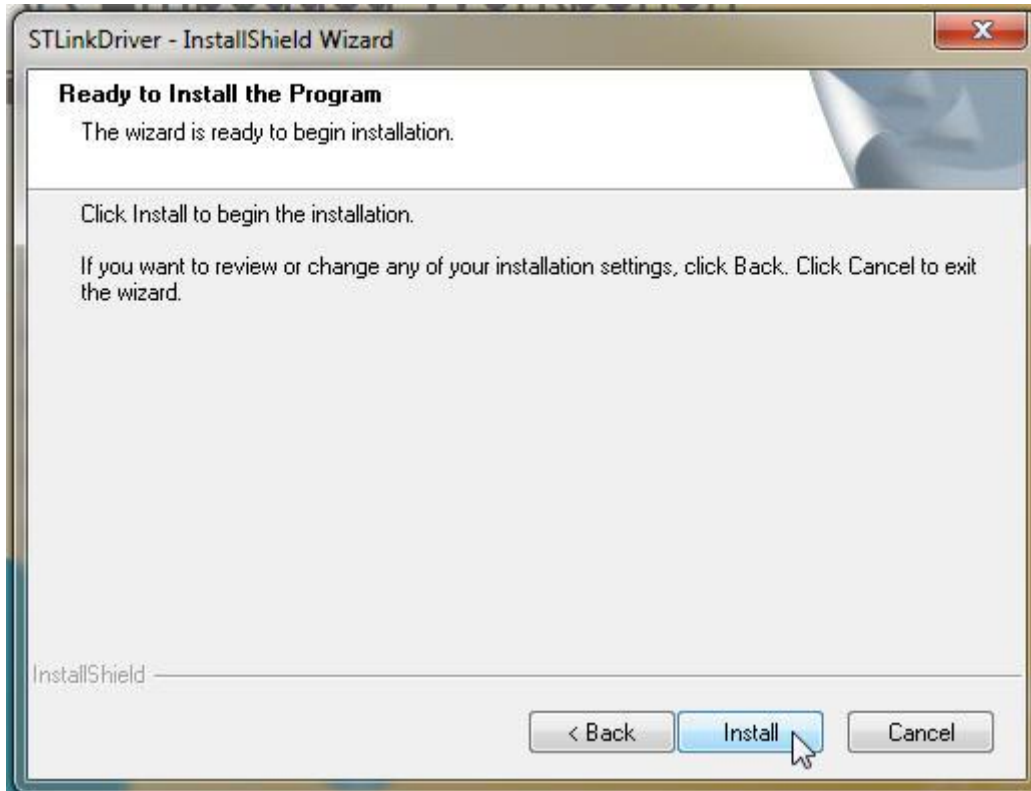
IAR Embedded Workbench IDE máme nainstalovaný,

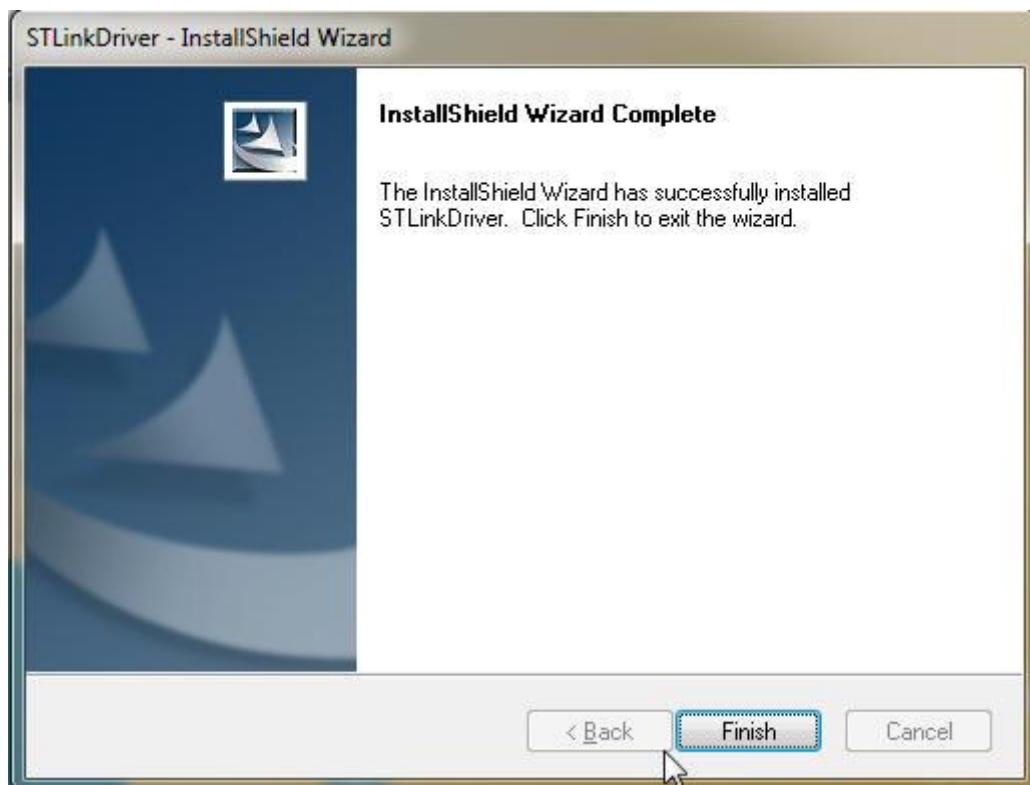


Ještě nainstalujeme drivery ST link pro USB

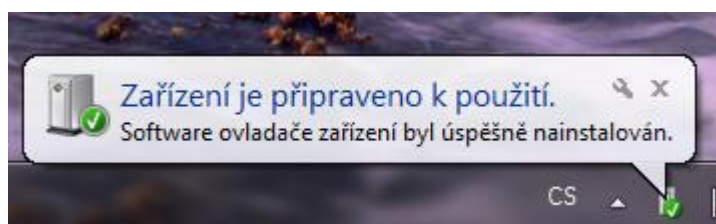
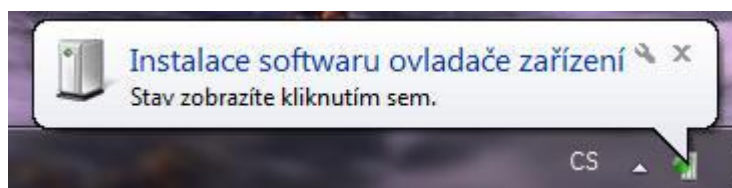


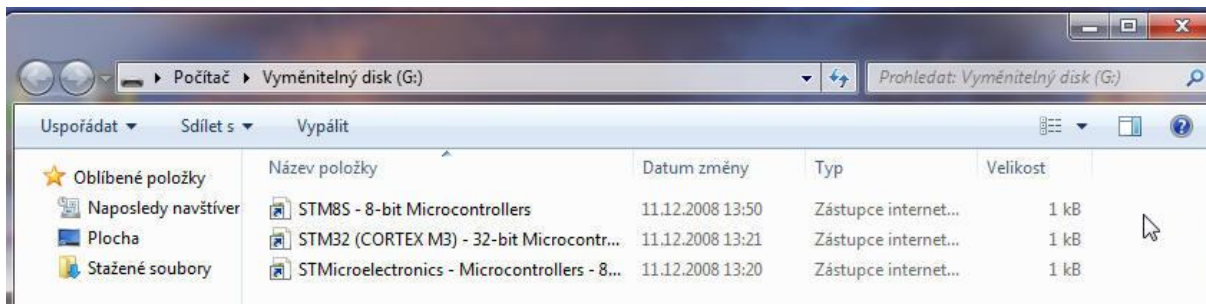






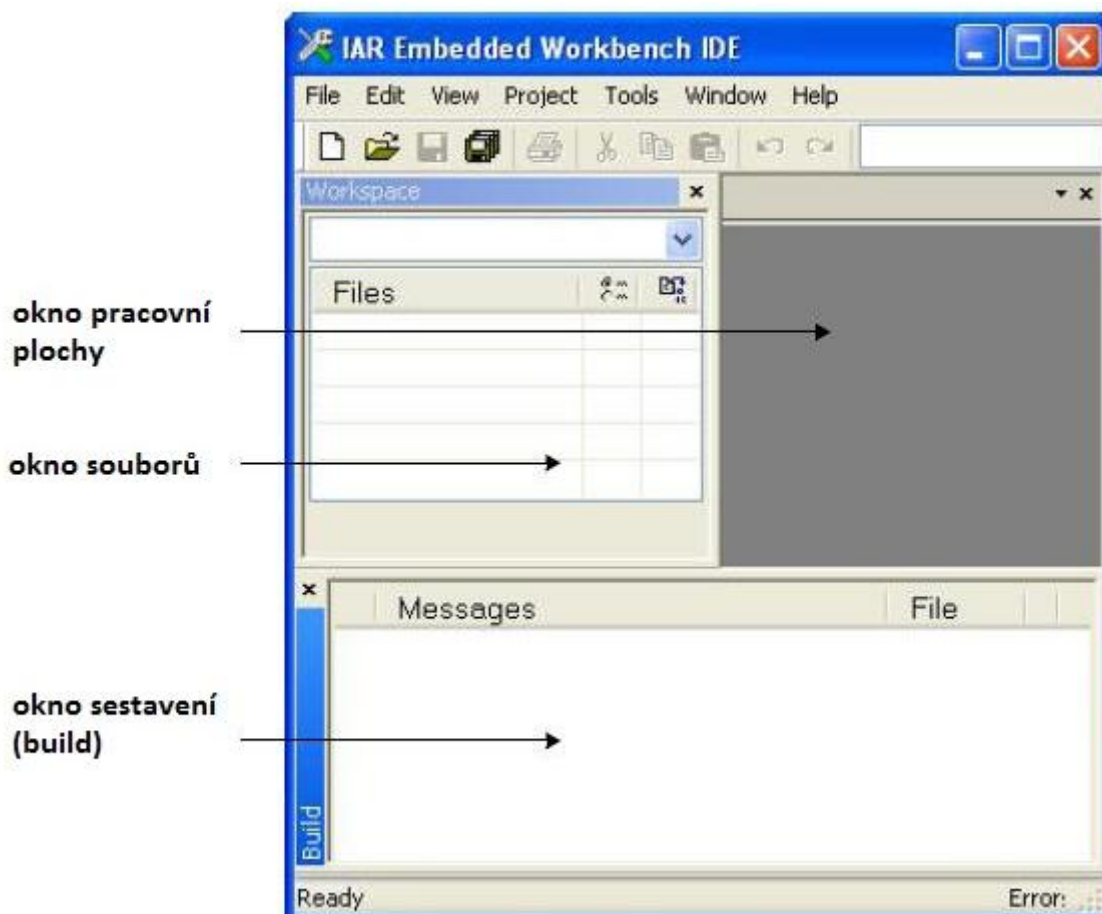
Instalace USB driveru pro ST je ukončena, zbývá kliknout na tlačítko **Finish**. Po připojení **STM32VL Discovery** se spustí Instalace software ovladače zařízení



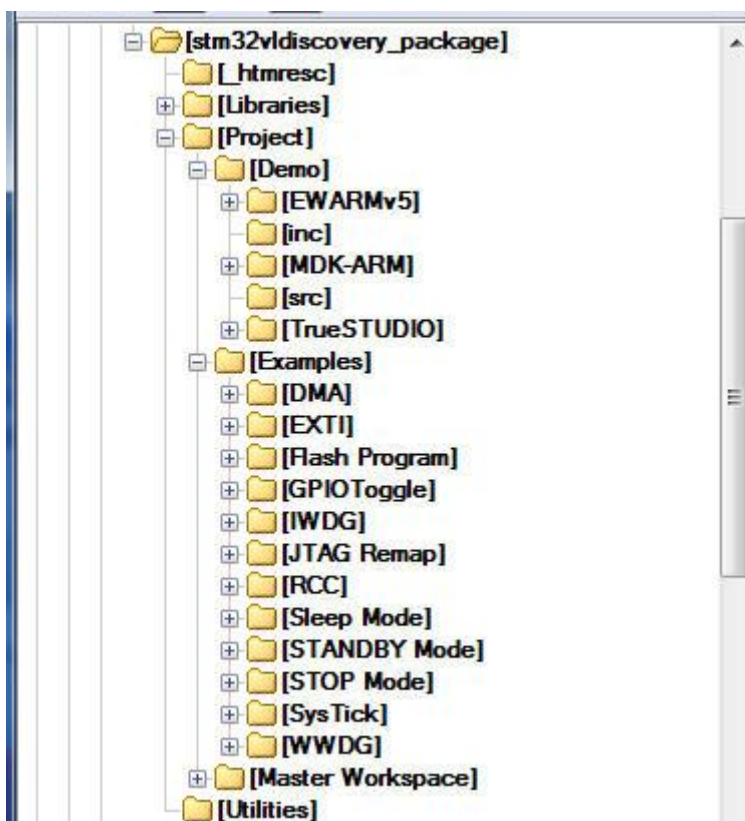


3.3 Začínáme pracovat s existujícím IAR projektem

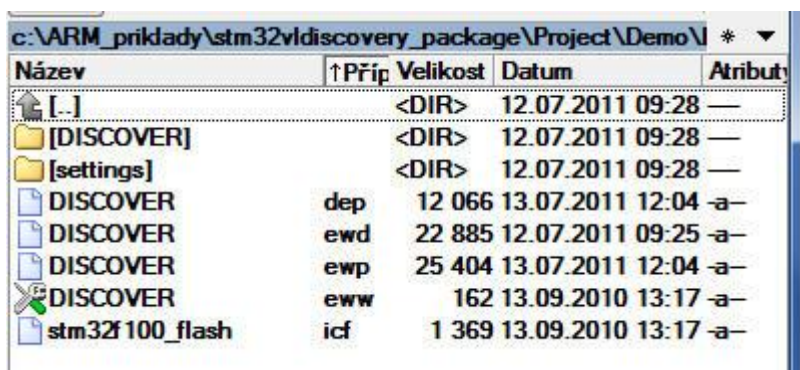
IAR Embedded Workbench IDE se skládá z několika částí:



Naší první ukázkou práce s *IAR Embedded Workbench* bude práce s již existujícím projektem **Demo** otevíraným pomocí souboru **DISCOVER.eww**. Najdeme jej v **stm32vldiscovery_package**



v podadresáři **Projects** , v jeho podsložce **Demo**, v podsložce **EWARMv5**



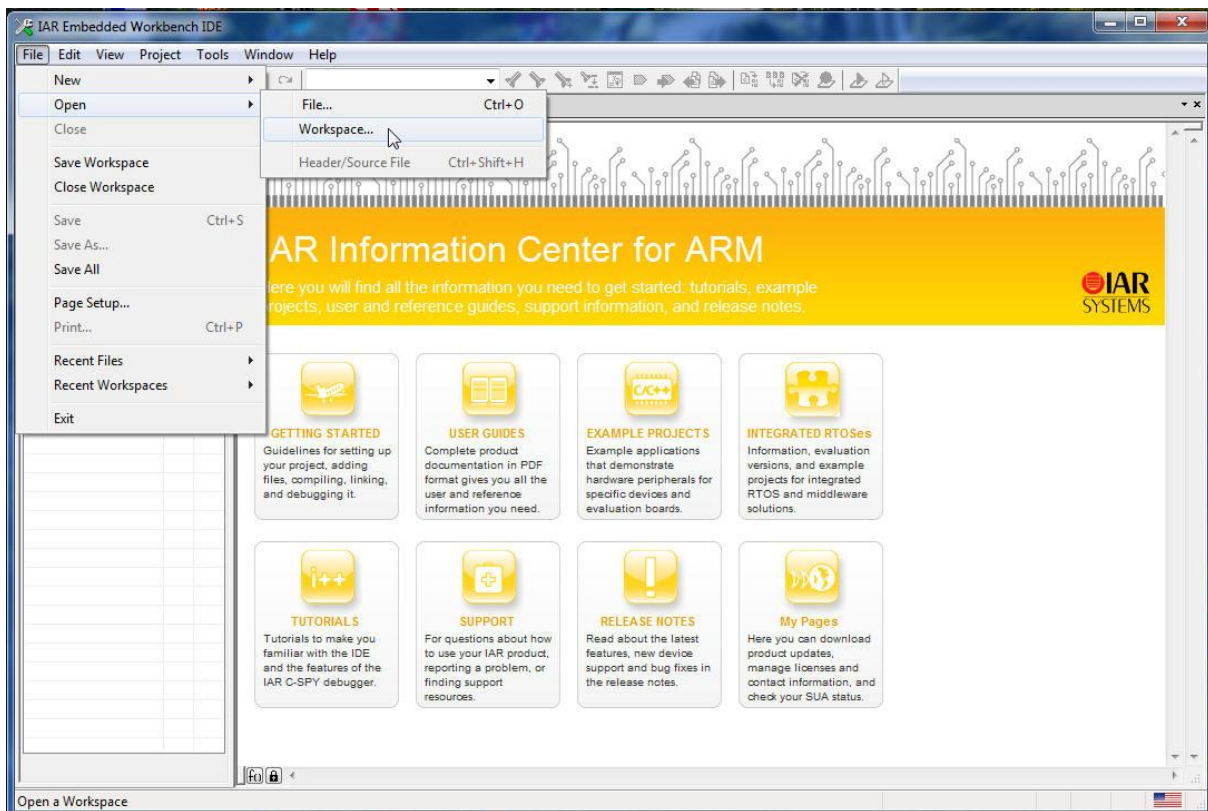
Všimněme si, že adresář **Demo** obsahuje podadresáře **EWARMv5**, **MDK-ARM** a **TrueSTUDIO** a že stejné podadresáře najdeme i ve všech podadresářích v příkladech **Examples**. Tím spolu s umístěním společných adresářů **Libraries** a **Utilities** a dílčích adresářů **src** a **inc** je docíleno toho, že tyto adresáře jsou společné pro projekty v prostředí **IAR Embedded Workbench**, stejně jako v prostředí **Keil uVision4** a v prostředí **Atollic TrueSTUDIO**.

V těchto prostředích najdeme i nastavení cest k souborům v výše uvedených adresářích. Dobře nám poslouží jako vzor pro nastavování cest ve vlastních projektech.

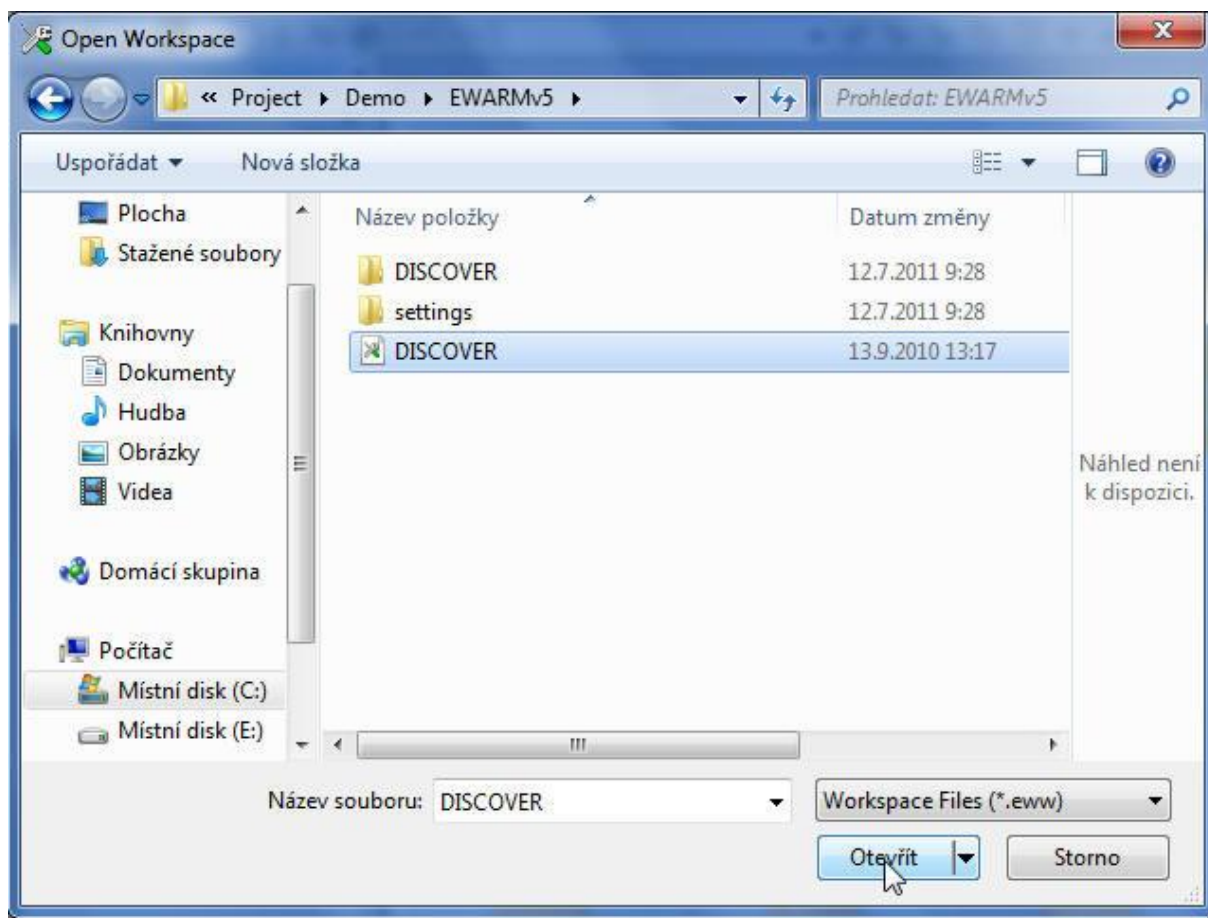
Balíček **stm32vldiscovery_package** získáme <http://www.st.com/stm32vldiscovery> v dolní části **Design Support** <http://www.st.com/internet/evalboard/product/250863.jsp> pod nápisem firmware jako **stm32vldiscovery_package.zip**.

Jeho popis pak najdeme v aplikační poznámce **AN3268 Application Note STM32VLDISCOVERY firmware package** ze <http://www.st.com/stonline/products/literature/an/17901.pdf>

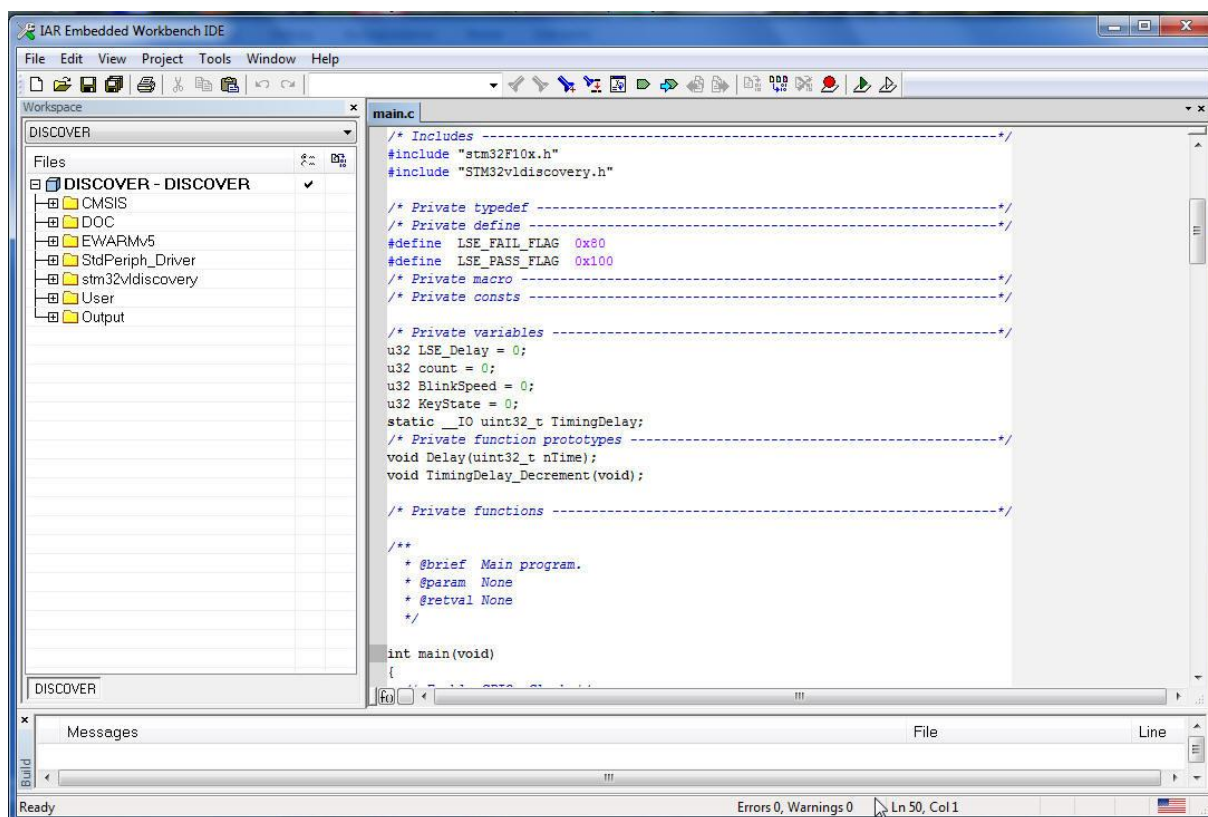
Po stažení **stm32vldiscovery_package.zip** z tohoto souboru nejprve získáme adresář **stm32vldiscovery_package**, který umístíme do nějaké složky s našimi projekty. Poté spustíme **IAR Embedded Workbench** a v něm v menu **File – Open -- Workspace**



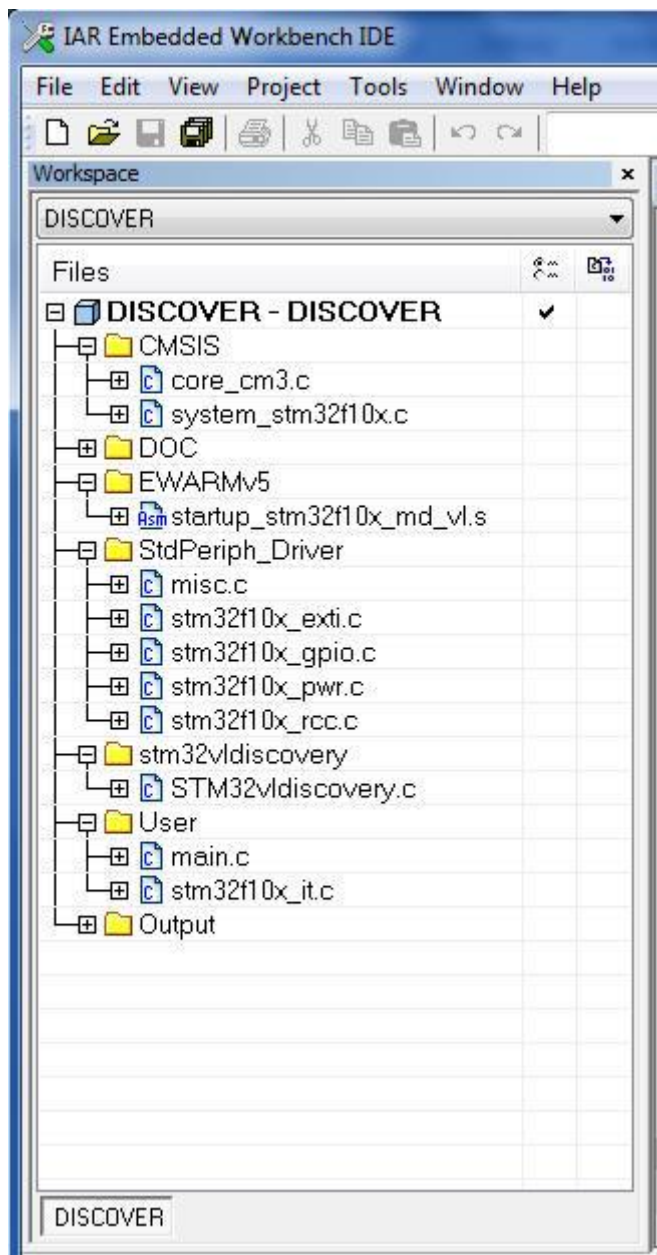
Poté vybereme workspace **DISCOVER**



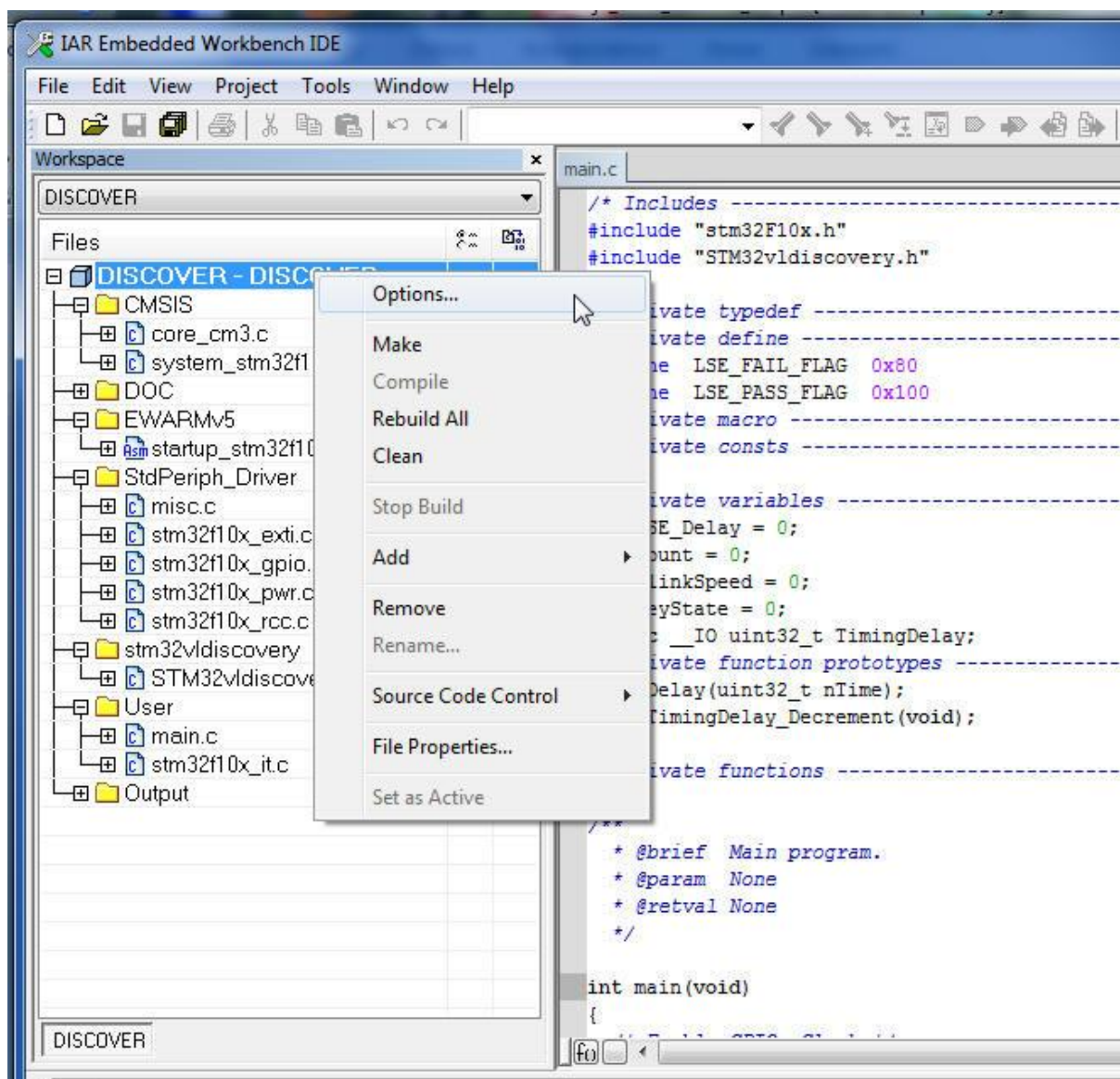
Klikneme na **Otevřít** , dostaneme



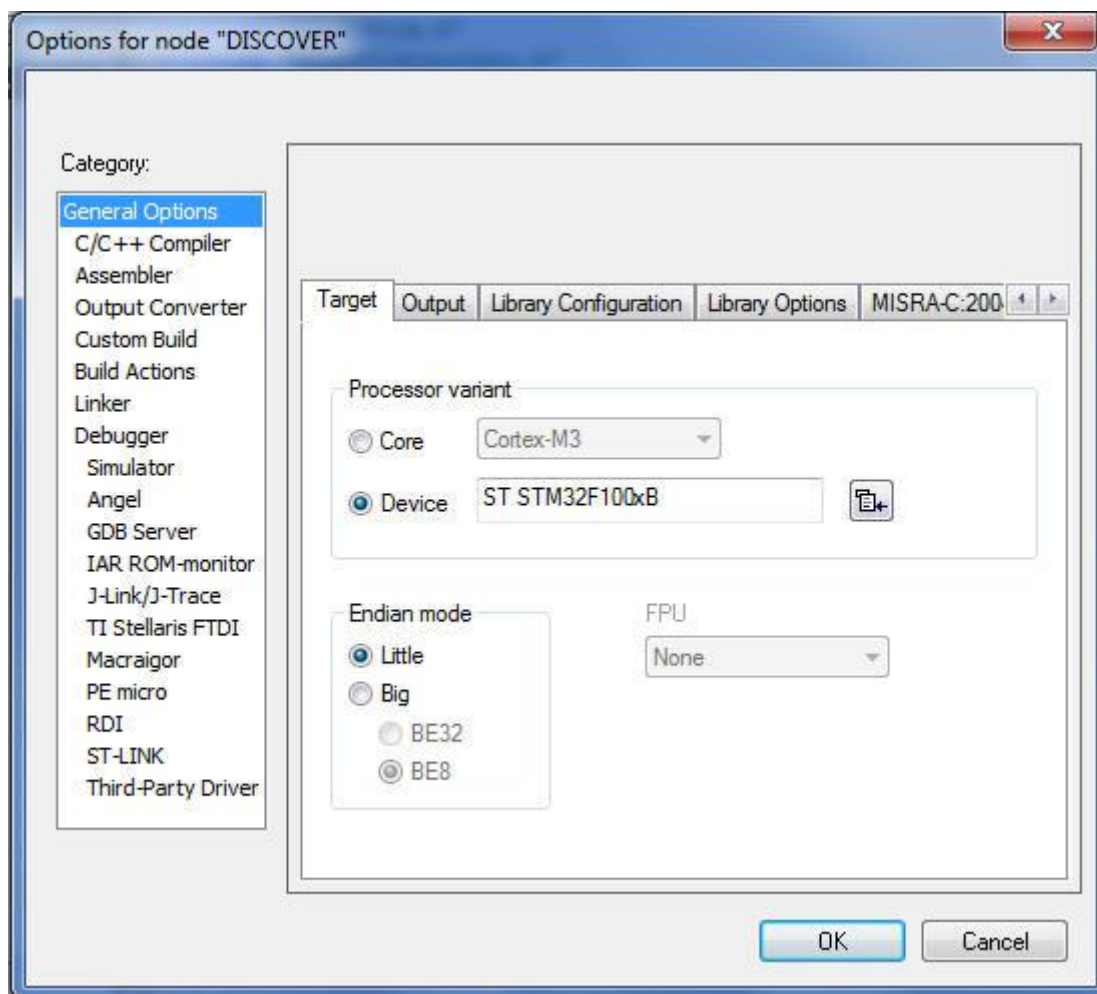
V levé části si všimneme obsahu okna souborů



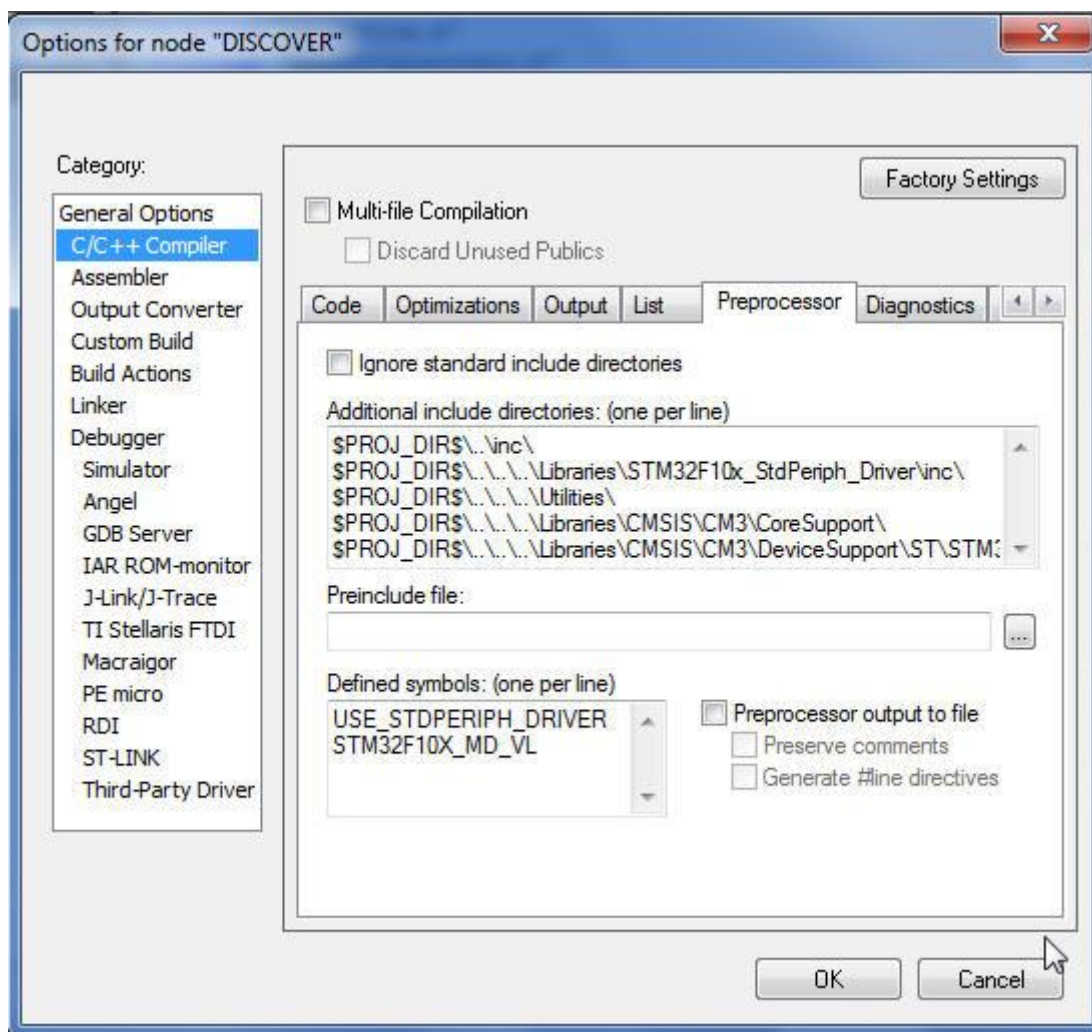
V něm označíme text **DISCOVER – DISCOVER** a pravým tlačítkem rozvineme místní menu



V něm vybereme položku **Options**, dostaneme okno se záložkami. Jednotlivé záložky otevíráme a nastavujeme jejich obsah dle následujících obrázků.



Vybereme CPU



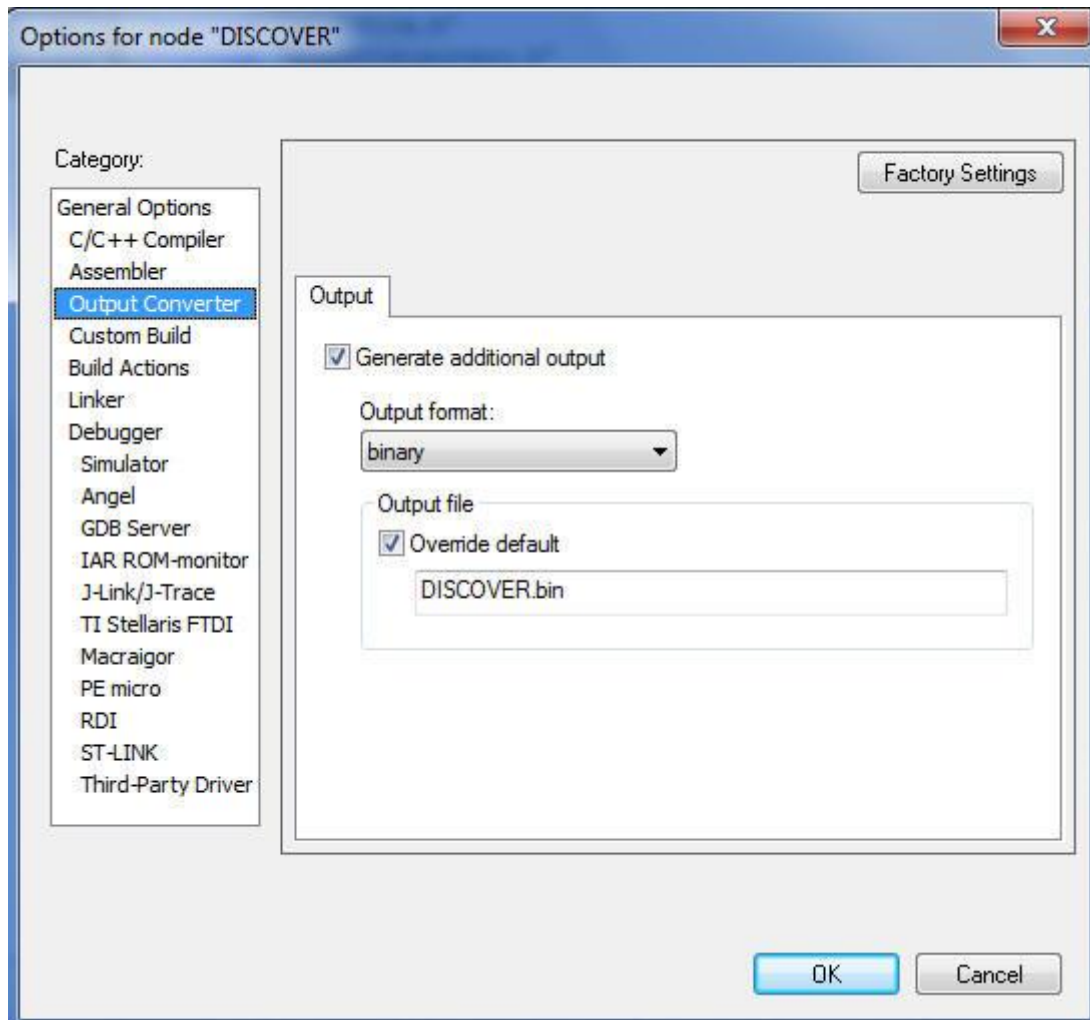
Důležité pro budoucí projekty (budeme upravovat)

Additional include directories:

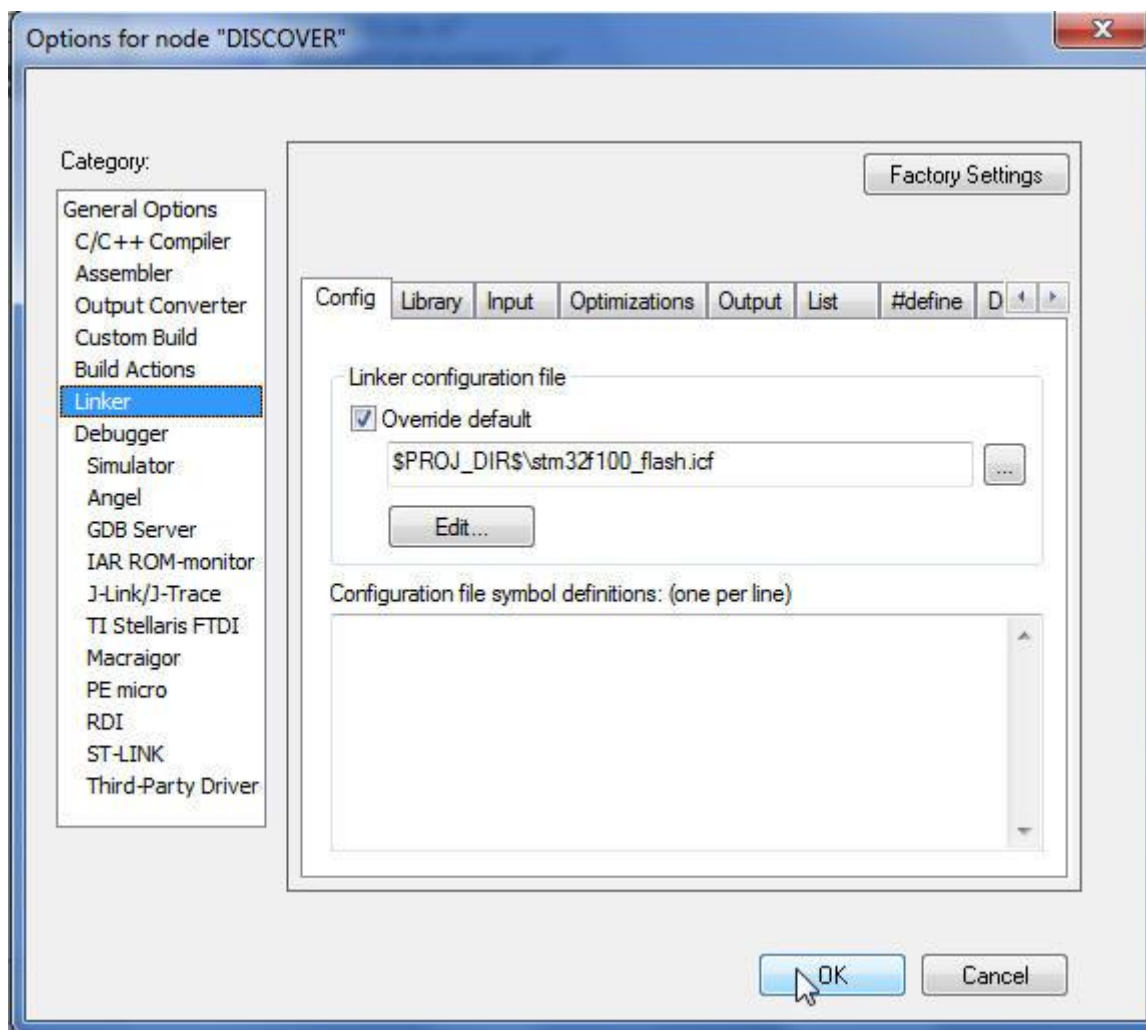
```
$PROJ_DIR$..\inc\
$PROJ_DIR$..\..\Libraries\STM32F10x_StdPeriph_Driver\inc\
$PROJ_DIR$..\..\Utilities\
$PROJ_DIR$..\..\Libraries\CMSIS\CM3\CoreSupport\
$PROJ_DIR$..\..\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
```

Defined symbols:

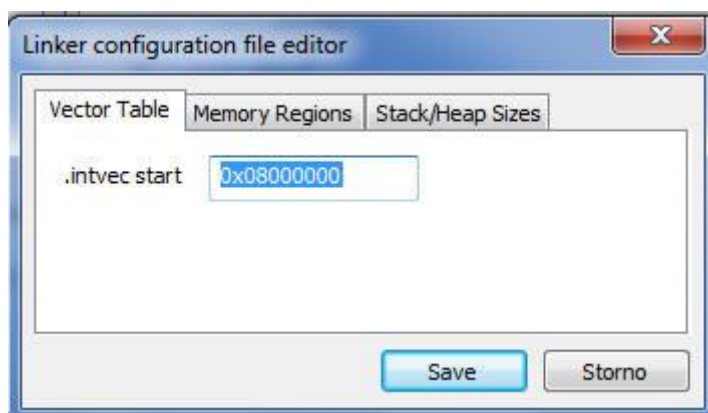
```
USE_STDPERIPH_DRIVER
STM32F10X_MD_VL
```

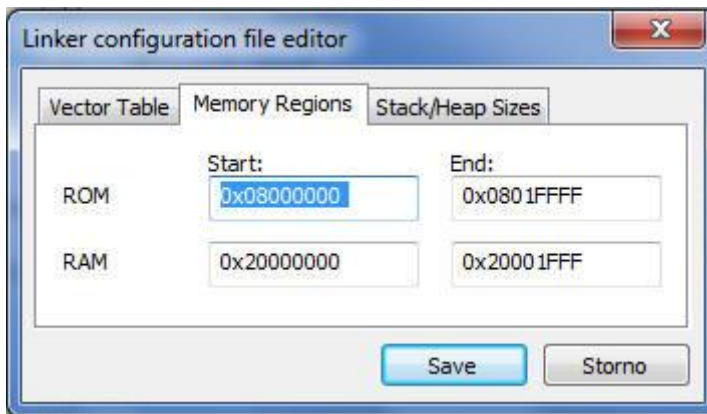


Zvolíme formát výstupního souboru bingy a klikneme na **OK**

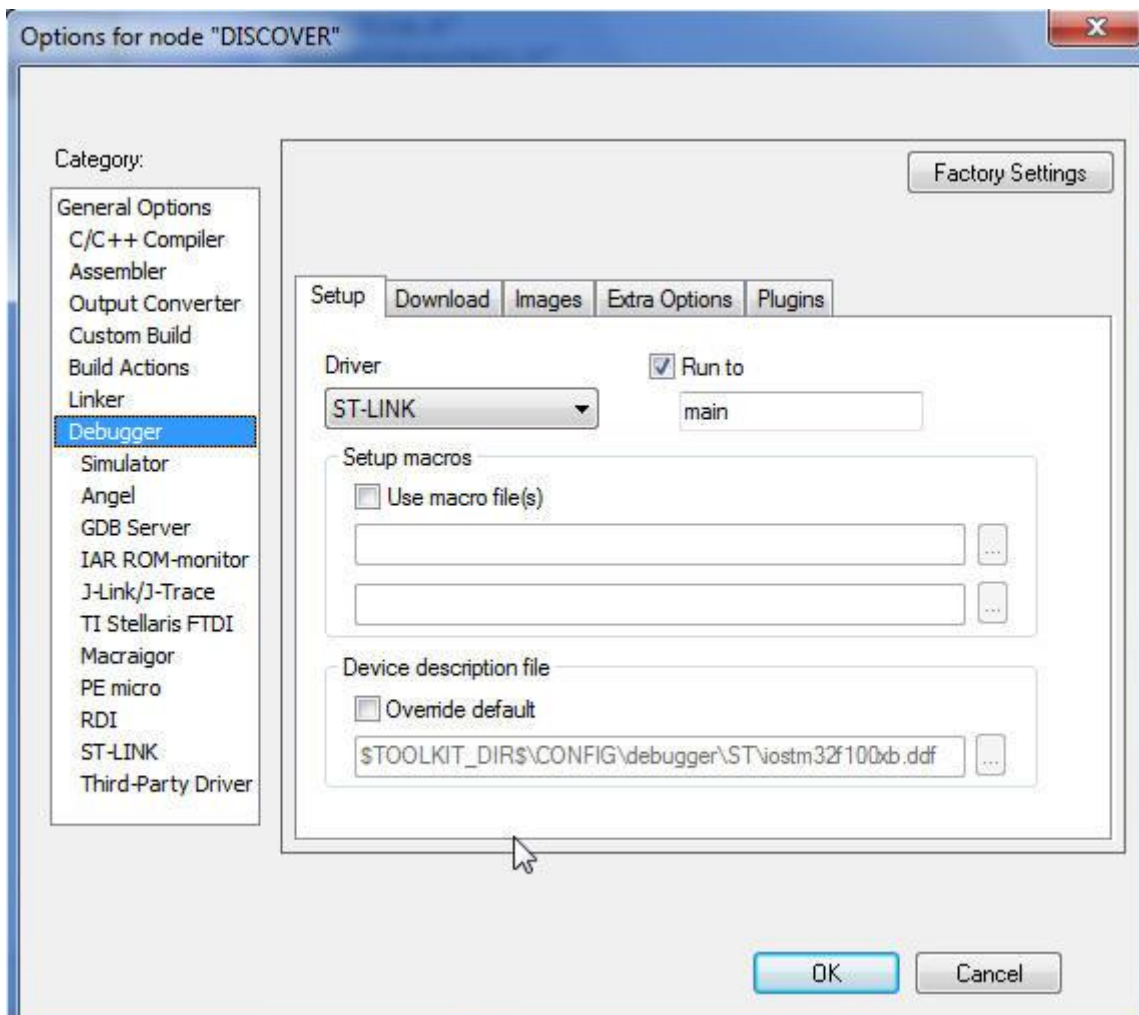


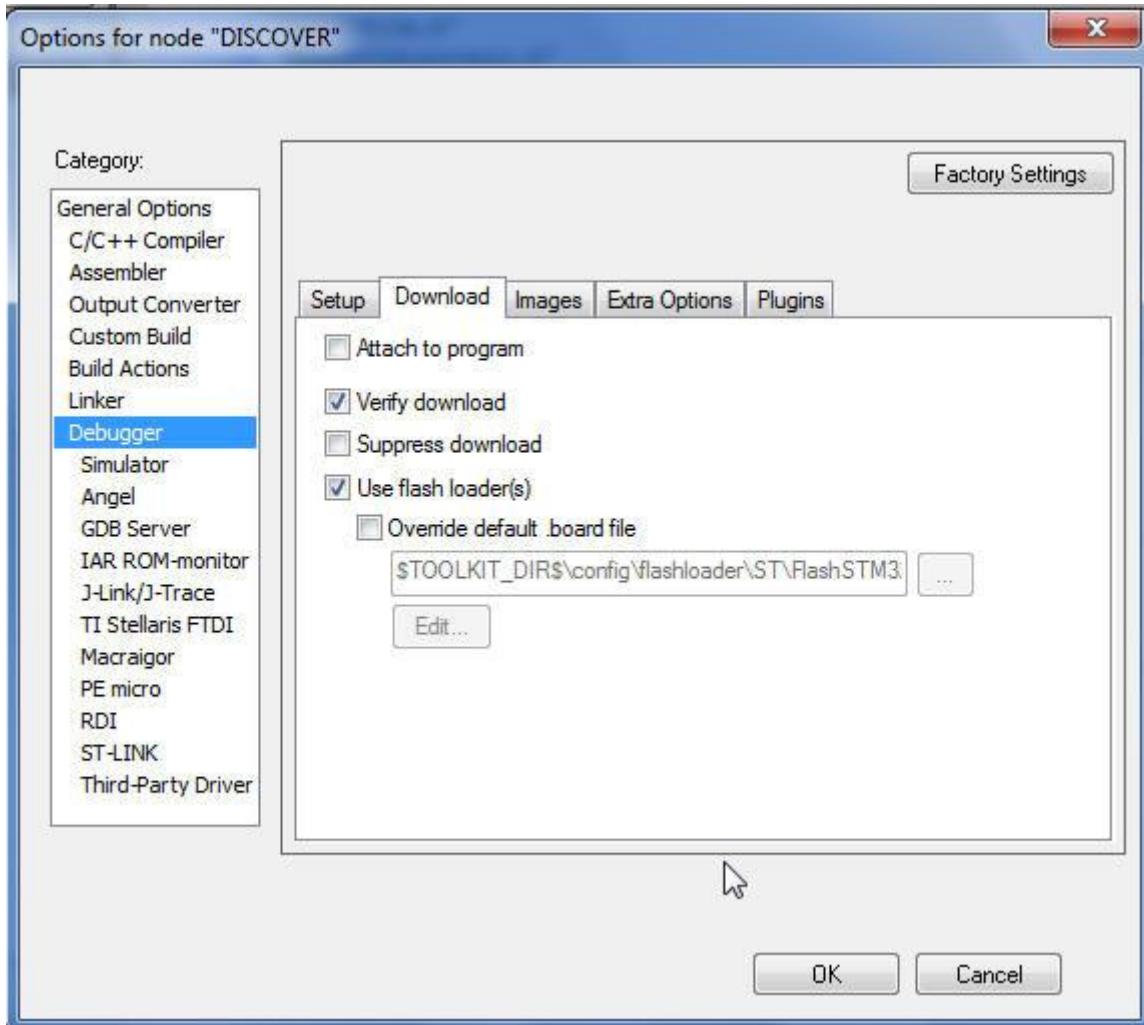
Ještě konfigurace linkeru



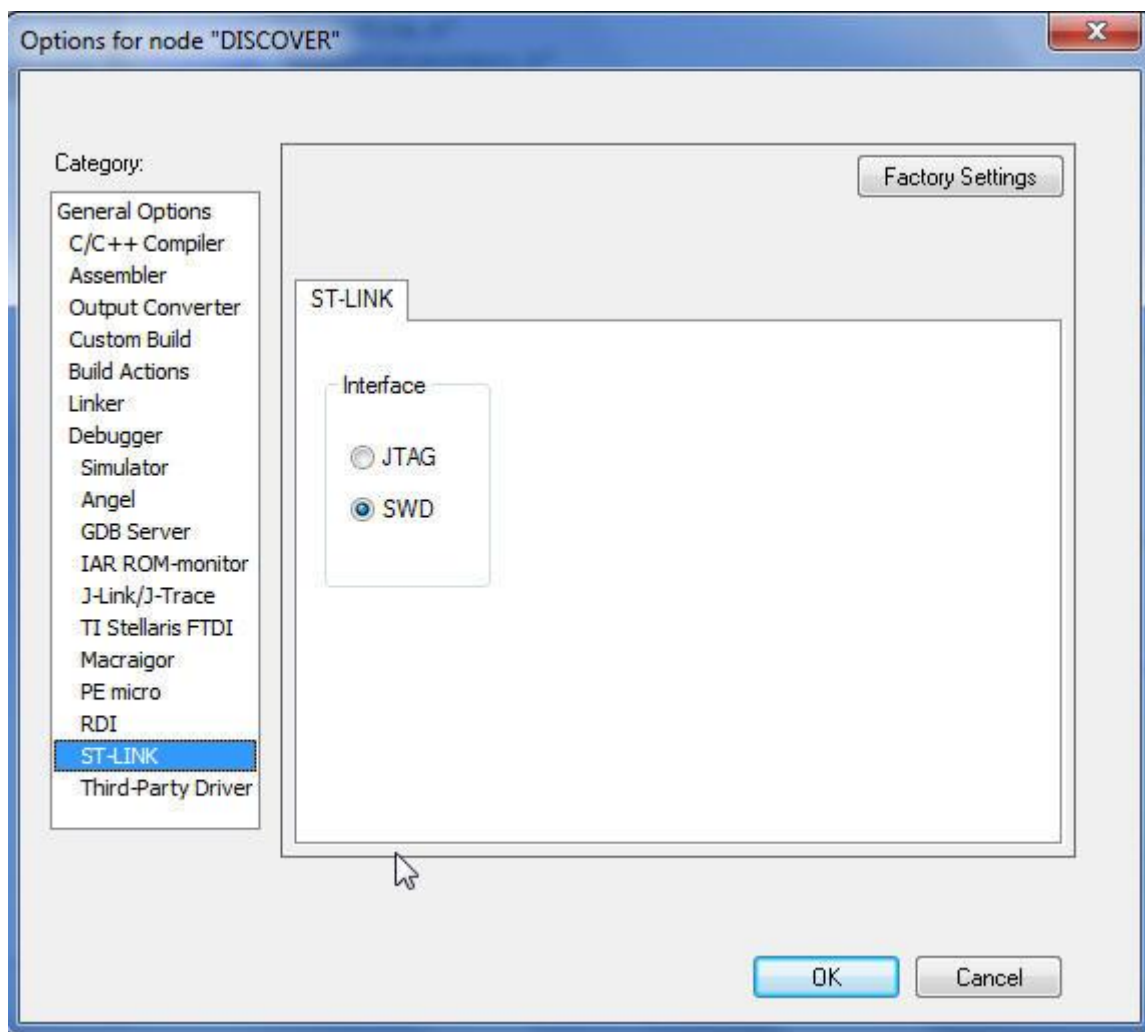


A nastavení **Debuggeru** na **ST-Link**

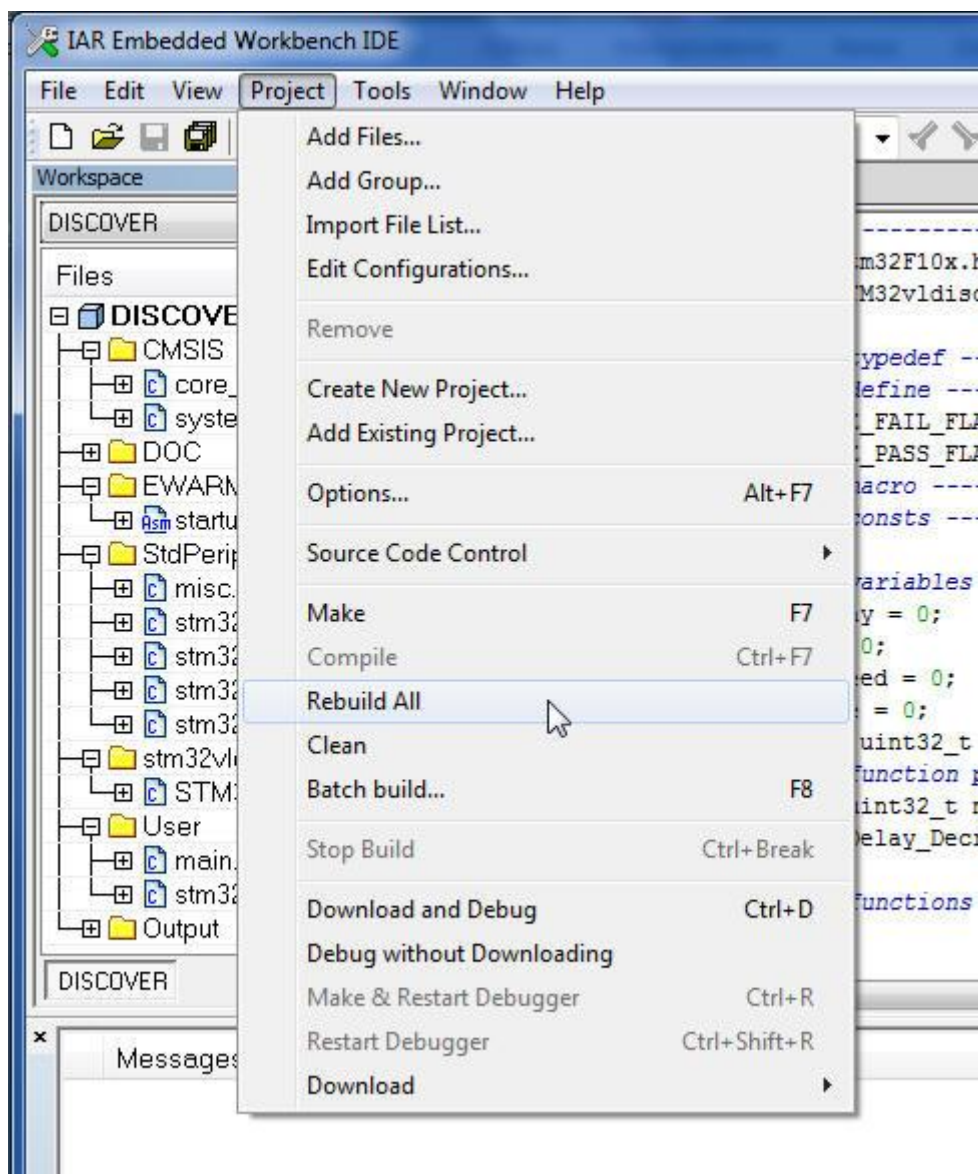




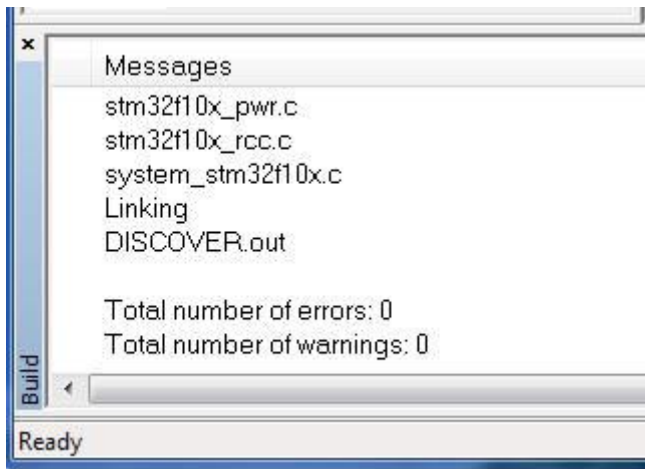
Nezapomeneme na **SWD**



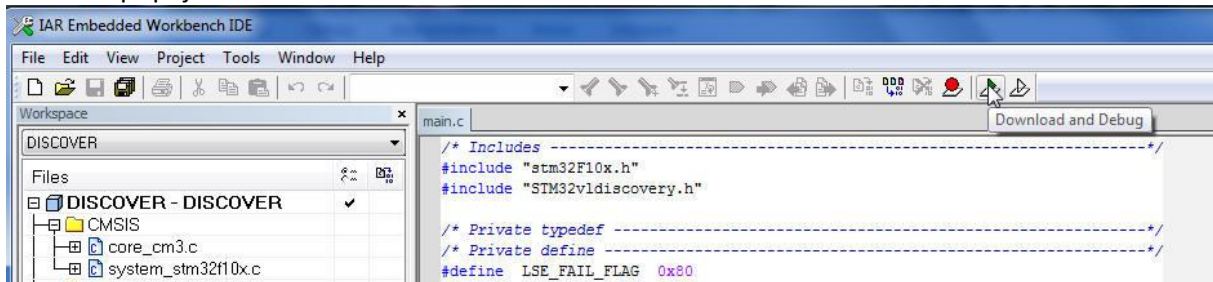
Nyní máme prohlédnutá veškerá nastavení, která s výhodou použijeme později ve vlastních projektech. Můžeme přistoupit k překladu, linkování a sestavení.



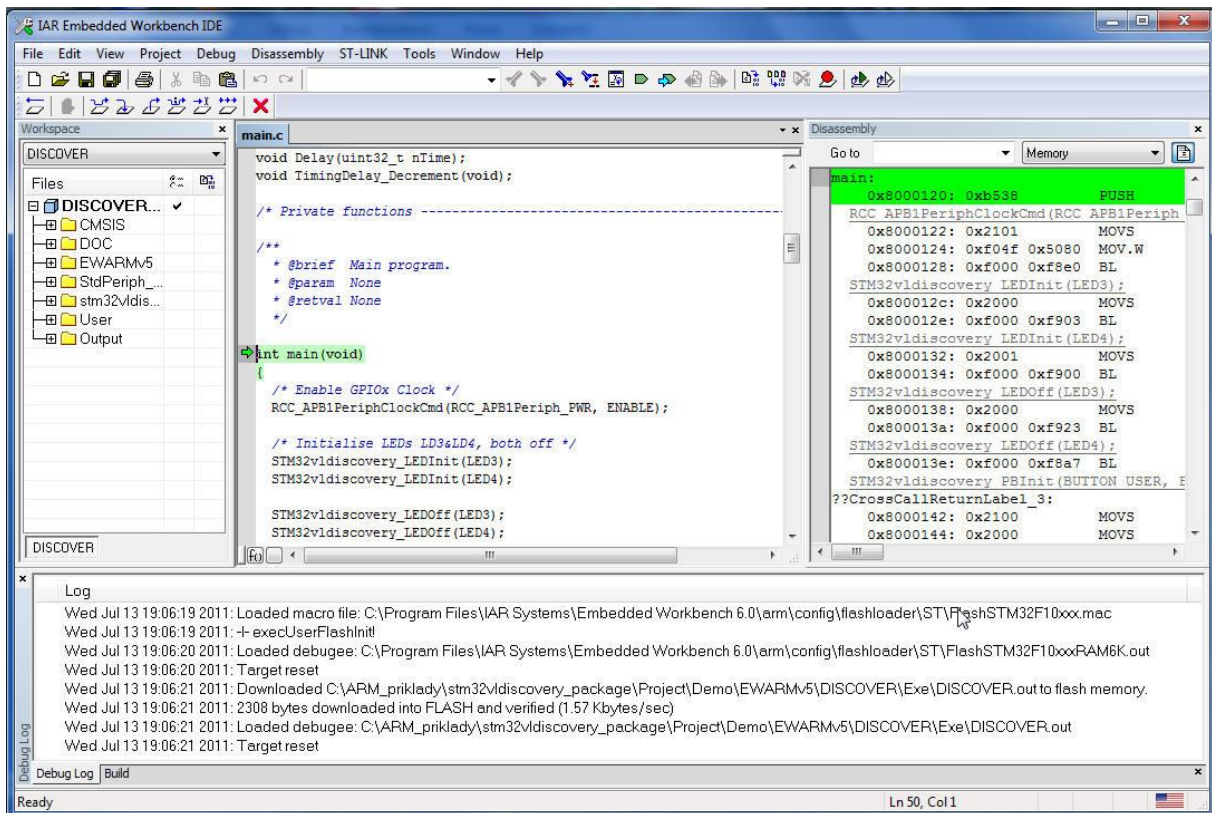
V menu proto vybereme **Project – Rebuild All** . Po chvíli dostaneme



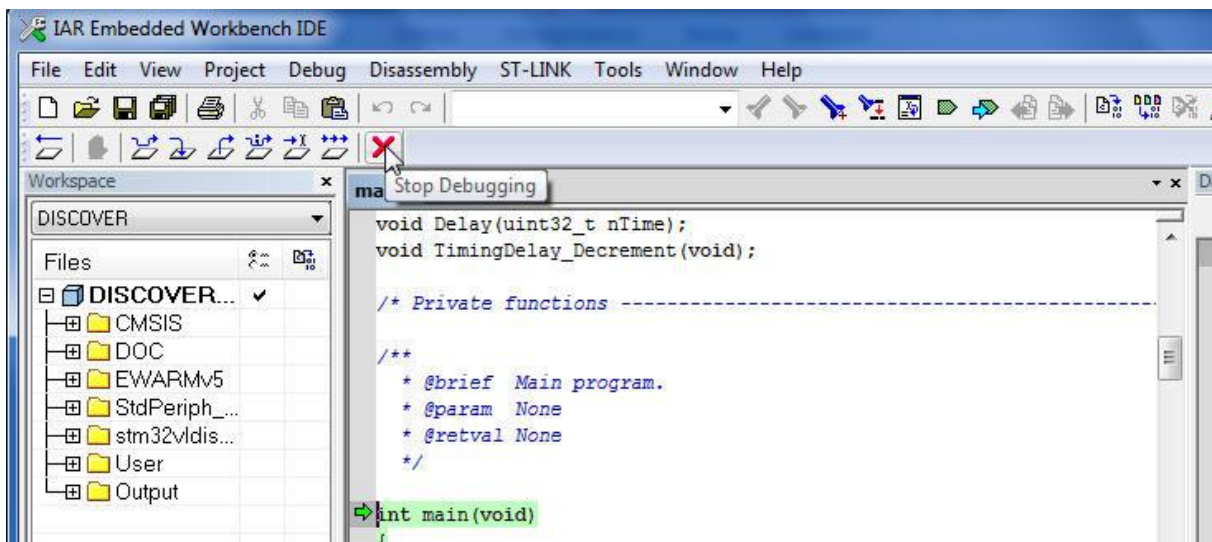
Přes USB připojíme startkit a



Po chvíli dostaneme

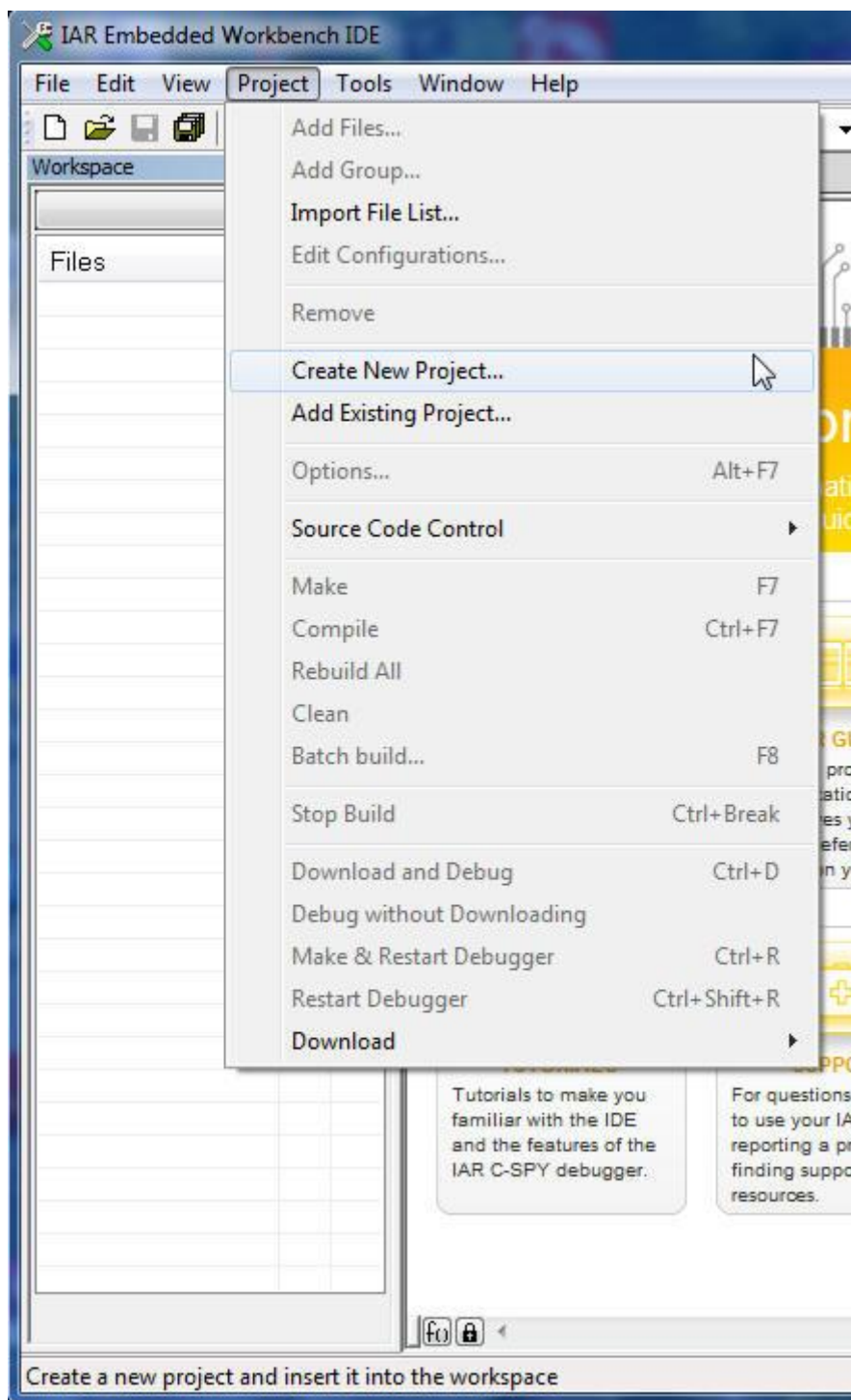


Klikneme a ikonku **Stop Debugging**

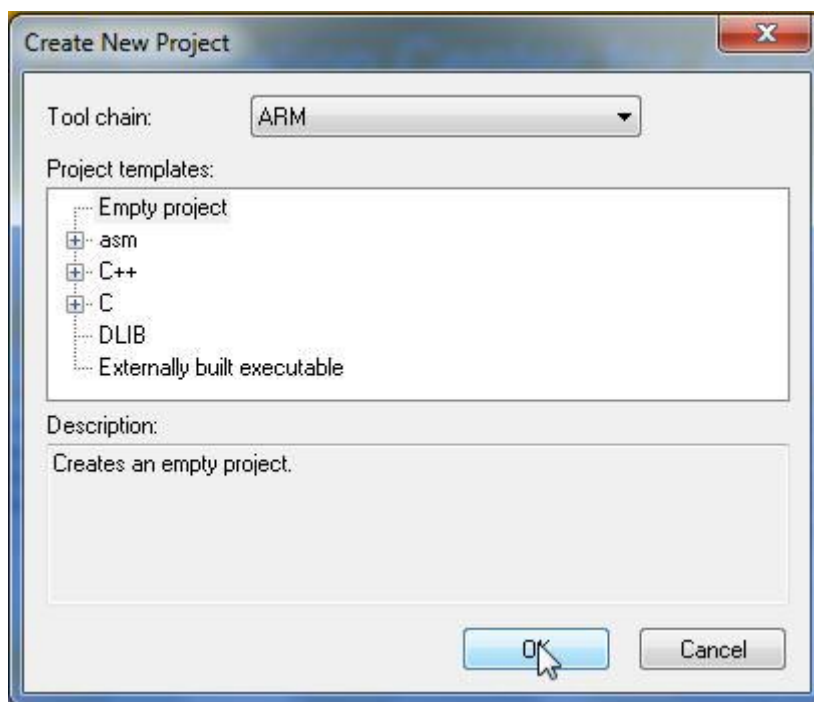


Pokud startkit odpojíme a znovu připojíme, spustí se program – blikání zelené LED PC9. Je to vlastně ten program, se kterým je **STM32VL DISCOVERY** dodán od výrobce.

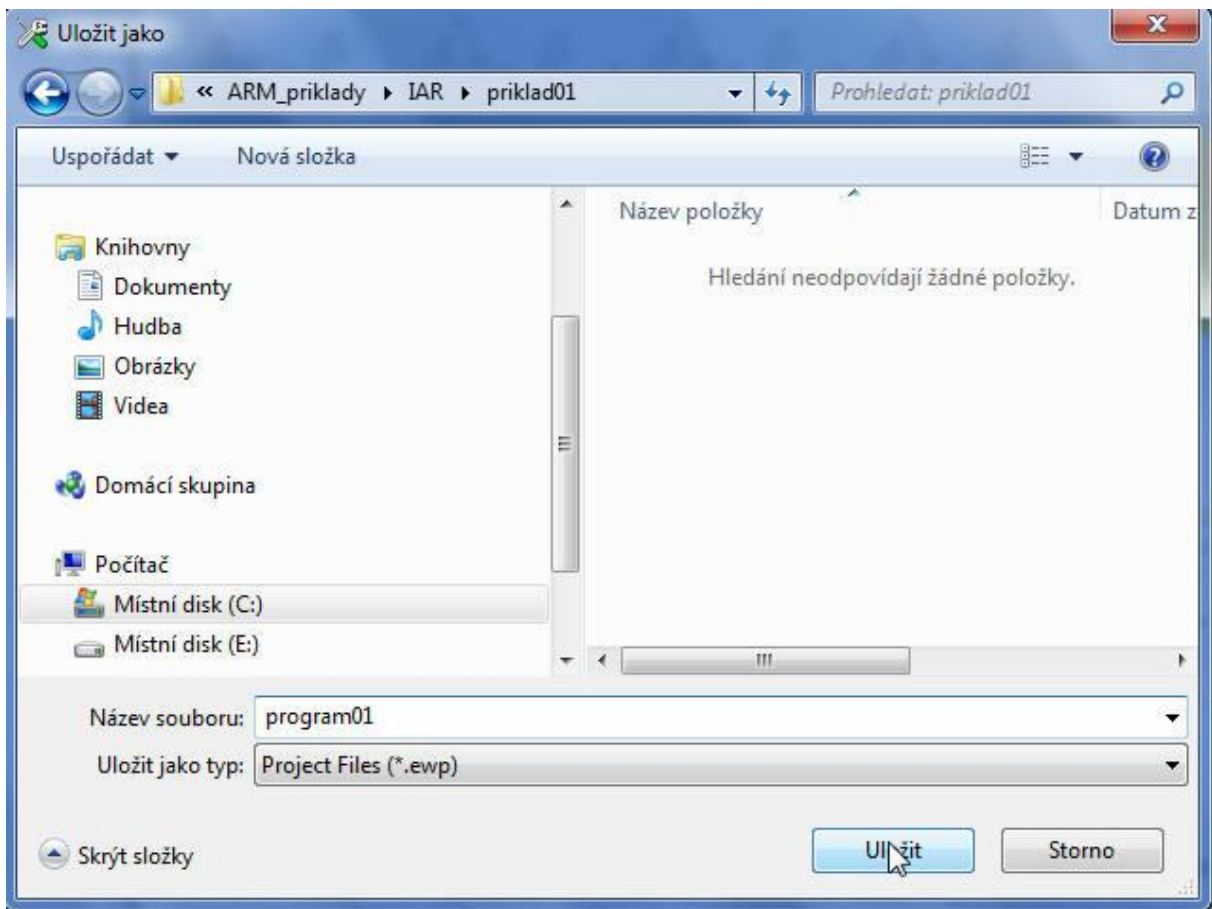
3.4.1 První vlastní IAR projekt (příklad01)



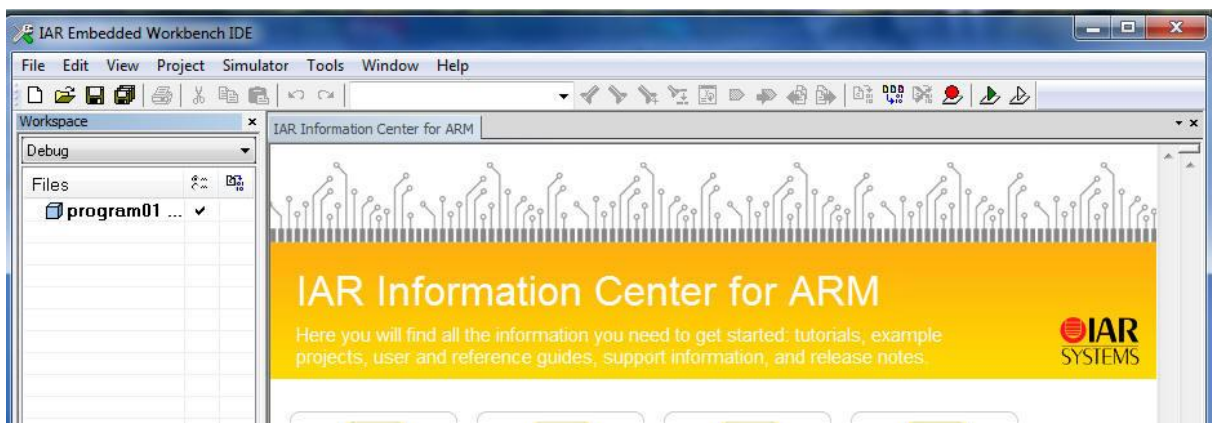
V menu vybereme **Project – Create New Project ...**



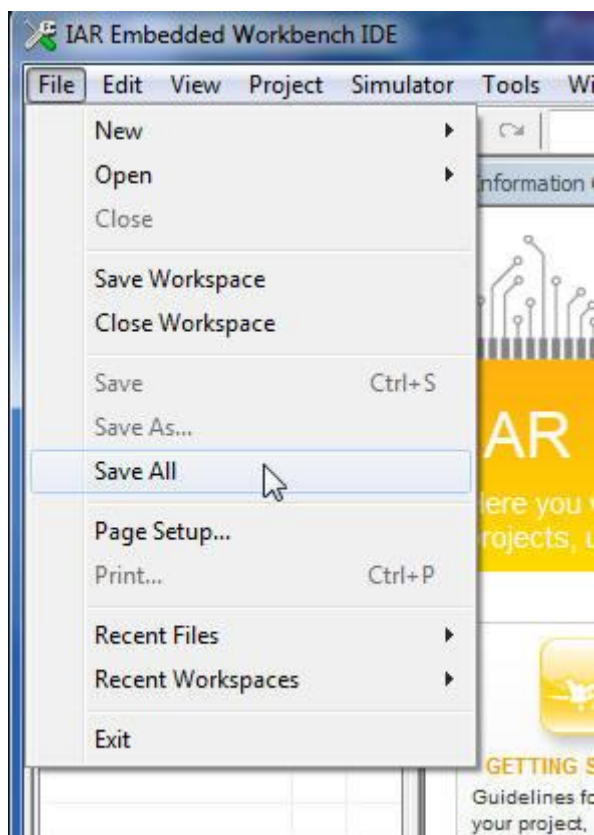
Vybereme ARM a klikneme na **OK**



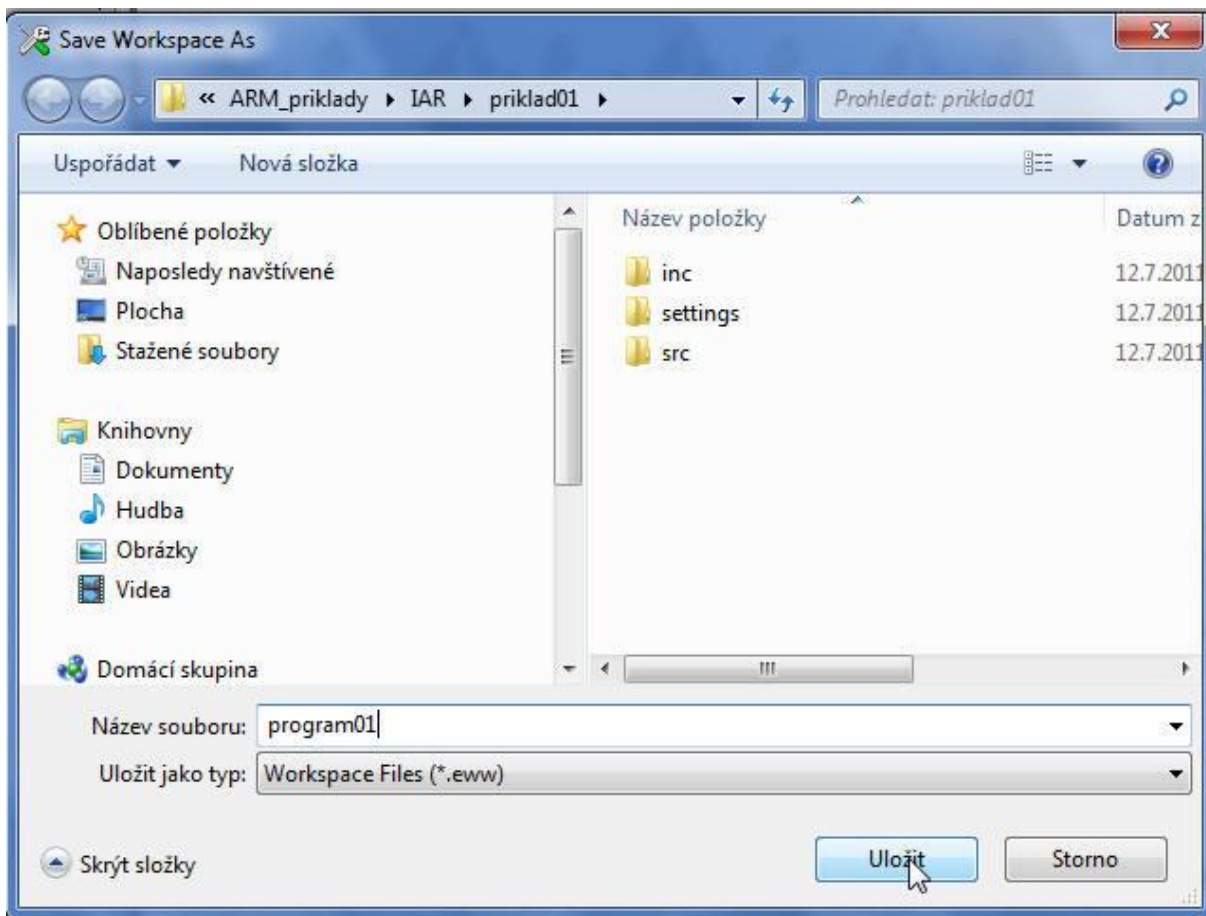
Zvolíme název projektu



Nyní projekt uložíme **File -- Save All**



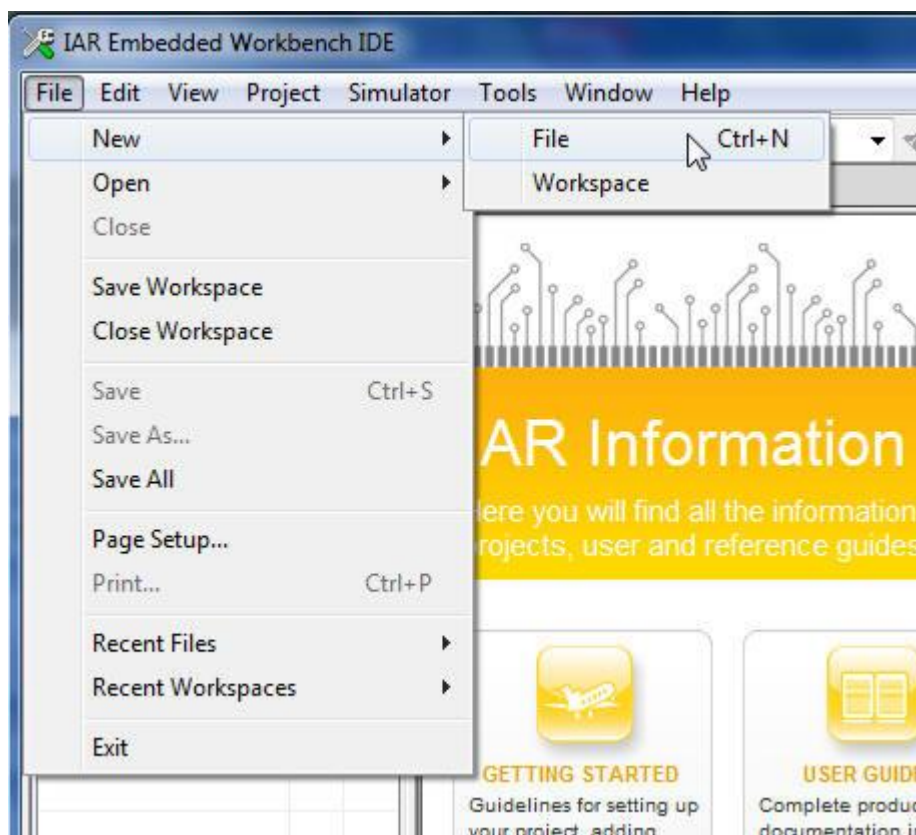
Jsme dotázáni na název workspace



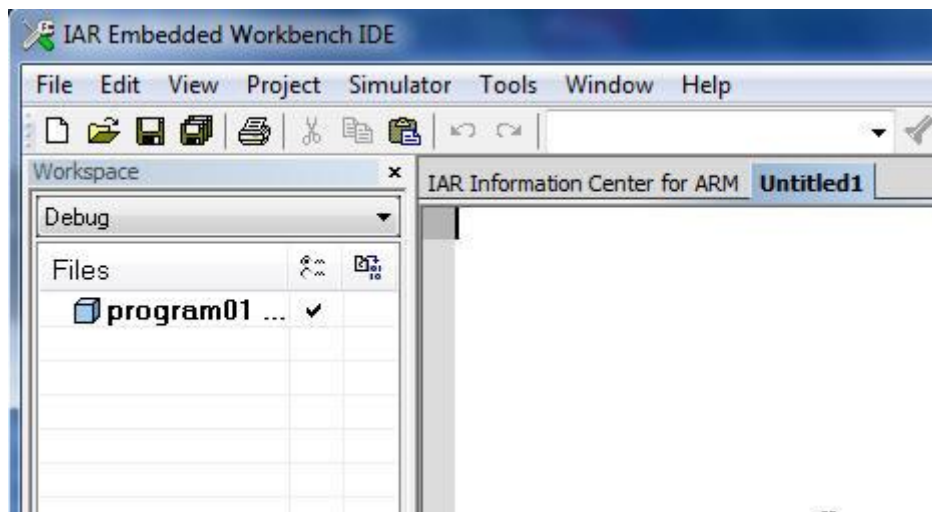
Zvolíme např. program01 a klikneme na **Uložit**. Podíváme se na vytvořené soubory a adresáře projektu

| Název | Přípona | Velikost | Datum | Atribut |
|------------|---------|----------|------------------|---------|
| [.] | <DIR> | | 12.07.2011 11:40 | — |
| [inc] | <DIR> | | 12.07.2011 11:38 | — |
| [settings] | <DIR> | | 12.07.2011 11:40 | — |
| [src] | <DIR> | | 12.07.2011 11:38 | — |
| program01 | eww | 163 | 12.07.2011 11:40 | a- |
| program01 | ewp | 45 958 | 12.07.2011 11:39 | a- |
| program01 | ewd | 45 477 | 12.07.2011 11:39 | a- |
| program01 | dep | 309 | 12.07.2011 11:40 | a- |

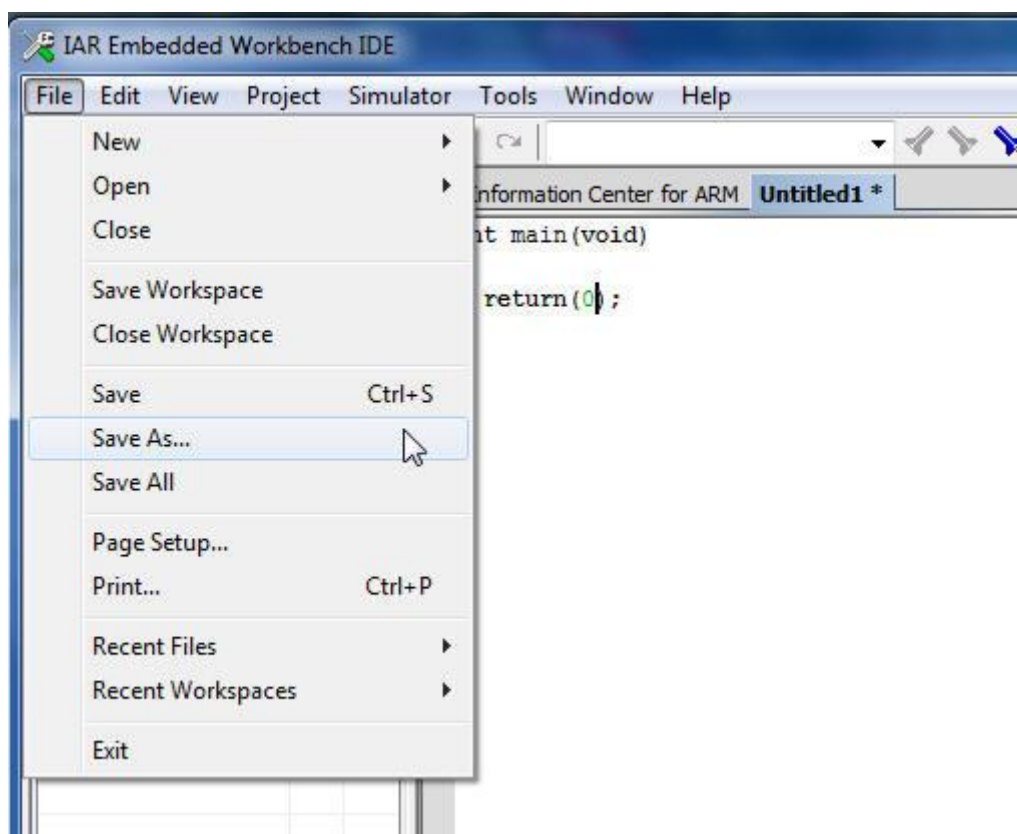
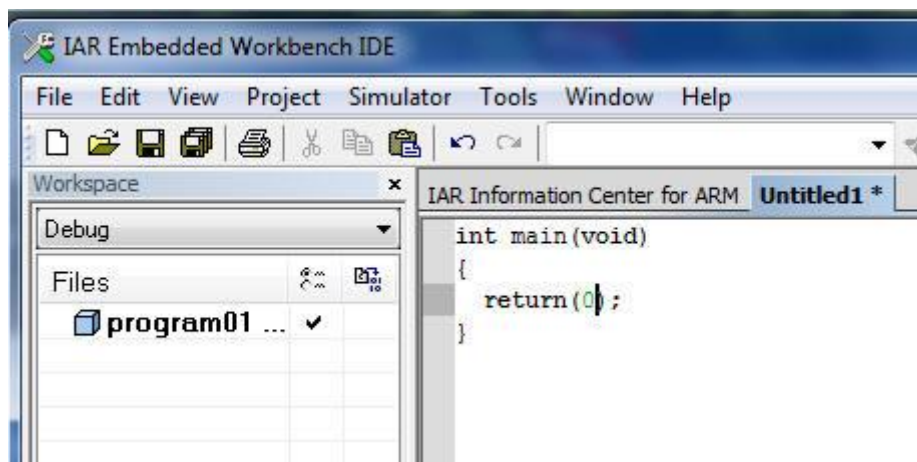
Pozn. Adresáře **inc** a **src** jsem předem vytvořil.



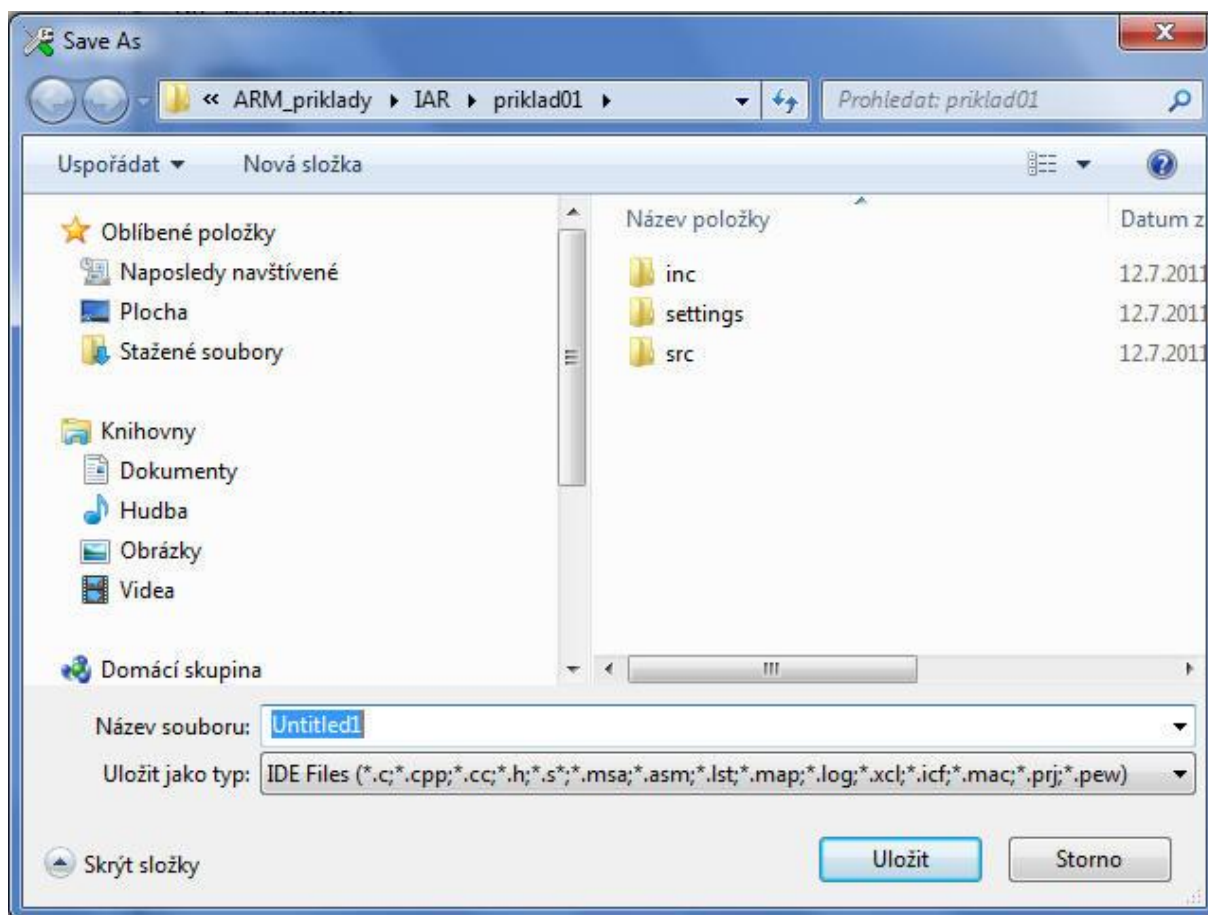
Dále vytvoříme nový soubor výběrem v menu **File – New -- File**



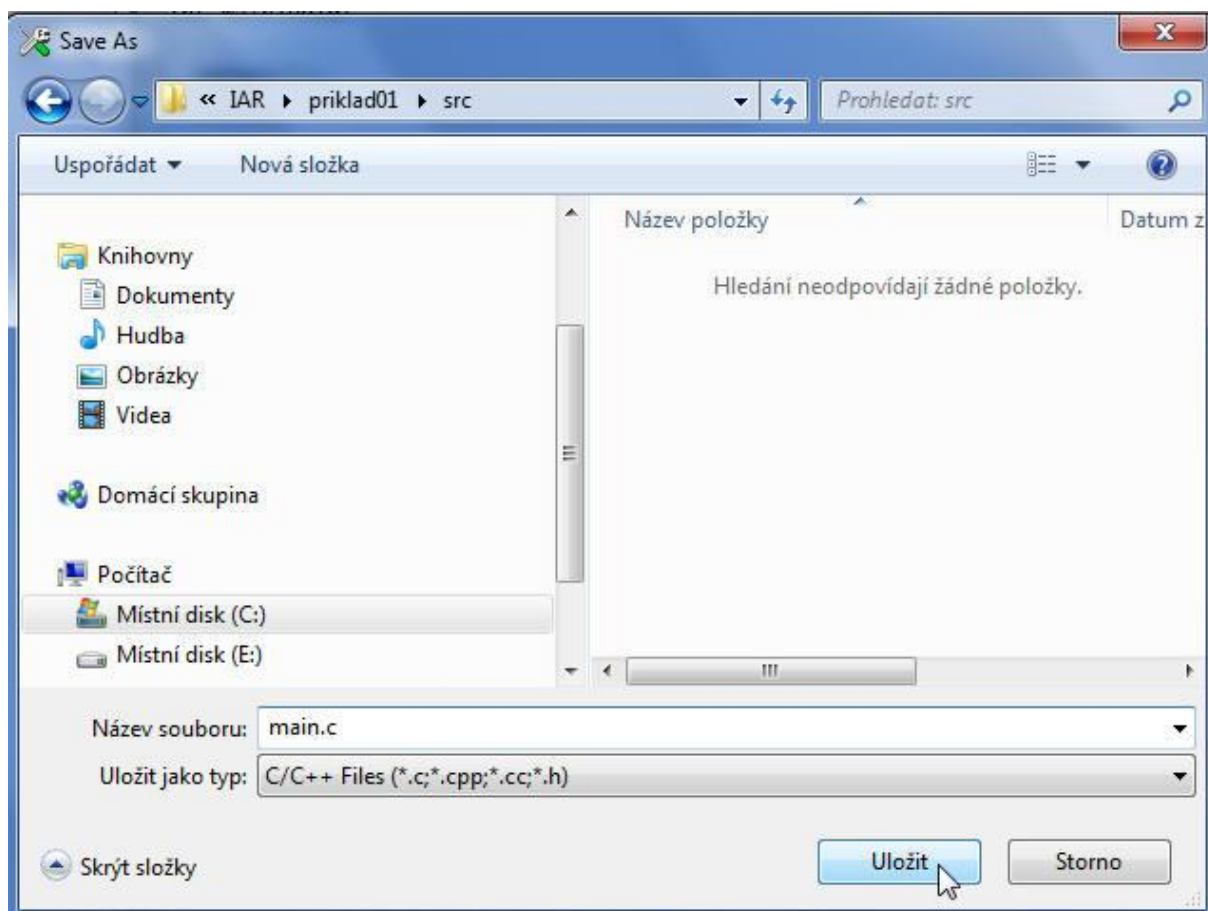
Do **Untitled1** napíšu krátký kód



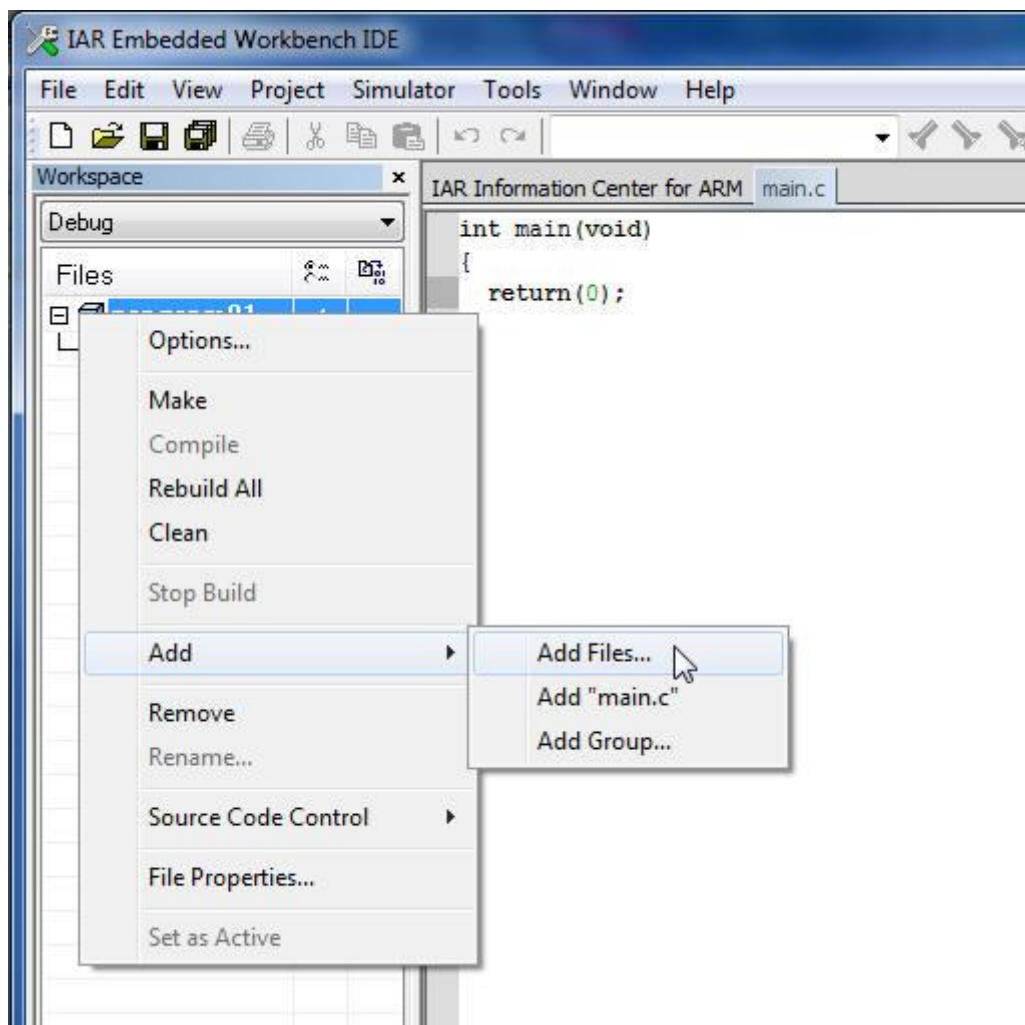
Poté uložíme **File – Save As...**



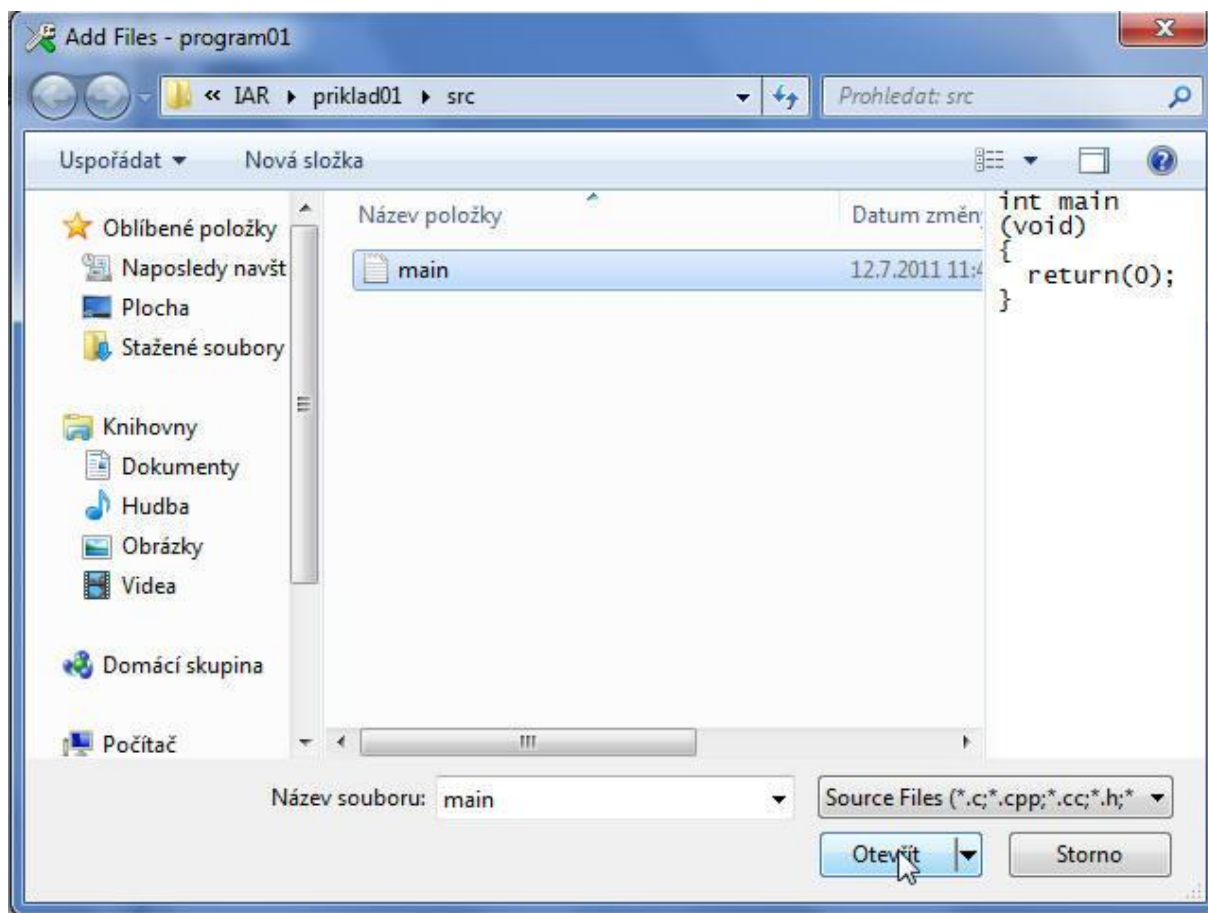
Zvolíme název nového souboru jako **main.c**



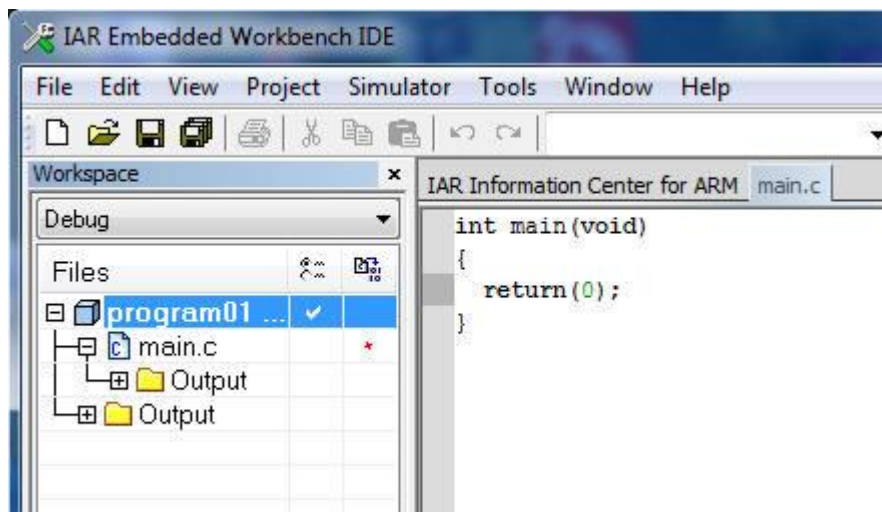
Klikneme na **Uložit**. Dále tento nový soubor přidáme do projektu. Proto název projektu označíme, pravým tlačítkem rozvineme menu

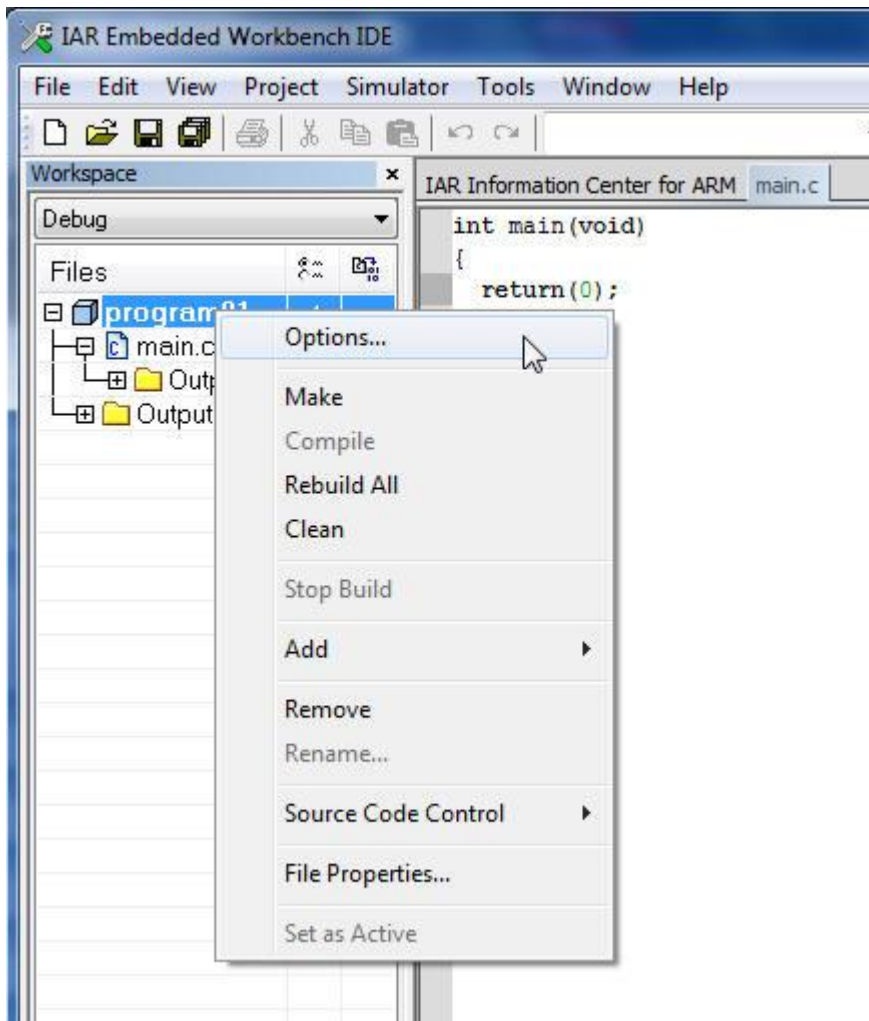


V místním menu vybereme **Add – Add Files...**

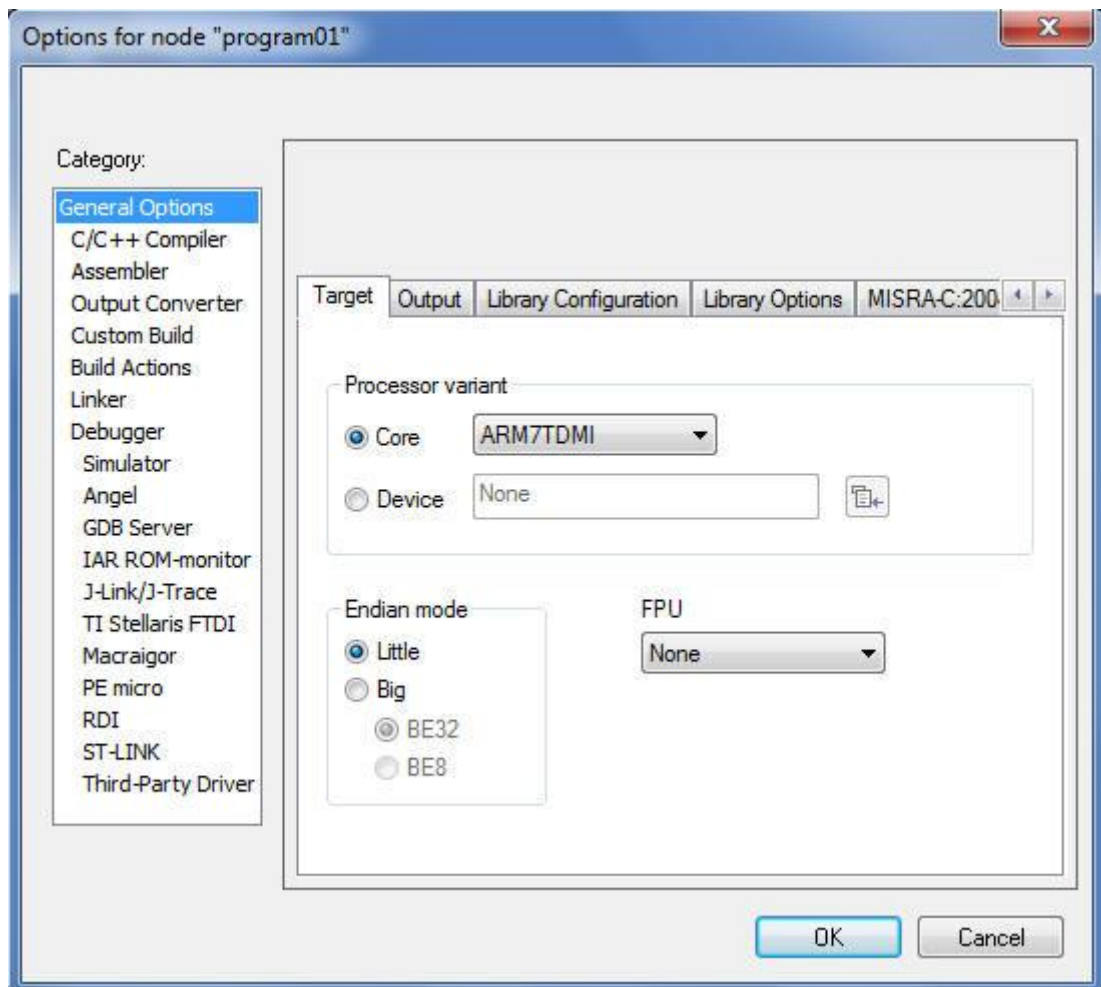


Vybereme soubor **main.c** a klikneme na **Otevřít**

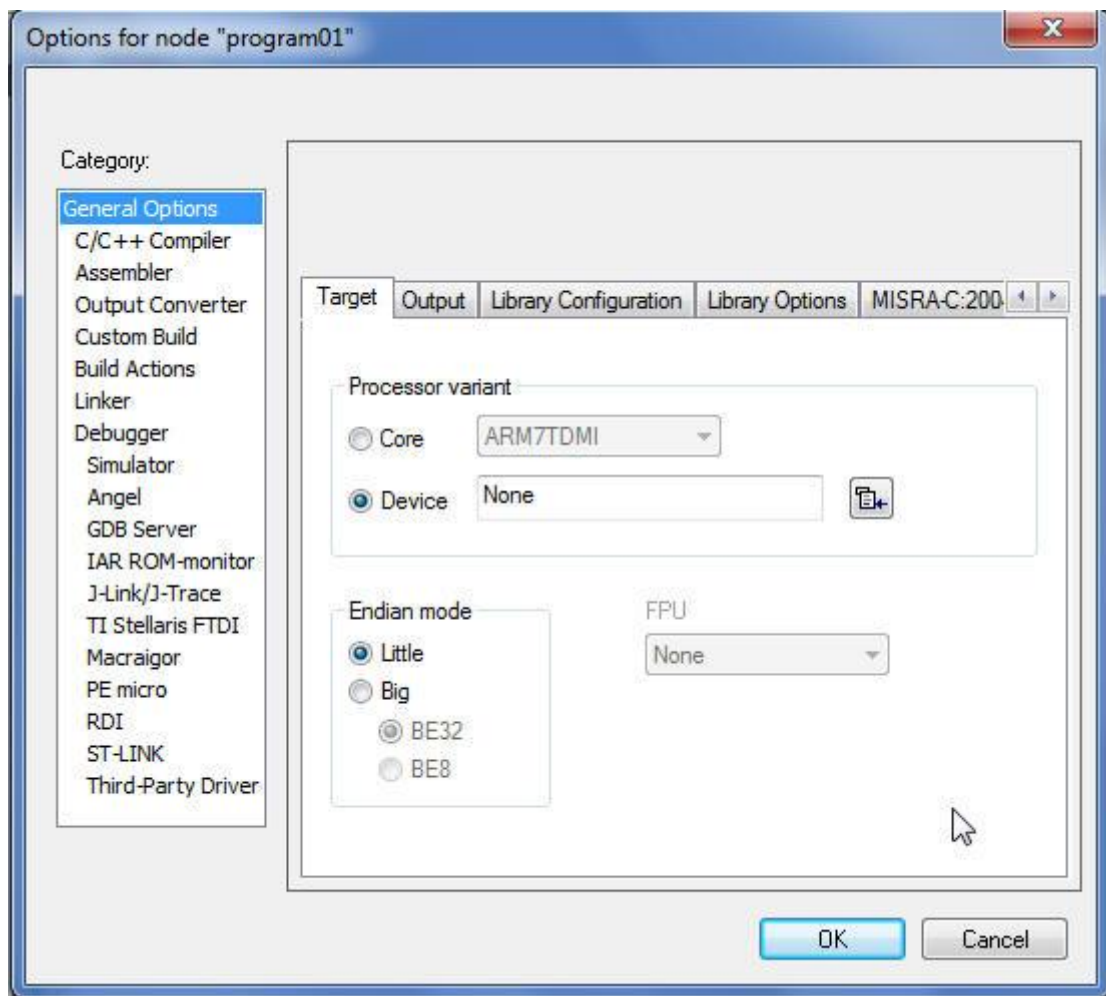




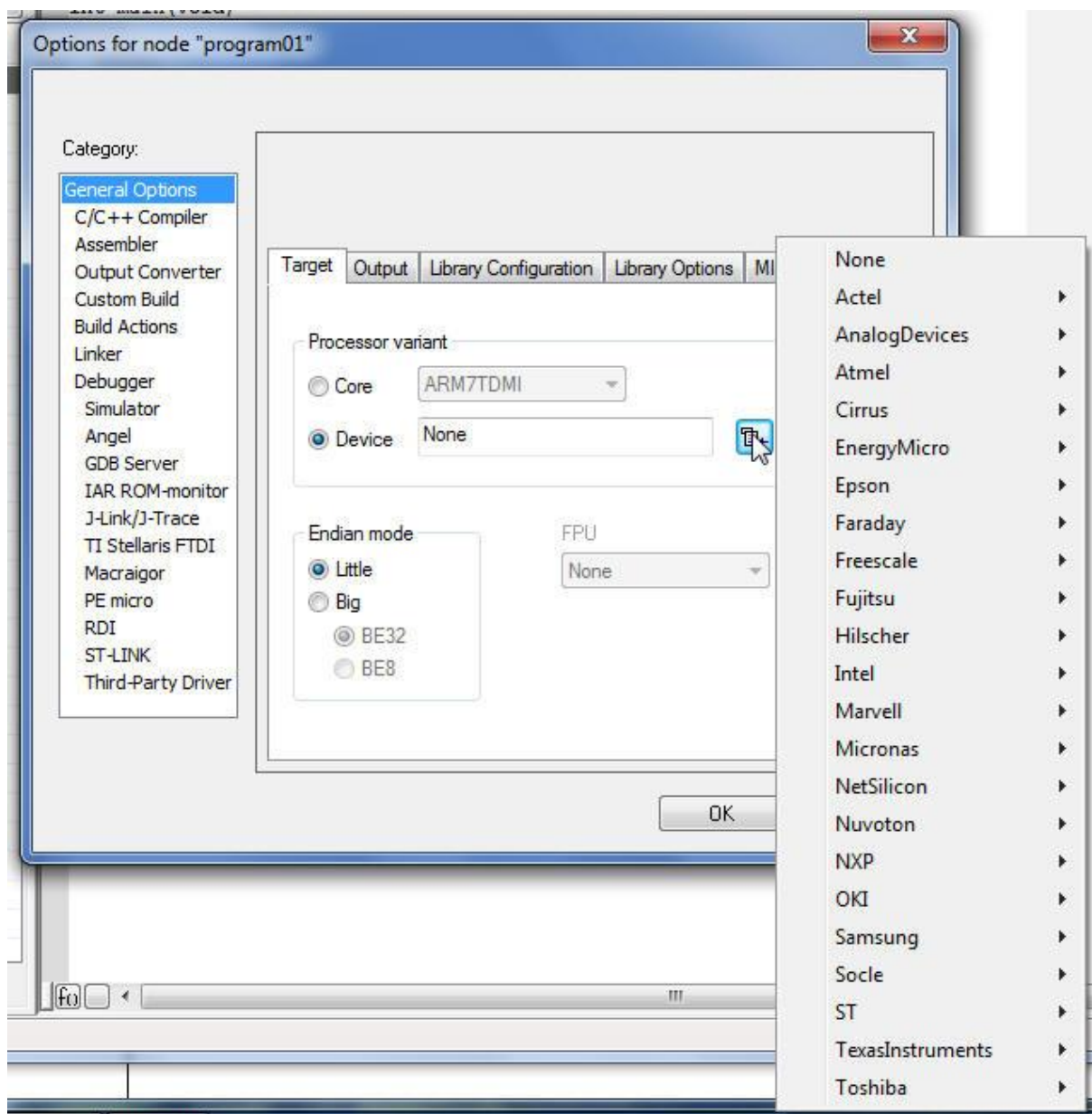
Znovu rozvineme místní menu a vybereme položku **Options**



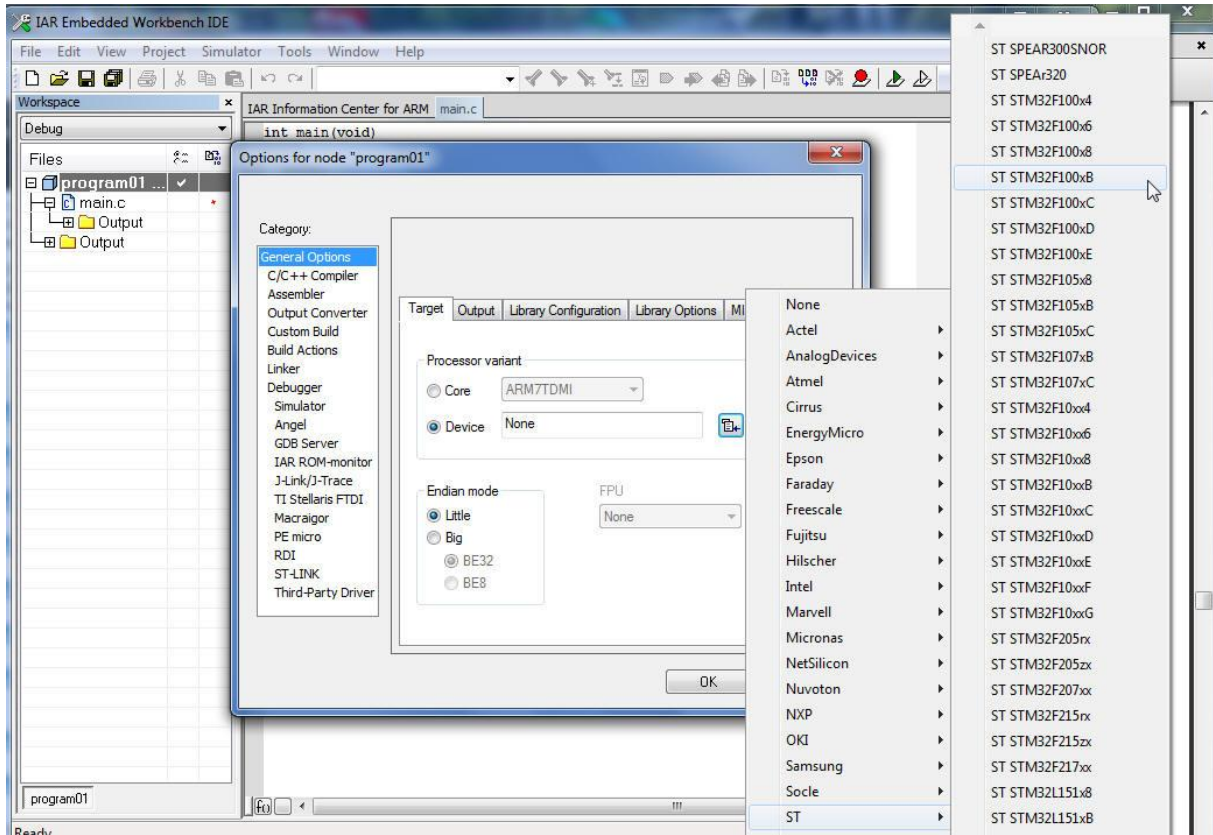
Překlikneme radiobutton **Processor variant** na **Device**



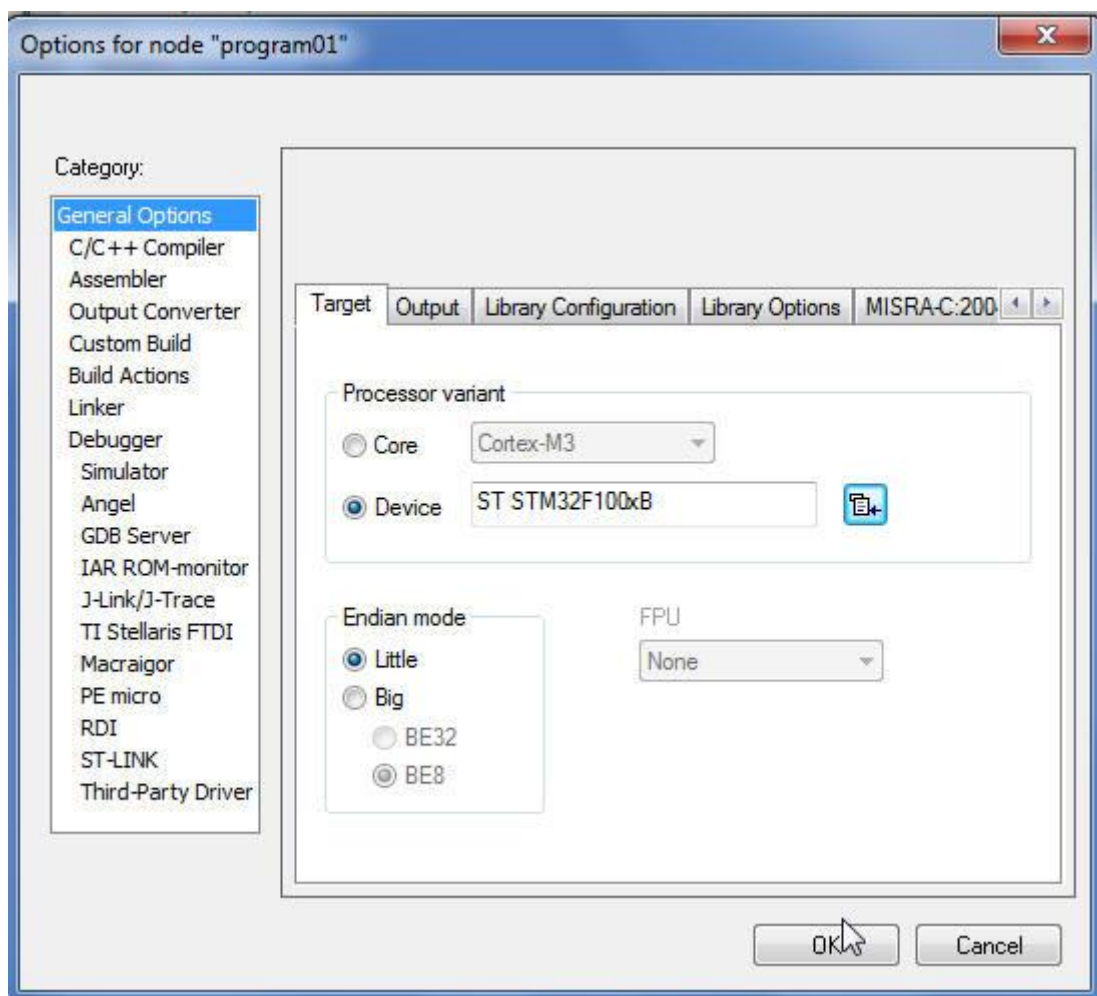
Klikneme na ikonku napravo od *textboxu device*



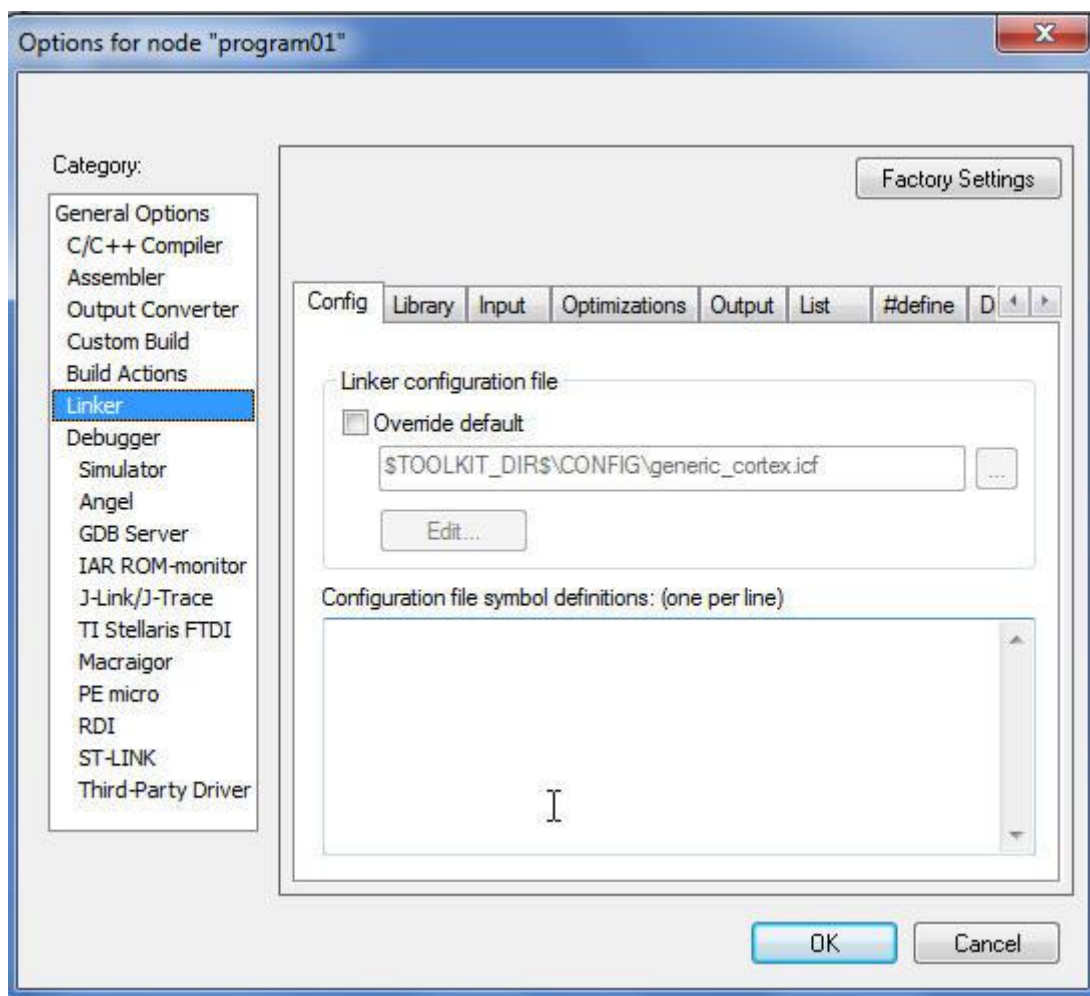
V rozvinutém menu vybereme **ST**



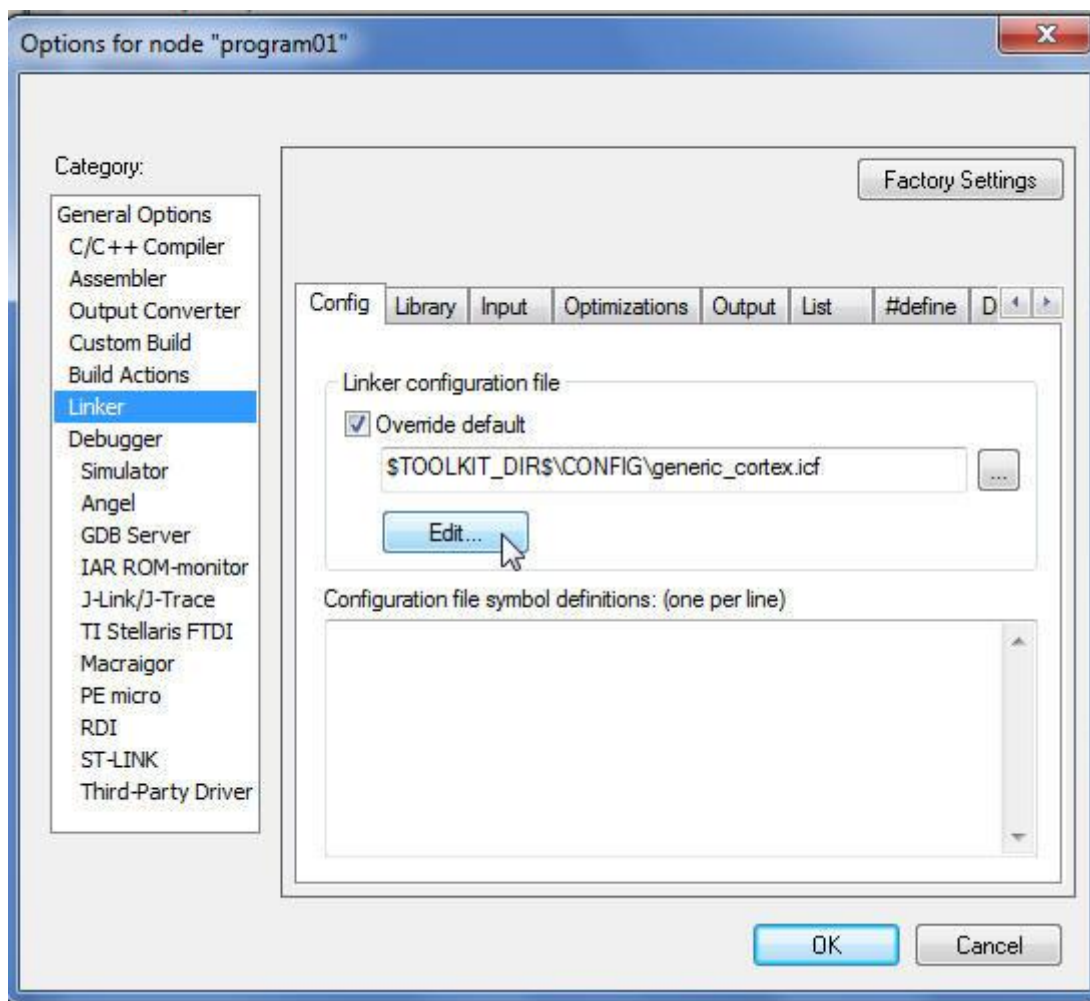
A v něm **STM32F100xB**



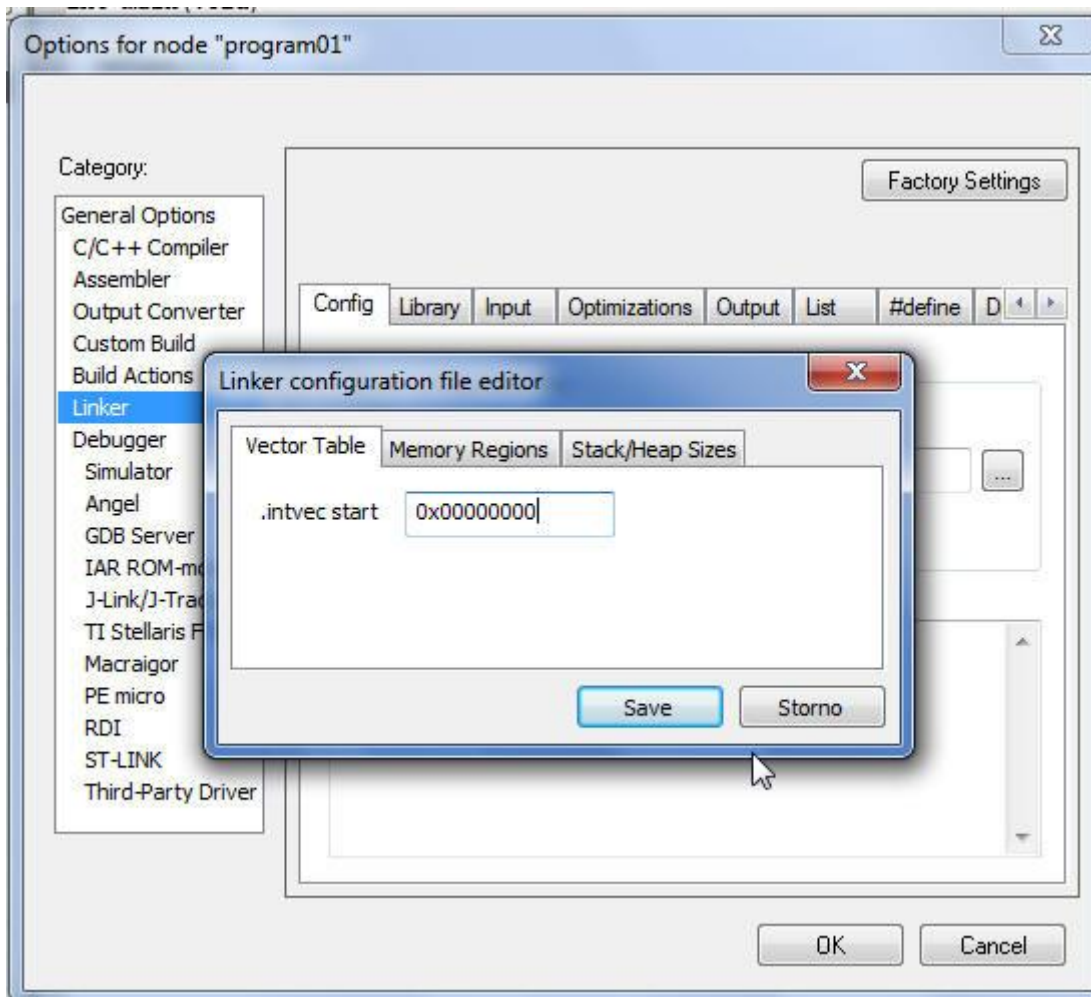
Poté vybereme v podokně **Category** položku **Linker**, dostaneme



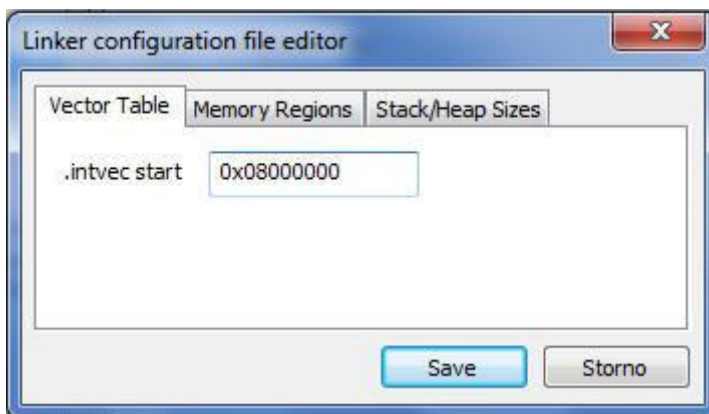
Zaškrtneme v **Linker configuration file** políčko **Override default**

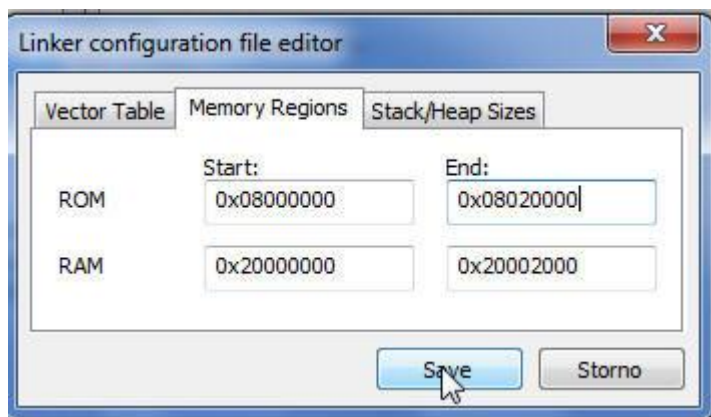
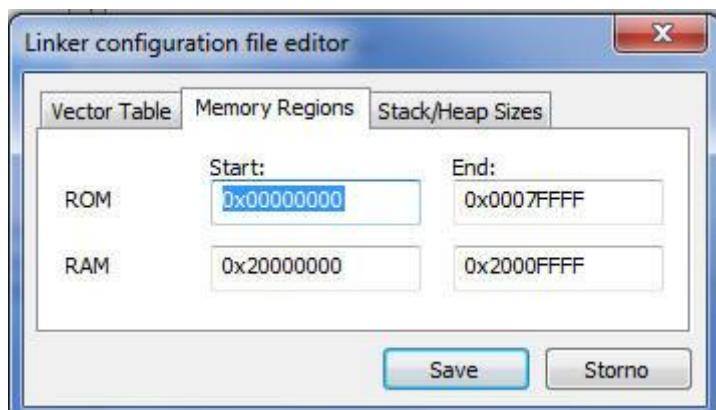


Vyplníme textbox dle obrázku a klikneme poté na **Edit**

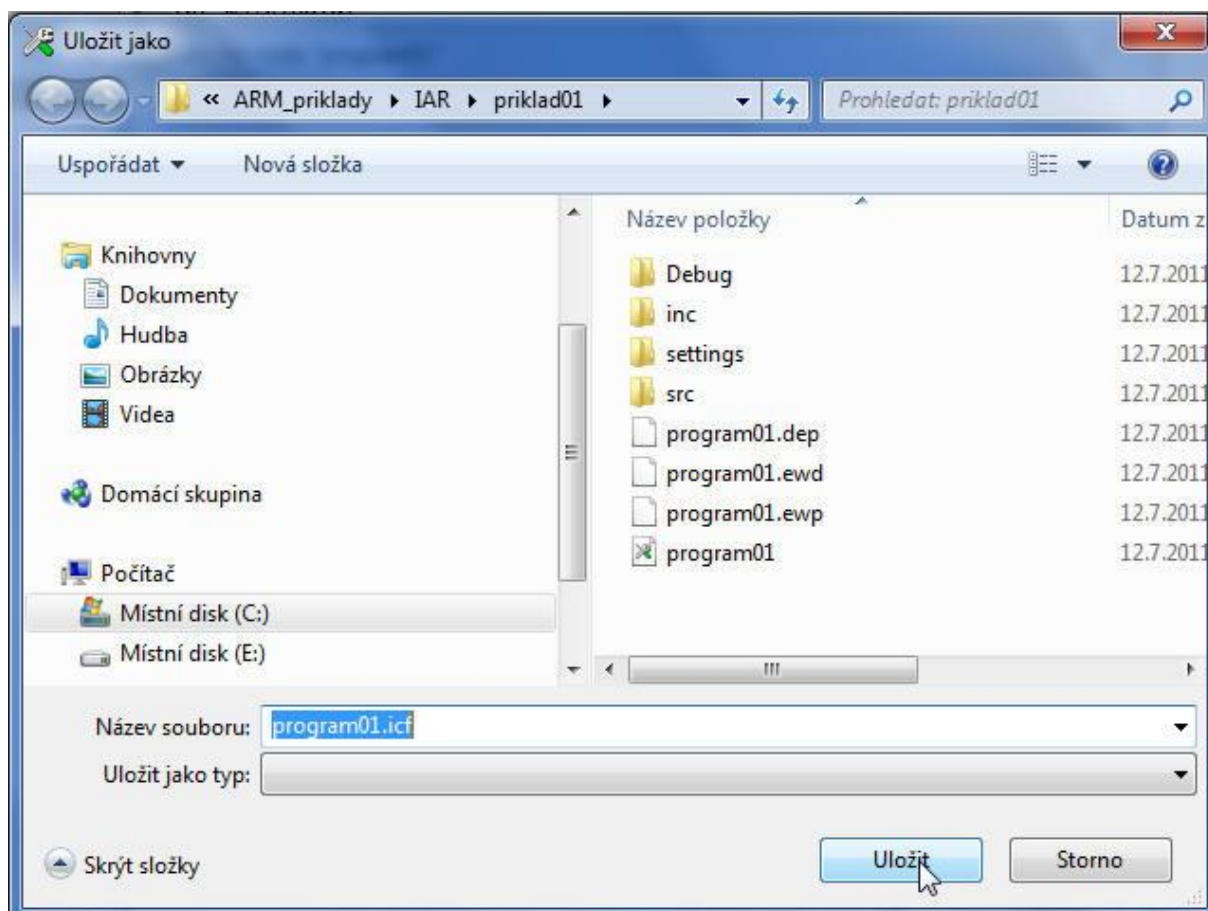


Vyplníme dle obrázků

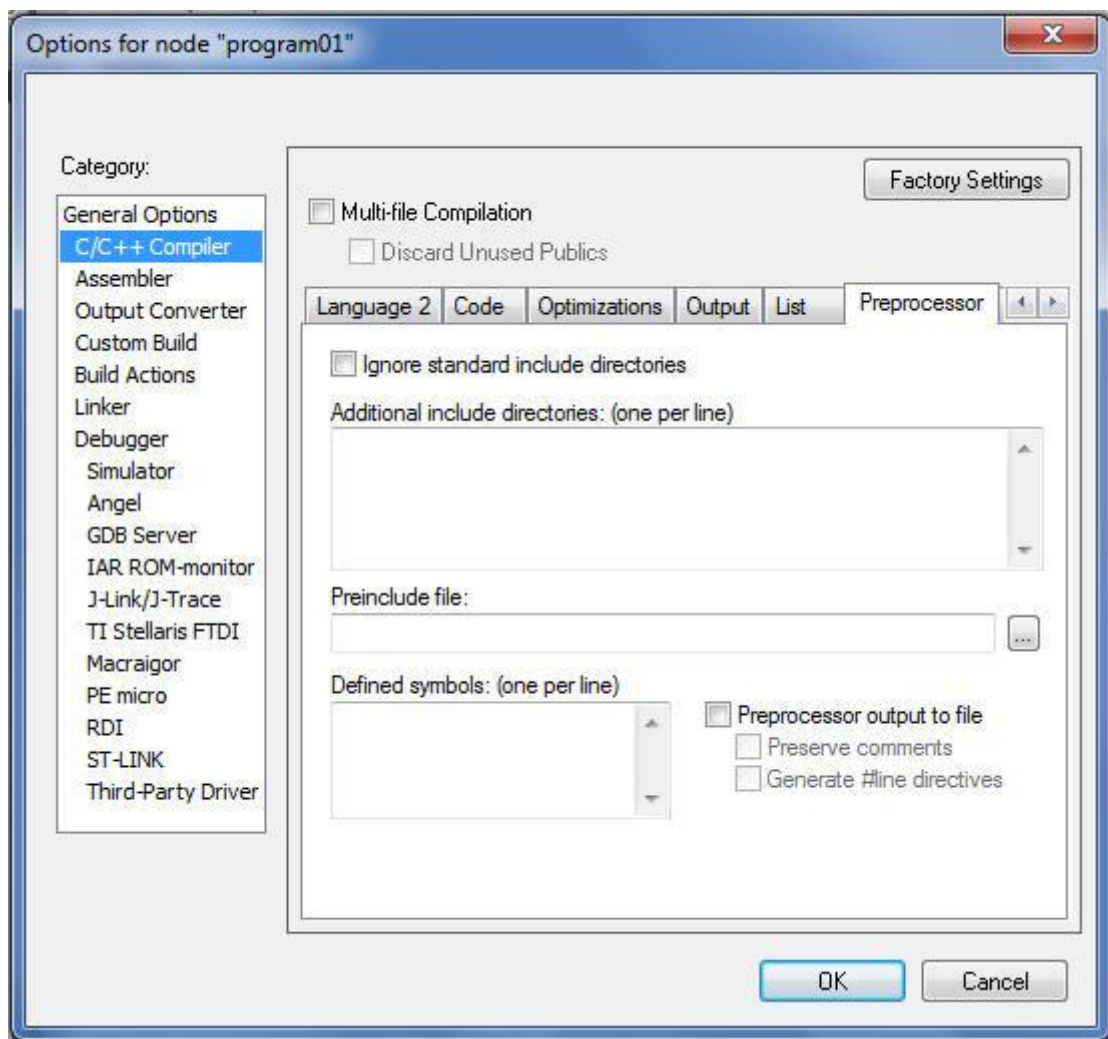




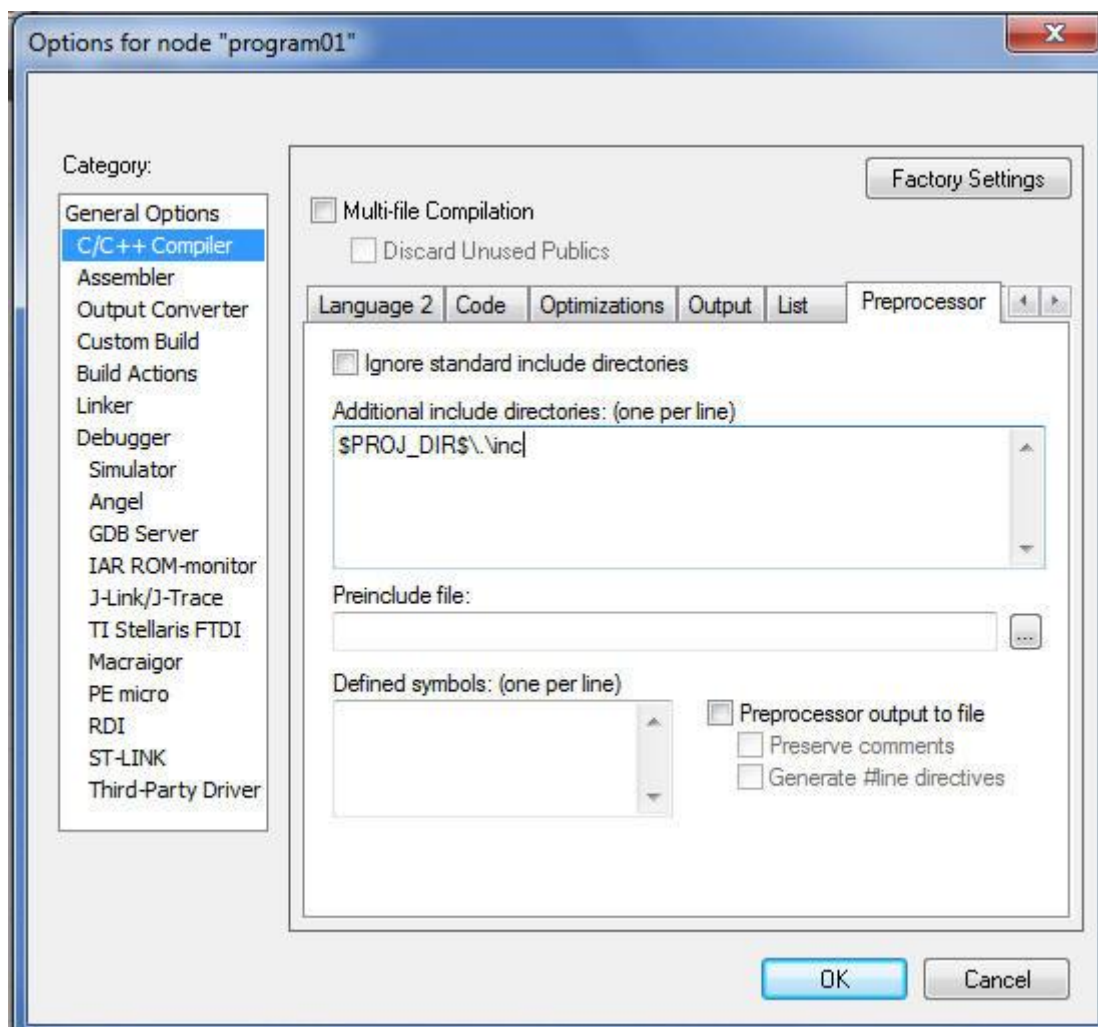
Nakonec klikneme na tlačítko **Save**, objeví se okno **Uložit jako**



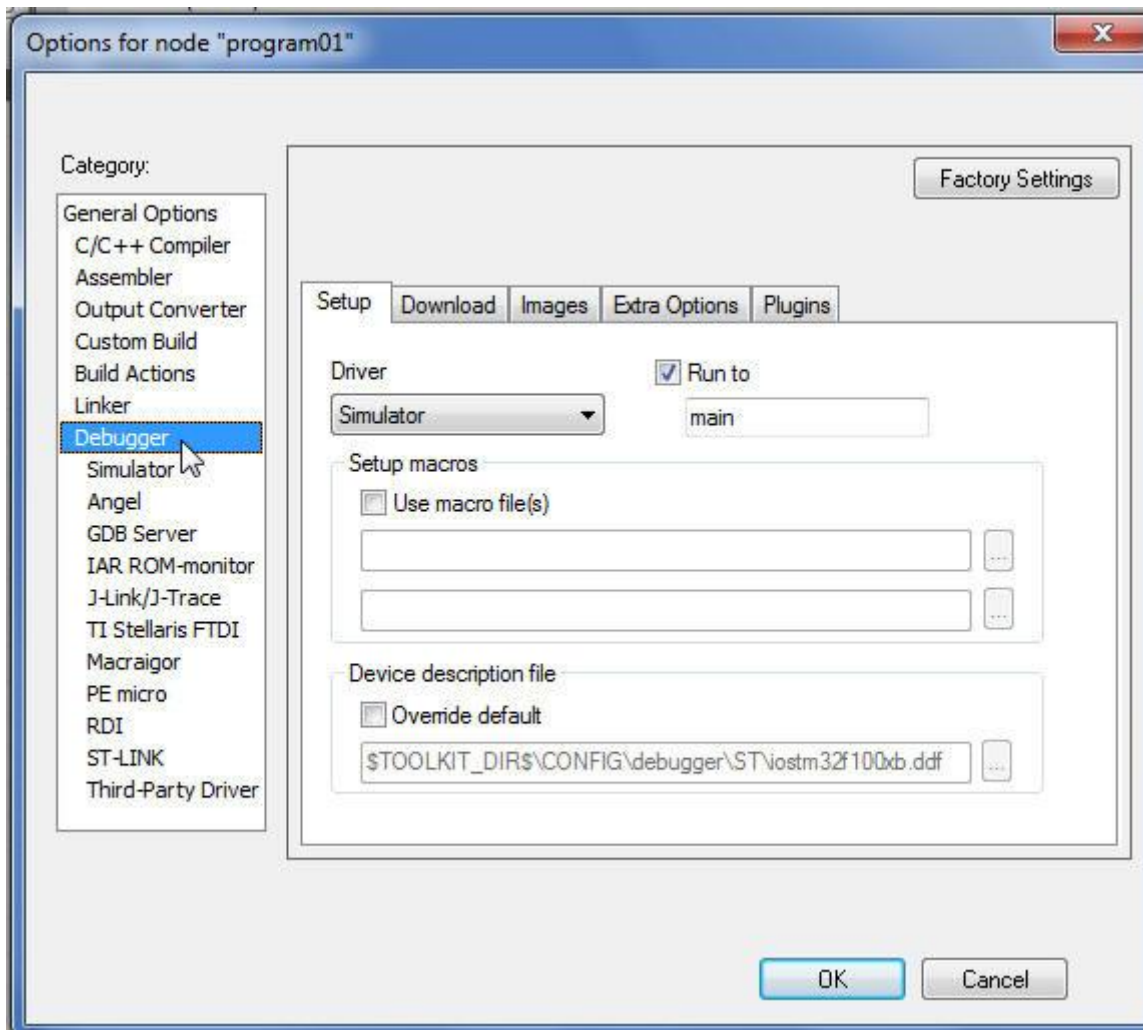
Vyplníme a stiskneme tlačítko **Uložit**



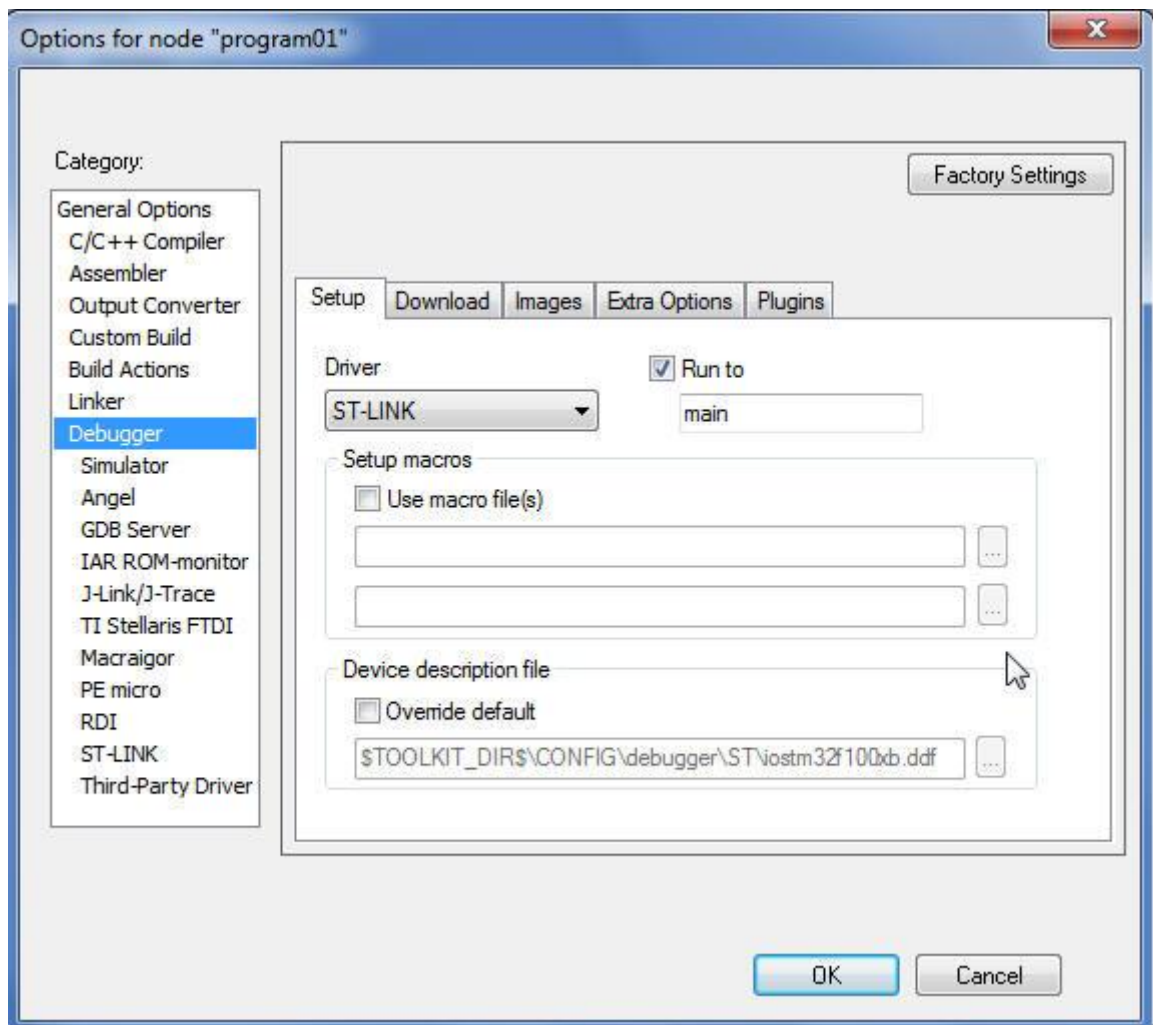
V **Options** ještě vyplníme vlastnosti C/C++ překladače

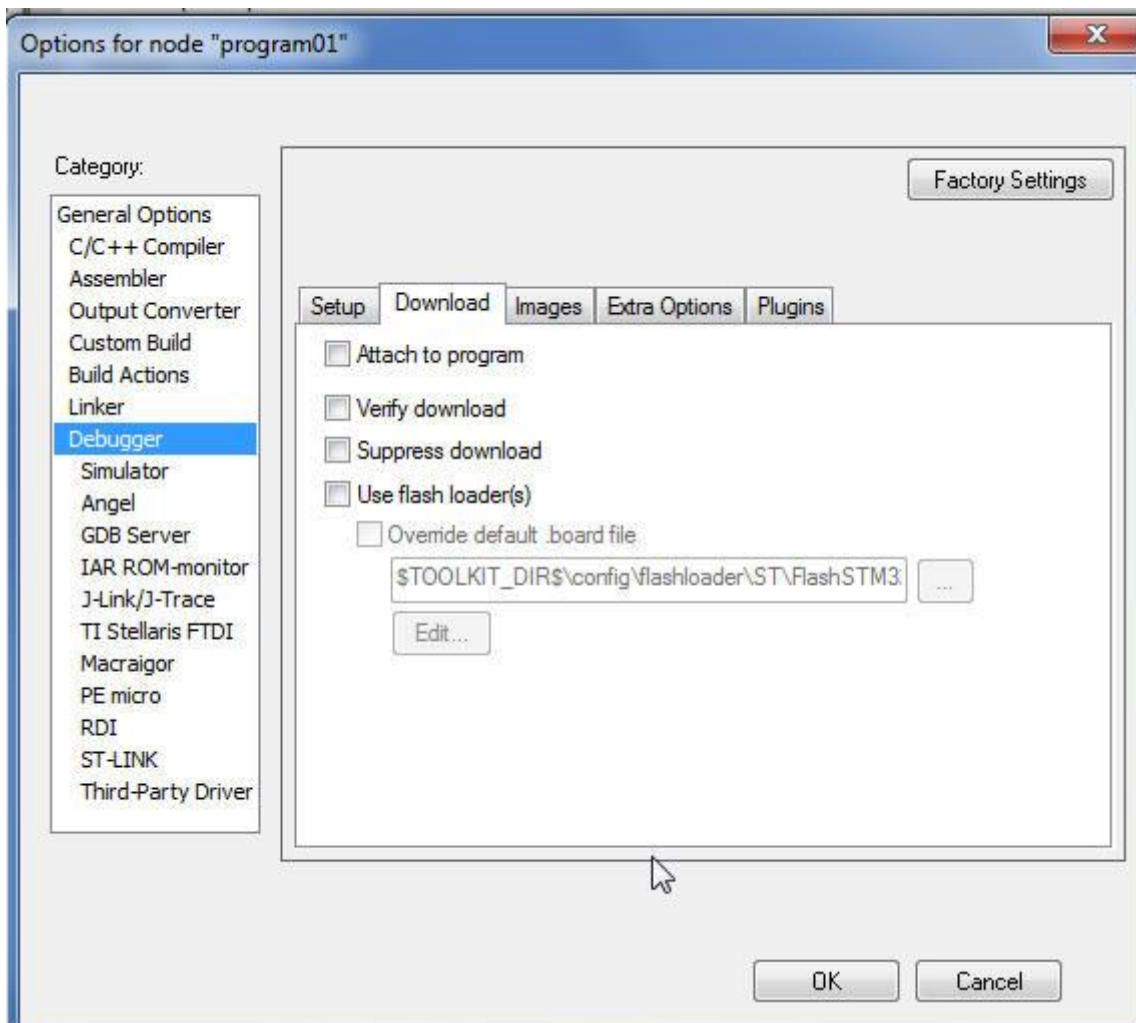


A dále nastavíme vlastnosti **Debuggeru**

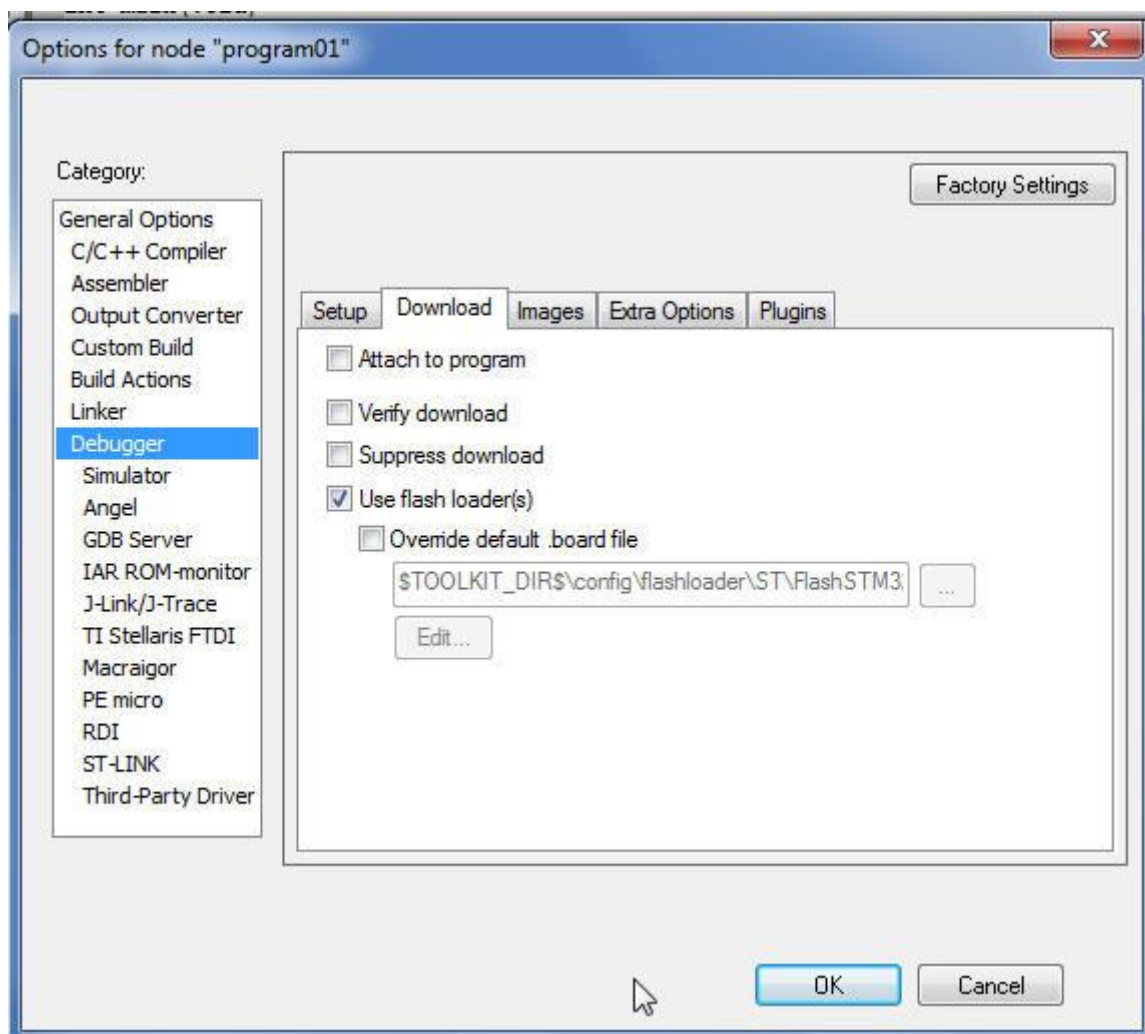


Vybereme driver **ST-Link**

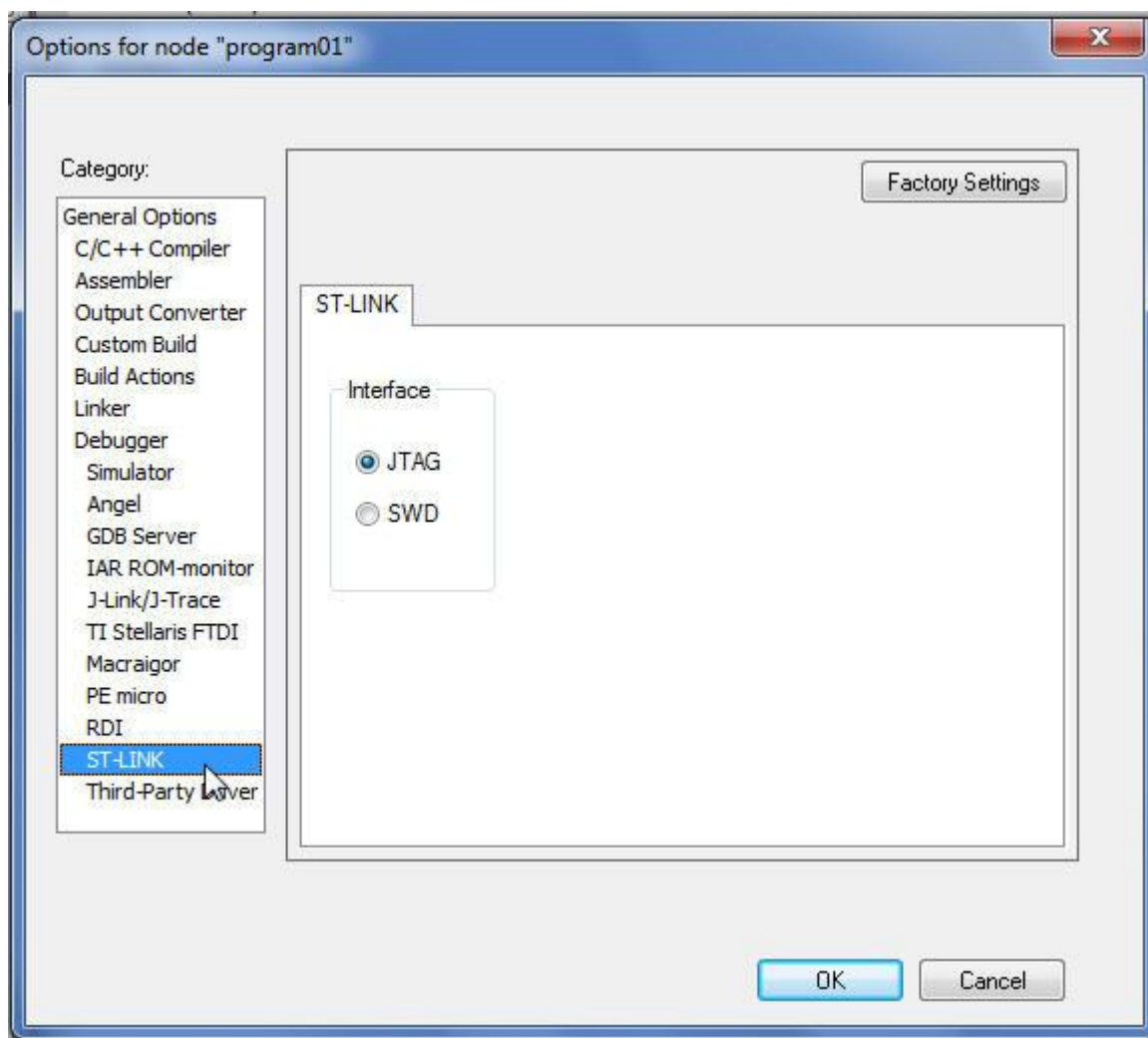




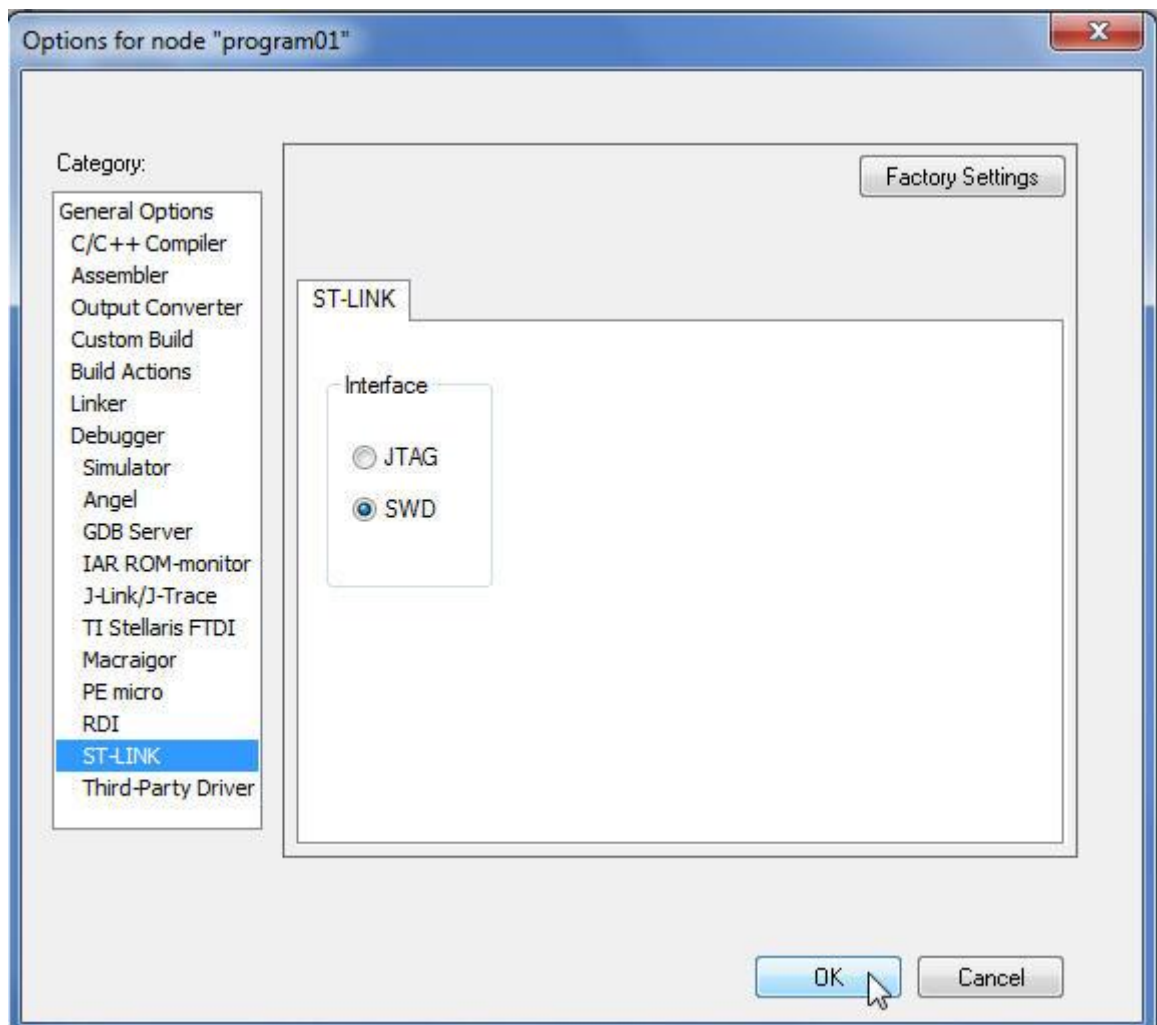
V záložce **Download** zaškrtneme **Use flash loaders**



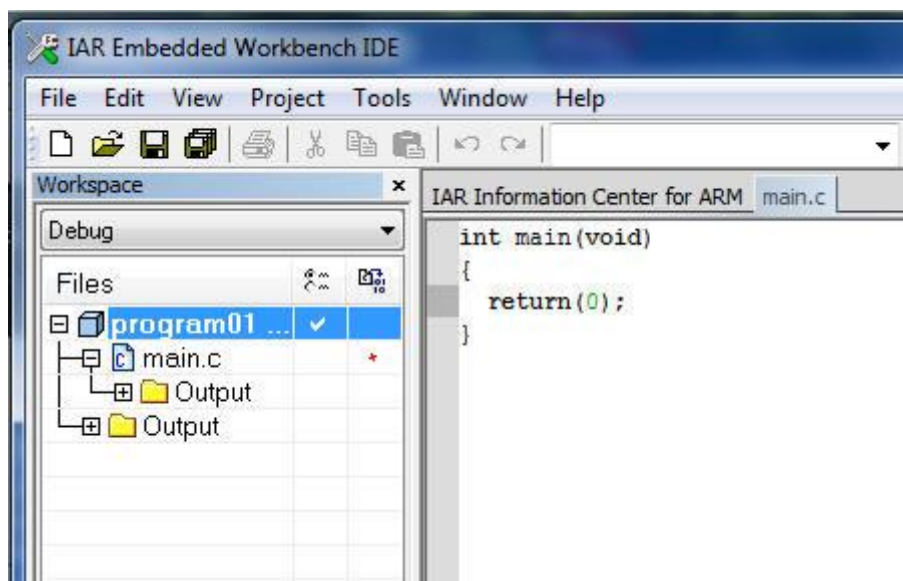
A poté vybereme v **Category** položku **ST-Link**



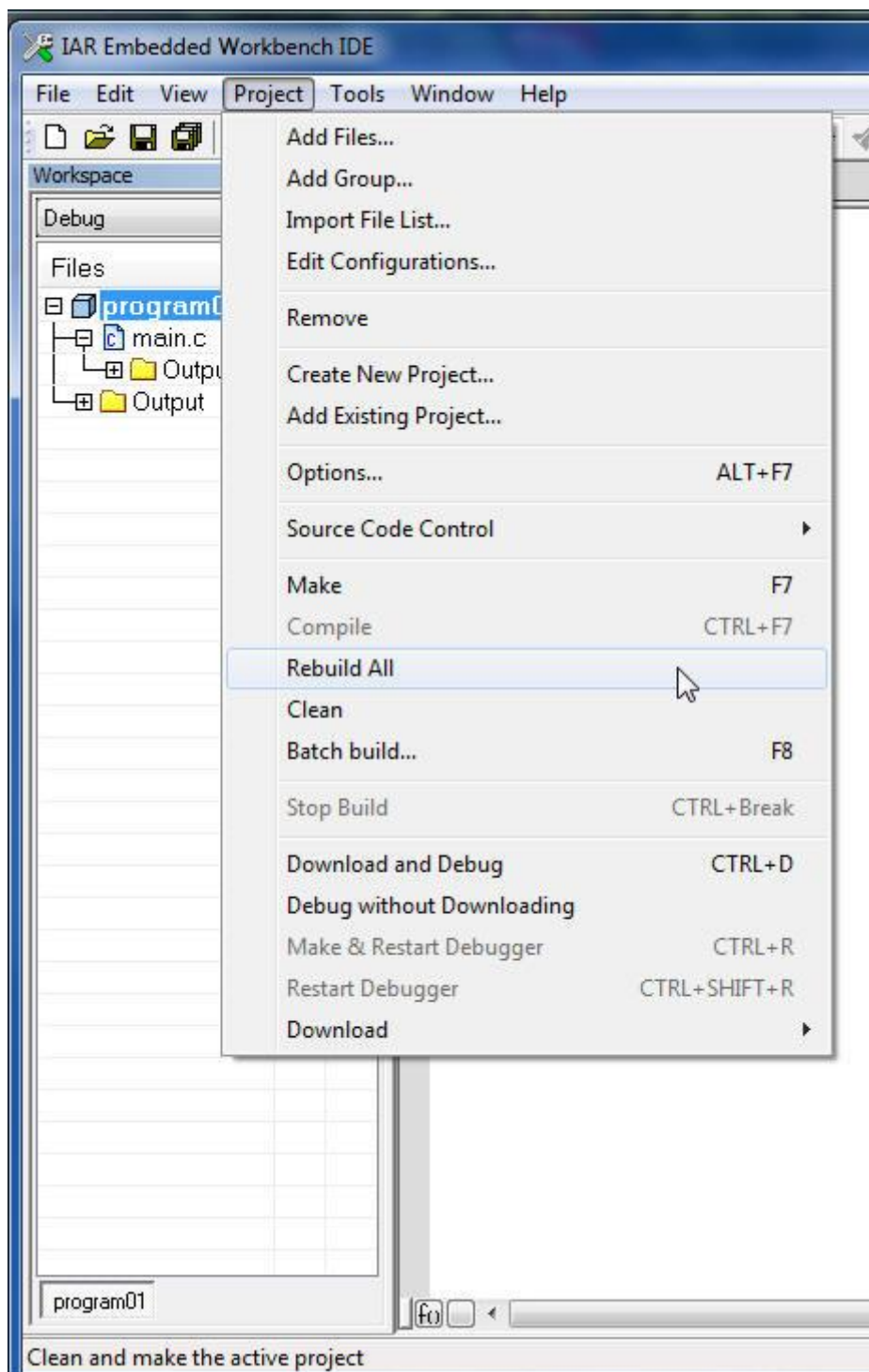
A radiobutton *interface* překlikneme na **SWD**



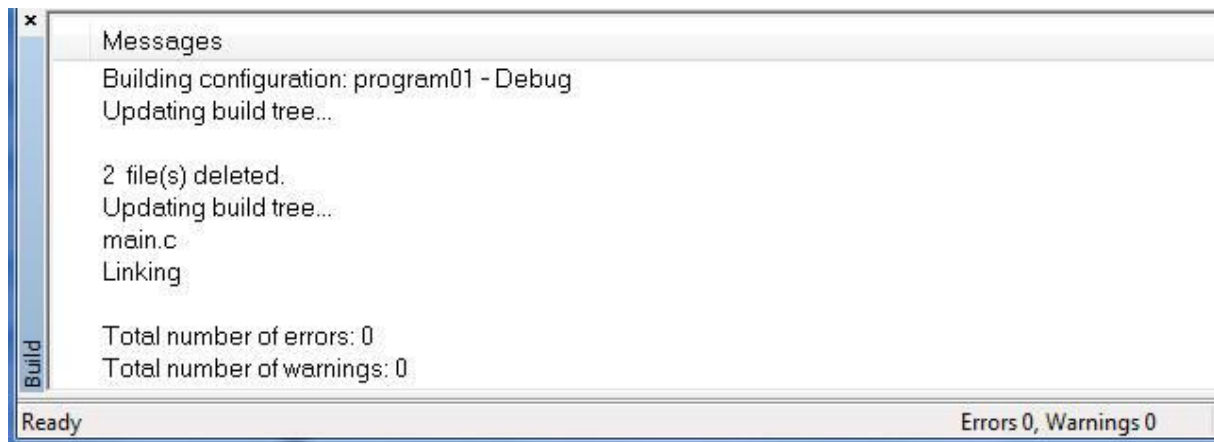
Poté již zbývá kliknout na tlačítko **OK**.



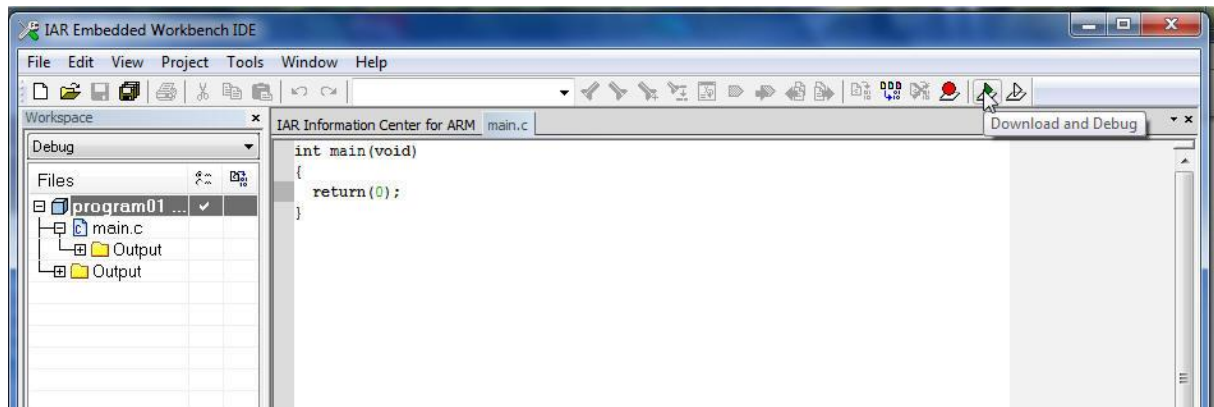
Soubor **main.c** zatím obsahuje jen minimální kód. V menu vybereme **Project – Rebuild All**



V tak jednoduchém programu pochopitelně nejsou chyby a tak se v okně zpráv objeví



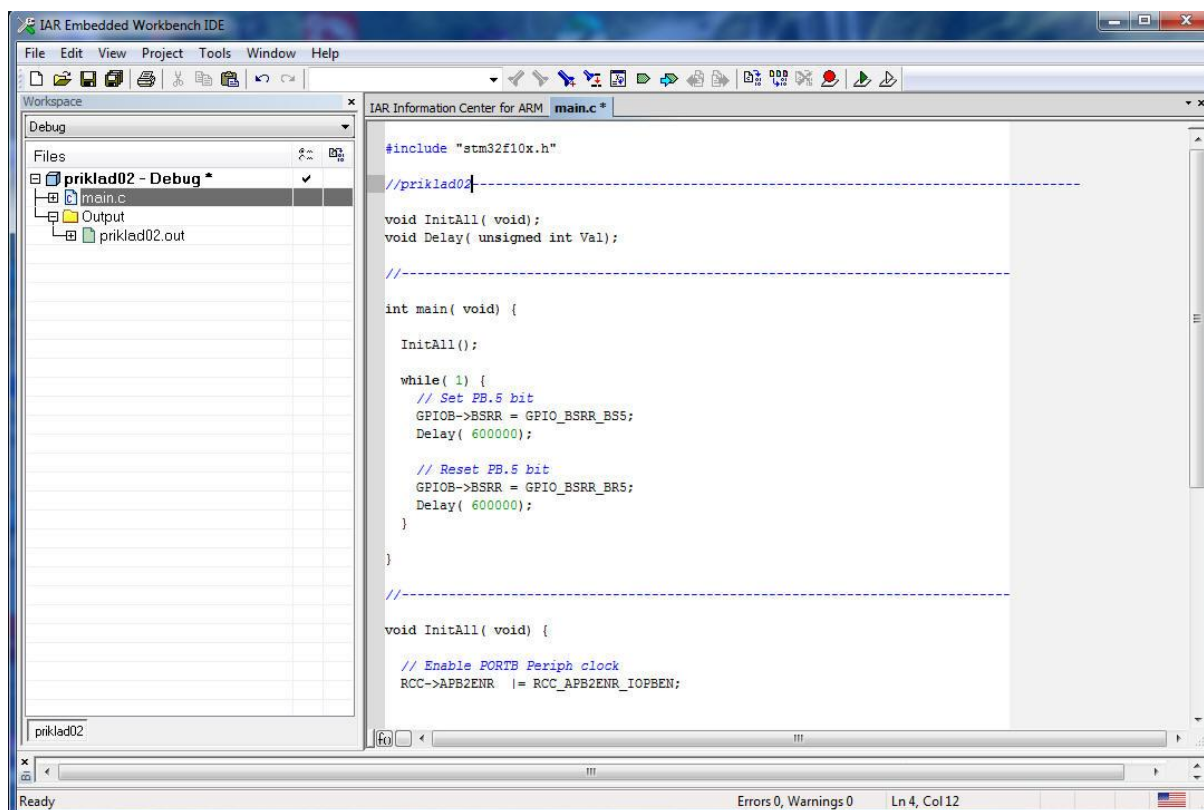
Nyní je možné **Download and Debug**:



Náš **main.c** však zatím neobsahoval žádný užitečný program a tak **Download and Debug** zatím vynecháme a doplníme kód **main.c** a dále přidáme linky na potřebné knihovny a další soubory.

3.4.2 Blikač LED PB5 (příklad02)

Začneme tím, že vytvoříme (postupem z předchozího příkladu) nový **příklad02** a workspace **příklad02**. Poté do něj přidáme i **main.c**



Zdrojový kód souboru **main.c**

```

#include "stm32f10x.h"

//priklad02-----

void InitAll( void);
void Delay( unsigned int Val);

//-----

int main( void) {

    InitAll();

    while( 1) {
        // Set PB.5 bit
        GPIOB->BSRR = GPIO_BSRR_BS5;
        Delay( 600000);

        // Reset PB.5 bit
        GPIOB->BSRR = GPIO_BSRR_BR5;
        Delay( 600000);
    }
}
    
```

```

}

//-----

void InitAll( void) {

    // Enable PORTB Periph clock
    RCC->APB2ENR    |= RCC_APB2ENR_IOPBEN;

    // Clear PB.5 control register bits
    GPIOB->CRL    &= ~(GPIO_CRL_MODE5 | GPIO_CRL_CNF5);

    // Configure PB.5 as Push Pull output at max 10Mhz
    GPIOB->CRL    |= GPIO_CRL_MODE5_0;

    return;
}

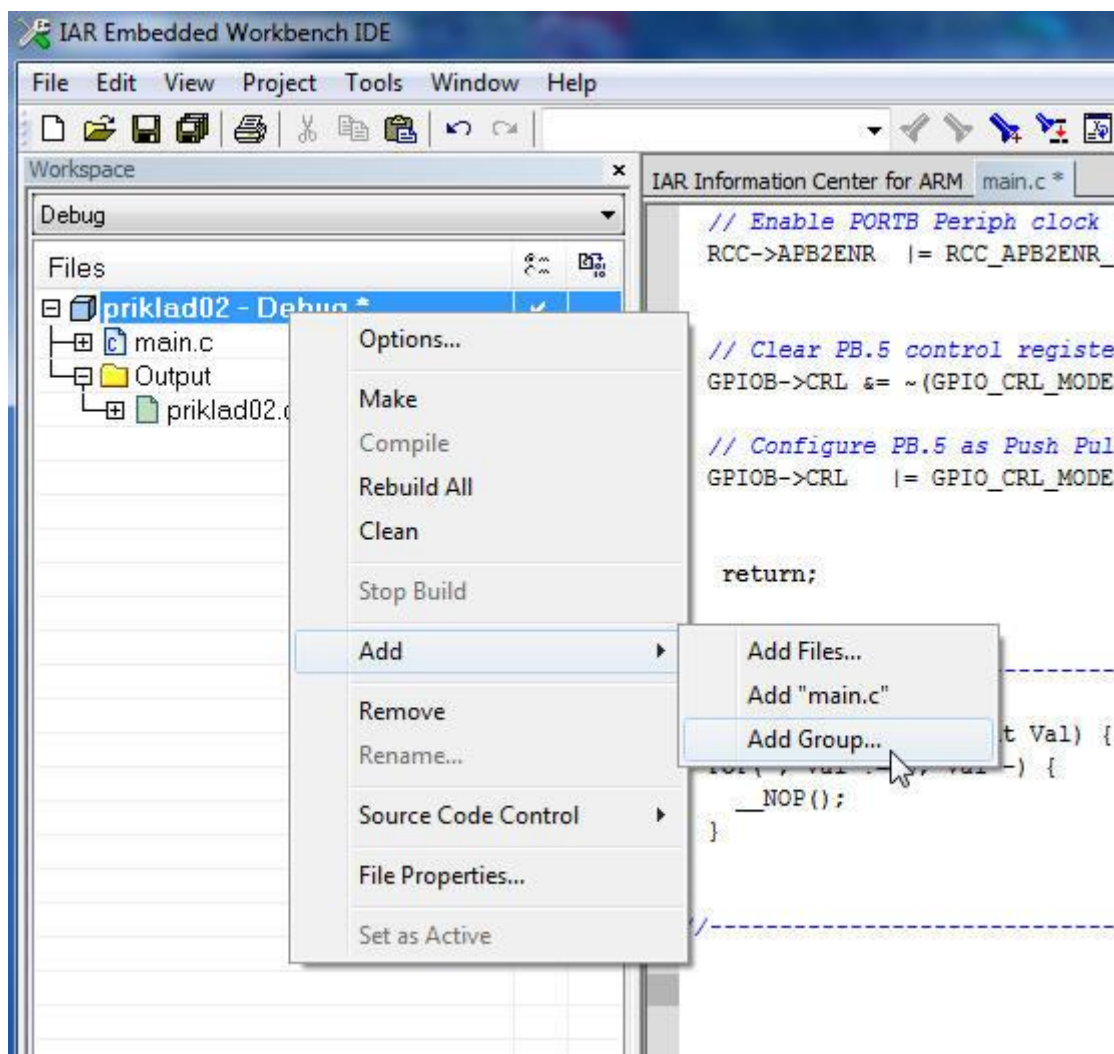
//-----

void Delay( unsigned int Val) {
    for( ; Val != 0; Val--) {
        __NOP();
    }
}

//-----

```

Do projektu musíme přidat knihovny a další soubory uložené v podadresářích. V místním menu (pravé tlačítko myši nad názvem projektu rozvine toto místní menu) vybereme **Add – Add Group**



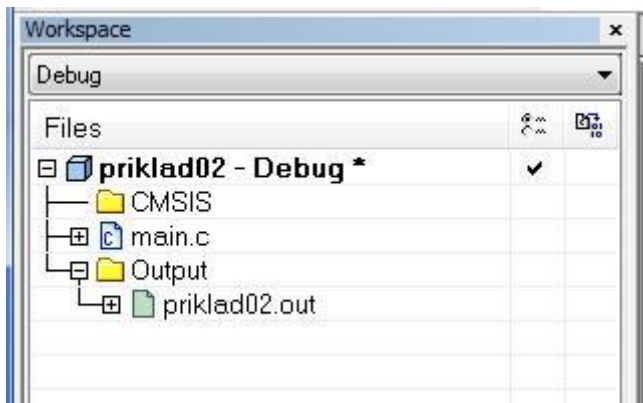
Dostaneme



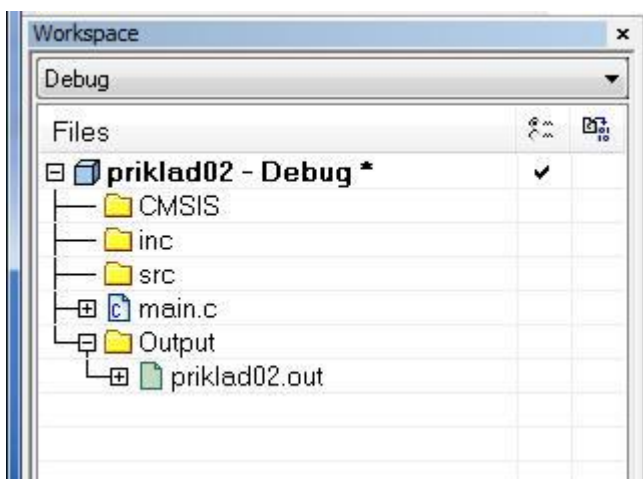
Vyplníme CMSIS



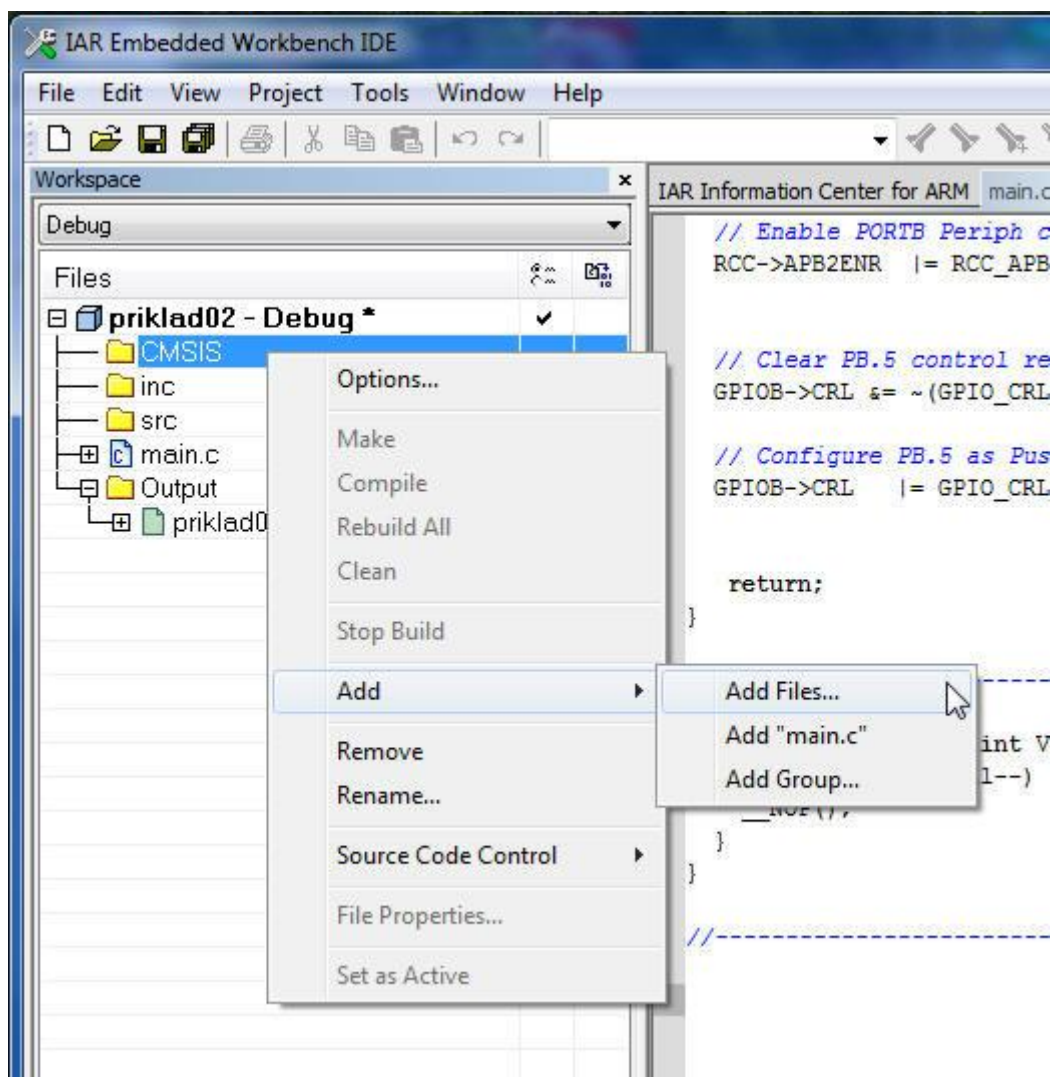
Klikneme na **OK**. Dostaneme



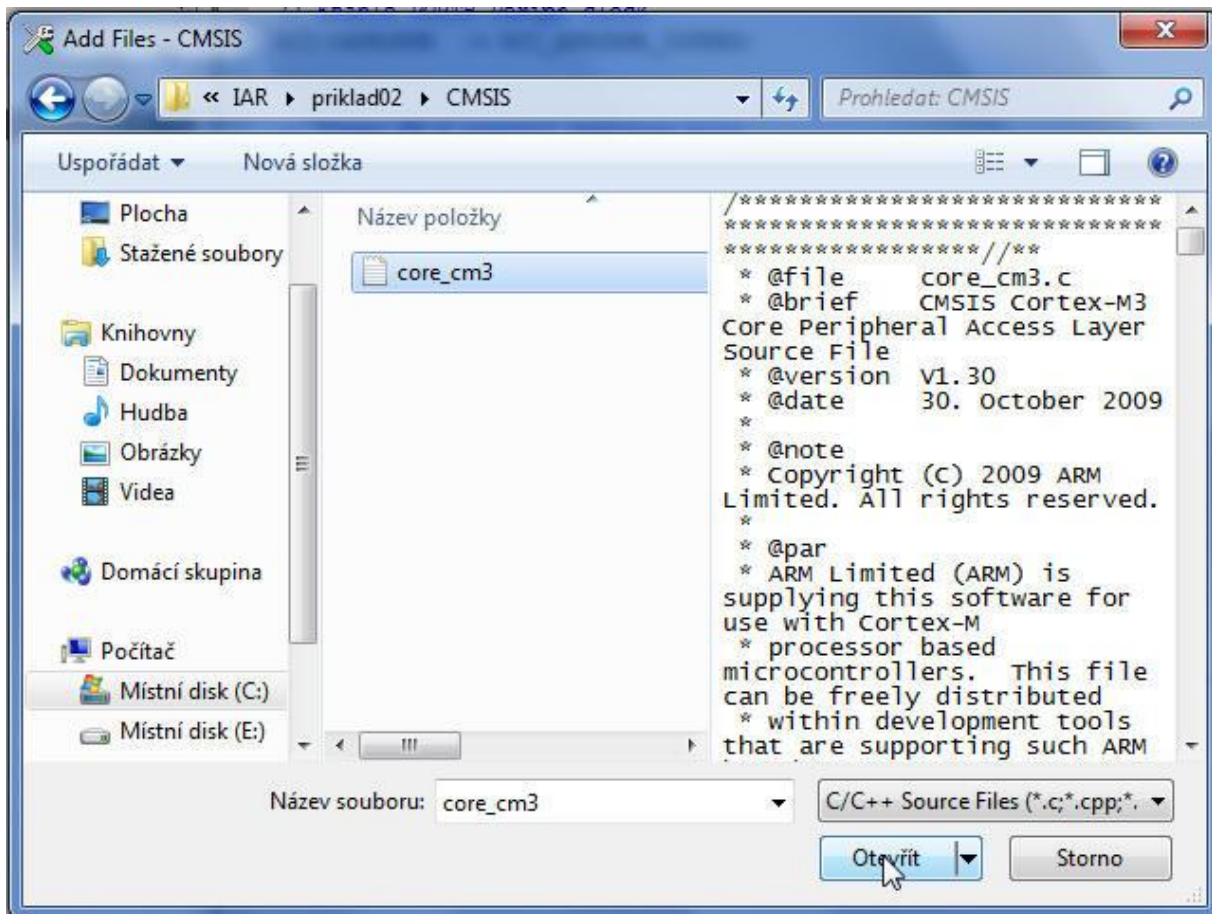
Tím jsme do projektu přidali podadresář **CMSIS** . Obdobně



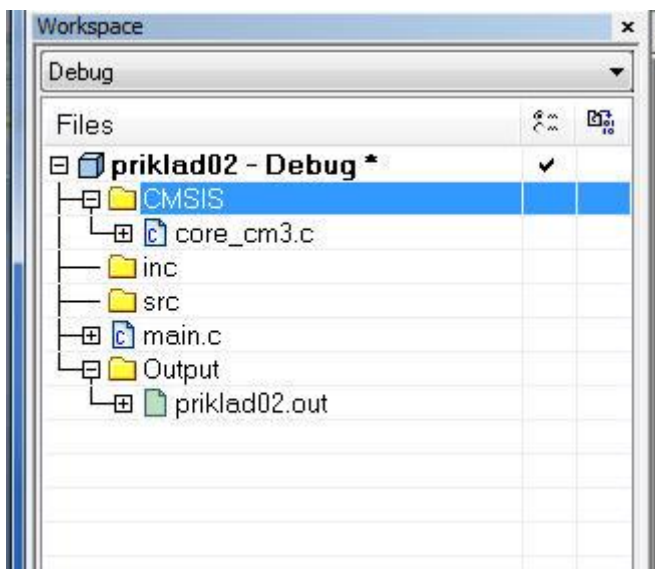
Do přidanych podadresářů ještě přidáme soubory. Rozvineme místní nabídku nad názvem **CMSIS**



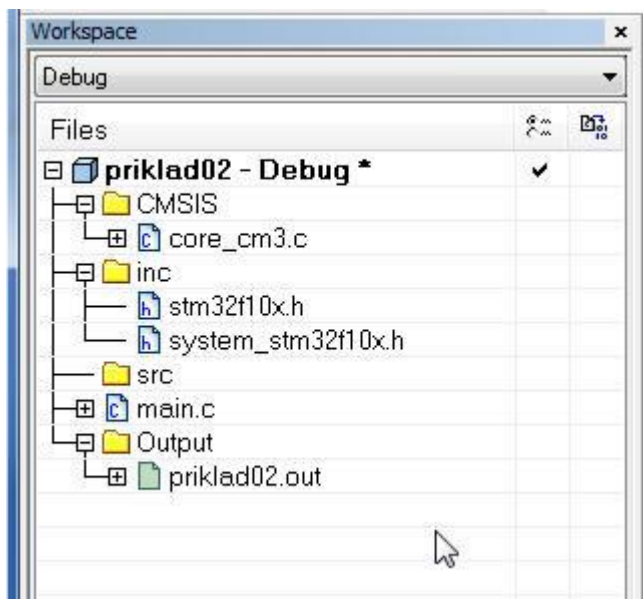
A v ní vybereme **Add – Add Files...**



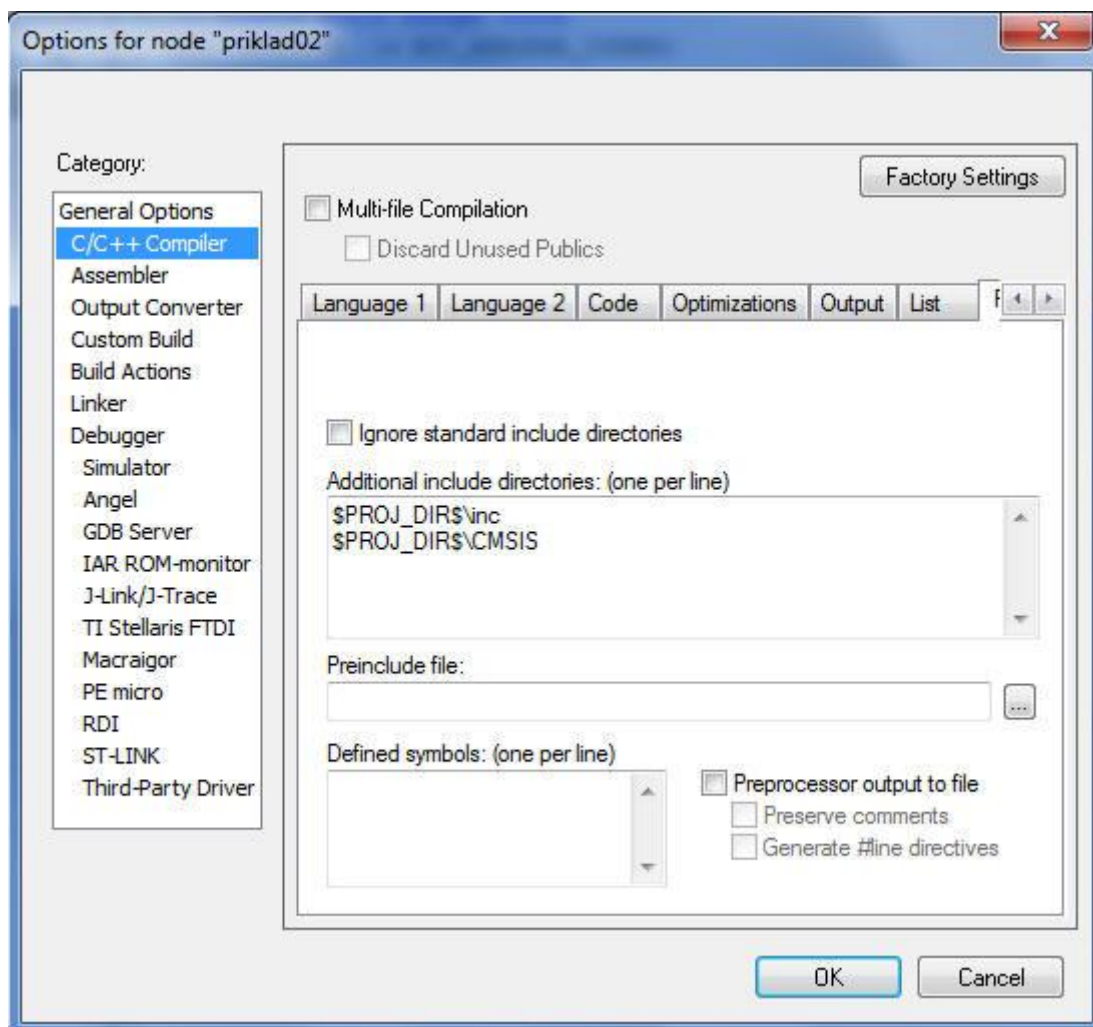
Přidáme soubor **core_cm3.c**



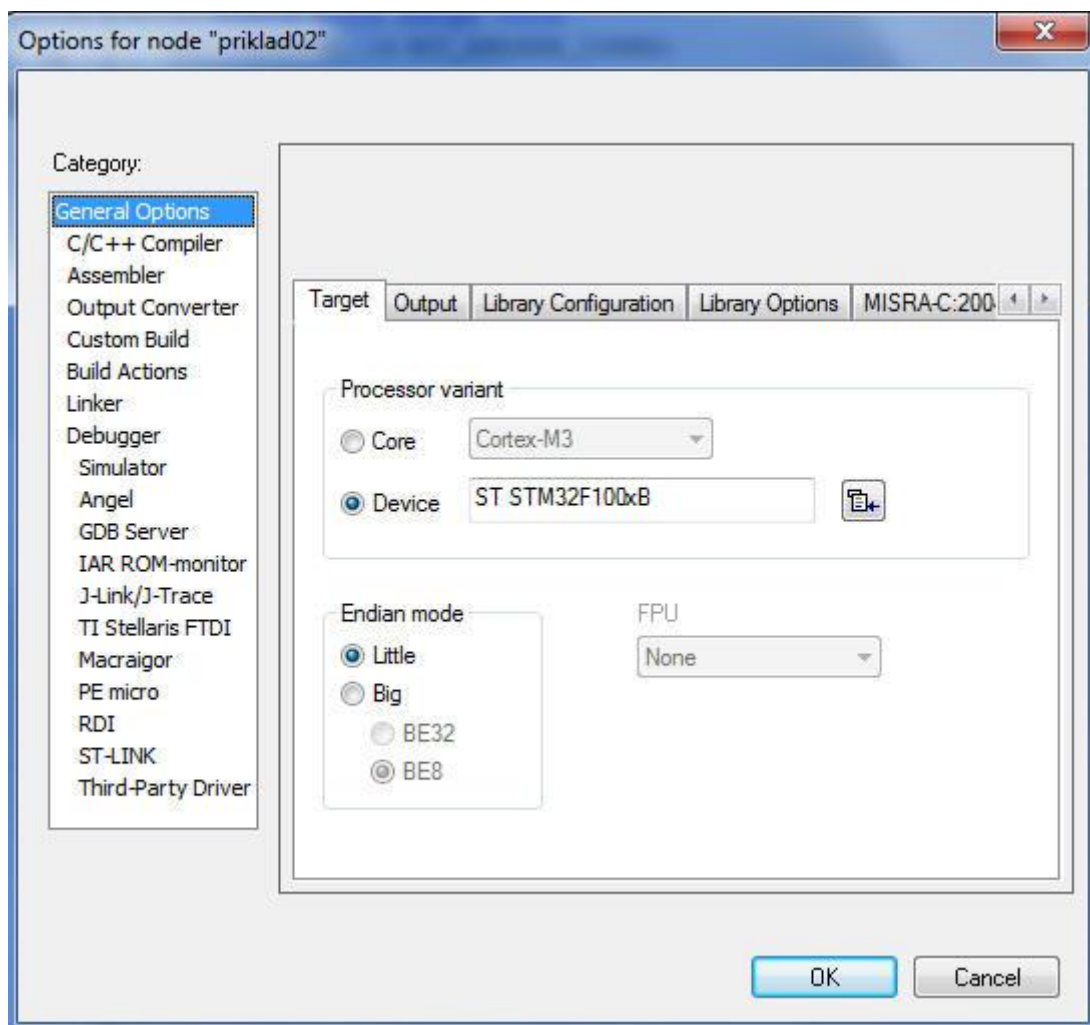
Obdobně doplníme o další soubory

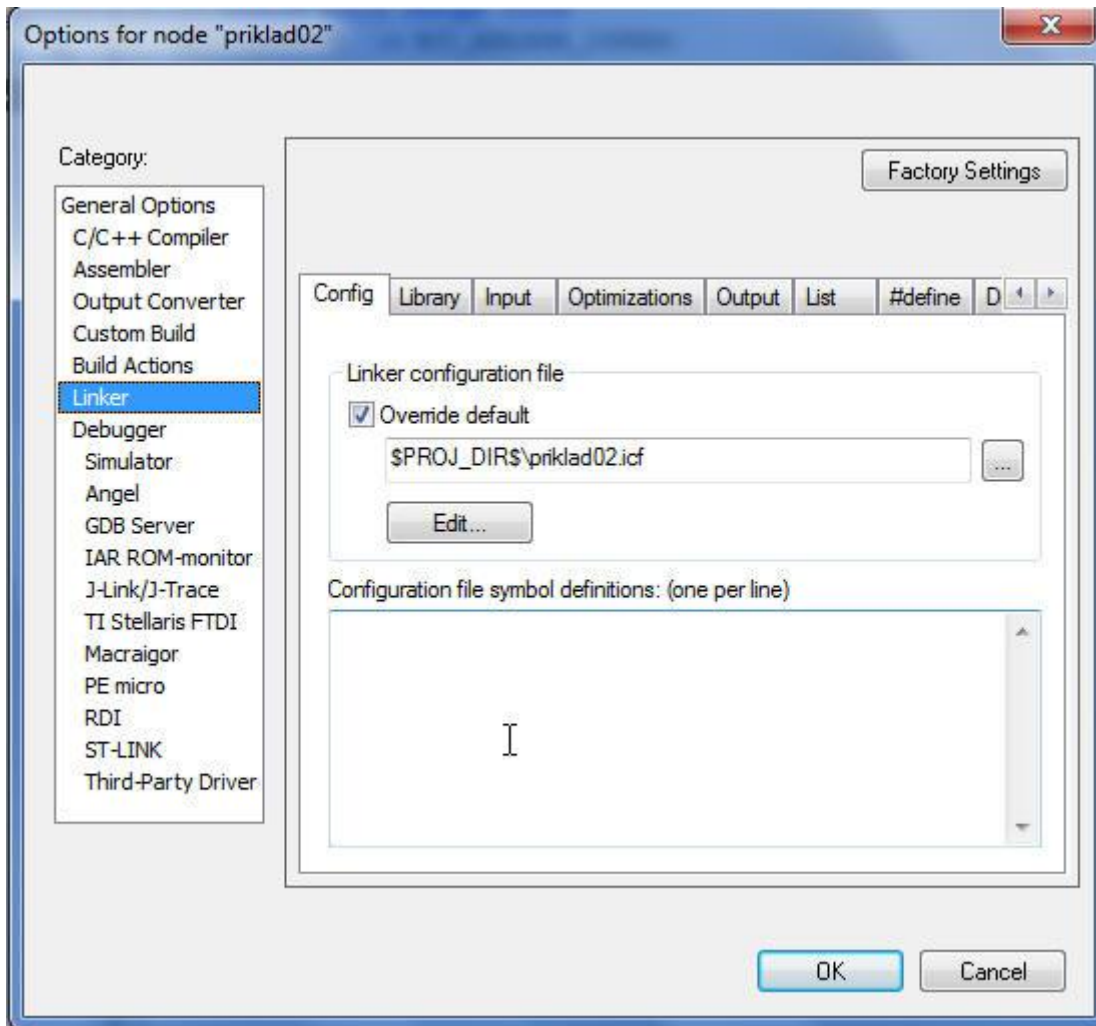


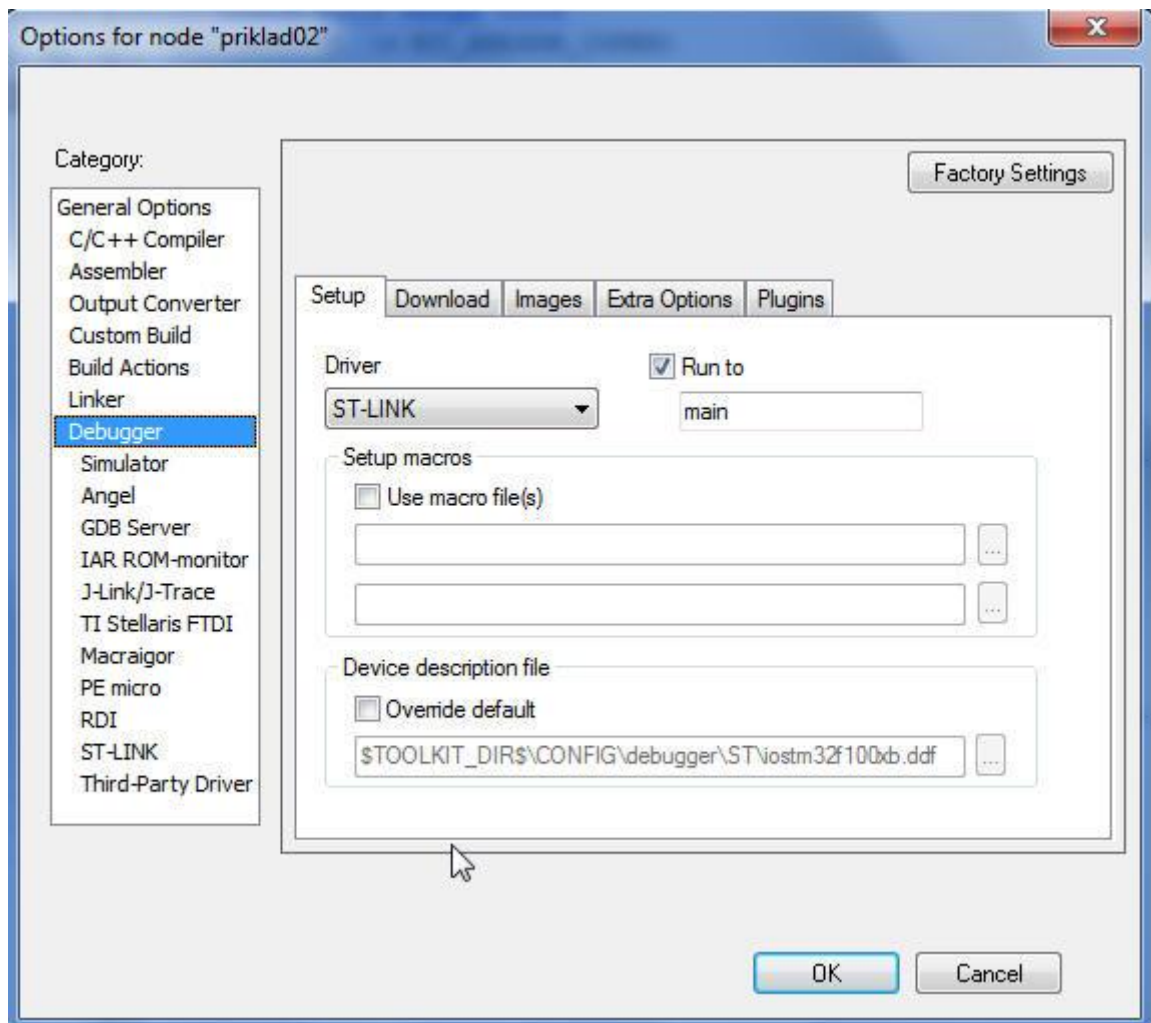
V **Options** nastavíme vlastnosti kompilátoru

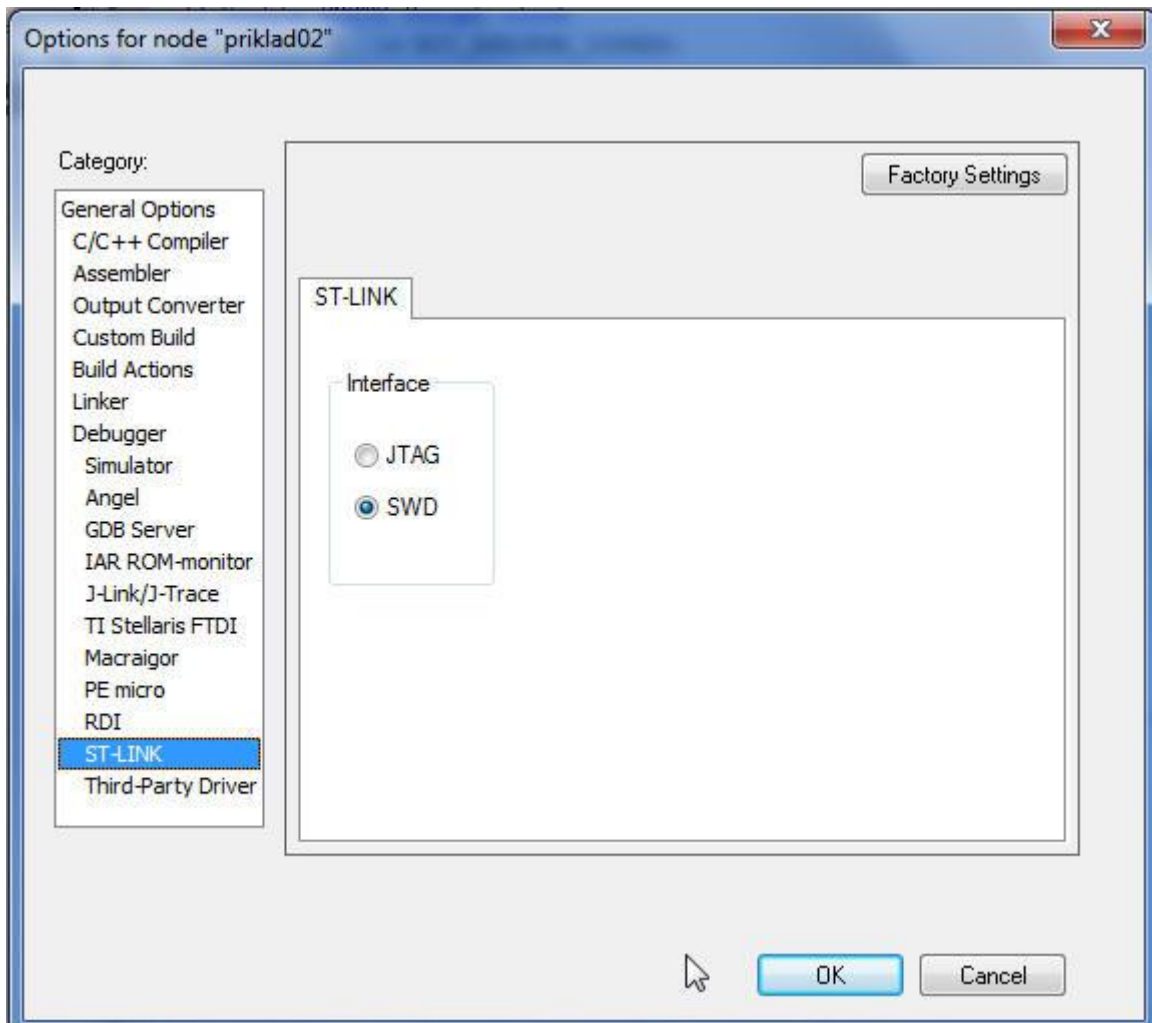


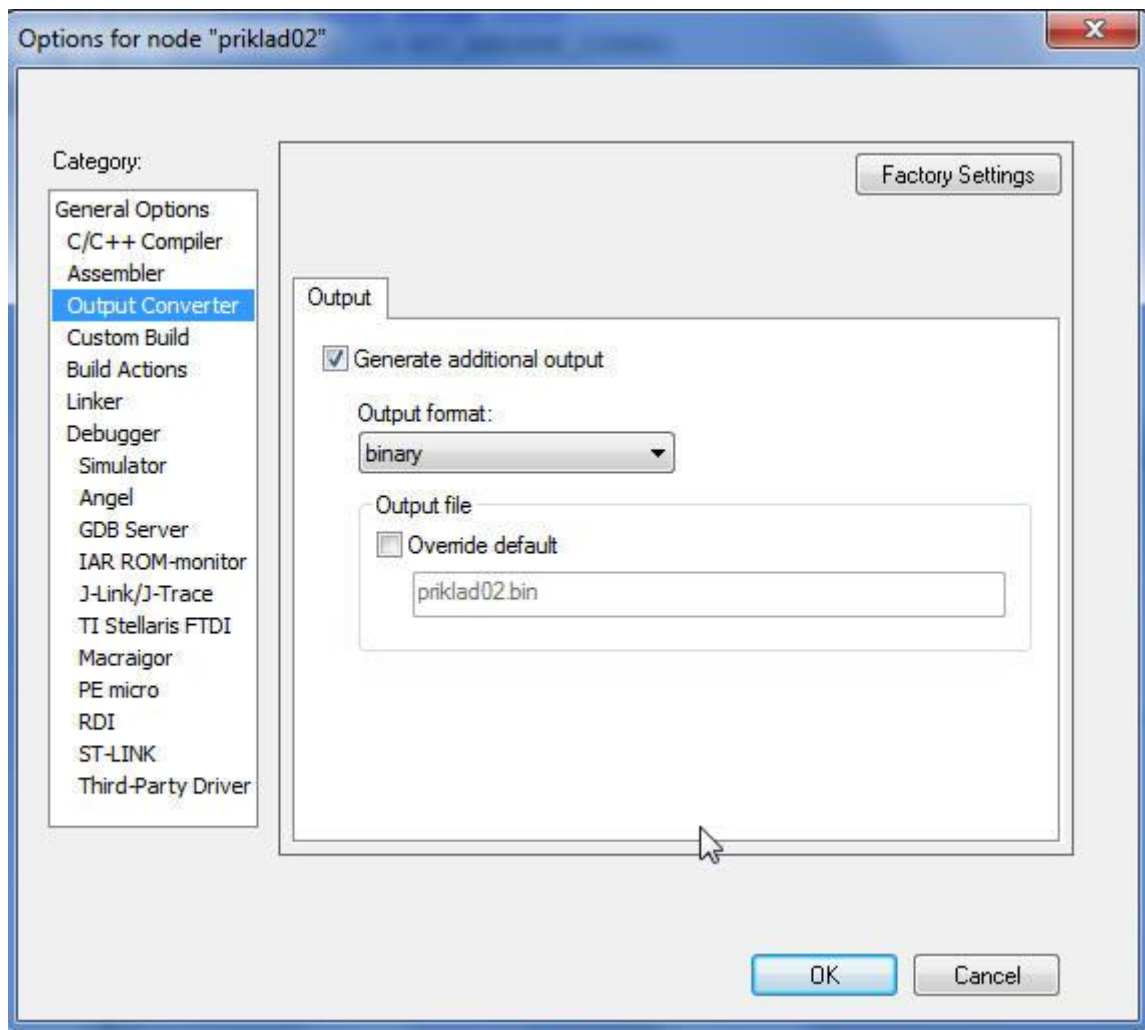
Ostatní nastavení **Options for node „příklad02“** budou stejné jako u prvního příkladu, tj



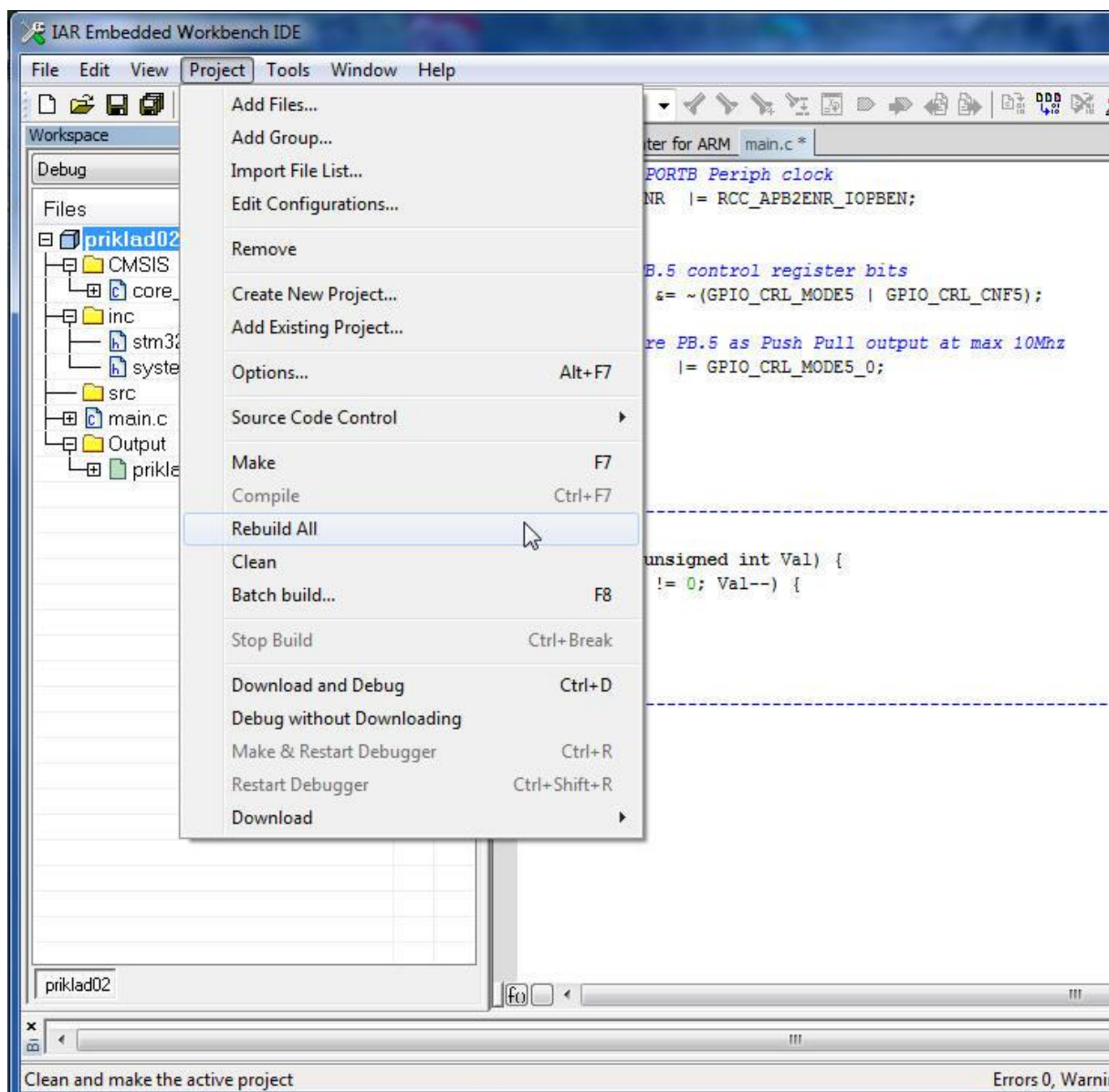




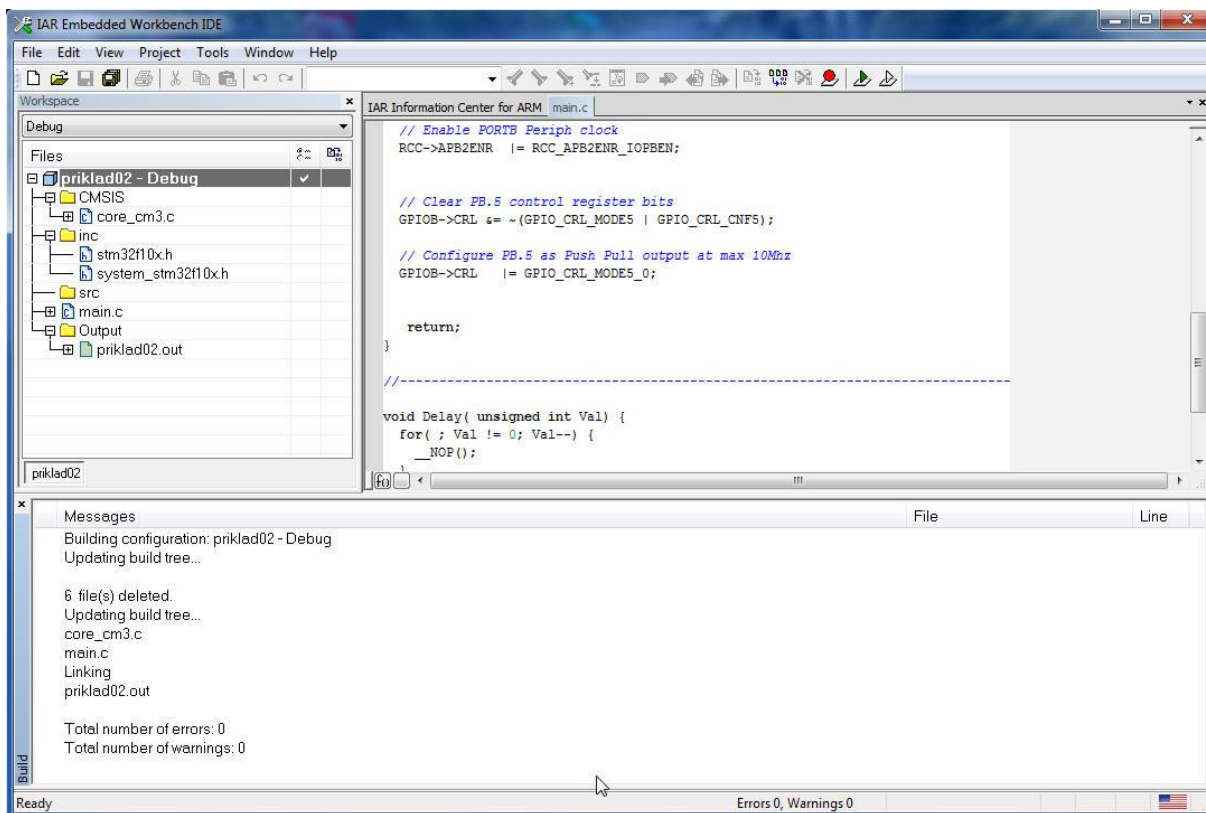




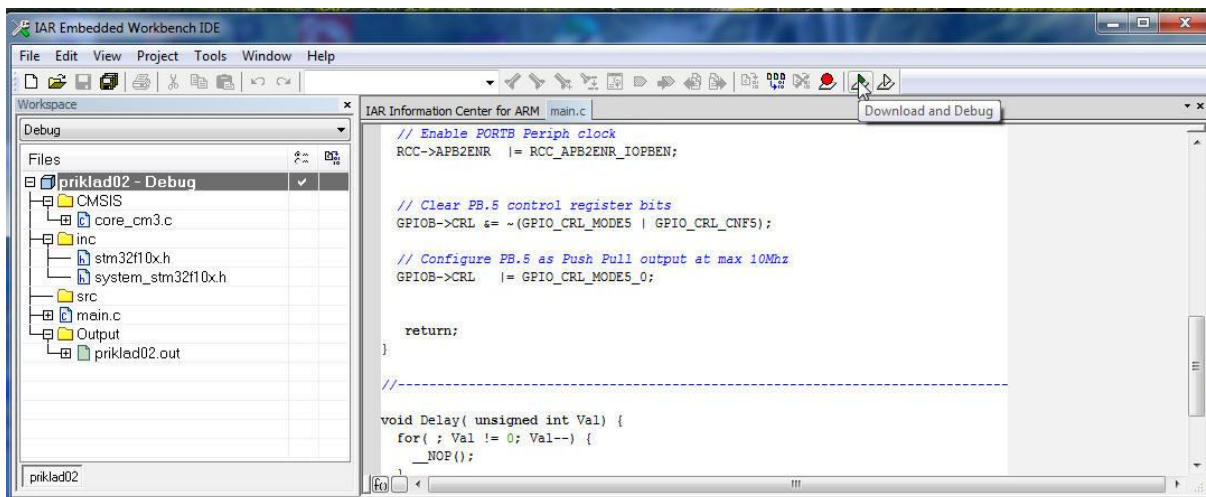
Nakonec v menu vybereme **Project – Rebuil All**

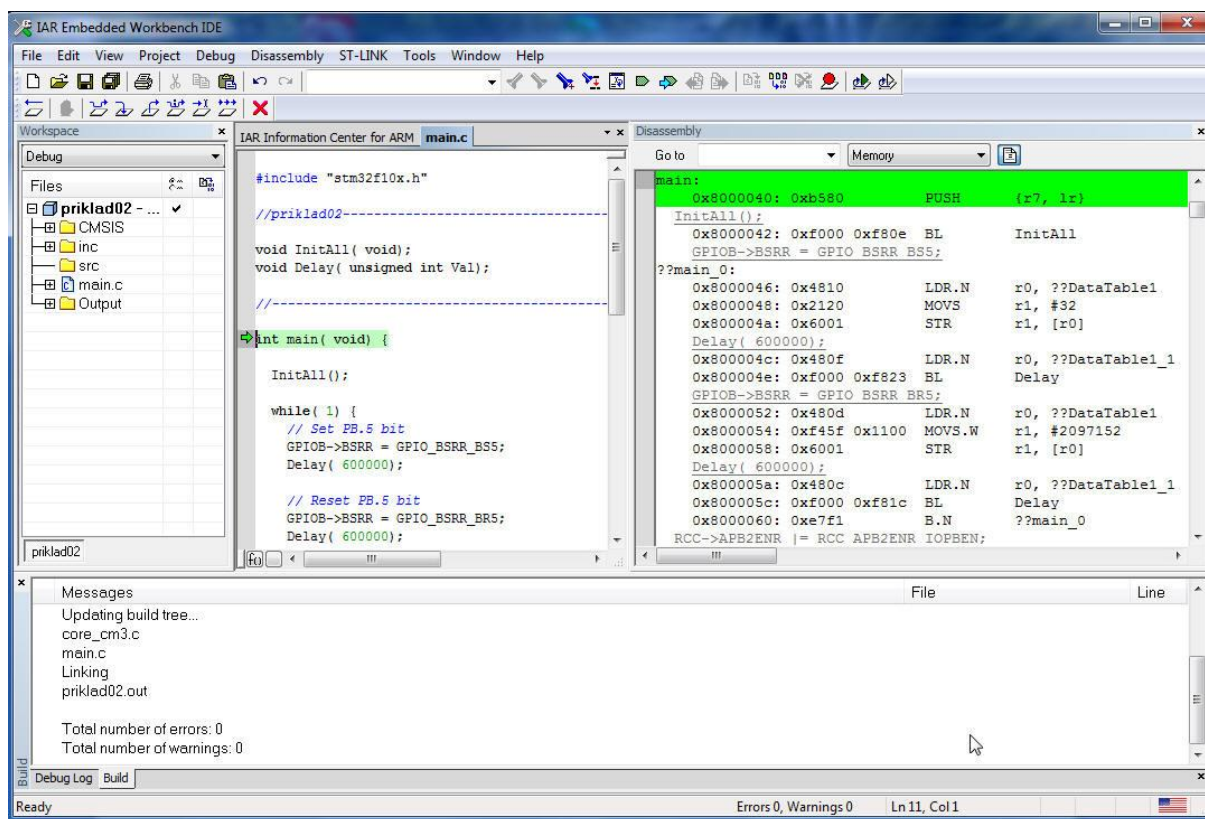


Projekt je úspěšně přeložen a spojen (linked)



Zbývá připojit přes USB startkit **STM32VL Discovery**. Dále

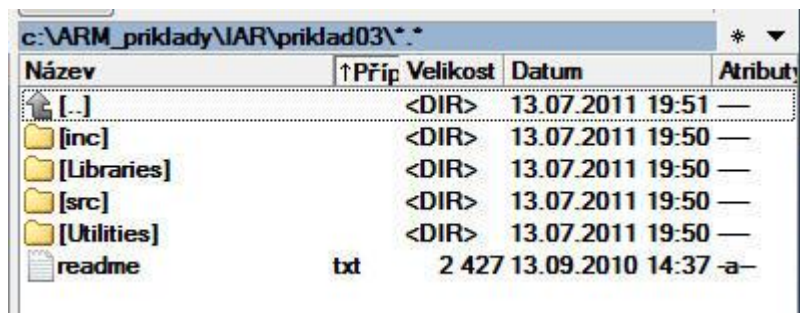




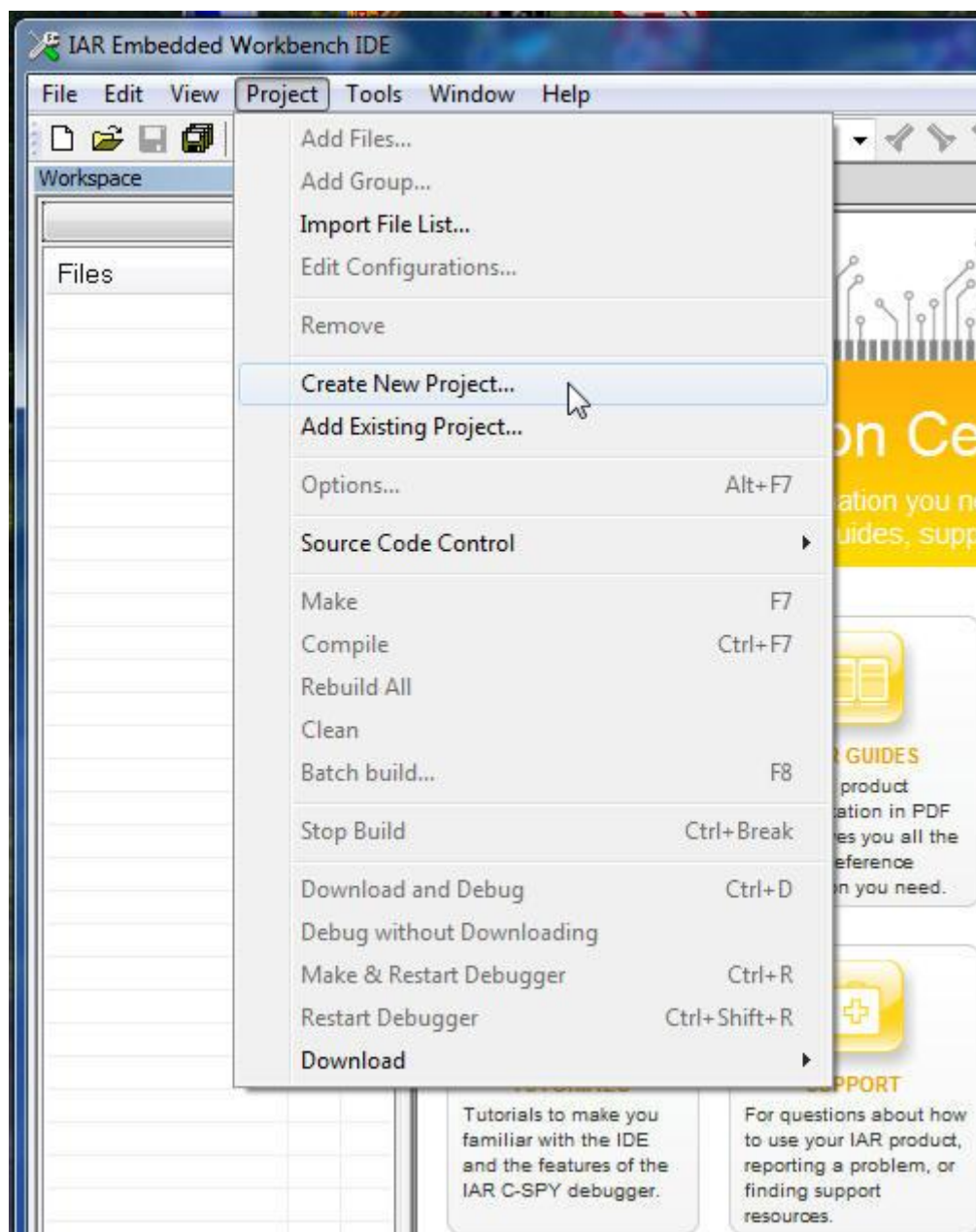
Program ukončíme, startkit resetujeme. LED připojená na PB5 podle očekávání bliká.

3.4.3 Blikač LED PC8 (příklad03)

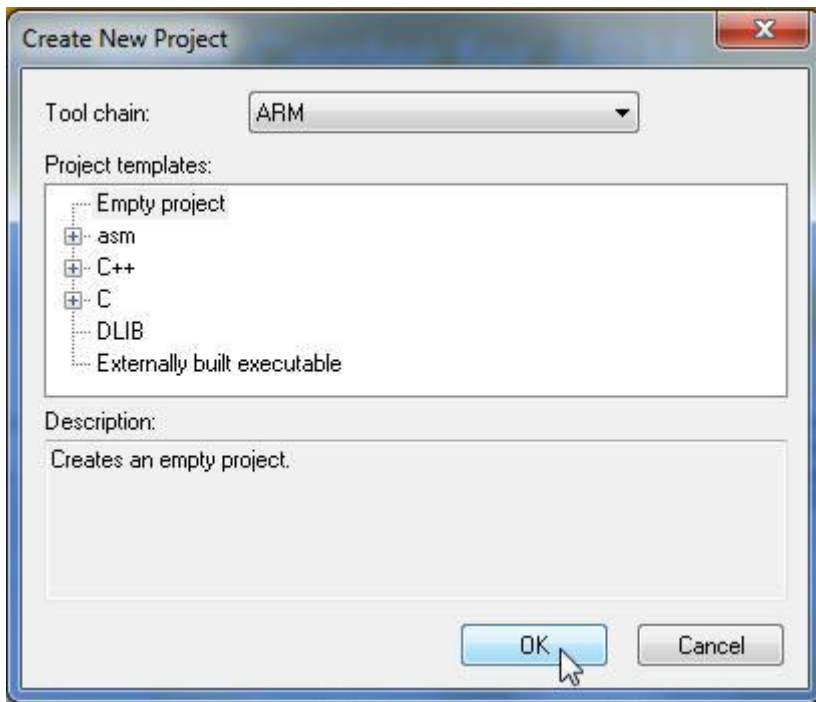
Nyní vytvoříme další projekt a workspace priklad03. Nejprve do adresáře priklad03 zkopírujeme adresáře Libraries a Utilities (včetně obsahu) z stm32vldiscovery_package a src a inc z projektu Demo (Discover)



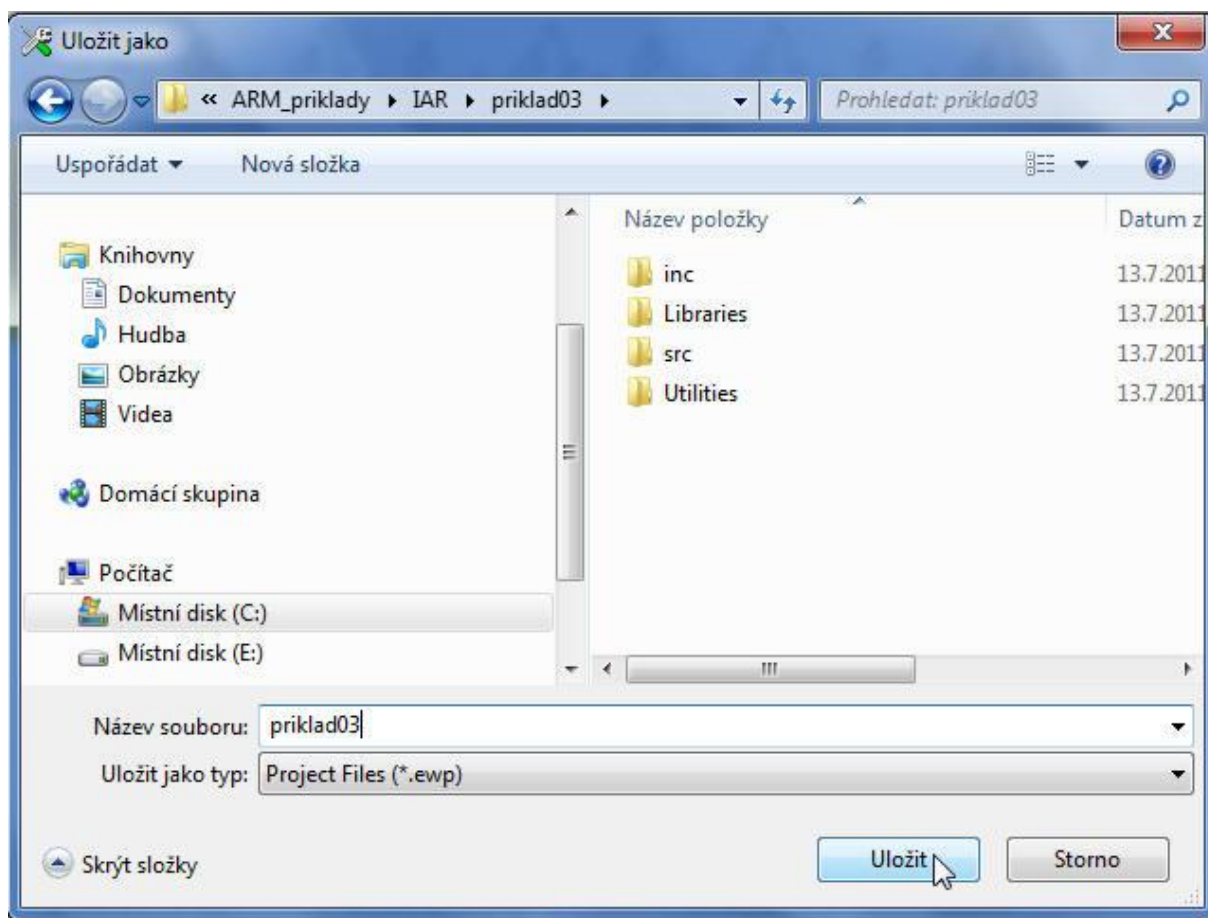
Poté již v IAR Embedded Workbench



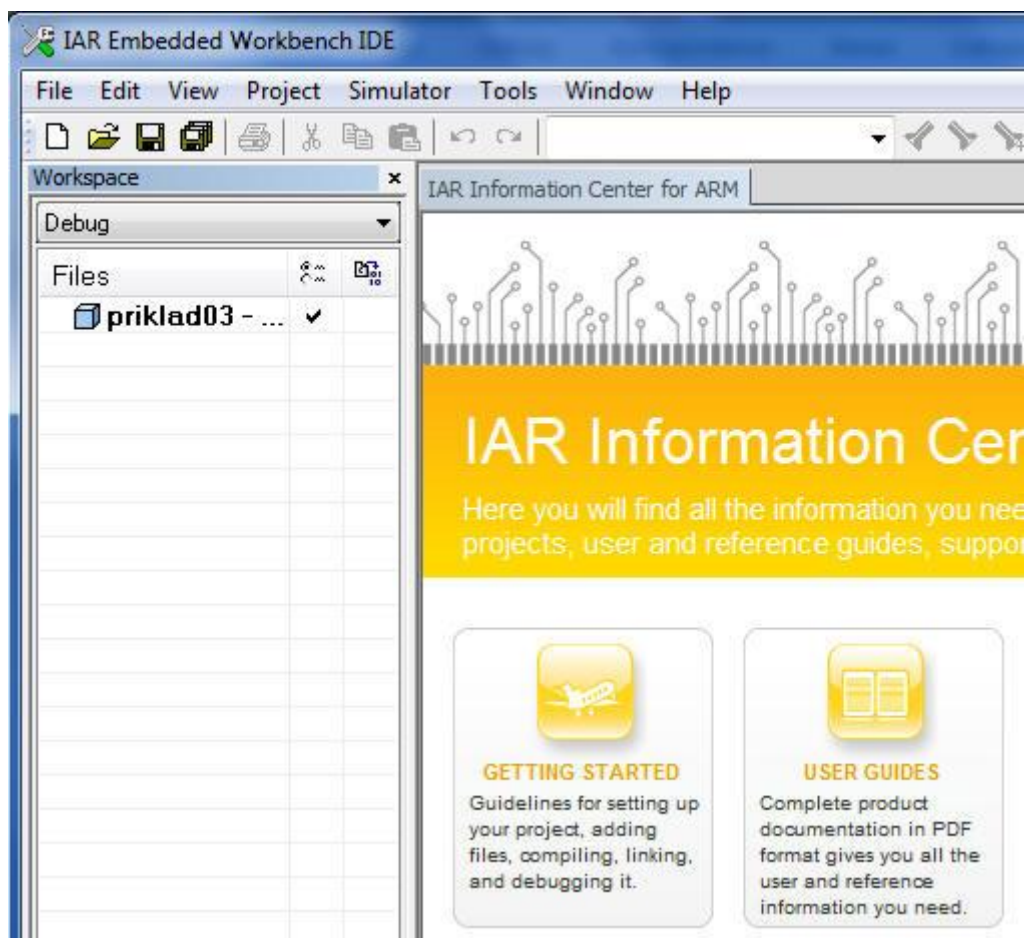
V menu vybereme **Project – Create New Project**



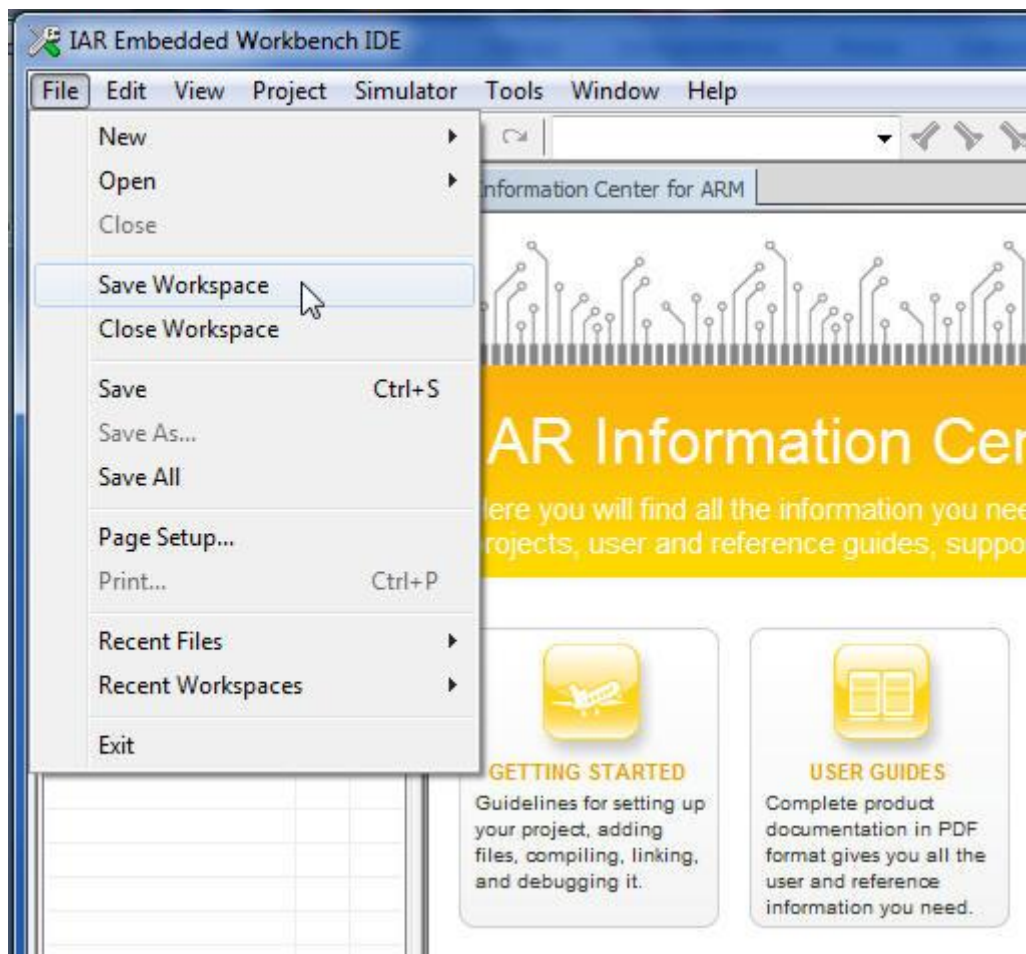
Vybereme **ARM** a klikneme na **OK**

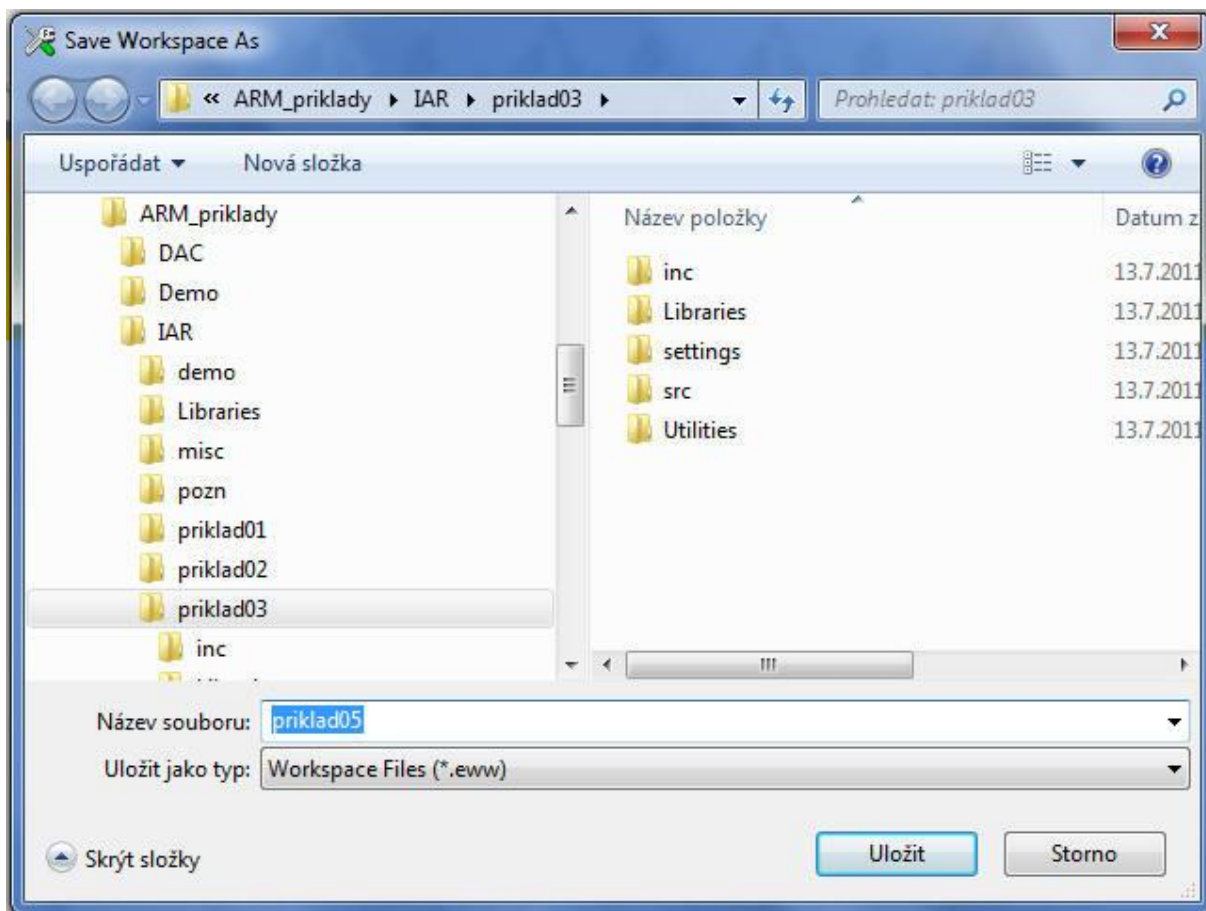


Zvolíme název projektu a klikneme na **Uložit**

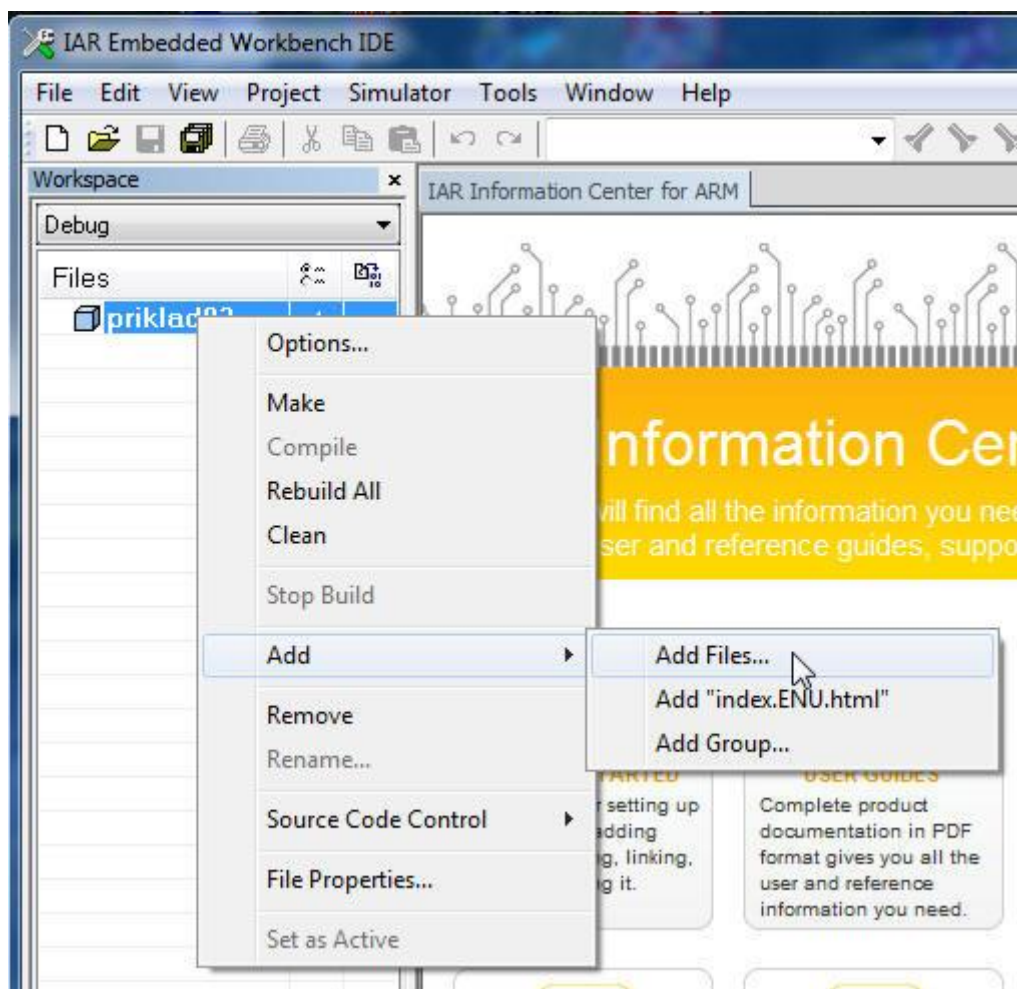


Uložíme nový workspace **File – Save Workspace**

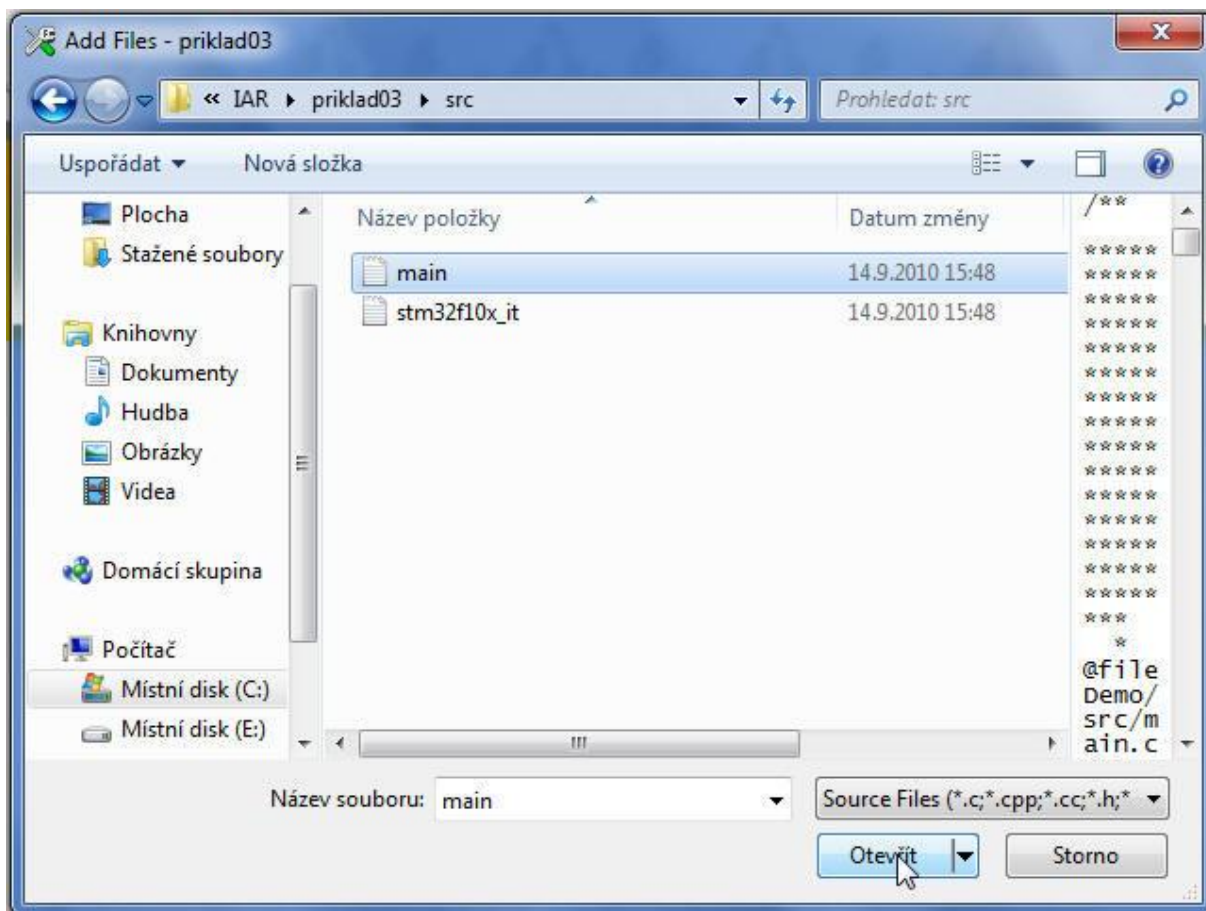




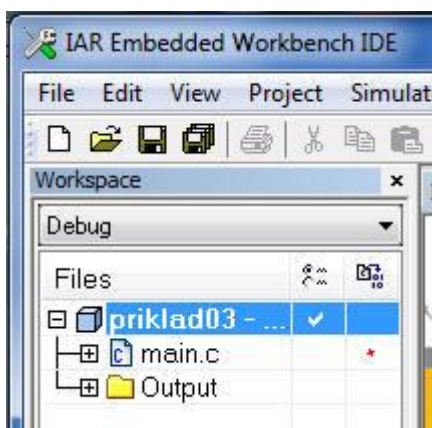
Vybereme název Workspace



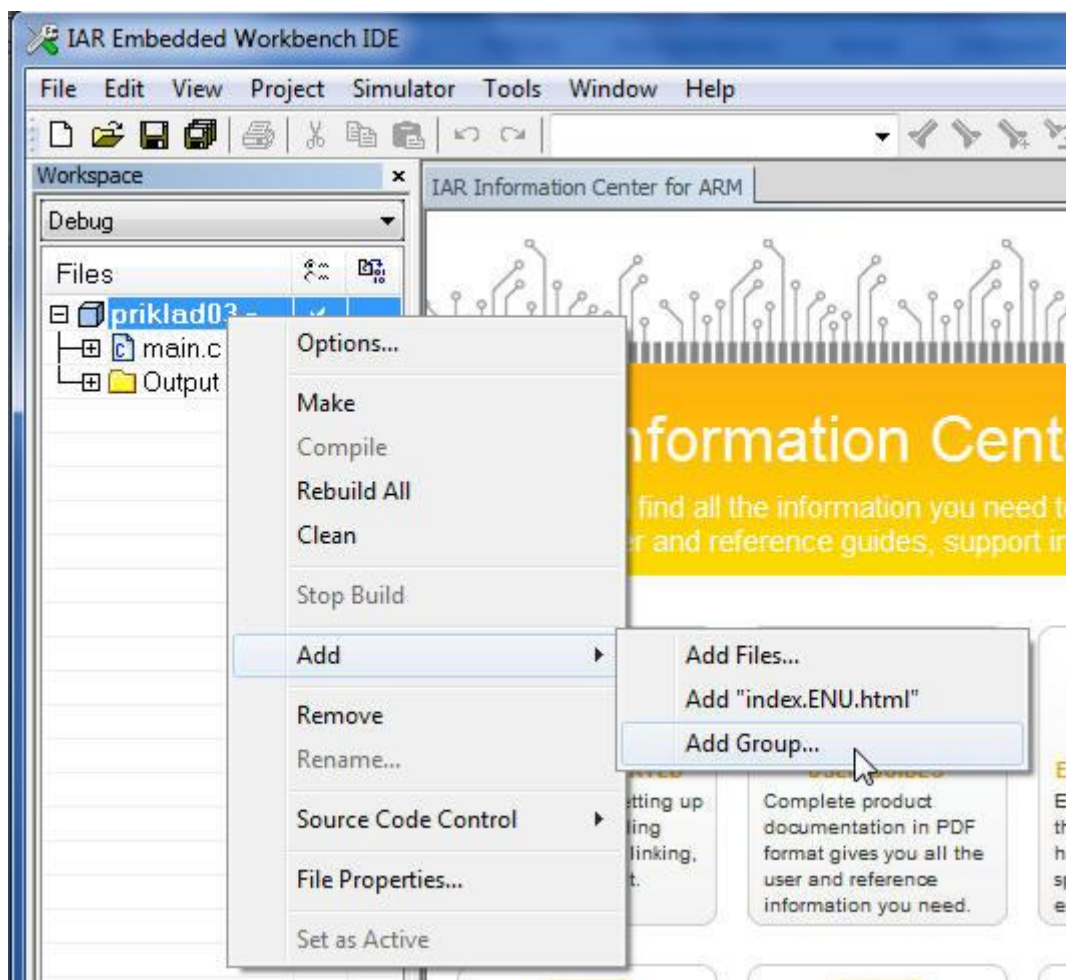
Přidáme soubor výběrem v místní nabídce **Add – Add Files...**



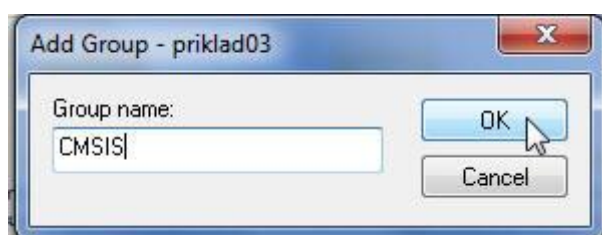
Vybereme **main.c**

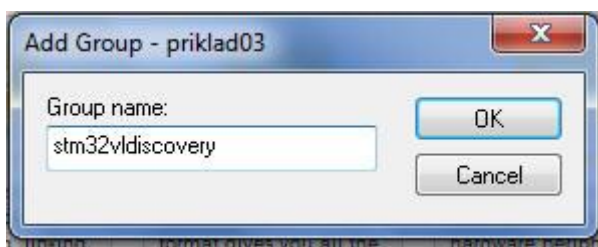
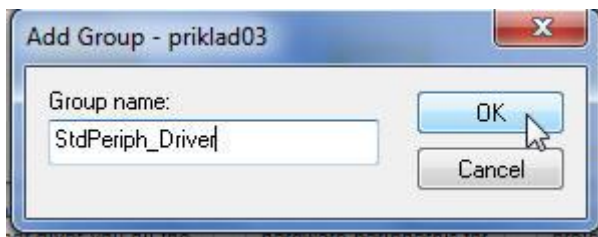
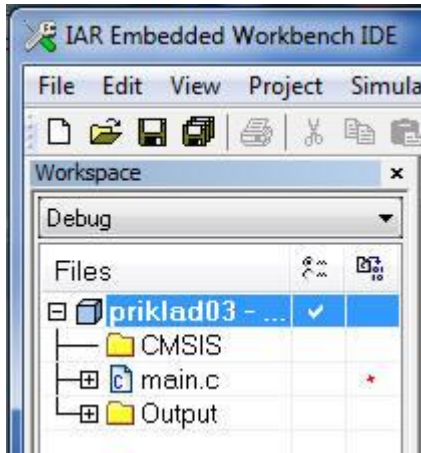


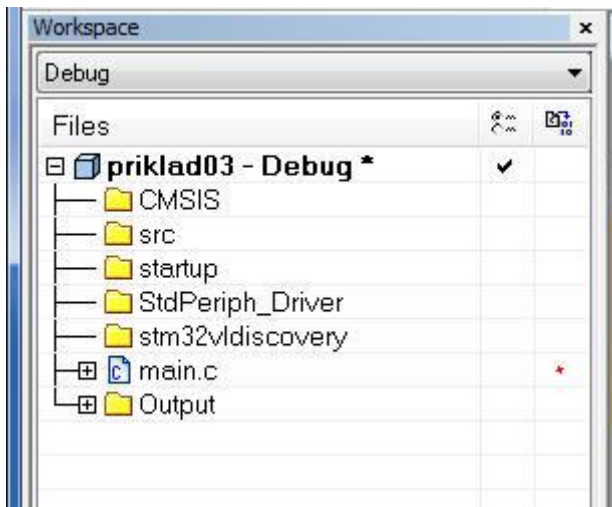
V místní nabídce vybereme **Options**



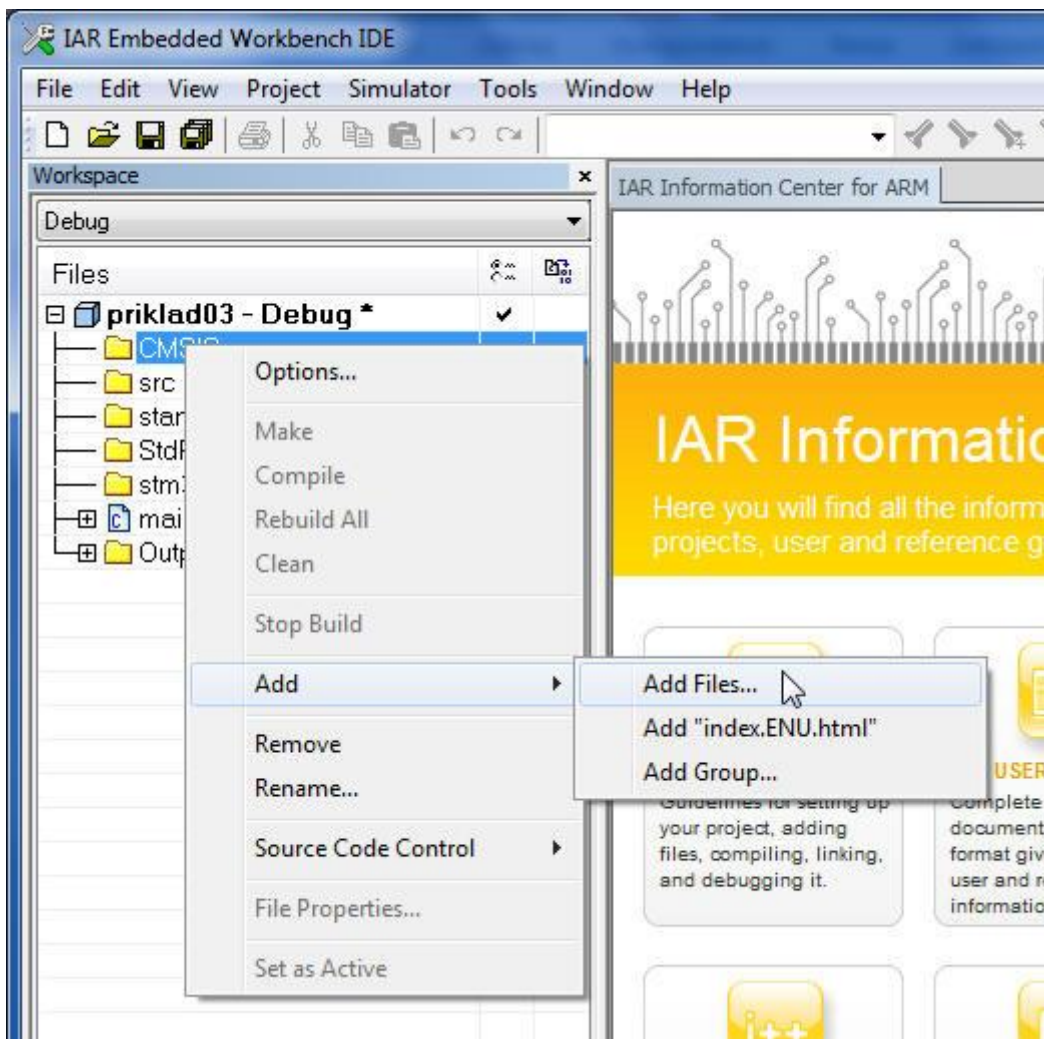
Přidáme podadresáře (Groups)

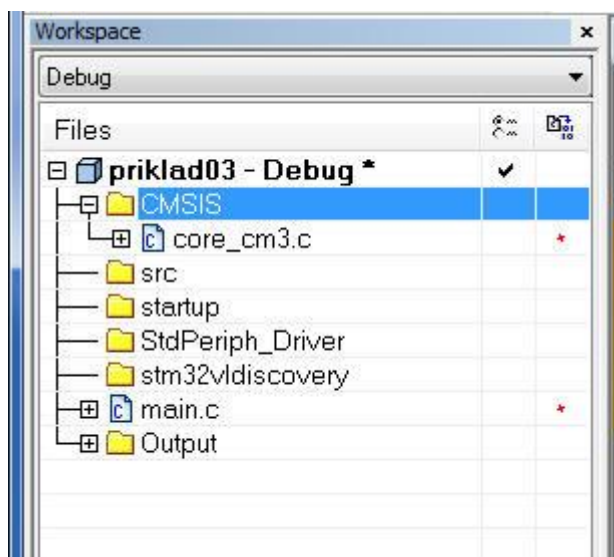
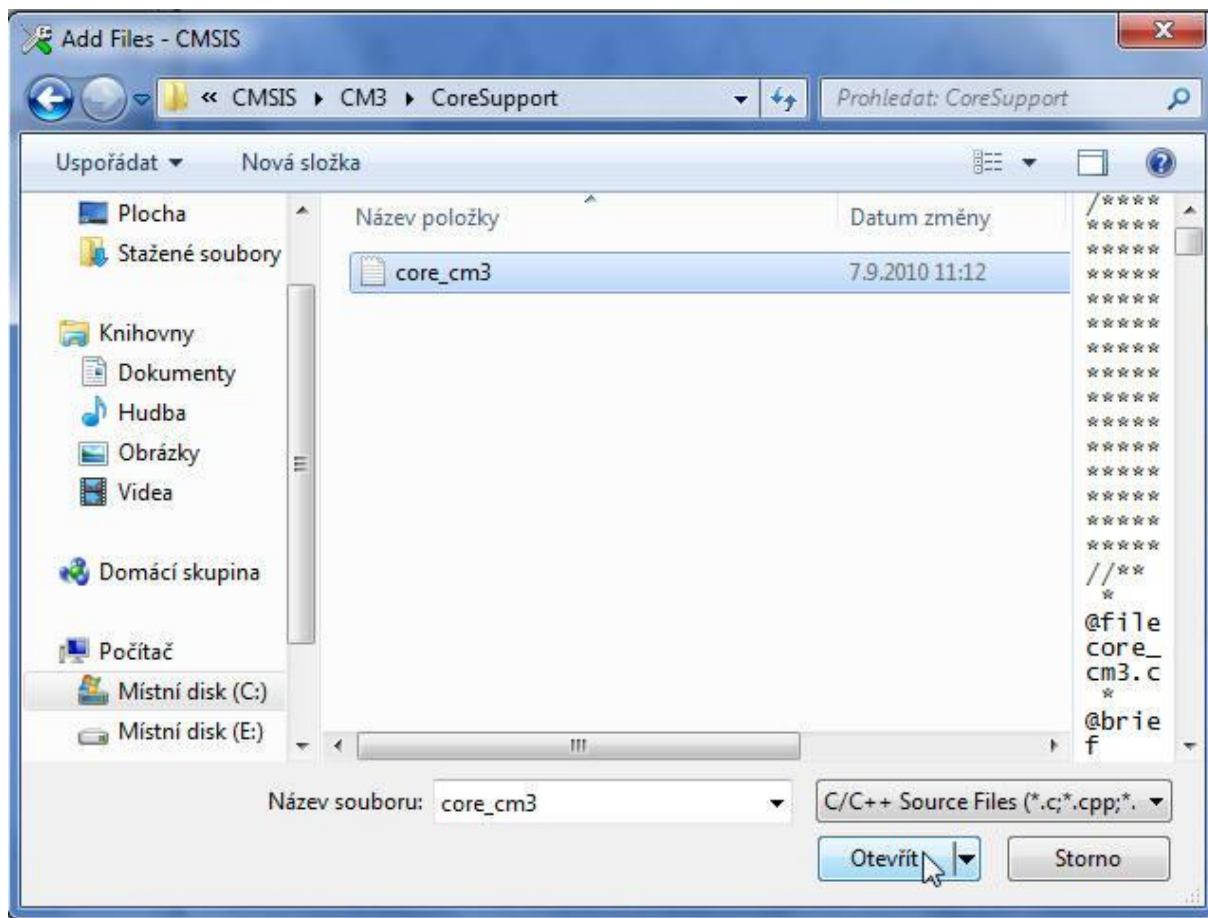




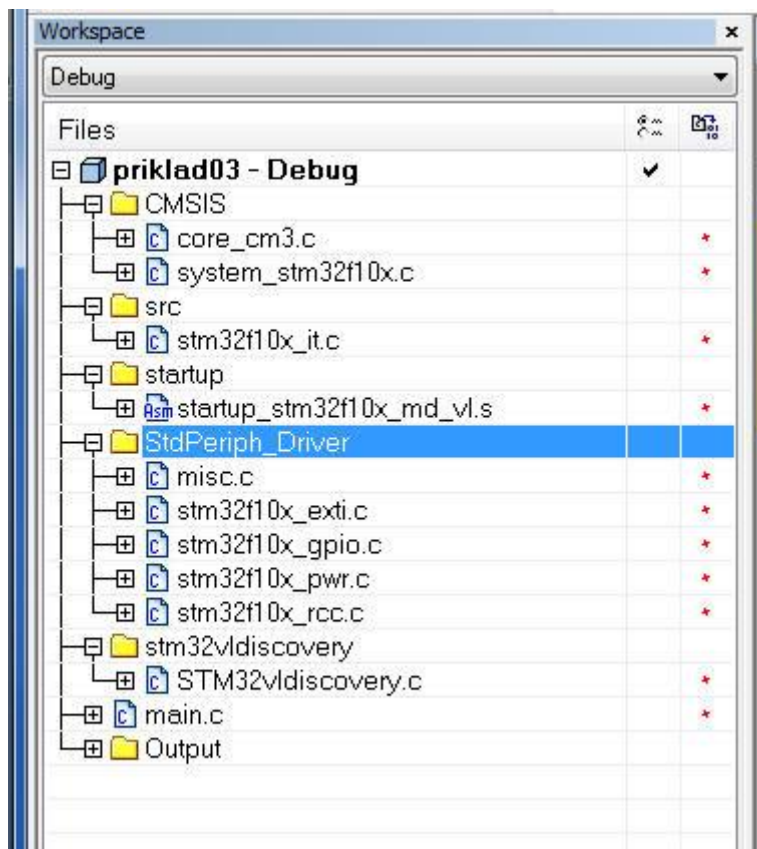


Dále soubory

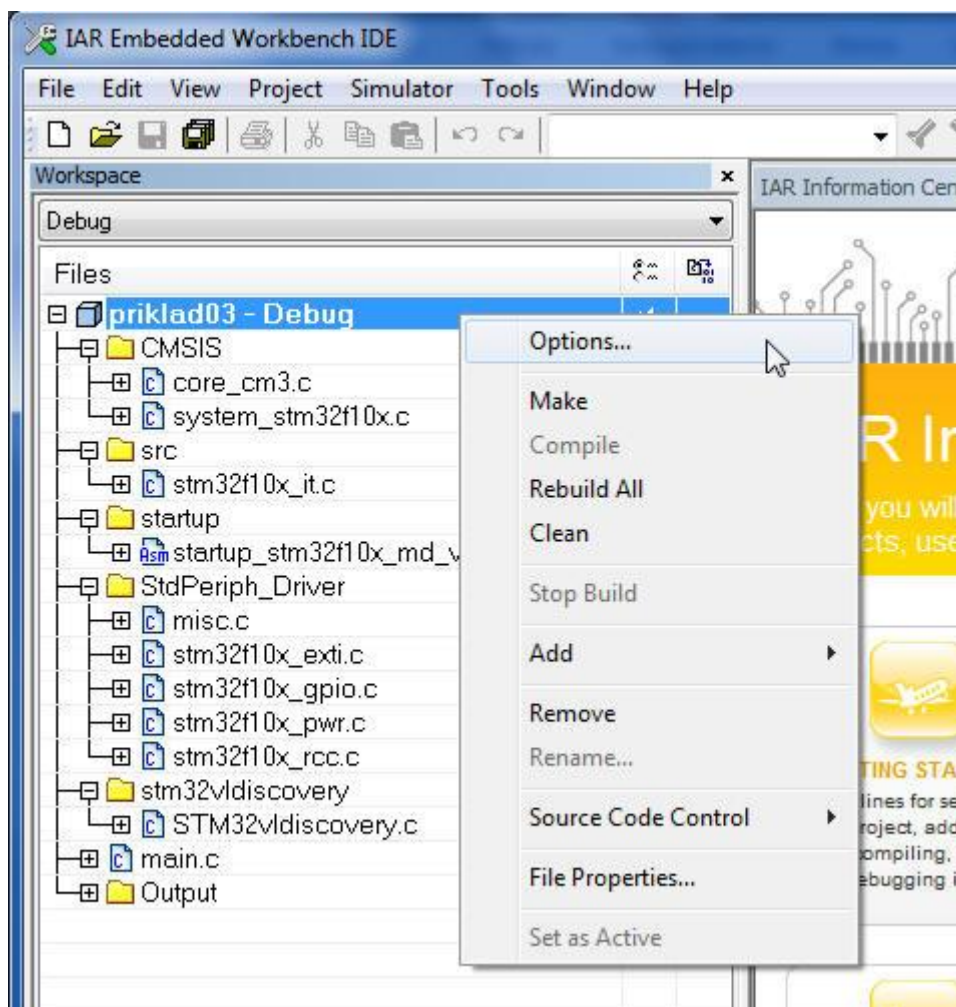


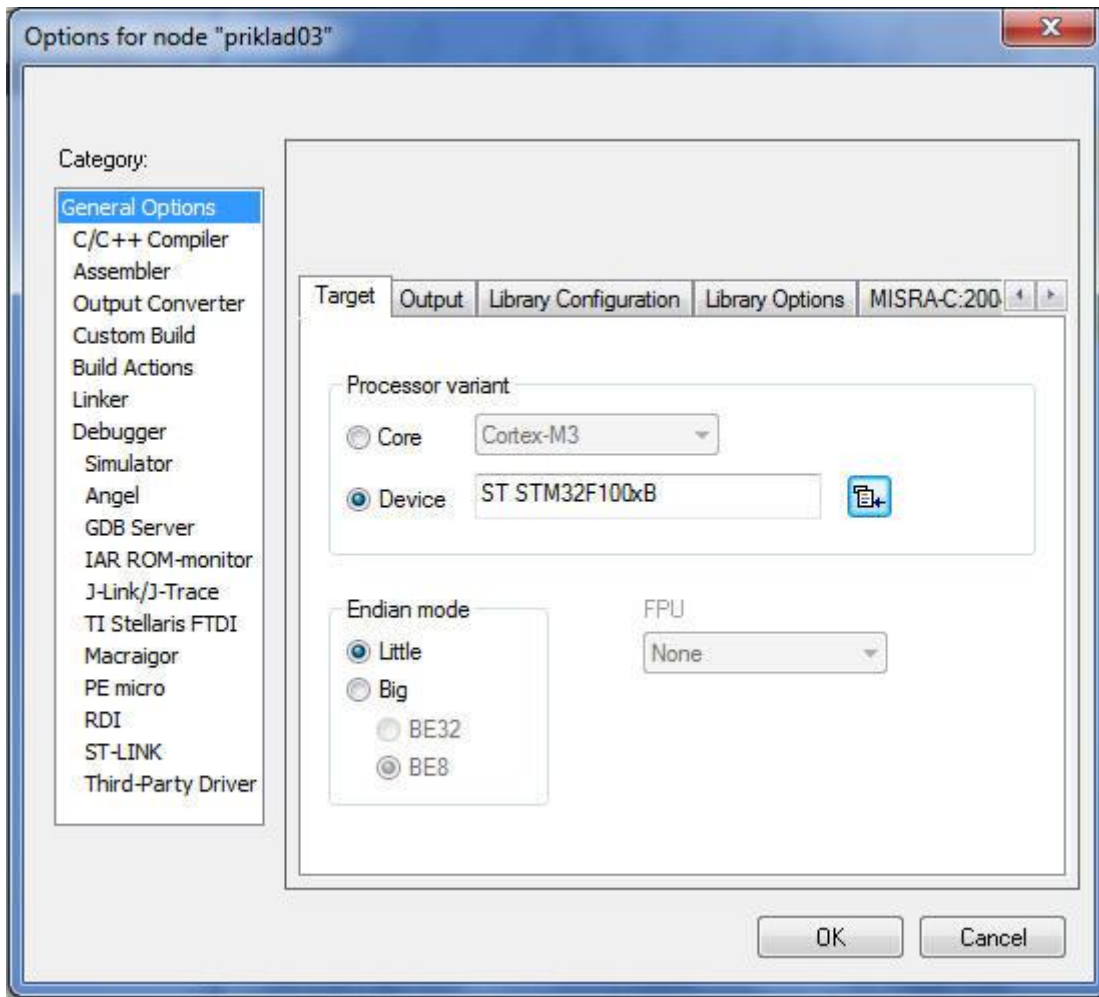


Nyní již máme potřebnou strukturu projektu

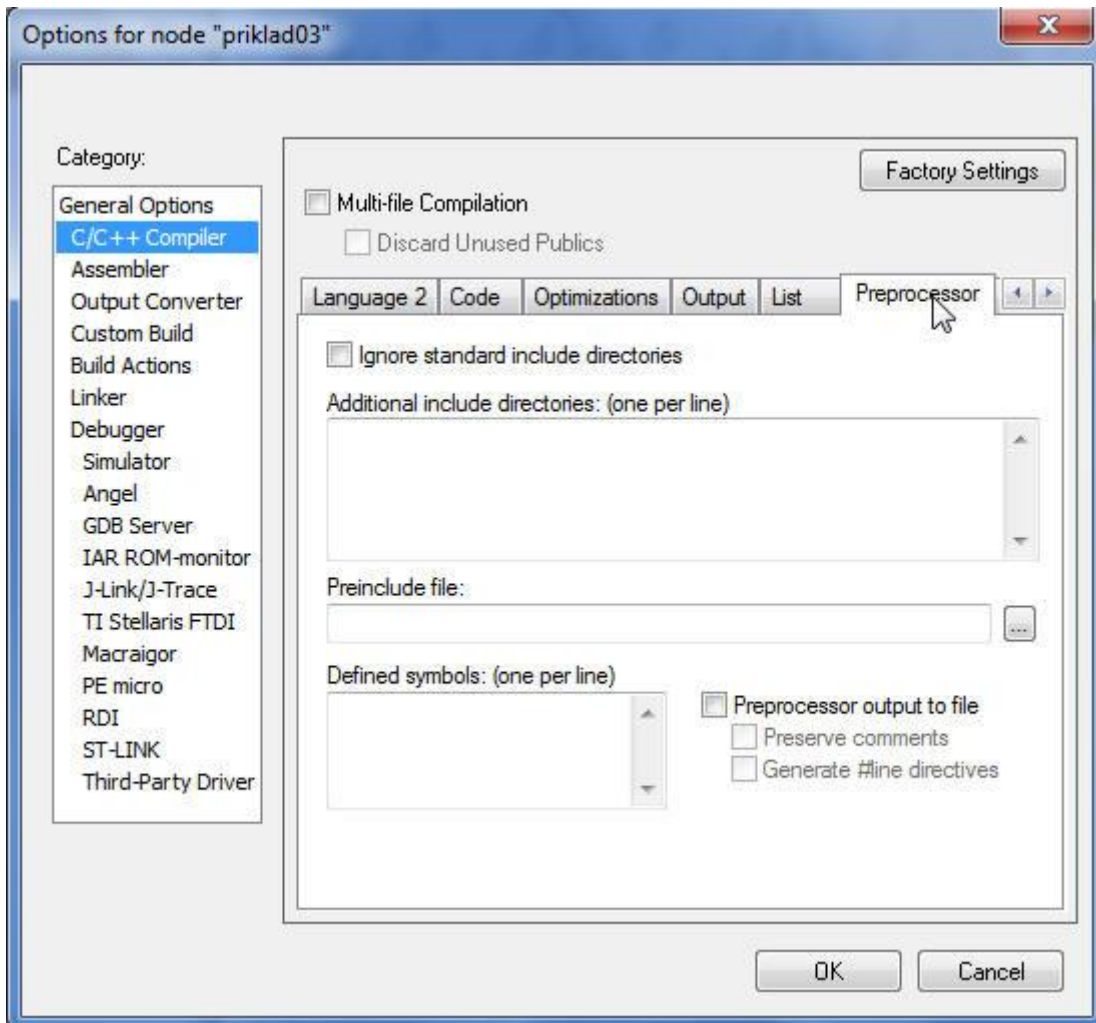


Všimněme si červených teček. Později se jimi budeme zabývat. Nyní ale nastavíme **Options**





Vybereme CPU STM32F100xB (podrobněji v předchozích kapitolách)

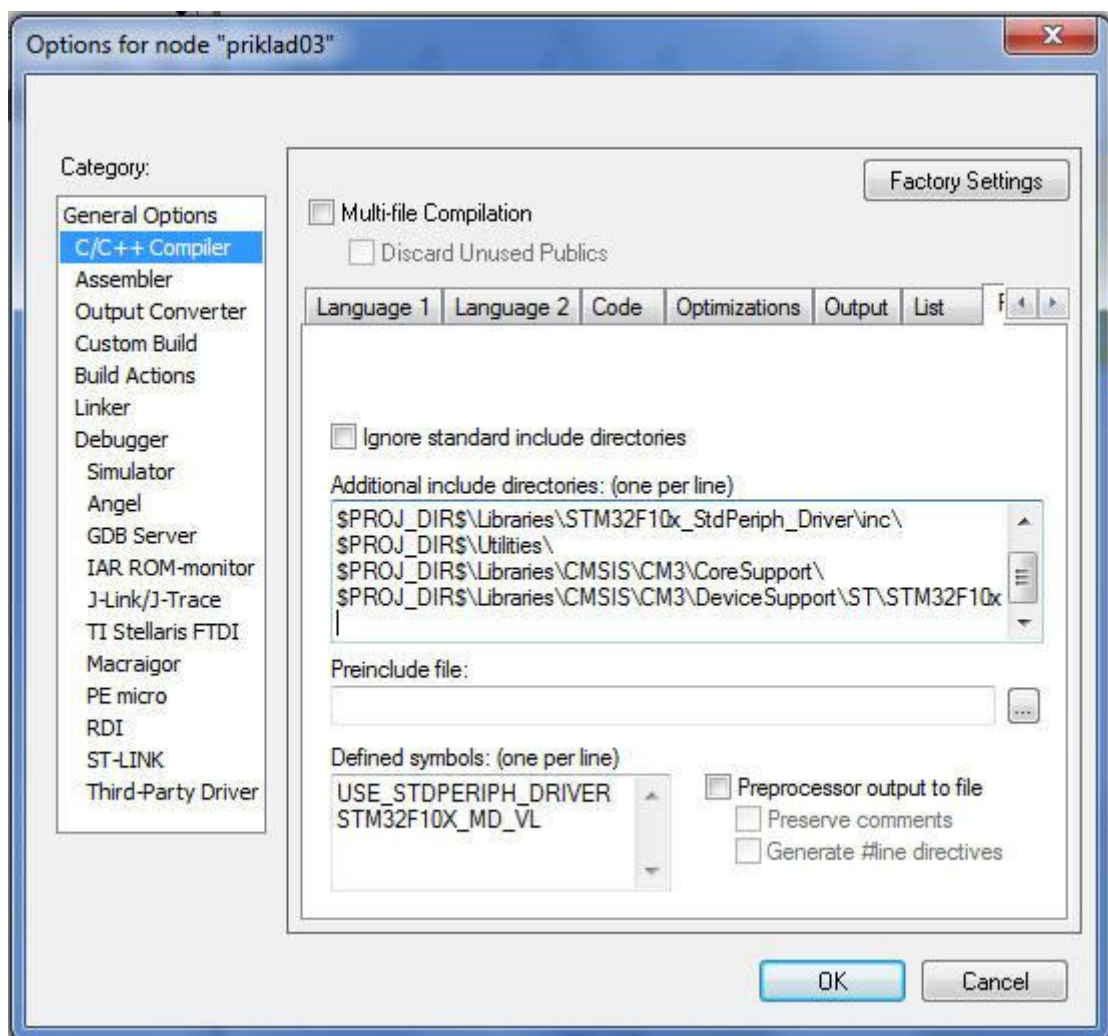


Additional include directories:

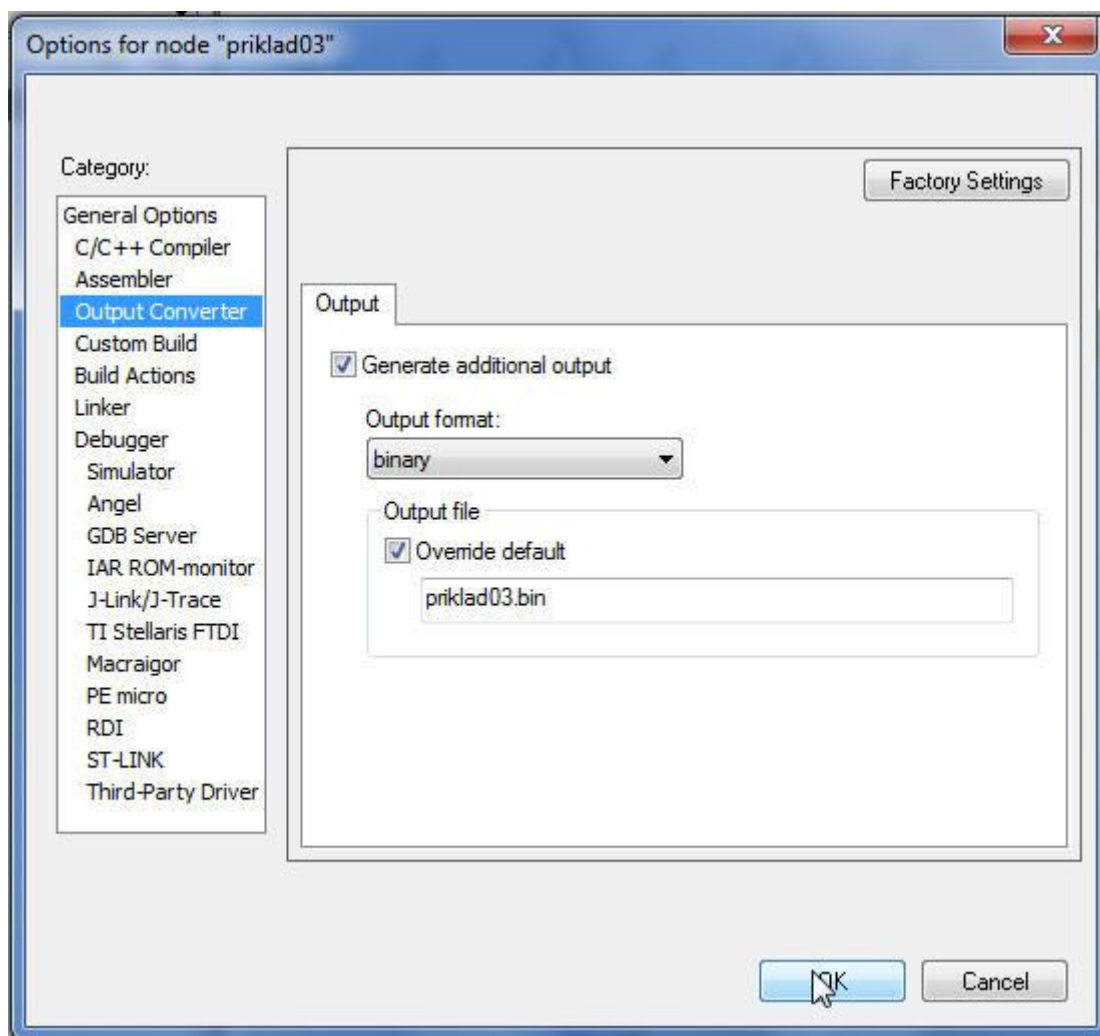
```
$PROJ_DIR$\inc\  
$PROJ_DIR$\Libraries\STM32F10x_StdPeriph_Driver\inc\  
$PROJ_DIR$\Utilities\  
$PROJ_DIR$\Libraries\CMSIS\CM3\CoreSupport\  
$PROJ_DIR$\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
```

Defined symbols:

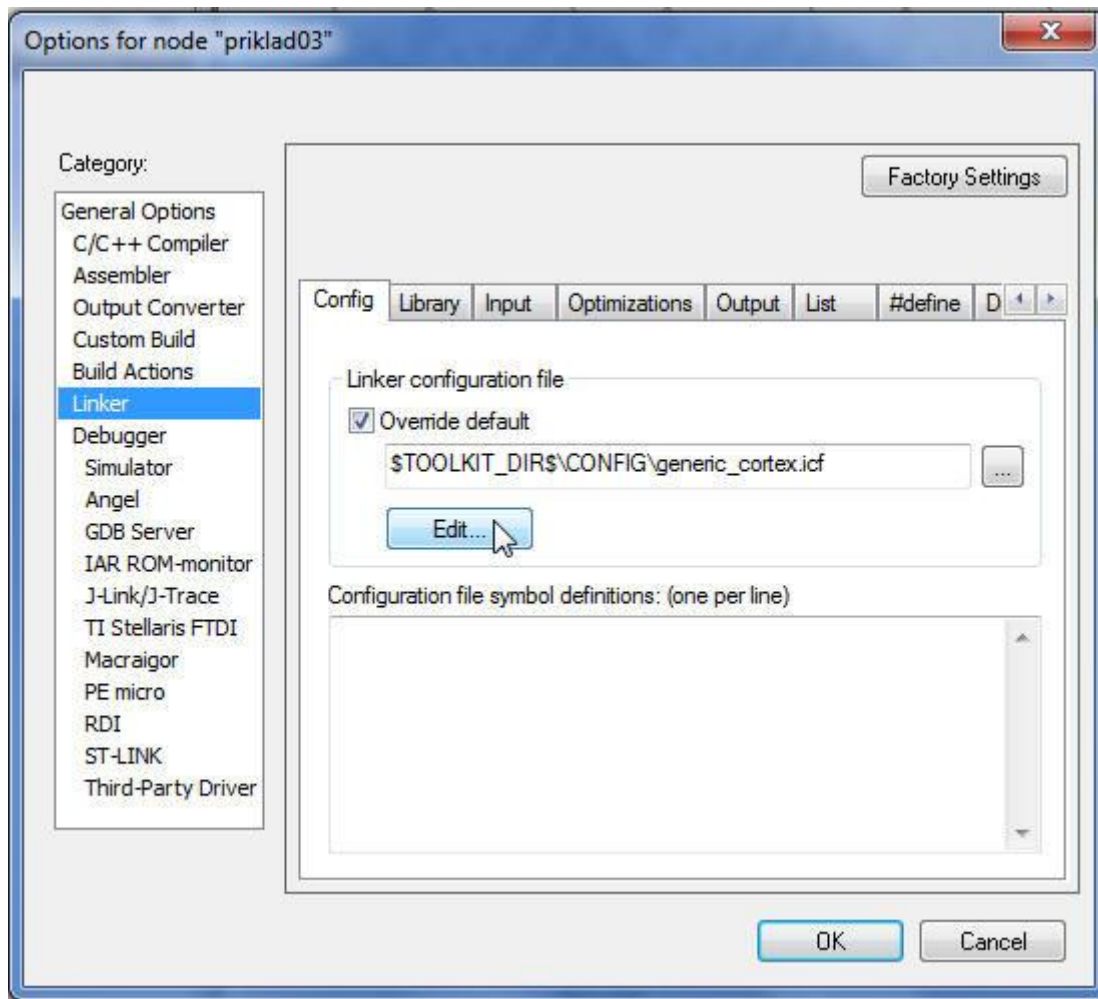
```
USE_STDPERIPH_DRIVER  
STM32F10X_MD_VL
```

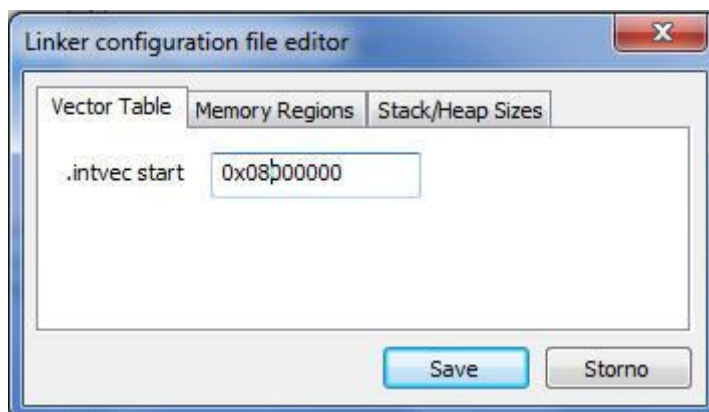
Vlastnosti překladače



Nastavíme výstup

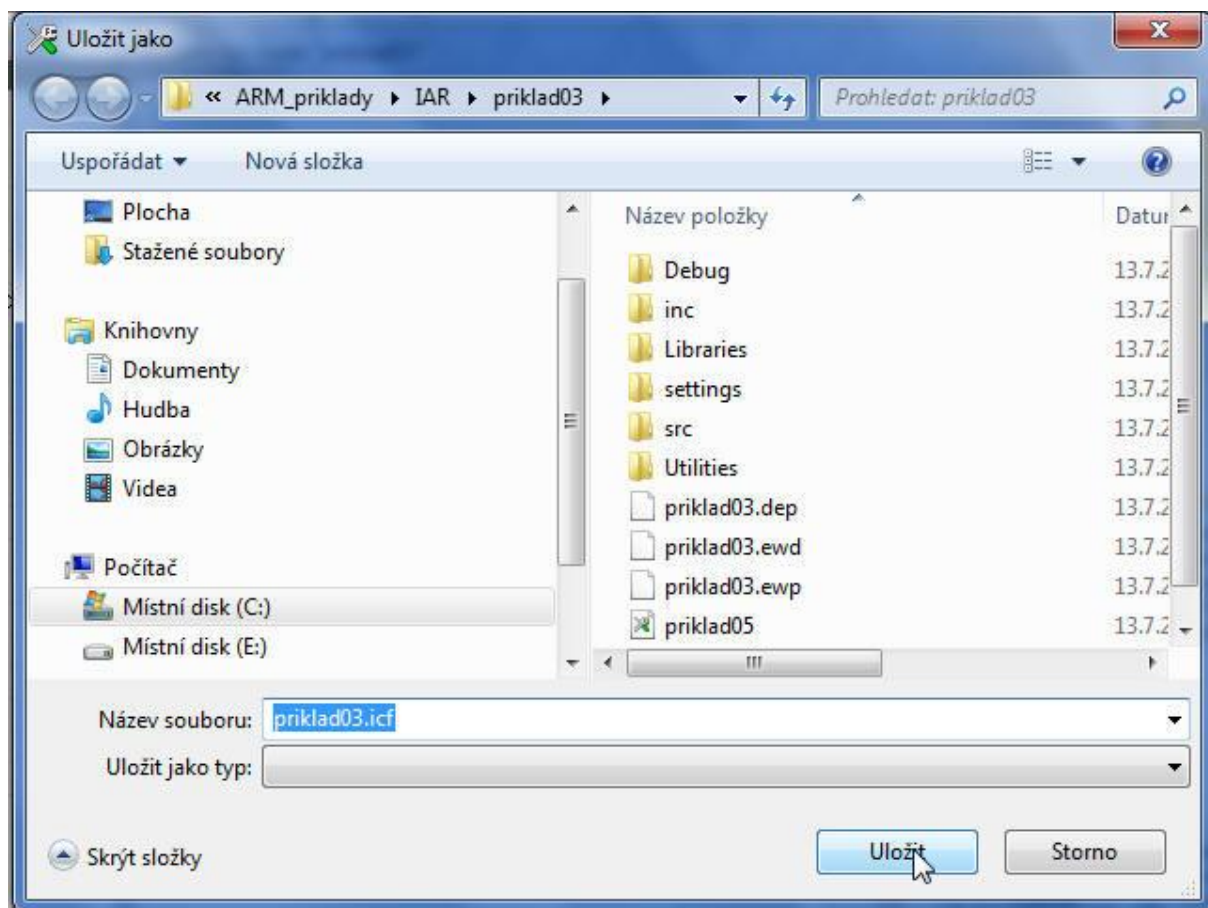


Nastavme linker

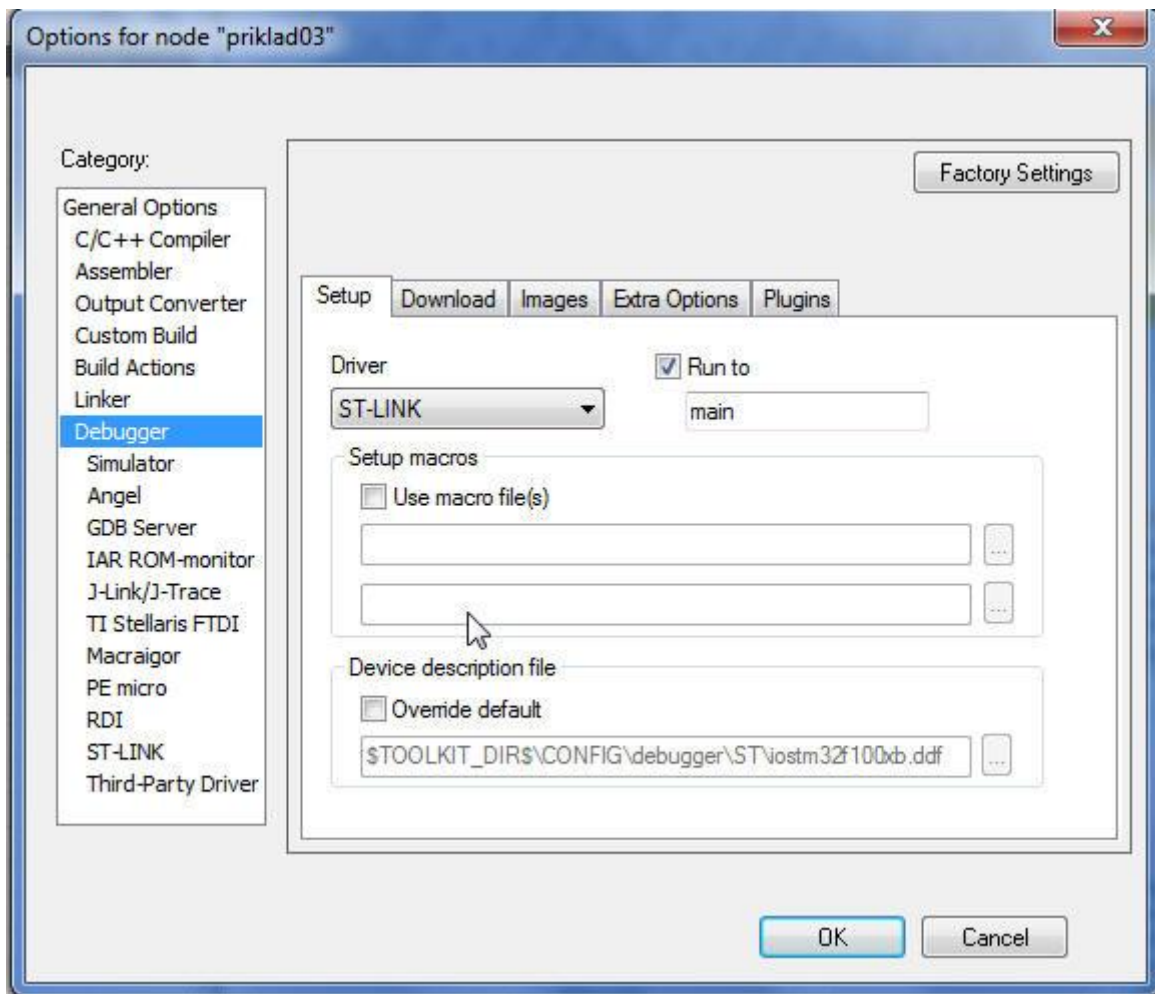




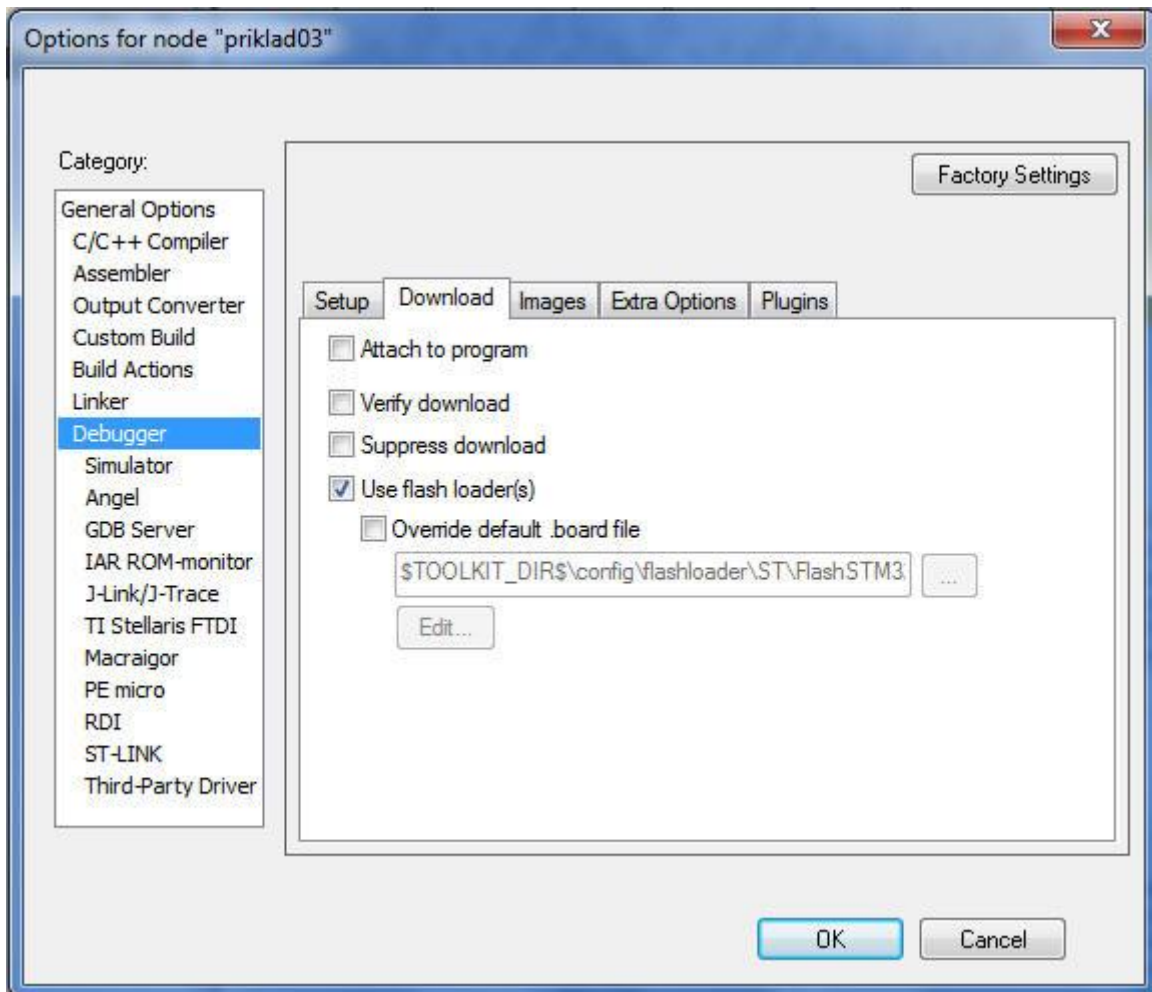
Po kliknutí na **Save** se objeví

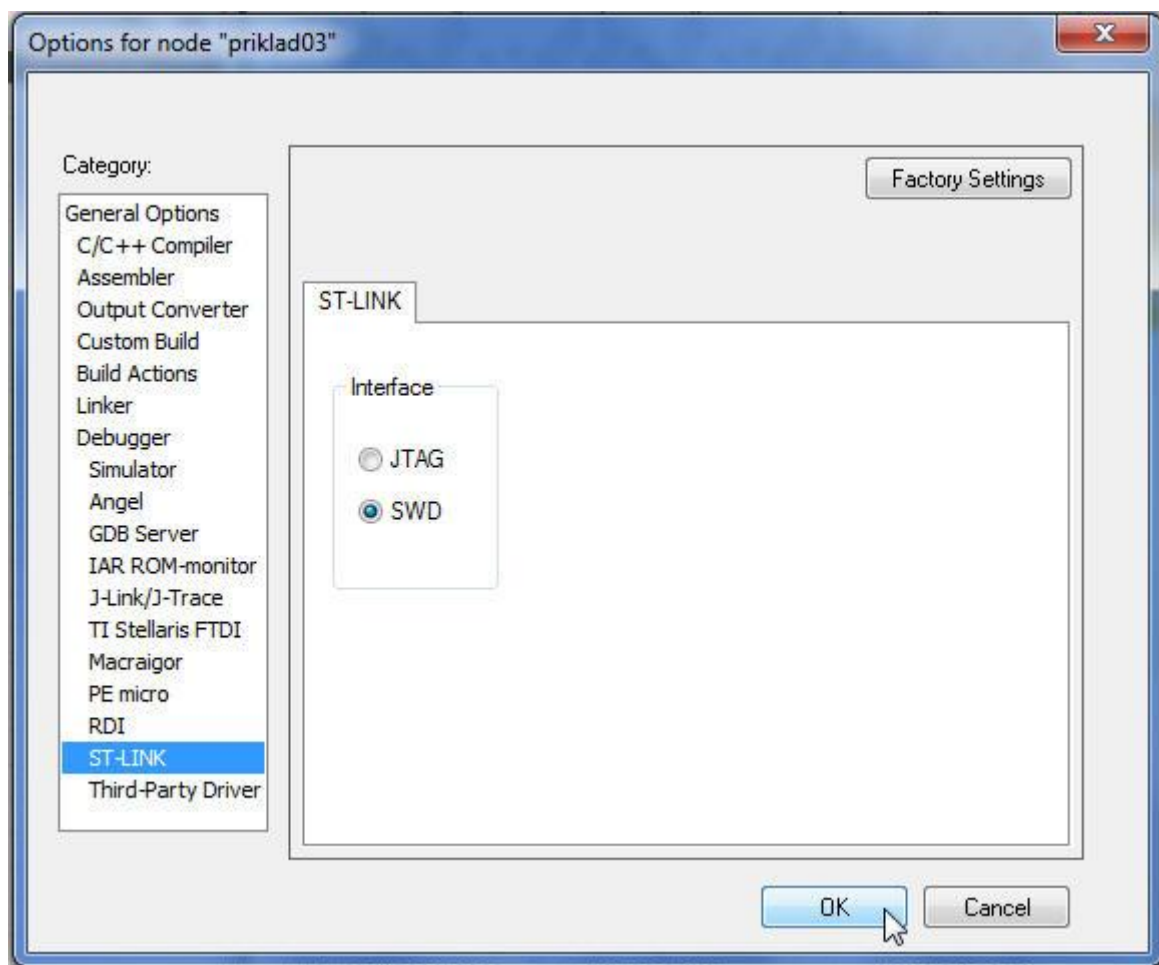


Vyplníme a klikneme na **Uložit**

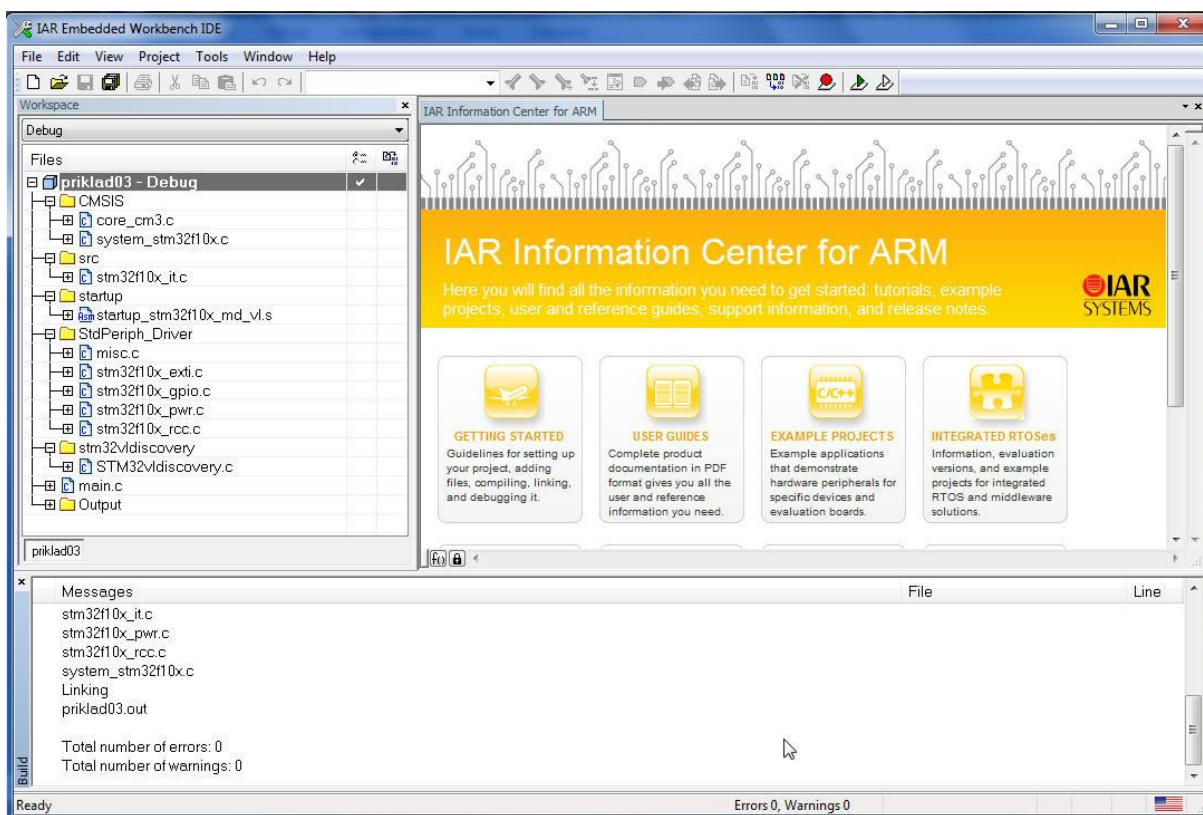


Dále nastavíme **Debugger**

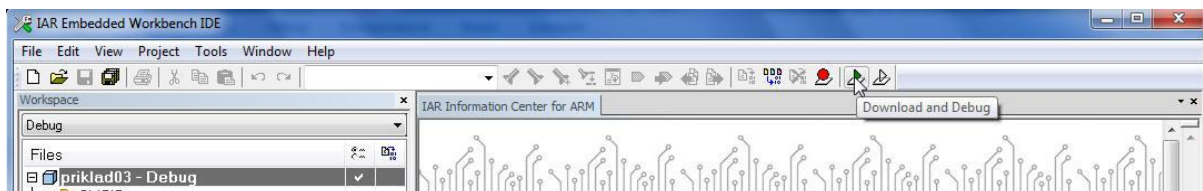


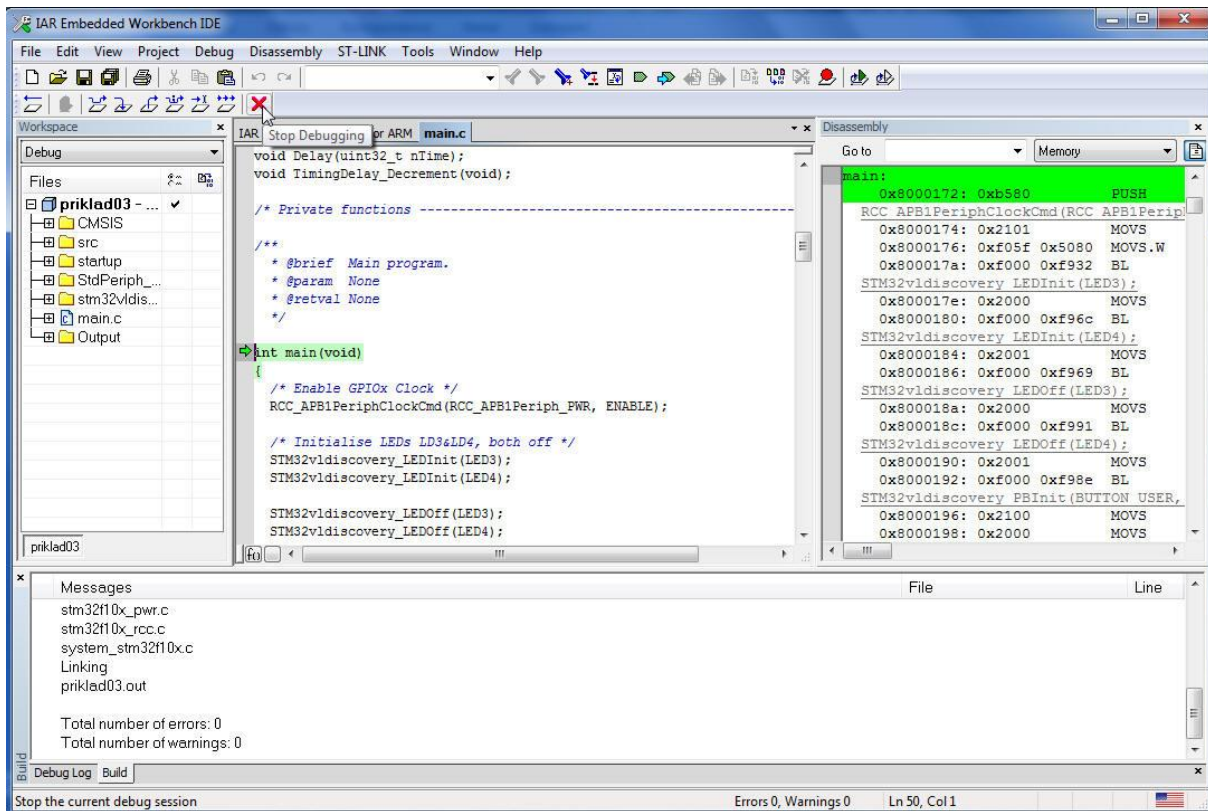


A ST-Link na SWD. Klikneme na **OK**. Poté v IDE spustíme překlad a sestavení (Build).



Překlad a sestavení proběhlo **OK**, zároveň vymizely červené tečky.





Odpojíme a připojíme startkit. Zelená led PC9 bliká. Nyní upravíme kód `main.c` tak, že vzájemně zaměníme výskyty LED3 a LED4

```

/* Includes -----
---*/
#include "stm32F10x.h"
#include "STM32vldiscovery.h"

/* Private typedef -----
---*/
/* Private define -----
---*/
#define LSE_FAIL_FLAG 0x80
#define LSE_PASS_FLAG 0x100
/* Private macro -----
---*/
/* Private consts -----
---*/

/* Private variables -----
---*/
u32 LSE_Delay = 0;
u32 count = 0;
u32 BlinkSpeed = 0;

```

```

u32 KeyState = 0;
static __IO uint32_t TimingDelay;
/* Private function prototypes -----
---*/
void Delay(uint32_t nTime);
void TimingDelay_Decrement(void);

/* Private functions -----
---*/

/**
 * @brief Main program.
 * @param None
 * @retval None
 */

int main(void)
{
    /* Enable GPIOx Clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);

    /* Initialise LEDs LD3&LD4, both off */
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);

    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);

    /* Initialise USER Button */
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);

    /* Setup SysTick Timer for 1 msec interrupts */
    if (SysTick_Config(SystemCoreClock / 1000))
    {
        /* Capture error */
        while (1);
    }

    /* Enable access to the backup register => LSE can be enabled */
    PWR_BackupAccessCmd(ENABLE);

    /* Enable LSE (Low Speed External Oscillation) */
    RCC_LSEConfig(RCC_LSE_ON);

    /* Check the LSE Status */
    while(1)
    {
        if(LSE_Delay < LSE_FAIL_FLAG)
        {
            /* check whether LSE is ready, with 4 seconds timeout */
            Delay (500);
            LSE_Delay += 0x10;
            if(RCC_GetFlagStatus(RCC_FLAG_LSERDY) != RESET)
            {
                /* Set flag: LSE PASS */
            }
        }
    }
}

```

```

LSE_Delay |= LSE_PASS_FLAG;
/* Turn Off Led4 */
STM32vldiscovery_LEDOff(LED3);
/* Disable LSE */
RCC_LSEConfig(RCC_LSE_OFF);
break;
}
}

/* LSE_FAIL_FLAG = 0x80 */
else if(LSE_Delay >= LSE_FAIL_FLAG)
{
    if(RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
    {
        /* Set flag: LSE FAIL */
        LSE_Delay |= LSE_FAIL_FLAG;
        /* Turn On Led4 */
        STM32vldiscovery_LEDOn(LED3);
    }
    /* Disable LSE */
    RCC_LSEConfig(RCC_LSE_OFF);
    break;
}
}

/* main while */
while(1)
{
    if(0 == STM32vldiscovery_PBGetState(BUTTON_USER))
    {
        if(KeyState == 1)
        {
            if(0 == STM32vldiscovery_PBGetState(BUTTON_USER))
            {
                /* USER Button released */
                KeyState = 0;
                /* Turn Off LED4 */
                STM32vldiscovery_LEDOff(LED3);
            }
        }
    }
    else if(STM32vldiscovery_PBGetState(BUTTON_USER))
    {
        if(KeyState == 0)
        {
            if(STM32vldiscovery_PBGetState(BUTTON_USER))
            {
                /* USER Button released */
                KeyState = 1;
                /* Turn ON LED4 */
                STM32vldiscovery_LEDOn(LED3);
                Delay(1000);
                /* Turn OFF LED4 */
                STM32vldiscovery_LEDOff(LED3);
                /* BlinkSpeed: 0 -> 1 -> 2, then re-cycle */
            }
        }
    }
}

```

```

        BlinkSpeed ++ ;
    }
}
count++;
Delay(100);
/* BlinkSpeed: 0 */
if(BlinkSpeed == 0)
{
    if(4 == (count % 8))
        STM32vldiscovery_LEDOn(LED4);
    if(0 == (count % 8))
        STM32vldiscovery_LEDOff(LED4);
}
/* BlinkSpeed: 1 */
if(BlinkSpeed == 1)
{
    if(2 == (count % 4))
        STM32vldiscovery_LEDOn(LED4);
    if(0 == (count % 4))
        STM32vldiscovery_LEDOff(LED4);
}
/* BlinkSpeed: 2 */
if(BlinkSpeed == 2)
{
    if(0 == (count % 2))
        STM32vldiscovery_LEDOn(LED4);
    else
        STM32vldiscovery_LEDOff(LED4);
}
/* BlinkSpeed: 3 */
else if(BlinkSpeed == 3)
    BlinkSpeed = 0;
}
}

/**
 * @brief Inserts a delay time.
 * @param nTime: specifies the delay time length, in milliseconds.
 * @retval None
 */
void Delay(uint32_t nTime)
{
    TimingDelay = nTime;

    while(TimingDelay != 0);
}

/**
 * @brief Decrements the TimingDelay variable.
 * @param None
 * @retval None
 */
void TimingDelay_Decrement(void)
{

```

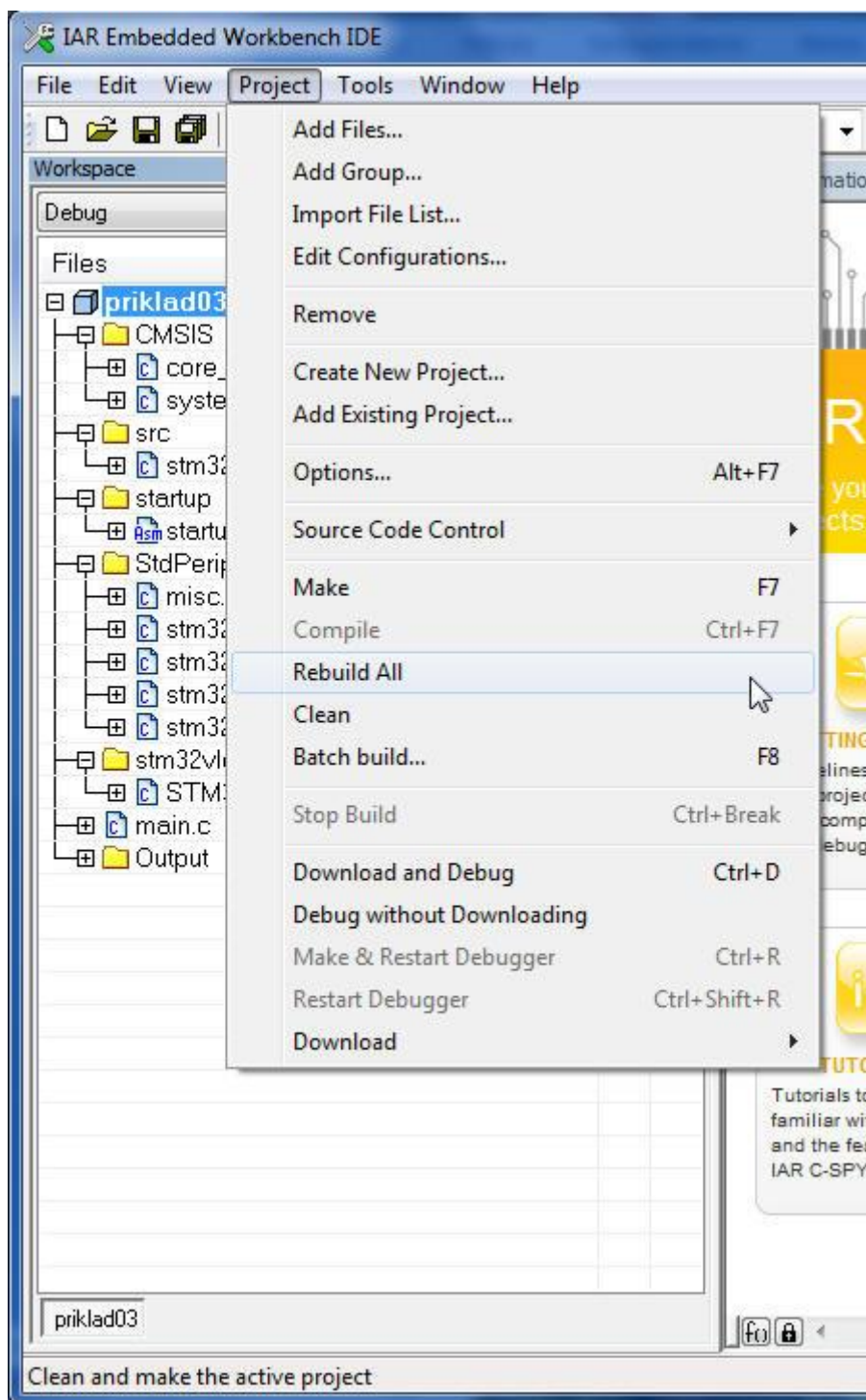
```

if (TimingDelay != 0x00)
{
    TimingDelay--;
}
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t* file, uint32_t line)
{
    /* User can add his own implementation to report the file name and line
    number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
    line) */

    /* Infinite loop */
    while (1)
    {
    }
}
#endif

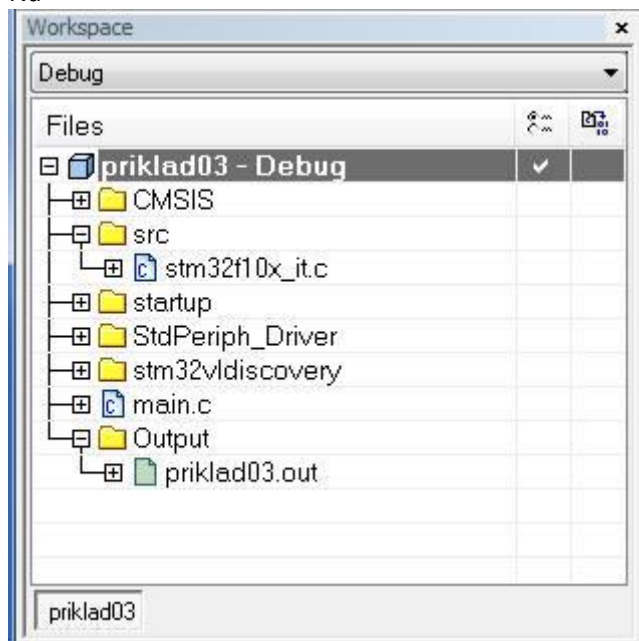
```



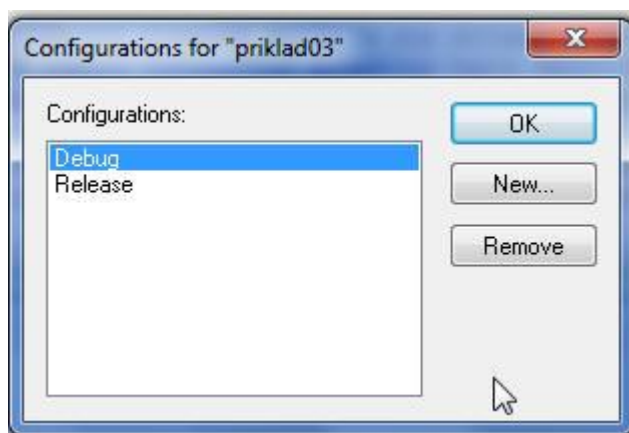
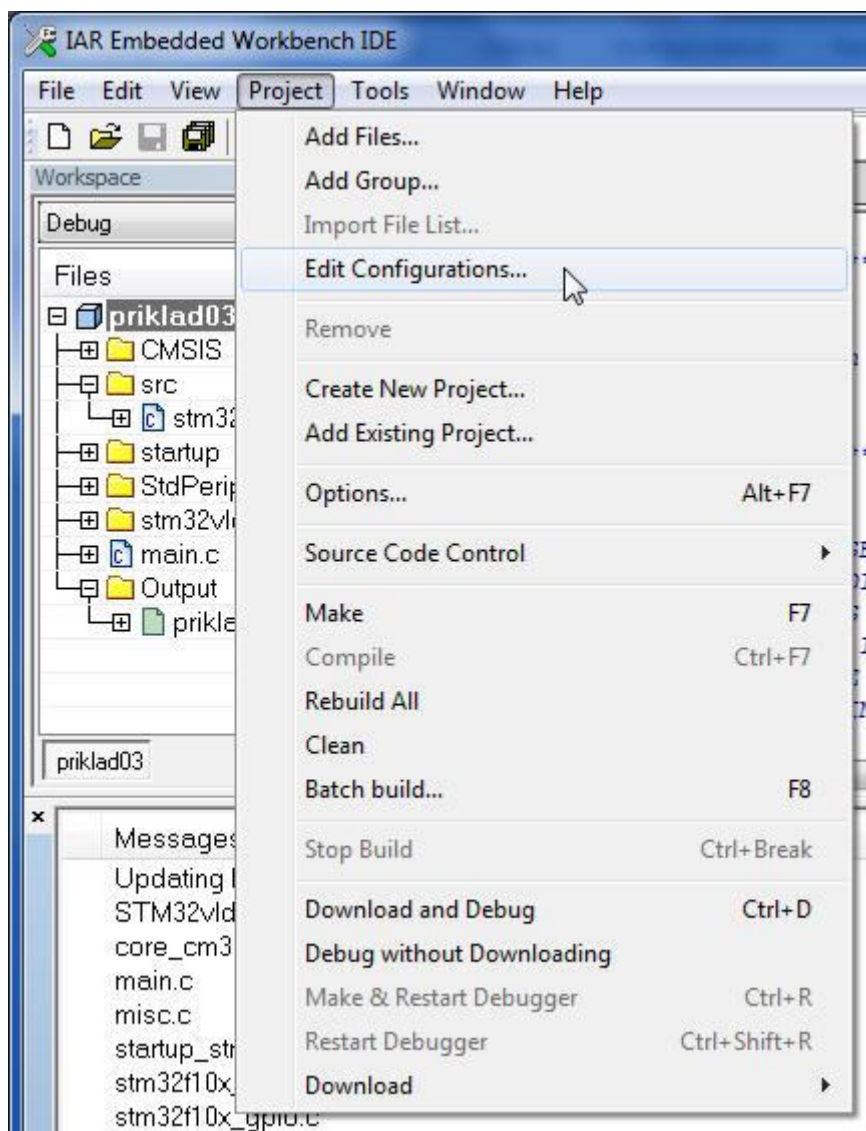
Po překladi a sestavení a spuštění programu v startkitu bliká modrá LED, po stisknutí USB tlačítka zelená a zrychlí se blikání modré.

POZNÁMKA:

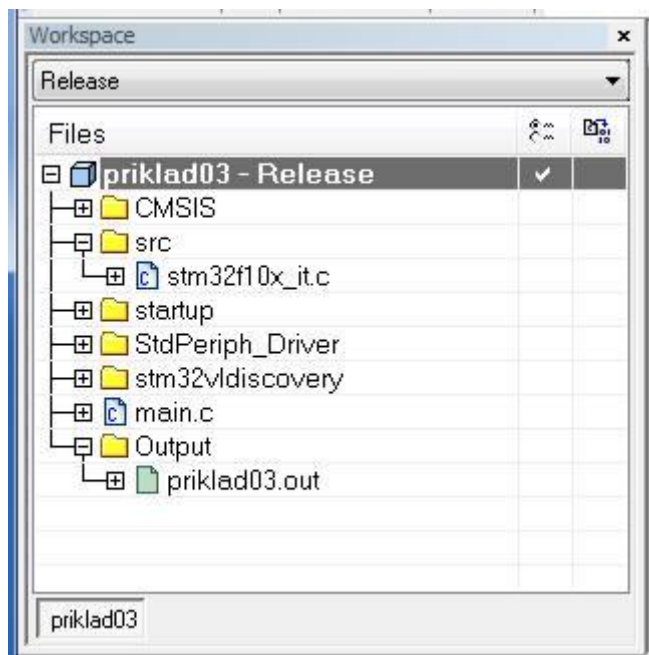
Na



Všimněme si popisu příklad03 – **Debug**. Pokud již budeme s naším programem spokojeni, budeme jistě chtít verzi **Release**.



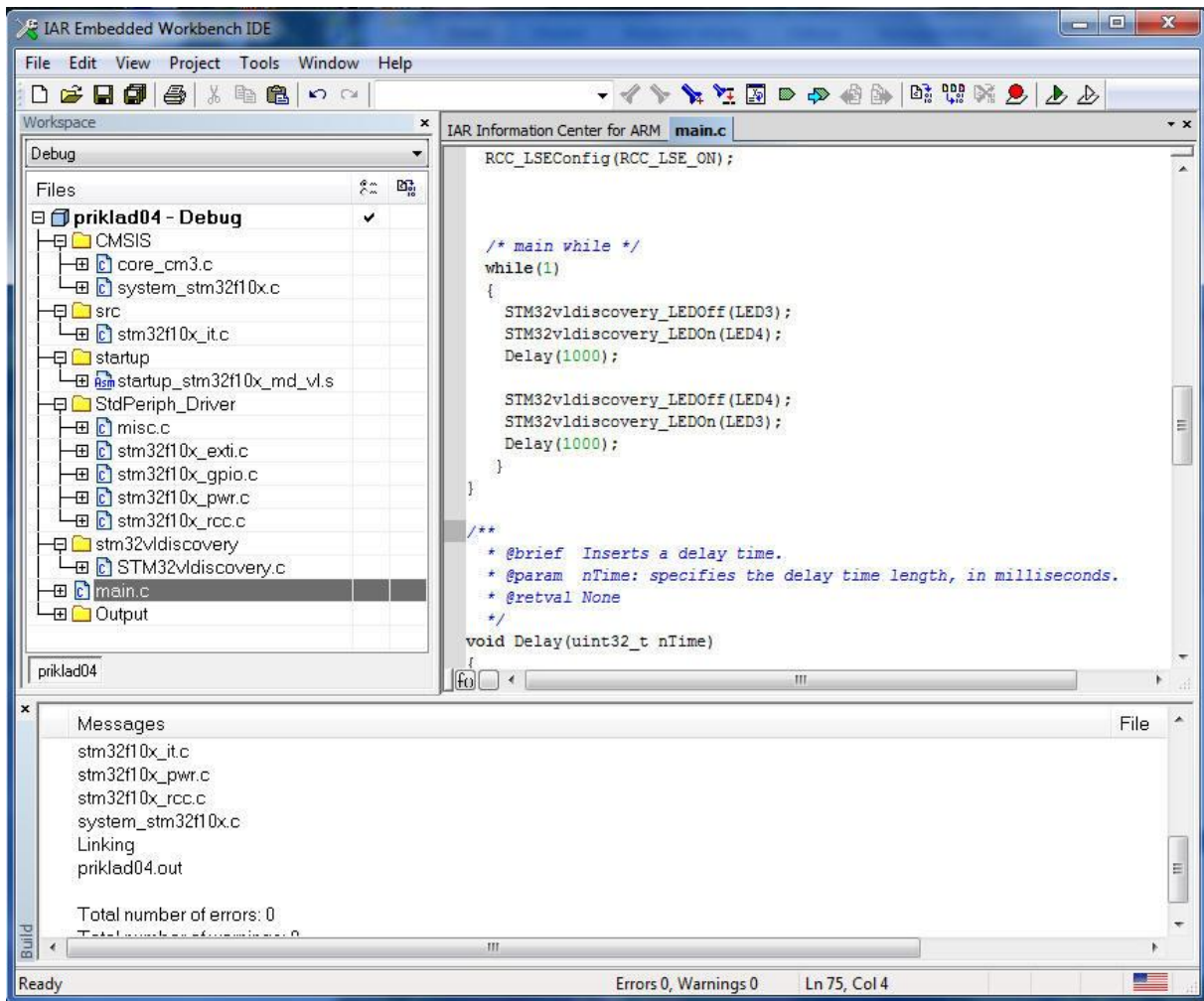
Přepneme na **Release**, potvrdíme **OK**.



Ve skutečnosti to tak jednoduché není. Výše uvedený obrázek se objeví s červenými tečkami, překlad hlásí velké množství chyb. Musíme totiž nastavit **Options**, stejně jako jsme to prováděli pro **Debug** verzi.

3.4.4 Další blikáč (příklad04)

Od předchozího příkladu se liší jen obsahem souboru **main.c** , takže postup bude stejný a uvedeme jen odlišnosti.



```

/**
 * priklad04 - stridave blikani zelene a modre LED
 */

/* Includes -----
---*/
#include "stm32F10x.h"
#include "STM32vldiscovery.h"

/* Private typedef -----
---*/
/* Private define -----
---*/
#define LSE_FAIL_FLAG 0x80
#define LSE_PASS_FLAG 0x100
/* Private macro -----
---*/

```

```

/* Private consts -----*/
---*/

/* Private variables -----*/
---*/
u32 LSE_Delay = 0;
u32 count = 0;
u32 BlinkSpeed = 0;
u32 KeyState = 0;
static __IO uint32_t TimingDelay;
/* Private function prototypes -----*/
---*/
void Delay(uint32_t nTime);
void TimingDelay_Decrement(void);

/* Private functions -----*/
---*/

/**
 * @brief Main program.
 * @param None
 * @retval None
 */

int main(void)
{
    /* Enable GPIOx Clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);

    /* Initialise LEDs LD3&LD4, both off */
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);

    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);

    /* Setup SysTick Timer for 1 msec interrupts */
    if (SysTick_Config(SystemCoreClock / 1000))
    {
        /* Capture error */
        while (1);
    }

    /* Enable access to the backup register => LSE can be enabled */
    PWR_BackupAccessCmd(ENABLE);

    /* Enable LSE (Low Speed External Oscillation) */
    RCC_LSEConfig(RCC_LSE_ON);

    /* main while */
    while(1)
    {

```

```

STM32vldiscovery_LEDOff(LED3);
STM32vldiscovery_LEDOn(LED4);
Delay(1000);

STM32vldiscovery_LEDOff(LED4);
STM32vldiscovery_LEDOn(LED3);
Delay(1000);
}
}

/**
 * @brief Inserts a delay time.
 * @param nTime: specifies the delay time length, in milliseconds.
 * @retval None
 */
void Delay(uint32_t nTime)
{
    TimingDelay = nTime;

    while(TimingDelay != 0);
}

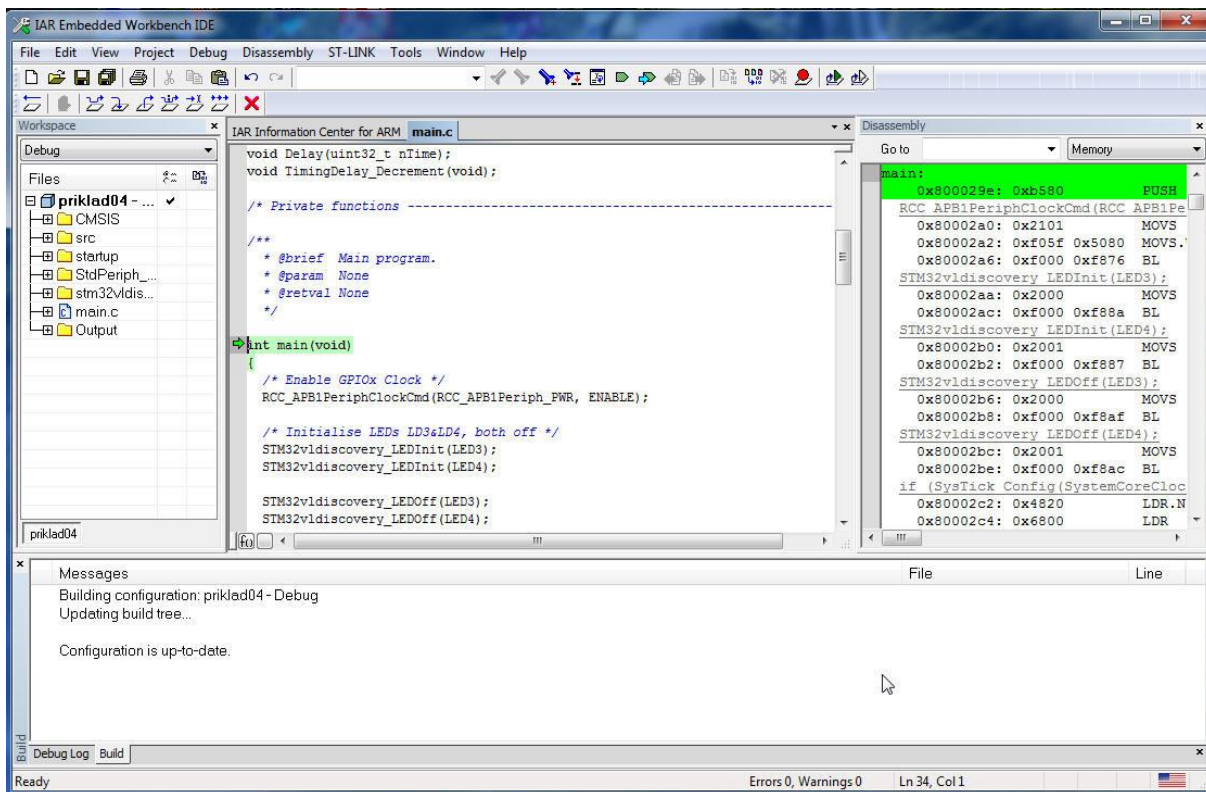
/**
 * @brief Decrements the TimingDelay variable.
 * @param None
 * @retval None
 */
void TimingDelay_Decrement(void)
{
    if (TimingDelay != 0x00)
    {
        TimingDelay--;
    }
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 *         where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t* file, uint32_t line)
{
    /* User can add his own implementation to report the file name and line
    number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
    line) */

    /* Infinite loop */
    while (1)
    {
    }
}
}

```

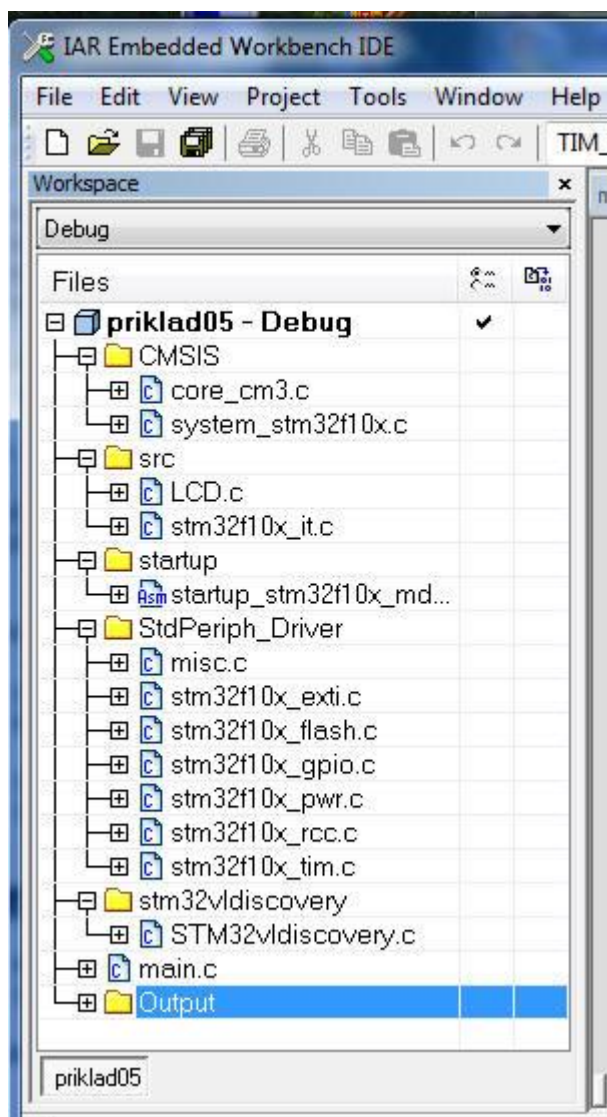
#endif



Odpojíme a znovu připojíme startkit – zelená a modrá LED se opravdu střídavě rozsvěčují a zhasínají.

3.4.5 LCD displej poprvé (příklad05)

Použijeme stejné **LCD.c** a **LCD.h**, jako v programech s uVision4 Keil a stejné adresáře a podadresáře, jako v předchozích programech, takže jen stručně: přidáme **stm32f10x_flash.c** a **stm32f10x_tim.c**



```
/**
 * priklad05 - LCD displej na PB10 az PB15 viz LCD.h
 */

/* Includes -----
---*/
#include "stm32F10x.h"
#include "STM32vldiscovery.h"
#include <stdint.h>
#include "LCD.h"
```

```

/* Private typedef -----*/
---*/
/* Private define -----*/
---*/
#define LSE_FAIL_FLAG 0x80
#define LSE_PASS_FLAG 0x100
/* Private macro -----*/
---*/
/* Private consts -----*/
---*/

/* Private variables -----*/
---*/
u32 LSE_Delay = 0;
u32 count = 0;
u32 BlinkSpeed = 0;
u32 KeyState = 0;
static __IO uint32_t TimingDelay;

uint8_t LCD_Message1[] = "Test displeje.";

/* Private function prototypes -----*/
---*/
void Delay(uint32_t nTime);
void TimingDelay_Decrement(void);

/* Private functions -----*/
---*/

/**
 * @brief Main program.
 * @param None
 * @retval None
 */

int main(void)
{
    /* Enable GPIOx Clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);

    /* Initialise LEDs LD3&LD4, both off */
    STM32vldiscovery_LEDInit(LED3);
    //STM32vldiscovery_LEDInit(LED4);

    STM32vldiscovery_LEDOff(LED3);
    //STM32vldiscovery_LEDOff(LED4);

    /* Setup SysTick Timer for 1 msec interrupts */
    if (SysTick_Config(SystemCoreClock / 1000))
    {
        /* Capture error */
        while (1);
    }
}

```

```

/* Enable access to the backup register => LSE can be enabled */
PWR_BackupAccessCmd(ENABLE);

/* Enable LSE (Low Speed External Oscillation) */
RCC_LSEConfig(RCC_LSE_ON);
LCD_Inicialization();
printLCD(LCD Message1);

/* main while */
while(1)
{
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOn(LED4);
    Delay(1000);

    STM32vldiscovery_LEDOff(LED4);
    STM32vldiscovery_LEDOn(LED3);
    Delay(1000);
}
}

/**
 * @brief Inserts a delay time.
 * @param nTime: specifies the delay time length, in milliseconds.
 * @retval None
 */
void Delay(uint32_t nTime)
{
    TimingDelay = nTime;

    while(TimingDelay != 0);
}

/**
 * @brief Decrements the TimingDelay variable.
 * @param None
 * @retval None
 */
void TimingDelay_Decrement(void)
{
    if (TimingDelay != 0x00)
    {
        TimingDelay--;
    }
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */

```



```
void assert_failed(uint8_t* file, uint32_t line)
{
    /* User can add his own implementation to report the file name and line
    number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
    line) */

    /* Infinite loop */
    while (1)
    {
    }
}
#endif
```

A ještě LCD.c a LCD.h

```
/* STM32 VL Discovery Library
 * LCD driver LCD.c
 * ver 1.0
 * SPSE Jecna 2011 Vladimir Vana
 */

/* Includes -----
---*/
#include "LCD.h"
#include "stm32f10x_tim.h"

/** @defgroup Private_TypesDefinitions */
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructureMili =
{36000, TIM_CounterMode_Up, 0, 0, 0};
/** @defgroup Private_Defines */

/** @defgroup Private_Macros */

/** @defgroup Private_Variables */

/** @defgroup Private_FunctionPrototypes*/

void delay_ms(uint16_t time);
void LCD_Inicialization(void);
void SendCommandLCD(uint8_t cmd);
void SendCharLCD(uint8_t character);
void LCD_Clear(void);
void LCD_CursorHome(void);
void printLCD(uint8_t* message);
void SendBitsLCD(uint8_t bits);
void Send4bitLCD(uint8_t cmd);
void FirstRow(void);
void SecondRow(void);
/** @defgroup Private_Functions */
```

```

/**
 * @brief Nastaveni LCD a pinu.
 * @retval : None
 */
void LCD_Inicialization(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(LCD_PORT_CLK, ENABLE);

    GPIO_InitStructure.GPIO_Pin = LCD_E | LCD_RS | LCD_D4 | LCD_D5 | LCD_D6 |
LCD_D7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(LCD_PORT, &GPIO_InitStructure);

    GPIO_ResetBits(LCD_PORT, LCD_RS);
        //RS=0:Instruction
    GPIO_ResetBits(LCD_PORT, LCD_E);
        //E=0:Disable
    //delay_us(200000);
    delay_ms(200);
    Send4bitLCD(0x30);
    //delay_us(8000); // dle datasheetu cekat vice nez 4.1 ms
    delay_ms(8);
    GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
    Send4bitLCD(0x30);
    //delay_us(1000);
    delay_ms(1);
    GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
    Send4bitLCD(0x30);
    //delay_us(1000);
    delay_ms(1);
    GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
    // delay_us(1000);
    delay_ms(1);
    // ted by mel byt display v 8-bitovem modu a definitivne ho prepneme do
4-bitoveho modu!
    Send4bitLCD(0x20);
    //delay_us(1000);
    delay_ms(1);
    GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
    //delay_us(1000);
    delay_ms(1);
    // nyni jsme ve 4-bitovem rezimu
    // Prikaz 0 0 1 DL N F - -
    // N = pocet radek 1=2 radky, 0=1 radka
    // DL = bitu komunikace 1=8bitu, 0=4bity
    // F = Font 1=5x10bodu, 0=5x8 bodu
    SendCommandLCD(0x28);
    //delay_us(1000);
    delay_ms(1);
    // zvedej sam adresu, posunuj kurzor
    SendCommandLCD(0x0E); //nyni OK
    //delay_us(1000);

```

```

delay_ms(1);
//PosliPrikazLCD(0b00000110);
LCD_Clear(); // vymazani LCD a navrat kurzoru na prvni pozici prvnioho
radku
}

/**
 * @brief Smaze LCD a nastavi LCD kurzor na home pozici
 * @retval : None
 */
void LCD_Clear(void)
{
    SendCommandLCD(0x01);
}

/**
 * @brief Nastavi LCD kurzor na home pozici
 * @retval : None
 */
void LCD_CursorHome(void)
{
    SendCommandLCD(0x02);
}

/**
 * @brief Posle prikaz do LCD
 * @param cmd prikaz do LCD
 * @retval : None
 */
void SendCommandLCD(uint8_t cmd) //v promenne cmd mame zaslaný prikaz
{
    GPIO_ResetBits(LCD_PORT, LCD_RS);
    //RS=0:Instruction
    SendBitsLCD(cmd); // posle bity jako instrukci
}

/**
 * @brief Posle znak do LCD
 * @param character znak
 * @retval : None
 */
void SendCharLCD(uint8_t character) //v promenne cmd je znak poslaný na LCD
{
    GPIO_SetBits(LCD_PORT, LCD_RS); //RS=1:Data

    SendBitsLCD(character); // posle bity jako data
}

/**
 * @brief Posle bity do LCD
 * @param bits znak nebo prikaz
 * @retval : None
 */
void SendBitsLCD(uint8_t bits)

```

```

{
    Send4bitLCD(bits); // zaslani bitu 4-7
    bits = bits<<4; // bitovy posun na zaslani bitu 0 az 3
    Send4bitLCD(bits); // zaslani bitu 0-3
}

/**
 * @brief Posle retezec znaku do LCD
 * @param message pointer na retezec
 * @retval : None
 */
void printLCD(uint8_t* message)
{
    uint8_t i=0;
    if (!message) return;
    while(message[i]!='\0')
    {
        if(i%20==0){
            if(i%40==0){ // pokud je na konci druheho radku prejde na
zacatek prvniho
                FirstRow();
                delay_ms(1);
            }else{ // pokud je na konci prvniho radku
prejde na zacatek druheho
                SecondRow();
            }
        }
        SendCharLCD(message[i]);
        i++;
    }
}

/**
 * @brief Prejde na prvni radek
 * @retval : None
 */
void FirstRow(void)
{
    SendCommandLCD(0x80); // prejde na zacatek prvniho radku - na adresu
prvniho znaku
    delay_ms(1);
}

/**
 * @brief Prejde na druhy radek
 * @retval : None
 */
void SecondRow(void)
{
    SendCommandLCD(0xC0); // prejde na zacatek druheho radku - na
adresu prvniho znaku
    delay_ms(1);
}

/**
 * @brief Posle po 4 bitech prikaz nebo znak do LCD

```

```

* @param command
* @retval : None
*/
void Send4bitLCD(uint8_t cmd)
{
    GPIO_ResetBits(LCD_PORT, LCD_E);
    delay_ms(1);
    if (cmd & 0x10) // vymaskovani 4 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D4);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D4);
    }
    if (cmd & 0x20) // vymaskovani 5 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D5);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D5);
    }
    if (cmd & 0x40) // vymaskovani 6 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D6);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D6);
    }
    if (cmd & 0x80) // vymaskovani 7 bitu
    {
        GPIO_SetBits(LCD_PORT, LCD_D7);
    } else {
        GPIO_ResetBits(LCD_PORT, LCD_D7);
    }
    GPIO_SetBits(LCD_PORT, LCD_E);
    //E=1:Enable
    delay_ms(1);
    GPIO_ResetBits(LCD_PORT, LCD_E); //E=0:Disable
}

/**
* @brief Cekat n milisekund
* @param time kolik ms se ma cekat
* @retval None
*/
void delay_ms(uint16_t time)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
    TIM_TimeBaseStructureMili.TIM_Period = ((time+1) * 2)-1;
    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructureMili);
    TIM_SelectOnePulseMode(TIM4, TIM_OPMode_Single);
    TIM_SetCounter(TIM4, 2);
    TIM_Cmd(TIM4, ENABLE);
    while (TIM_GetCounter(TIM4)){};
    TIM_Cmd(TIM4, DISABLE);
}

```

LCD.h

```

/* STM32 VL Discovery Library
 * LCD driver LCD.c
 * ver 1.0
 * Vd for SPSE Jecna 2011 lessons
 */

#ifndef LCD_H_
#define LCD_H_

/* Includes -----
---*/
#include "stm32f10x.h"

/** @defgroup Exported_Types */
/** @defgroup Exported_Functions */

void delay_ms(uint16_t time);
void LCD_Inicialization(void);
void SendCommandLCD(uint8_t cmd);
void SendCharLCD(uint8_t cmd);
void LCD_Clear(void);
void LCD_CursorHome(void);
void printLCD(uint8_t* message);
void Send4bitLCD(uint8_t cmd);
void FirstRow(void);
void SecondRow(void);

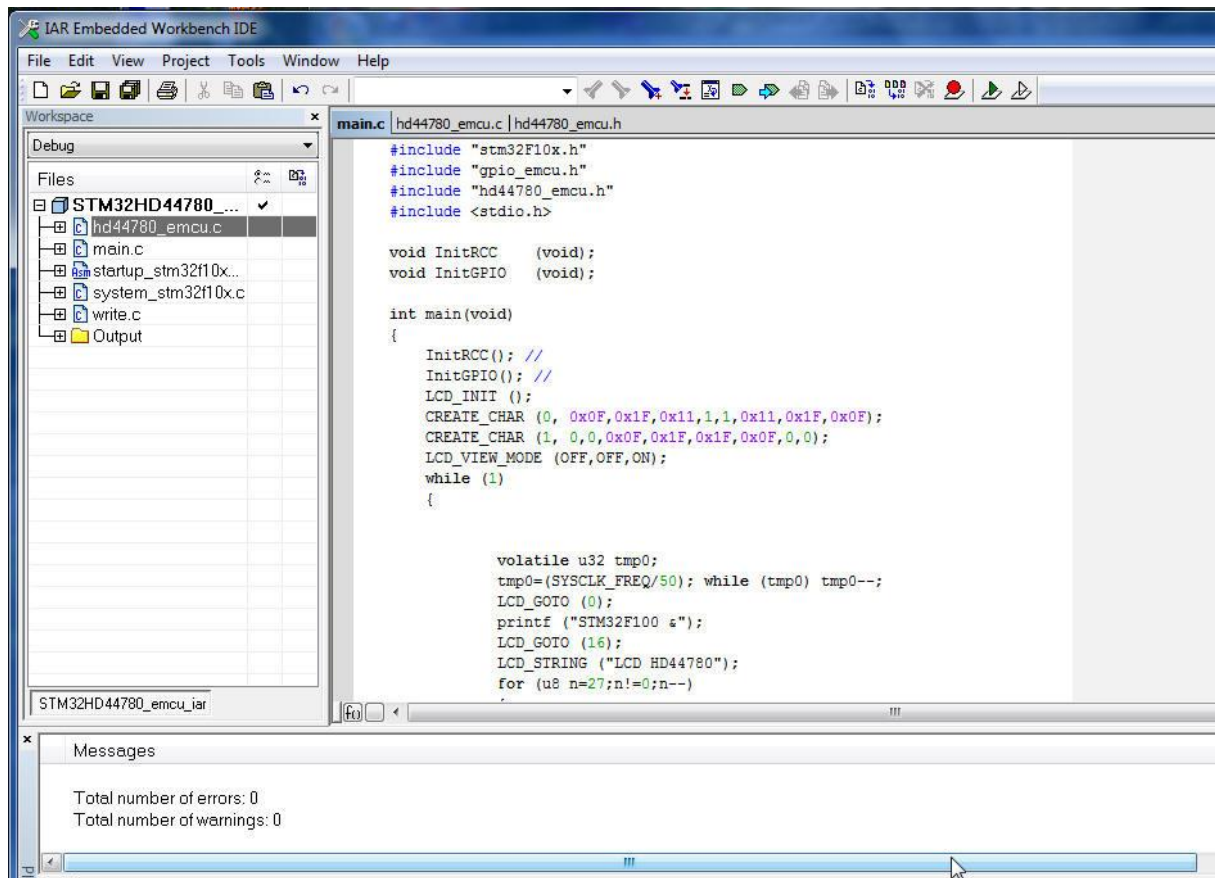
/** @defgroup Defines */
// piny pro ovladani displeje - zapojeni stejne jako na Katedre mereni CVUT
#define LCD_E GPIO_Pin_14
#define LCD_RS GPIO_Pin_15
#define LCD_D4 GPIO_Pin_10
#define LCD_D5 GPIO_Pin_11
#define LCD_D6 GPIO_Pin_12
#define LCD_D7 GPIO_Pin_13
#define LCD_PORT GPIOB
#define LCD_PORT_CLK RCC_APB2Periph_GPIOB

#endif /* LCD_H_ */

```

3.4.6 LCD displej podruhé (příklad06)

Trochu jiný přístup najdete v projektu na ruském serveru <http://eugenemcu.ru/publ/13-1-0-75>



The screenshot shows the IAR Embedded Workbench IDE interface. The main window displays the source code for a C program named 'main.c' located in the 'hd44780_emcu.c' directory. The code includes headers for the STM32F10x, GPIO, and the specific LCD driver, and defines initialization functions and a main loop that configures the LCD and prints a message.

```

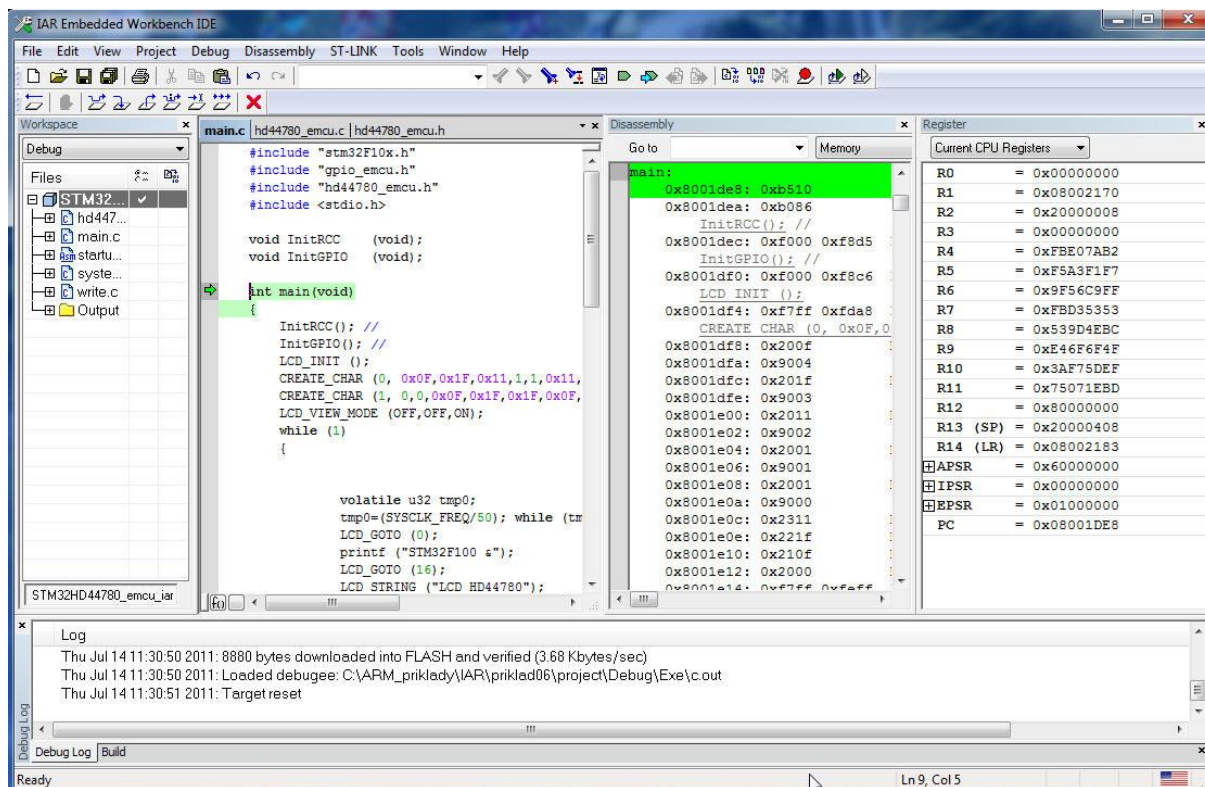
main.c | hd44780_emcu.c | hd44780_emcu.h
#include "stm32F10x.h"
#include "gpio_emcu.h"
#include "hd44780_emcu.h"
#include <stdio.h>

void InitRCC (void);
void InitGPIO (void);

int main(void)
{
    InitRCC(); //
    InitGPIO(); //
    LCD_INIT ();
    CREATE_CHAR (0, 0x0F, 0x1F, 0x11, 1, 1, 0x11, 0x1F, 0x0F);
    CREATE_CHAR (1, 0, 0, 0x0F, 0x1F, 0x1F, 0x0F, 0, 0);
    LCD_VIEW_MODE (OFF, OFF, ON);
    while (1)
    {

        volatile u32 tmp0;
        tmp0=(SYSCLK_FREQ/50); while (tmp0) tmp0--;
        LCD_GOTO (0);
        printf ("STM32F100 s");
        LCD_GOTO (16);
        LCD_STRING ("LCD HD44780");
        for (u8 n=27;n!=0;n--)
    }
}
    
```

The Messages window at the bottom shows: Total number of errors: 0, Total number of warnings: 0.



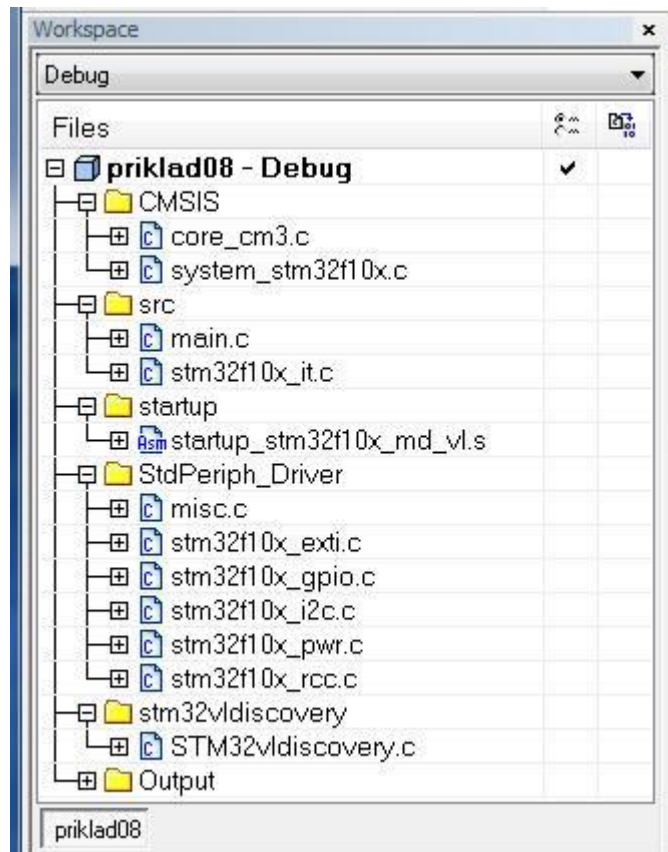
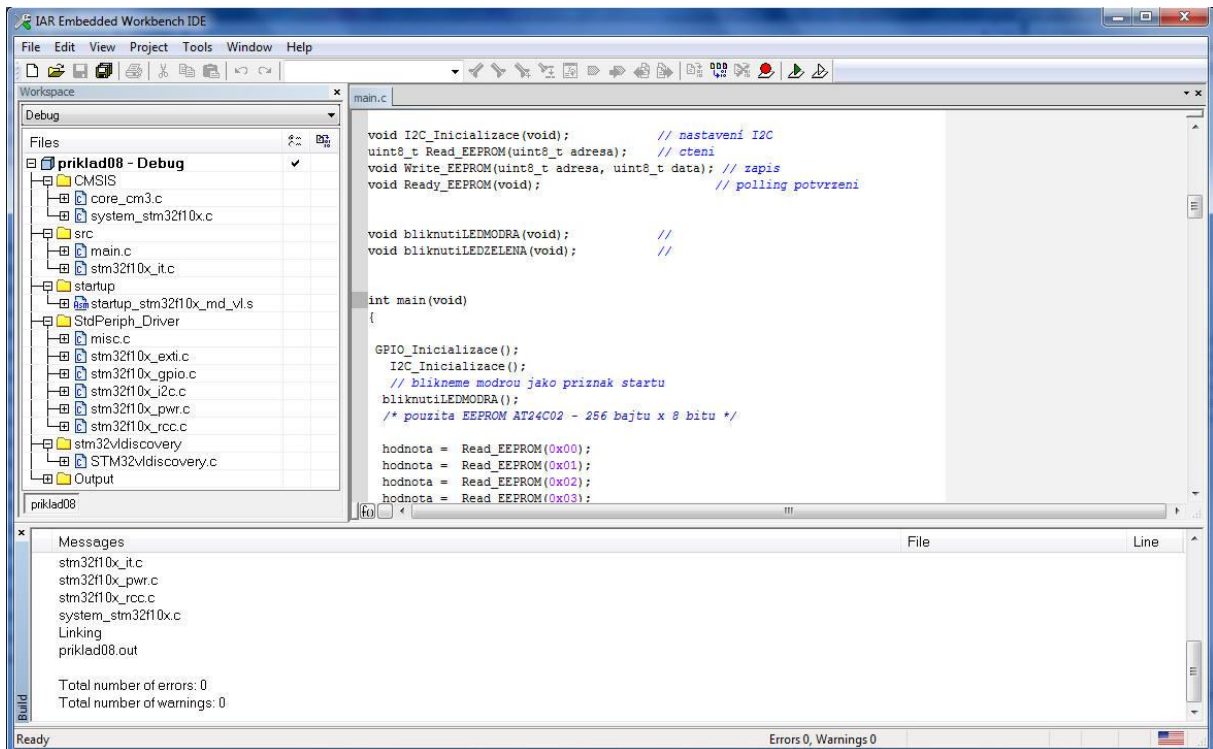
LCD zapojíme

| | | |
|-----|------|------|
| E | (6) | PC12 |
| R/W | (5) | PC11 |
| RS | (4) | PC10 |
| DB4 | (11) | PA8 |
| DB5 | (12) | PA9 |
| DB6 | (13) | PA10 |
| DB7 | (14) | PA11 |

3.4.7 LCD displej a klávesnice PS2 (příklad07)

převzato z ruského serveru <http://eugenemcu.ru/publ/13-1-0-75>

3.4.8 I2C komunikace s EEPROM AT24C022 (příklad08)



```

/*
 * I2C komunikace s EEPROM (AT24C02)
 * dle 29. dilu serialu o STM32
 * priklad 08          main.c
 * Vd pro SPSE Jecna Lectures
 */
/* Includes -----*/

#include "stm32F10x.h"
#include "stm32f10x_i2c.h"
#include "STM32vldiscovery.h"

/* Private typedef -----*/
GPIO_InitTypeDef  GPIO_InitStructure;

#define EEPROM_SLAVE_ADDRESS 0xA0    //protoze je to EEPROM
//SCL bude na PB6   DATA na PB7   AT24C02

//Tabulka I2C slave adres          (A0=0 je write do obvodu, A1=1 je read )
//Typ obvodu          A7      A6      A5      A4      A3      A2      A1
//8bitovy vstup/vystup  0      1      0      0      a      a      a      napr.
PCF8574
//EEPROM          1      0      1      0      a      a      a      napr.
PCF8582
//Hodiny/kalendar 1      0      1      0      0      0      a      napr. PCF8583
//8bitovy A/D a D/A  1      0      0      1      a      a      a      napr.
PCF8591
//PLL          1      1      0      0      0      0      a      a      napr.
TSA5055

/* Private variables -----*/
uint8_t hodnota=0;
static __IO uint32_t TimingDelay;
/* Private function prototypes -----*/
void Delay(__IO uint32_t nTick);
void TimingDelay_Decrement(void);
void GPIO_Inicializace(void); // nastavení vstupne/vystupnich pinu na kitu
void I2C_Inicializace(void); // nastavení I2C
uint8_t Read_EEPROM(uint8_t adresa); // cteni
void Write_EEPROM(uint8_t adresa, uint8_t data); // zapis
void Ready_EEPROM(void); // polling potvrzeni
void bliknutiLEDMODRA(void); //
void bliknutiLEDZELENA(void); //

int main(void)
{
    GPIO_Inicializace();
    I2C_Inicializace();
    // blikneme modrou jako priznak startu

```

```

bliknutiLEDMODRA();
/* pouzita EEPROM AT24C02 - 256 bajtu x 8 bitu */

hodnota = Read_EEPROM(0x00);
hodnota = Read_EEPROM(0x01);
hodnota = Read_EEPROM(0x02);
hodnota = Read_EEPROM(0x03);
hodnota = Read_EEPROM(0x04);

Write_EEPROM(0x00, 0xA1);
Ready_EEPROM();
bliknutiLEDZELENA();
Write_EEPROM(0x01, 0xB2);
Ready_EEPROM();
bliknutiLEDMODRA();
Write_EEPROM(0x02, 0xC3);
Ready_EEPROM();
bliknutiLEDZELENA();
Write_EEPROM(0x03, 0xD4);
Ready_EEPROM();
bliknutiLEDMODRA();

hodnota = Read_EEPROM(0x00);
hodnota = Read_EEPROM(0x01);
hodnota = Read_EEPROM(0x02);
hodnota = Read_EEPROM(0x03);
hodnota = Read_EEPROM(0x04);

// blikneme zelenou jako priznak konce
bliknutiLEDMODRA();

/* main while */
while(1)
{
    bliknutiLEDZELENA();
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void TimingDelay_Decrement(void)
{
    if (TimingDelay != 0x00)
    {
        TimingDelay--;
    }
}

void GPIO_Inicializace(void)
{
    STM32vldiscovery_LEDInit(LED3);
}

```

```

STM32vldiscovery_LEDInit(LED4);
STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
STM32vldiscovery_LEDOff(LED3);
STM32vldiscovery_LEDOff(LED4);
}

void bliknutiLEDMODRA(void)
{
    STM32vldiscovery_LEDOn(LED4); // zapnem LED4 - modra
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED4); // vypnem LED4 - modra
    Delay(0x5FFFF);
}

void bliknutiLEDZELENA(void)
{
    STM32vldiscovery_LEDOn(LED3); // zapnem LED3 - zelena
    Delay(0x5FFFF);
    STM32vldiscovery_LEDToggle(LED3); // vypnem LED3 - zelena
    Delay(0x5FFFF);
}

void I2C_Inicializace(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    I2C_InitTypeDef  I2C_InitStructure;
    I2C_DeInit(I2C1);

    /* I2C Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* GPIO Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    /* Configure I2C pins: SCL and SDA */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD; //GPIO_Mode_AF_PP
    GPIO_Init(GPIOB, &GPIO_InitStructure); //I2C EEPROM pripojime k PB
    //SCL bude na PB6  DATA na PB7  AT24C02

    /* I2C configuration */
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = 0xA0; //EEPROM
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = 10000;

    /* I2C Peripheral Enable */
    I2C_Cmd(I2C1, ENABLE);
    /* Apply I2C configuration after enabling it */
    I2C_Init(I2C1, &I2C_InitStructure);
}

```

```

uint8_t Read_EEPROM(uint8_t adresa)
{
    uint8_t Data;
    Data = 0;
    /* While the bus is busy */
    while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

    /* Send START condition */
    I2C_GenerateSTART(I2C1, ENABLE);

    /* Test on EV5 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

    /* Send EEPROM address for read */
    I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);

    /* Test on EV6 and clear it */
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    /* Clear EV6 by setting again the PE bit */
    I2C_Cmd(I2C1, ENABLE);

    /* Send the EEPROM's internal address to read from: MSB of the
address first */
    I2C_SendData(I2C1, adresa);

    /* Test on EV8 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    /* Send START condition a second time */
    I2C_GenerateSTART(I2C1, ENABLE);

    /* Test on EV5 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

    /* Send EEPROM address for read */
    I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Receiver);

    /* Test on EV6 and clear it */
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

    /* Disable Acknowledgement */
    I2C_AcknowledgeConfig(I2C1, DISABLE);

    /* Send STOP Condition */
    I2C_GenerateSTOP(I2C1, ENABLE);
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)) {}

    /* Read a byte from the EEPROM */
    Data = I2C_ReceiveData(I2C1);
}

```

```

        /* Enable Acknowledgement to be ready for another reception */
        I2C_AcknowledgeConfig(I2C1, ENABLE);

        return(Data);
    }

void Write EEPROM(uint8_t adresa, uint8_t data)
{
    /* Send START condition */
    I2C_GenerateSTART(I2C1, ENABLE);

    /* Test on EV5 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

    /* Send EEPROM address for write */
    I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);

    /* Test on EV6 and clear it */
    while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    /* Send the EEPROM's internal address to write to : MSB of the
address first */
    I2C_SendData(I2C1, adresa);

    /* Test on EV8 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    /* Send the byte to be written */
    I2C_SendData(I2C1, data);

    /* Test on EV8 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    /* Send STOP condition */
    I2C_GenerateSTOP(I2C1, ENABLE);
}

void Ready_EEPROM(void)
{
    __IO uint16_t temp = 0;

    do
    {
        /* Send START condition */
        I2C_GenerateSTART(I2C1, ENABLE);

        /* Read I2C_EE SR1 register to clear pending flags */
        temp = I2C_ReadRegister(I2C1, I2C_Register_SR1);

        /* Send EEPROM address */
        I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);
    }
}

```

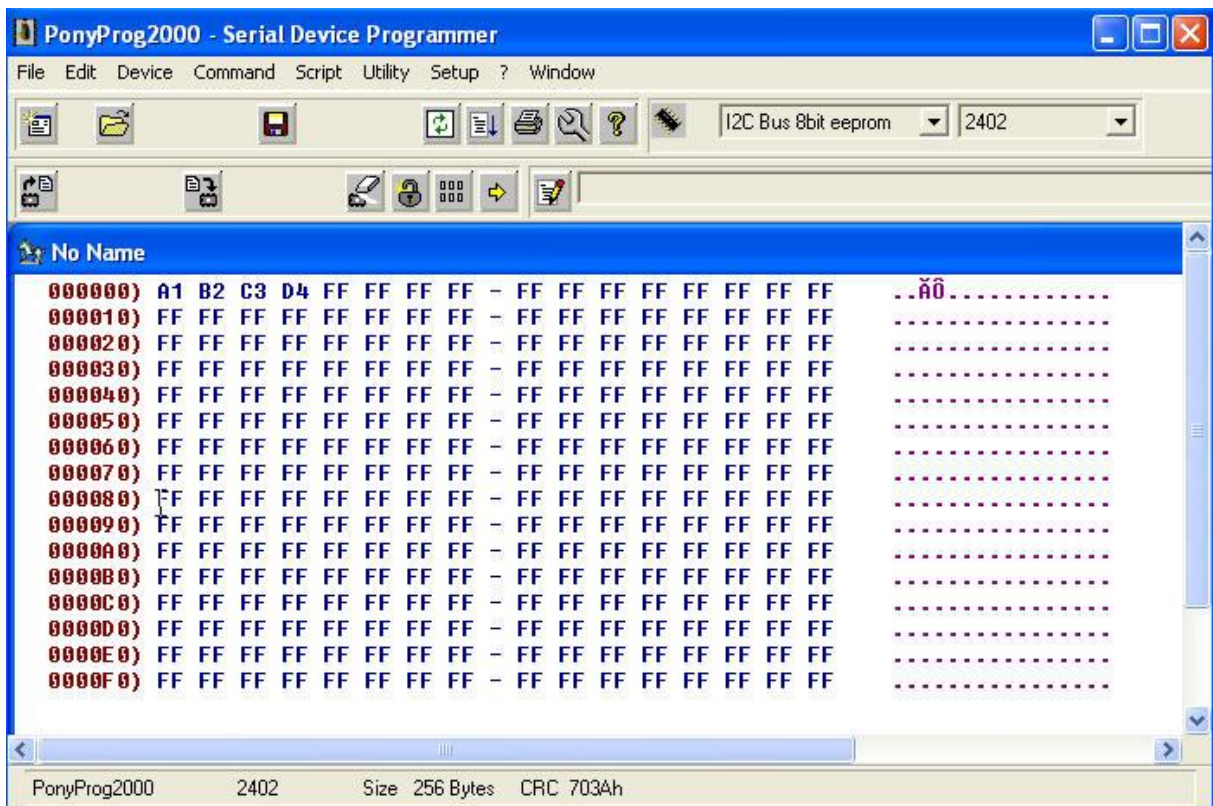
```

}while(!(I2C_ReadRegister(I2C1, I2C_Register_SR1) & 0x0002));

/* Clear AF flag */
I2C_ClearFlag(I2C1, I2C_FLAG_AF);

/* STOP condition */
I2C_GenerateSTOP(I2C1, ENABLE);
}

```

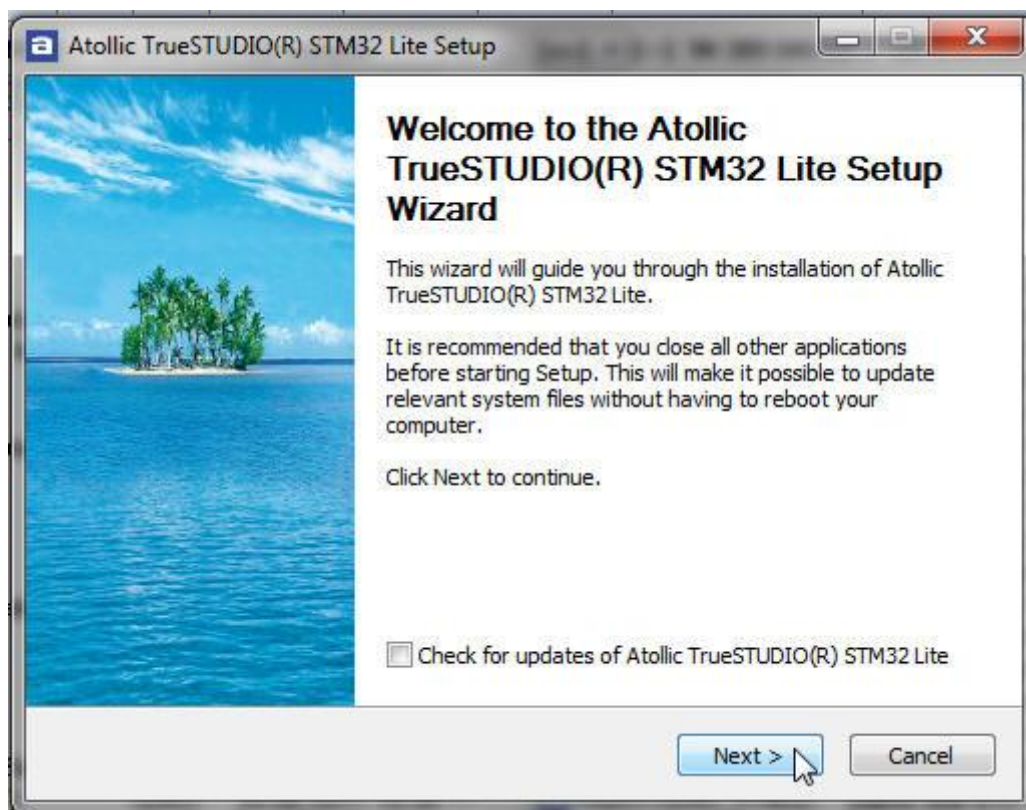


4 Práce v prostředí Atollic TrueStudio STM32 Lite

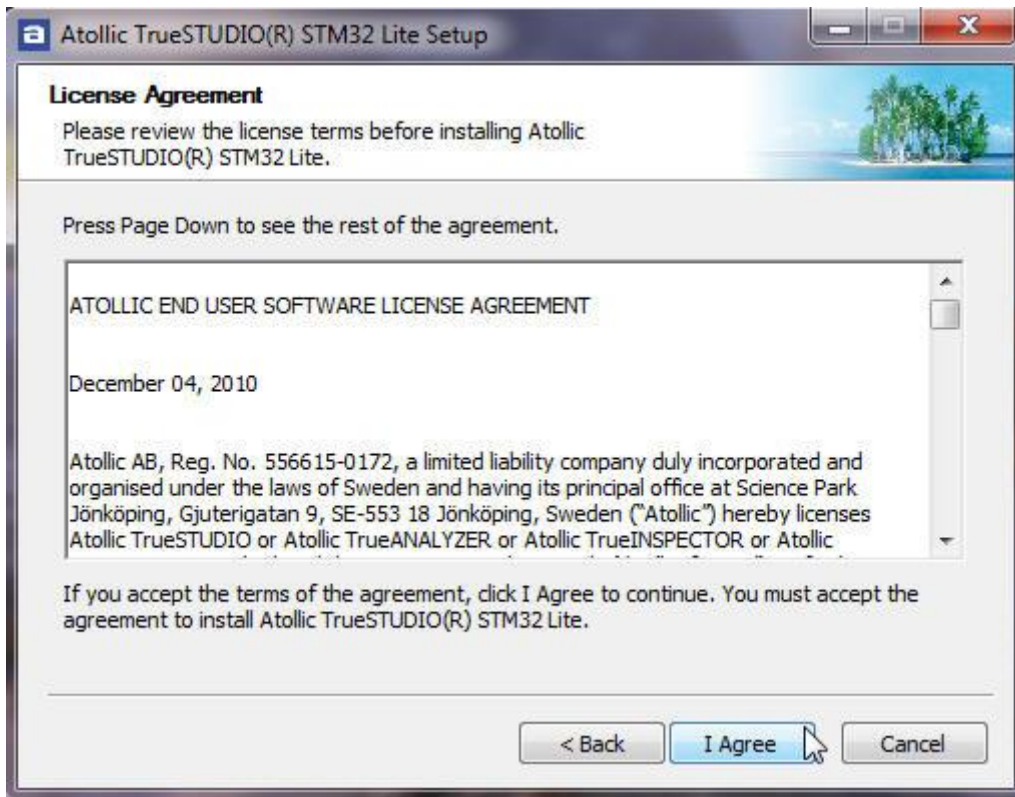
Atollic TrueSTUDIO®/STM32 Lite v2.1.0 je vývojové prostředí založené na Eclipse. Lze ho zdarma stáhnout z <http://www.atollic.com/index.php/targets/stm32> . Lite verze je free, není omezená časově ani délkou kódu, ale neobsahuje některé funkčnosti TrueSTUDIO Profesional.

4.1 Instalace Atollic TrueStudio STM32 Lite

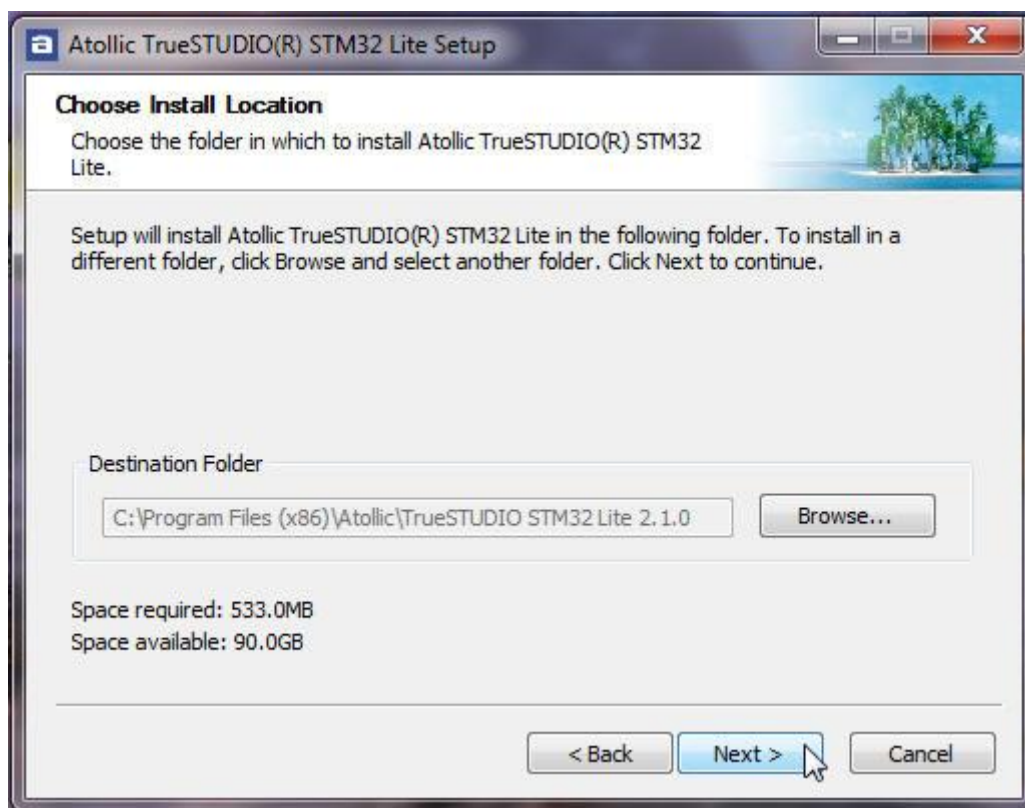
Spustíme instalační soubor, dostaneme



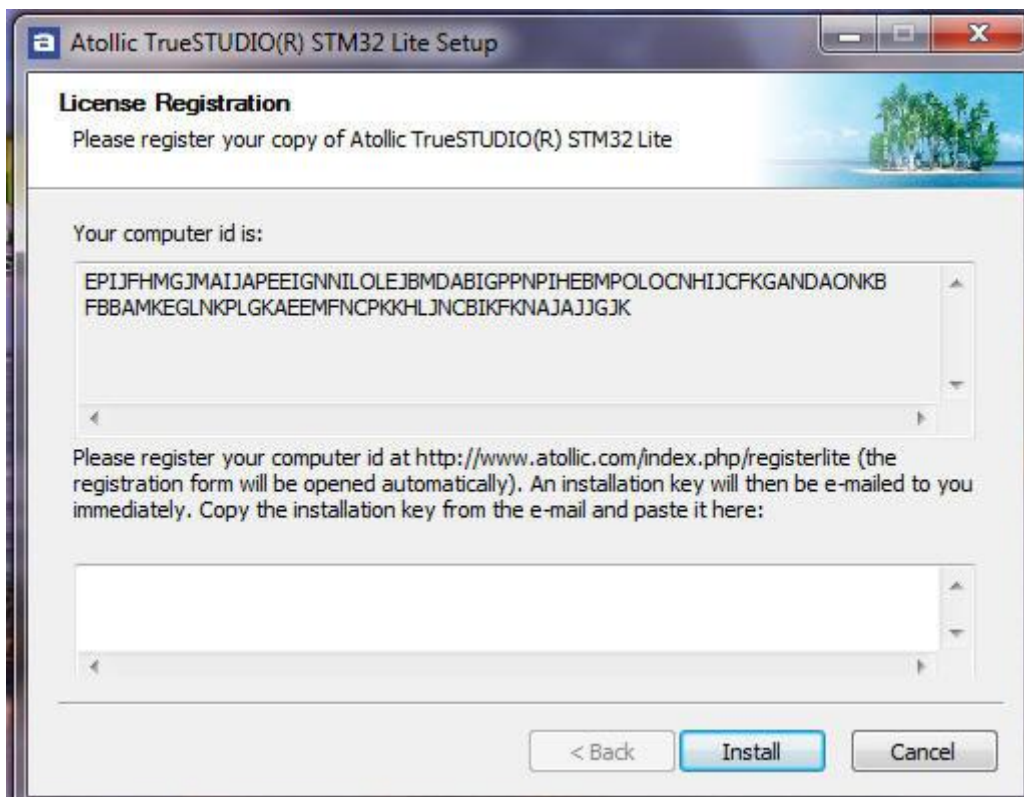
Klineme na **Next**



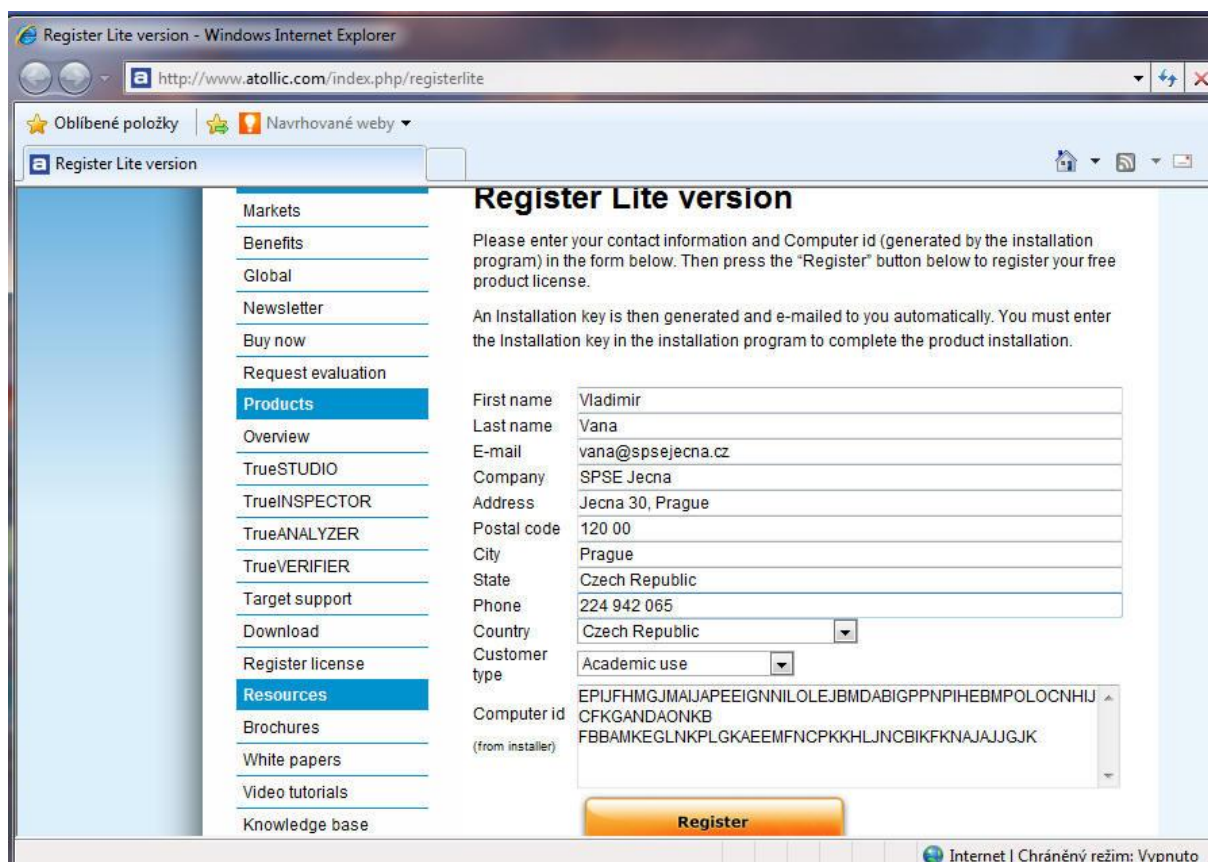
Odsouhlasíme licenční podmínky **I Agree**



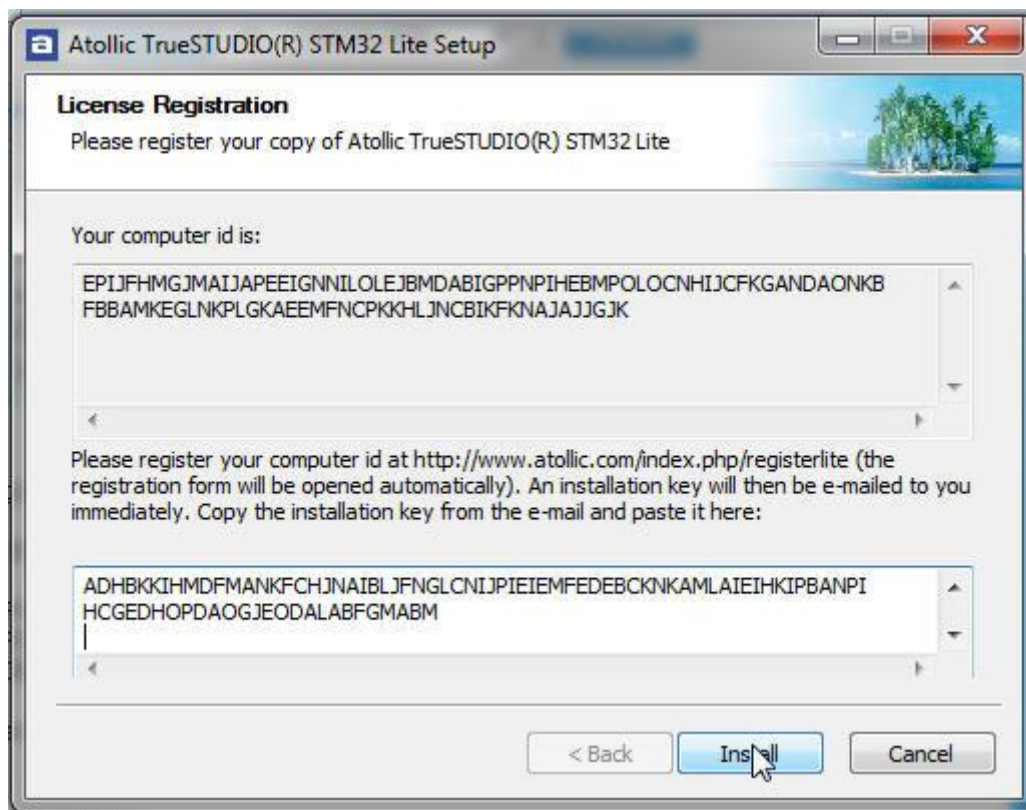
Ponecháme nabízené nastavení místa instalace a klikneme na **Next**. Dostaneme



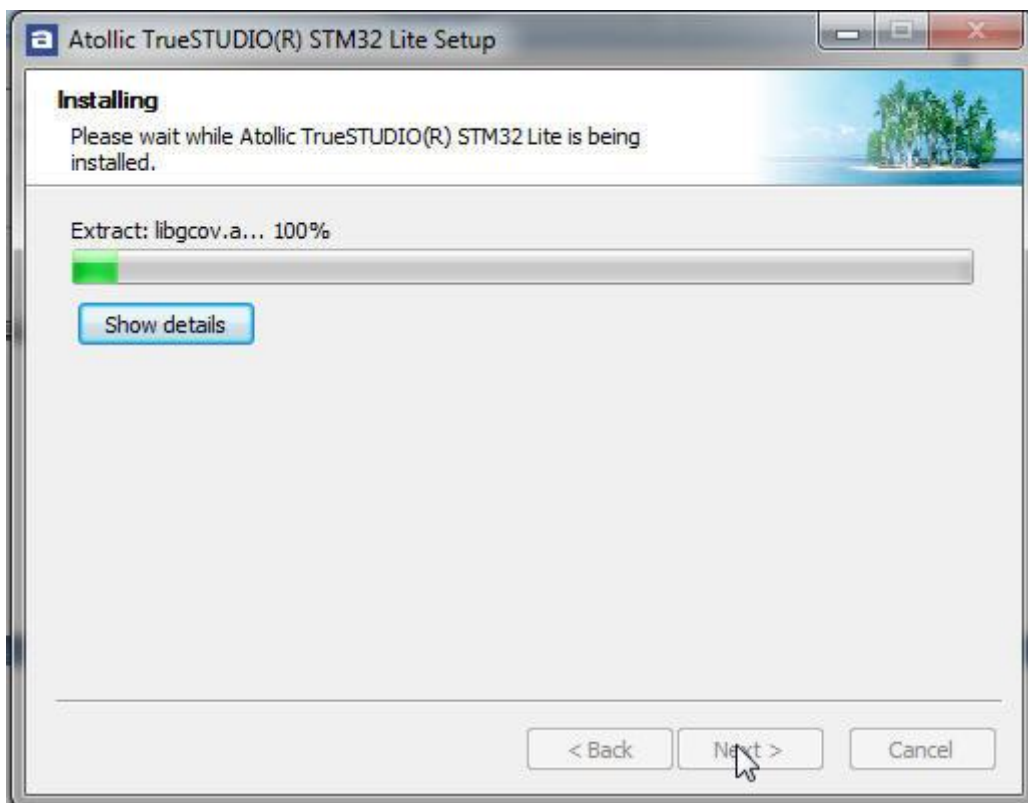
Obsah horního okna přeneseme do registračního formuláře na stránkách www.atollic.com



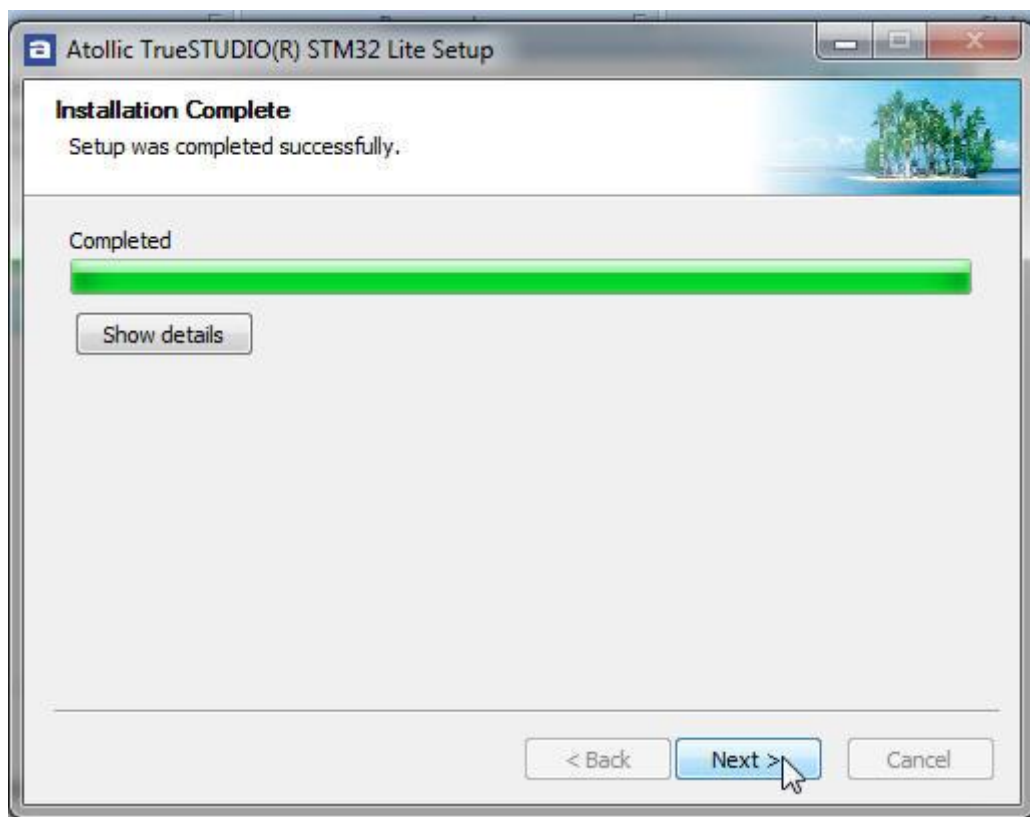
Po vyplnění všech údajů klikneme na register. E- mailem dostaneme řetězec, který vložíme do dolního textboxu v



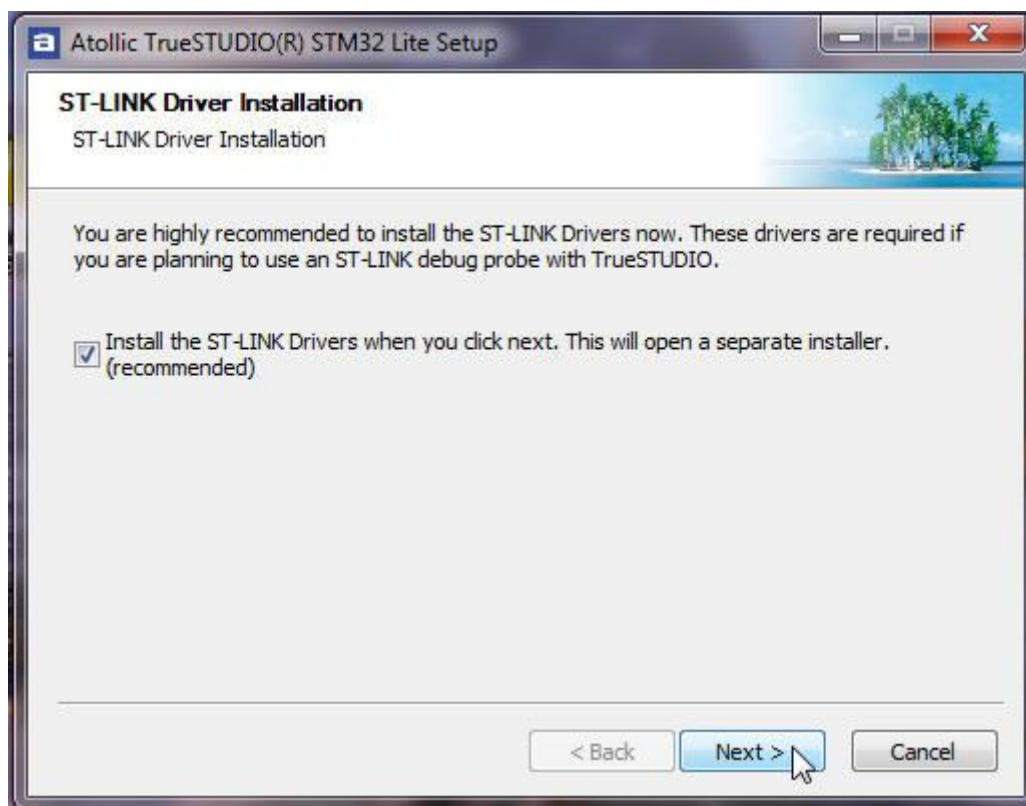
Nyní již můžeme spustit instalaci tlačítkem **Install**



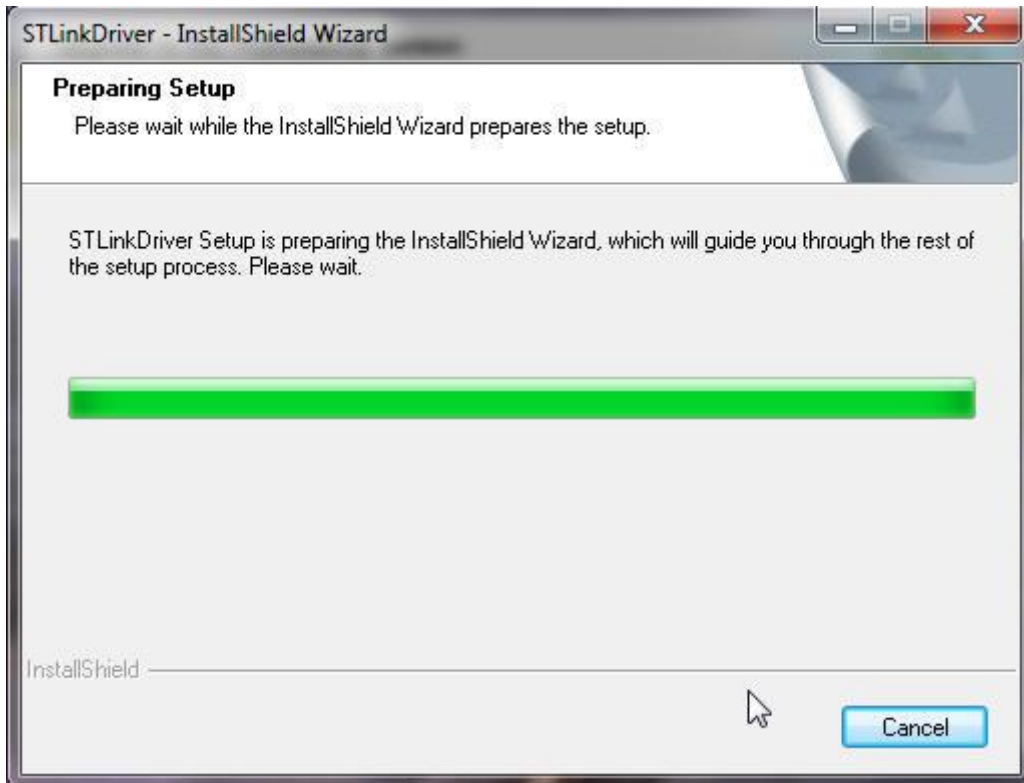
Instalace probíhá



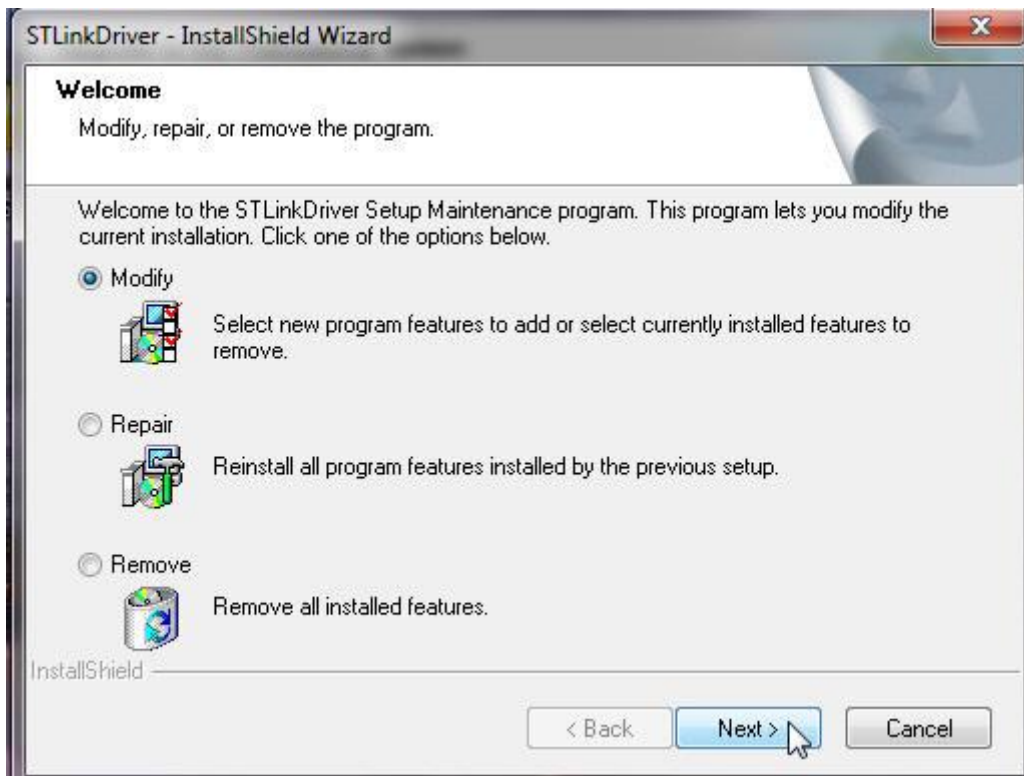
Klikneme na **Next**



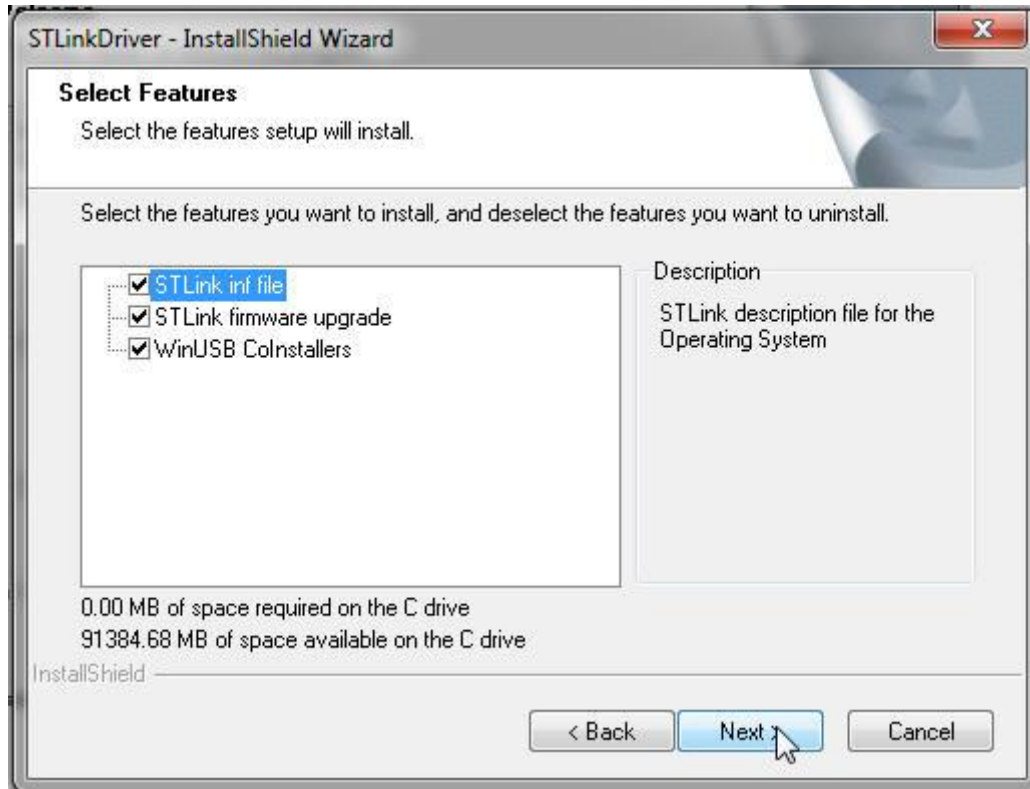
Zaškrtneme požadavek na instalaci ST-Link driverů a klikneme na **Next**



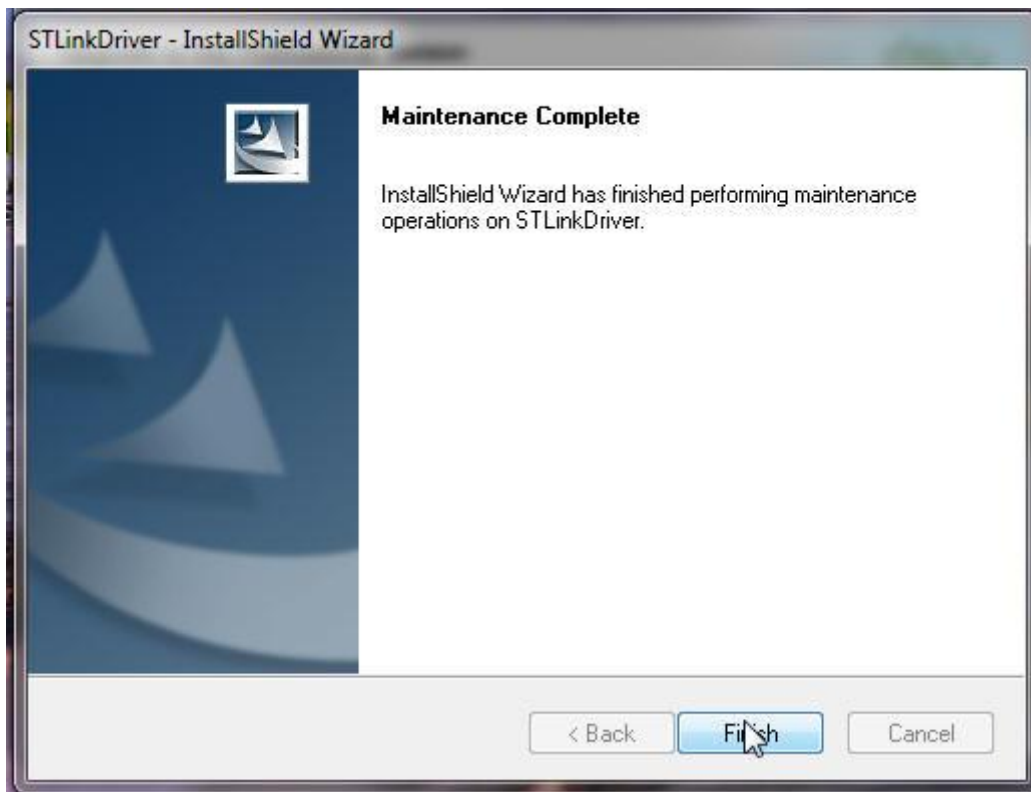
Nainstalují se drivery a poté se objeví



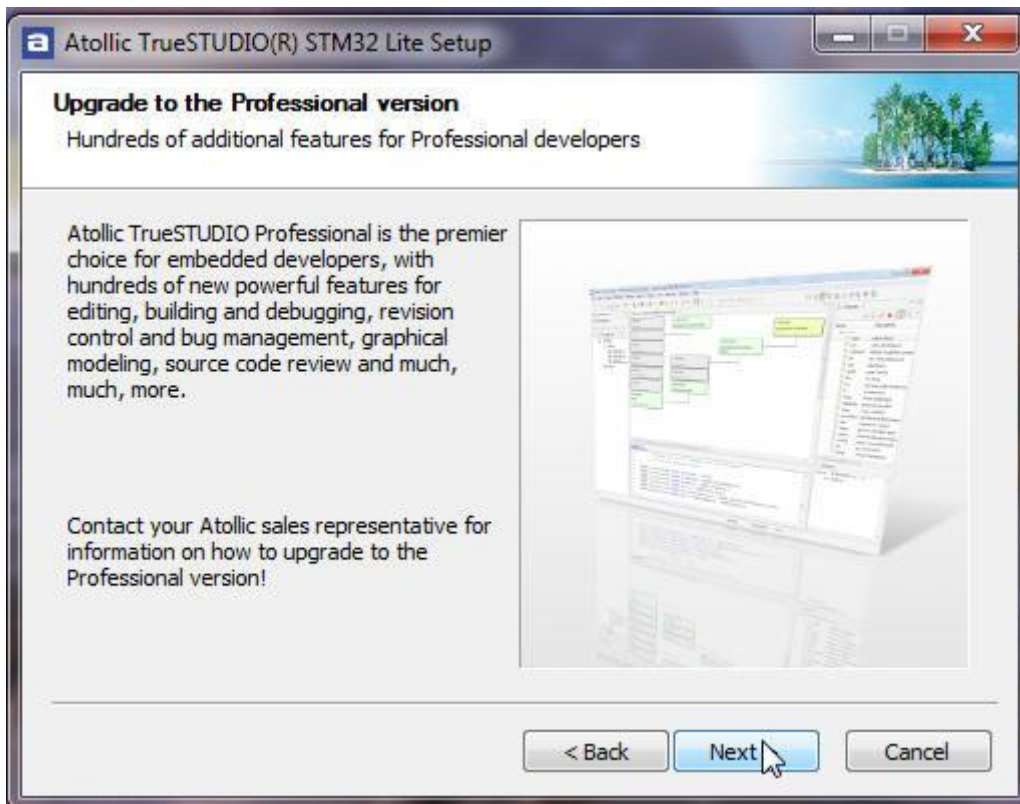
Klikneme na **Next**



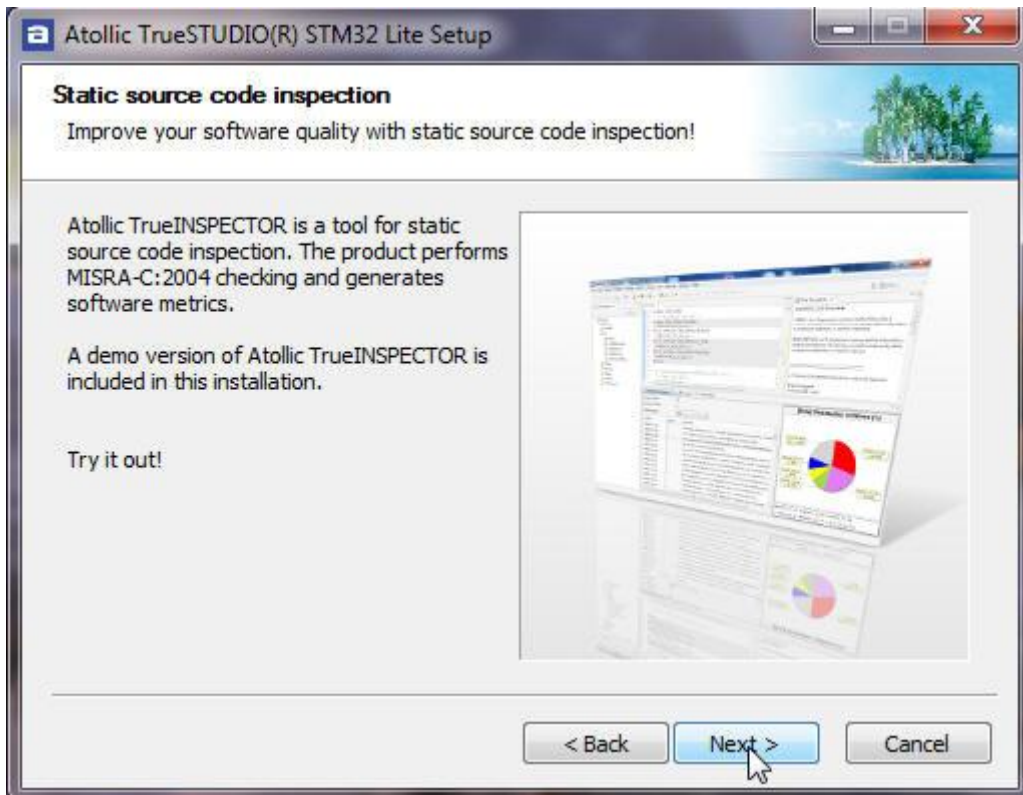
Opět **Next**



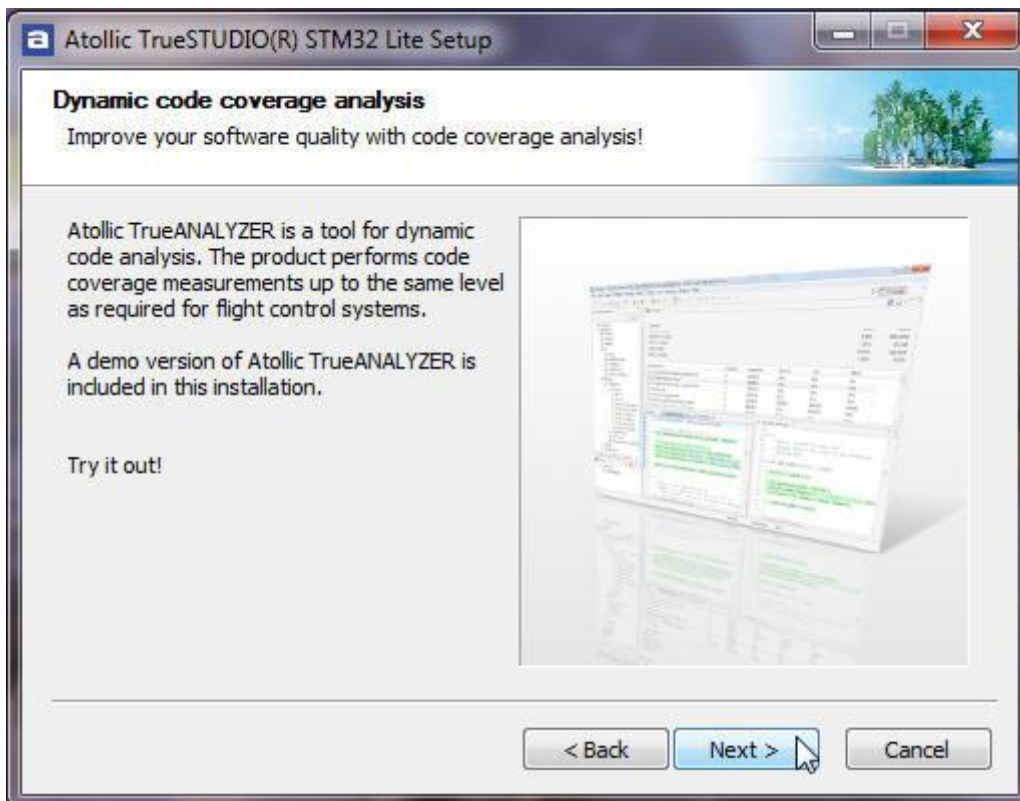
A nakonec tlačítko **Finish**



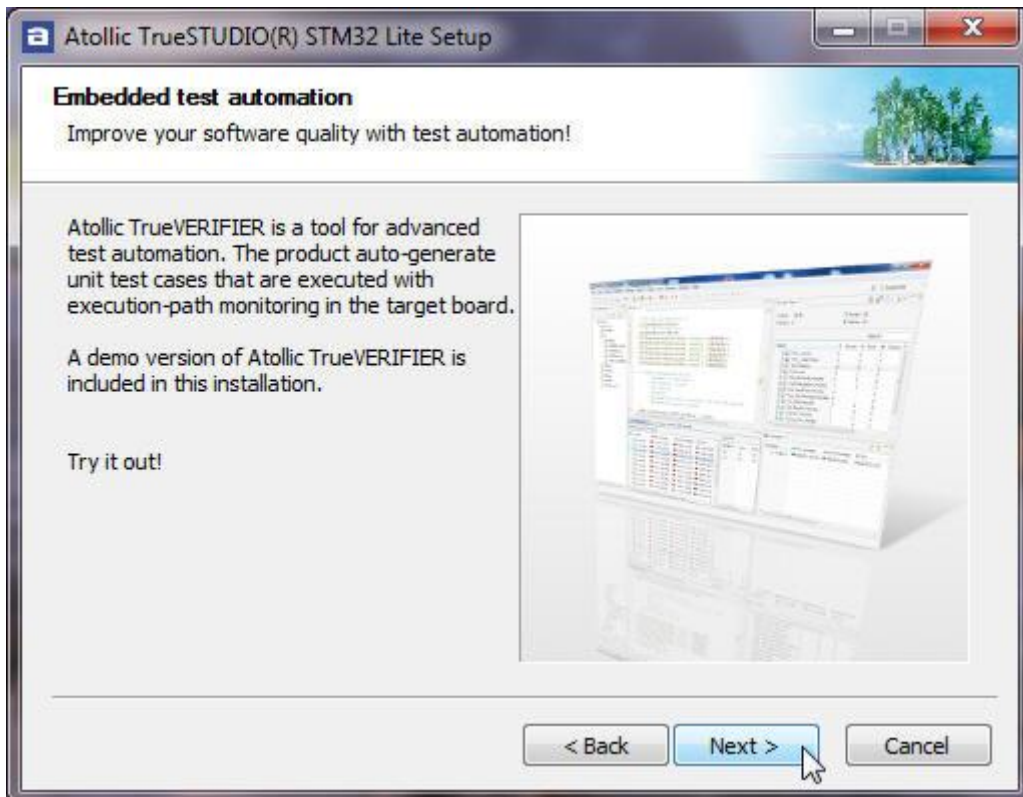
Opět budeme klikat na **Next**



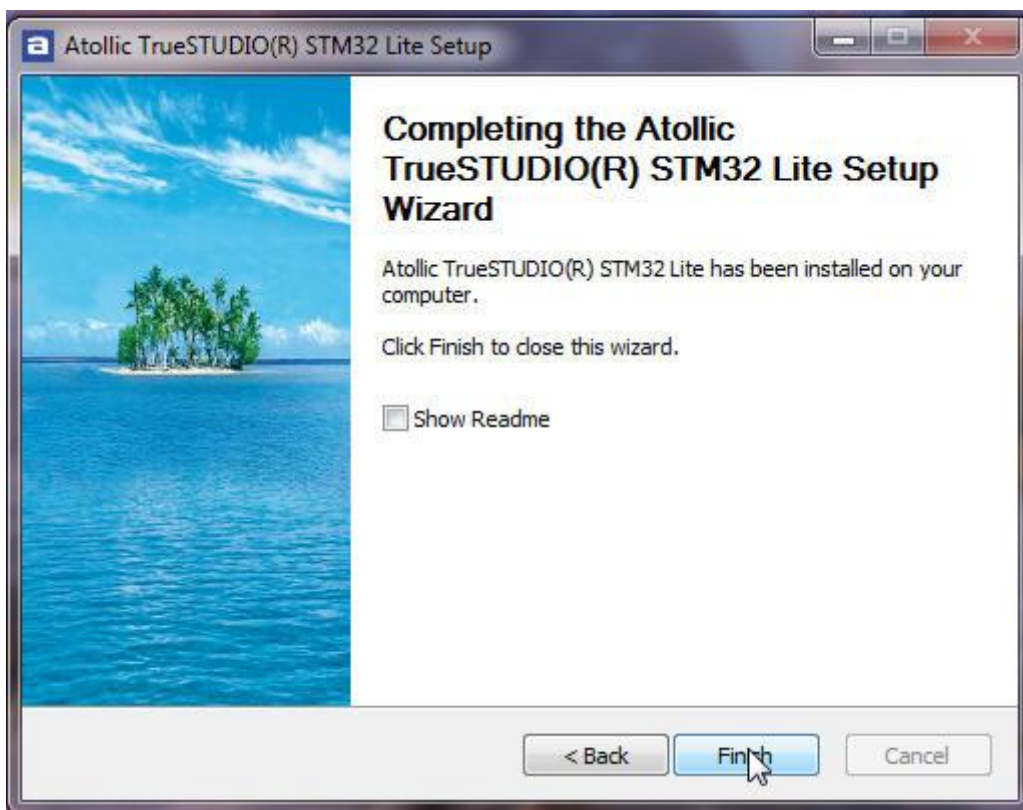
Znovu **Next**



Další **Next**

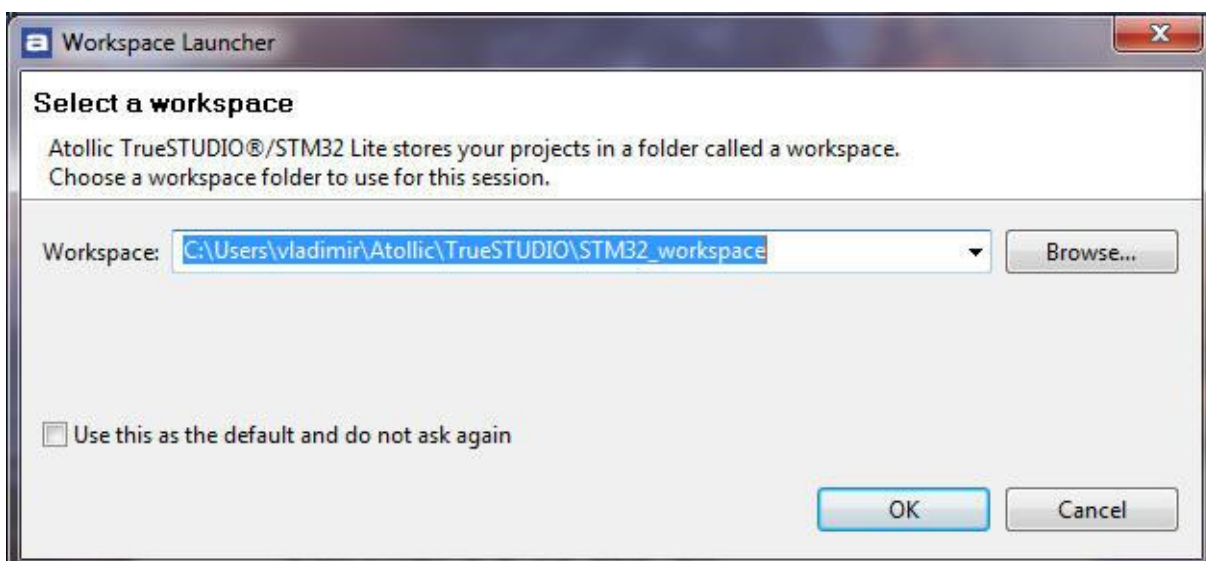


Opět Next

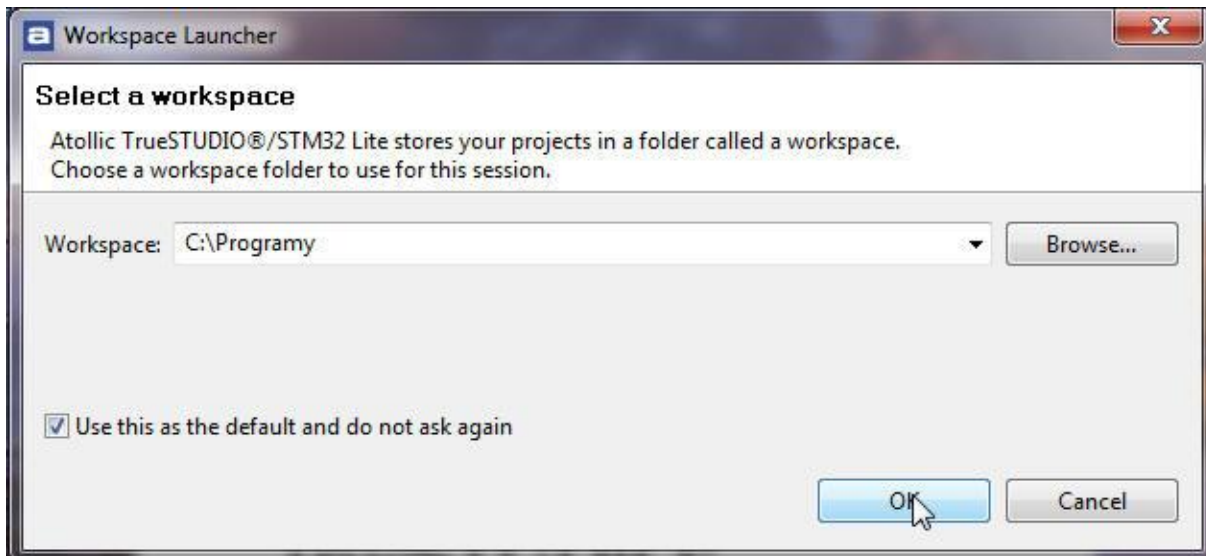


A **Finish**.

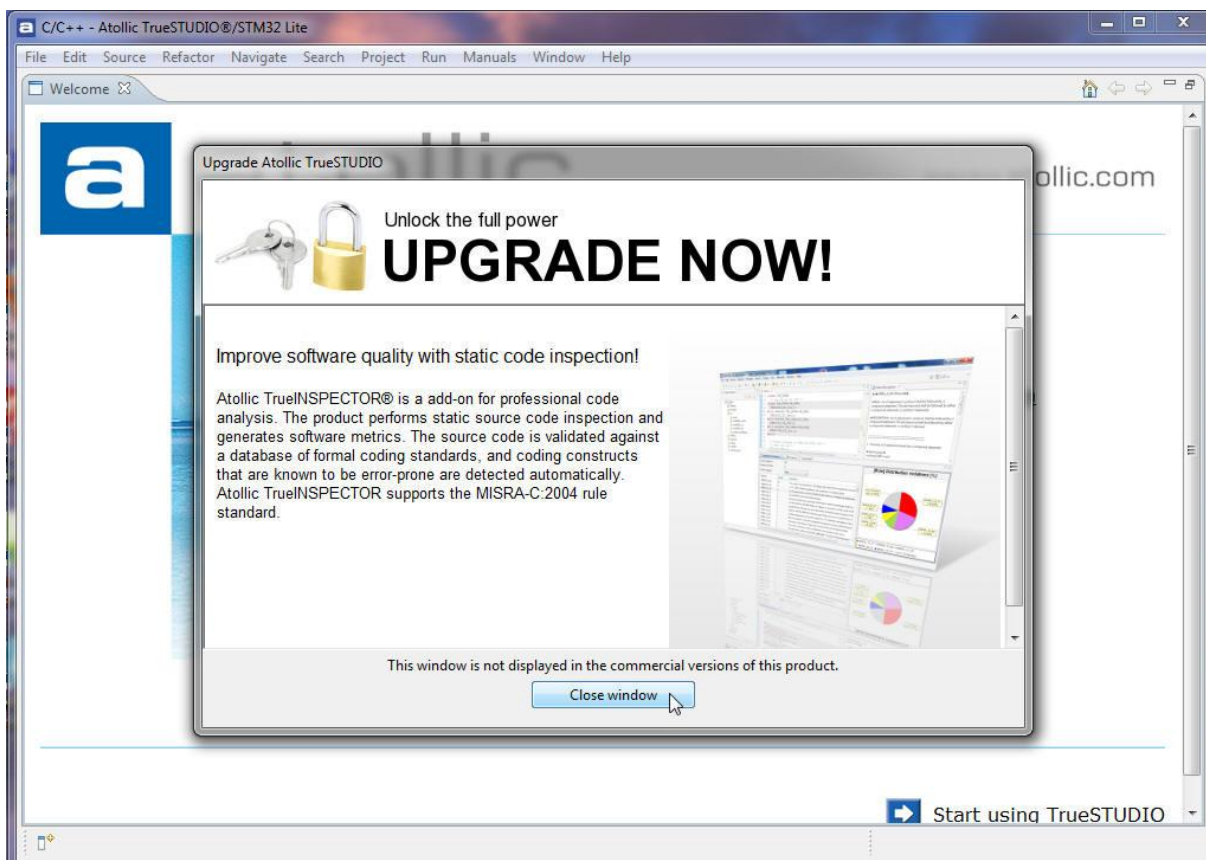
Spustíme nainstalovaný program. Zeptá se nás na umístění workspace (pokud znáte Eclipse, tak víte, že toto dělá Eclipse vždy po prvním spuštění a Atolic využívá právě Eclipse)



Místo nabízeného umístění workspace můžeme vybrat jiné

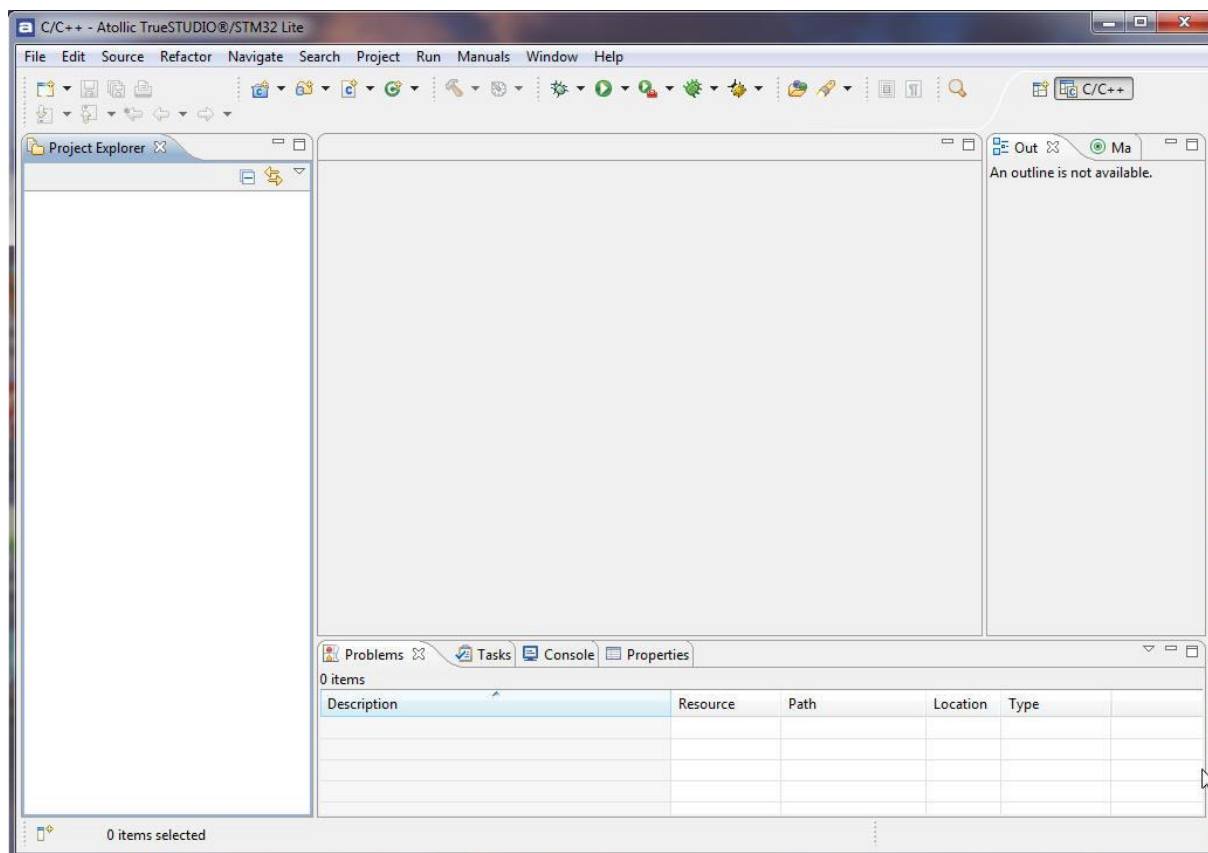


Klikneme na **OK**



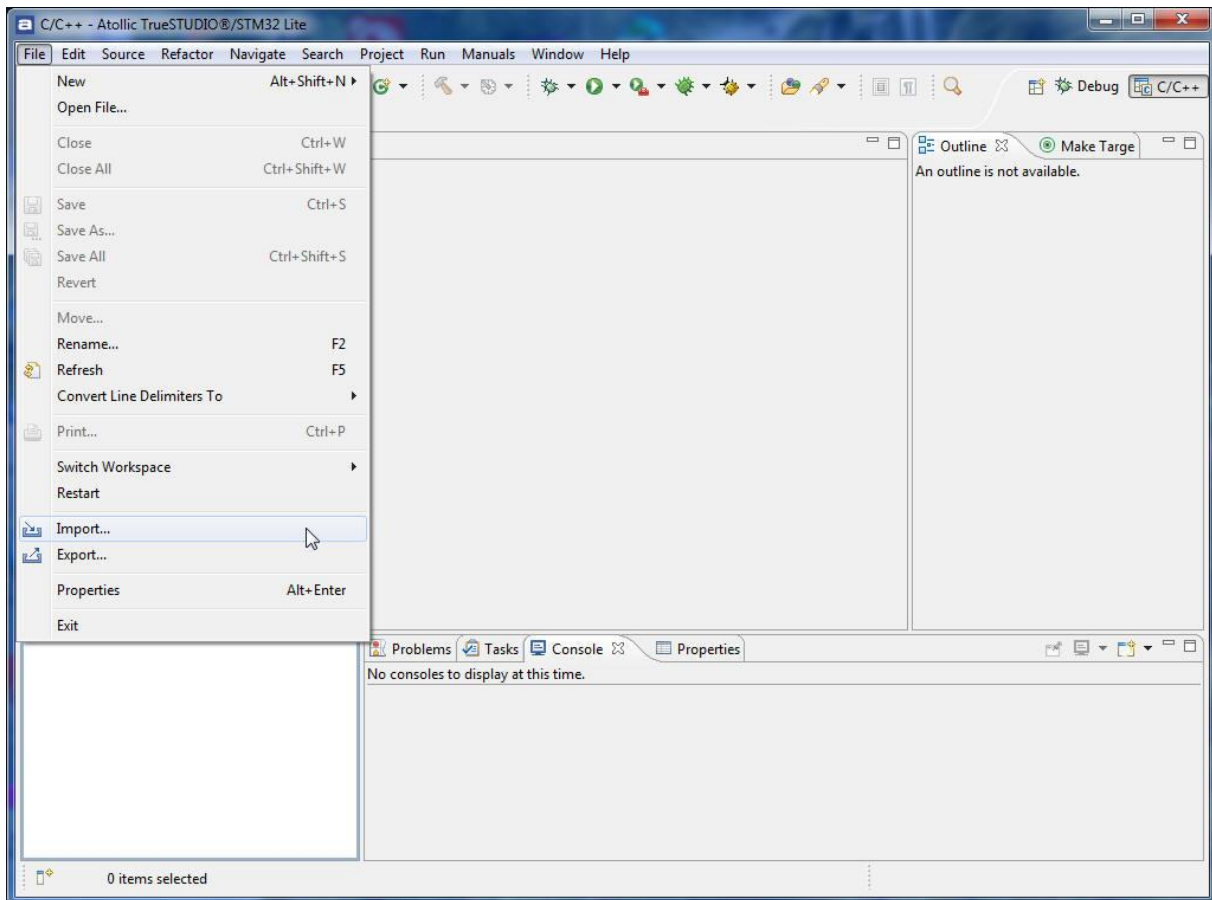
Spustí se vývojové prostředí Atollic v Lite verzi. Ta nás občas bude obtěžovat okny, která hned zavřeme.



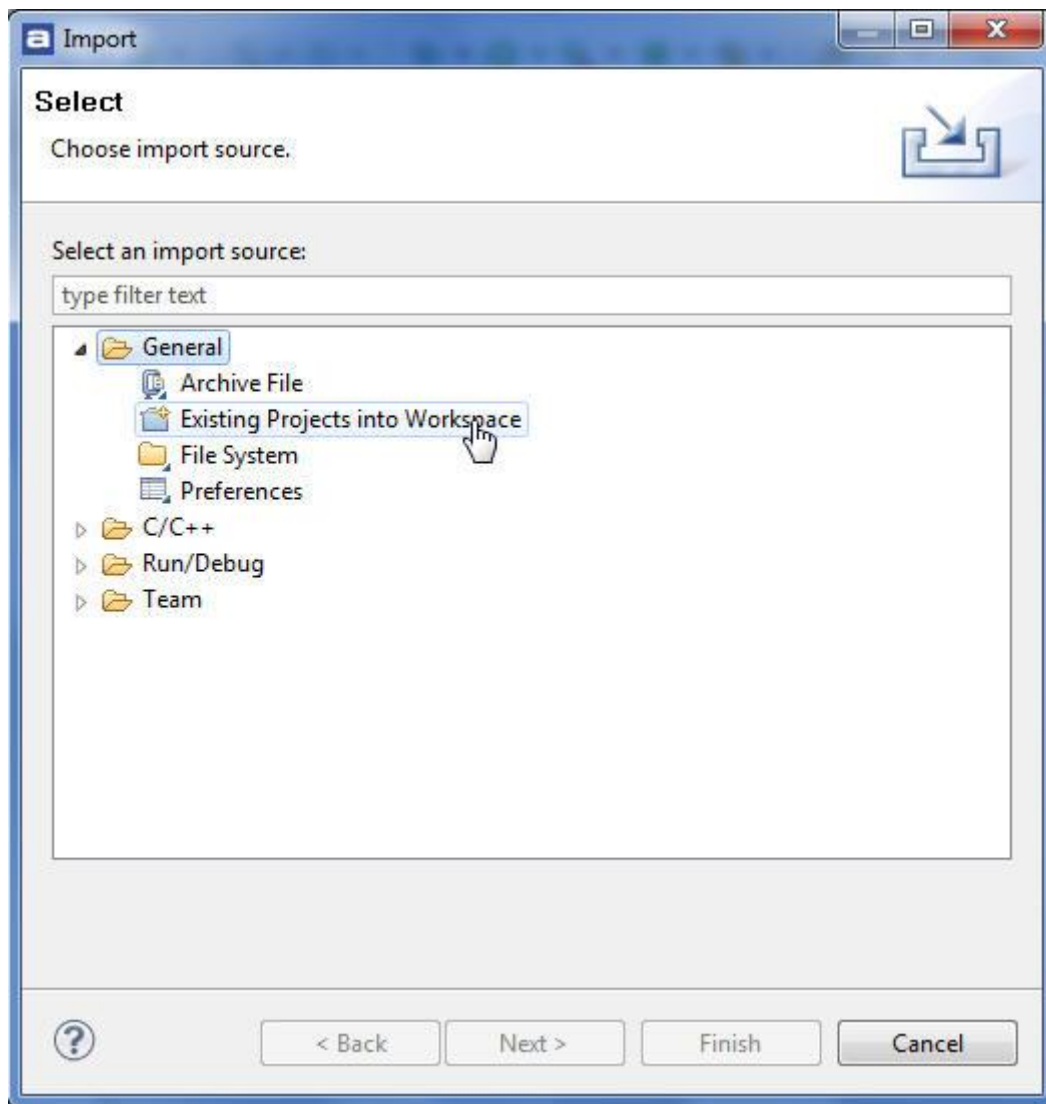


4.2 První program v TrueStudio – import hotového programu

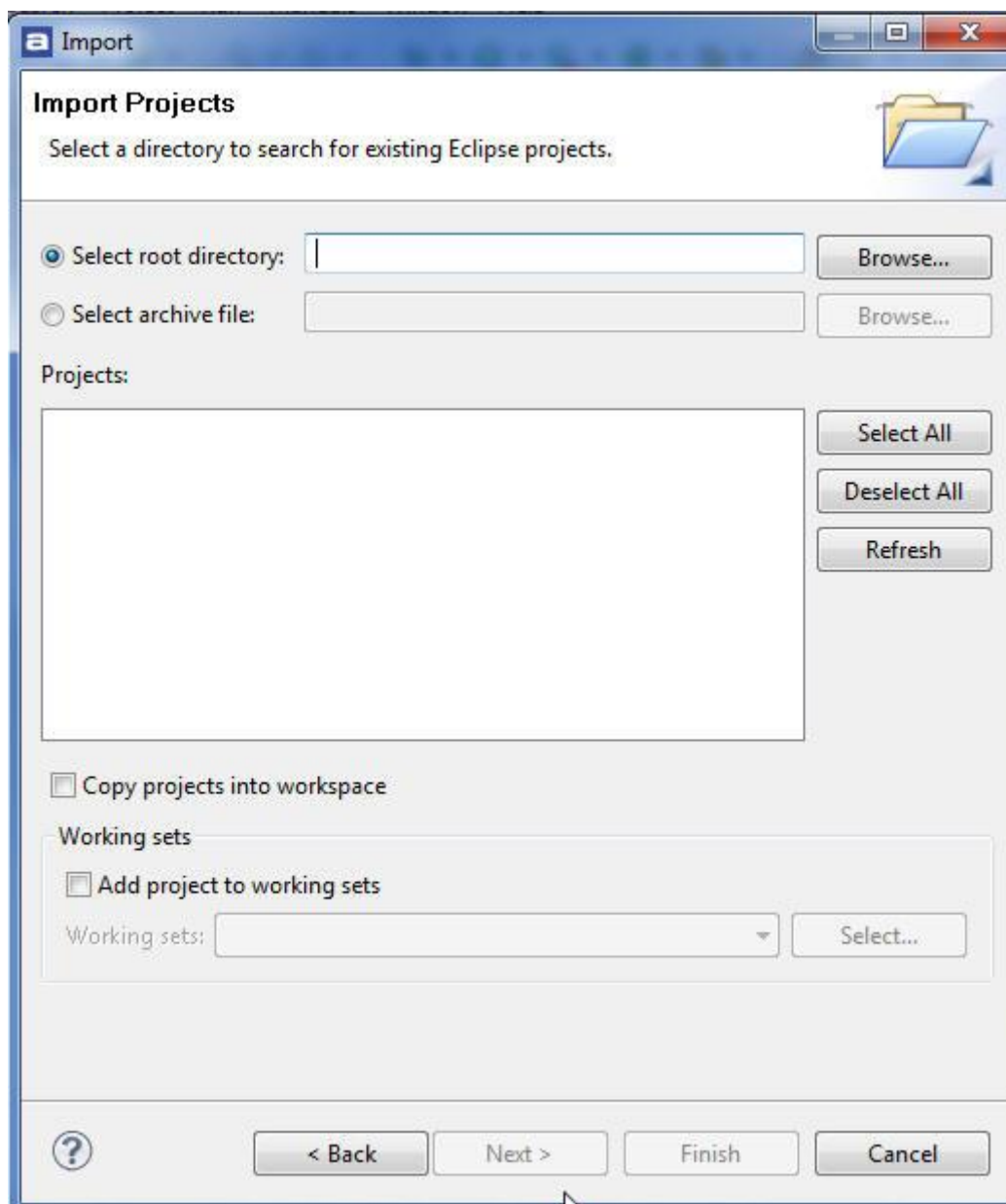
V menu vybereme **File --Import**



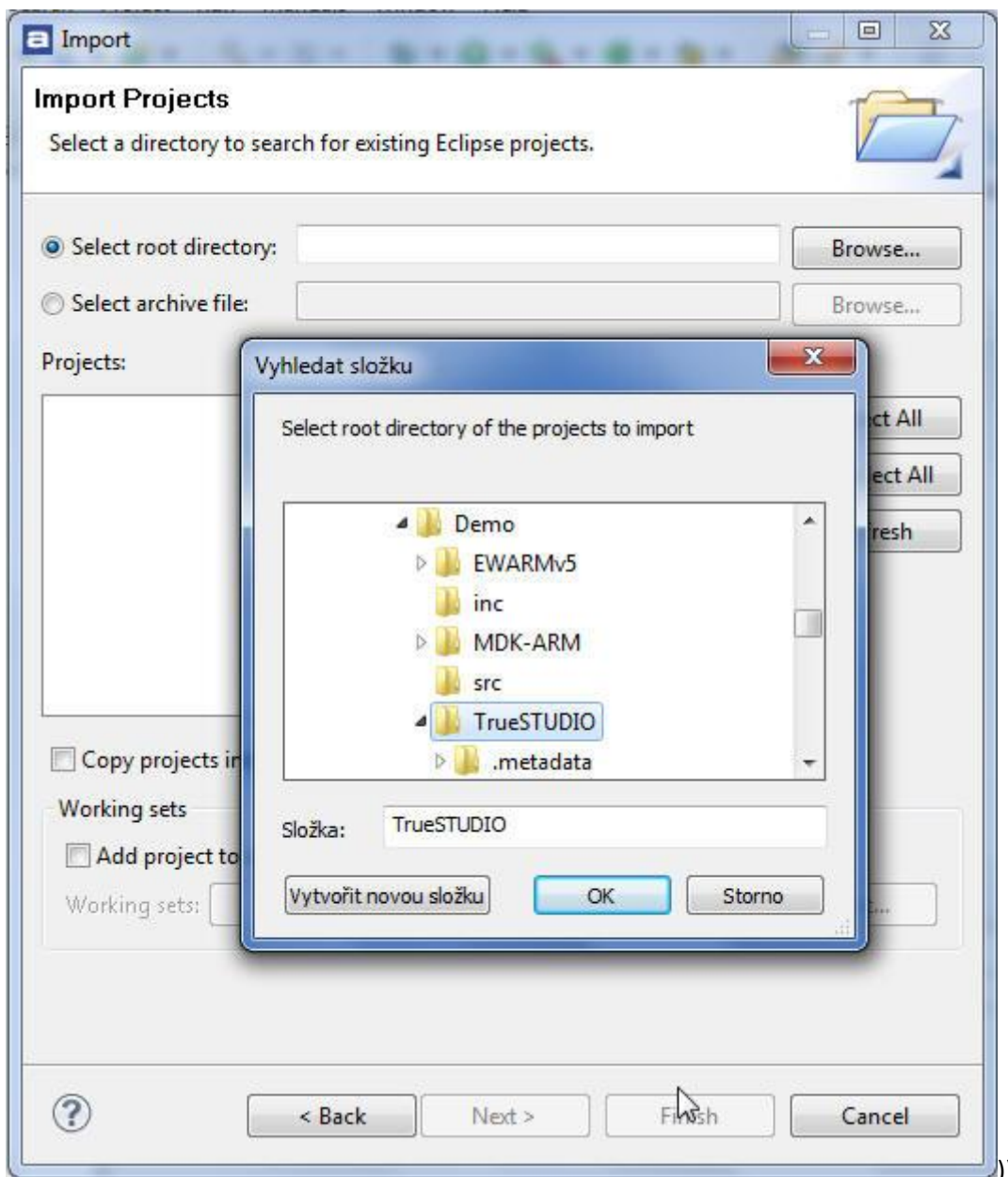
Objeví se okno **Select**



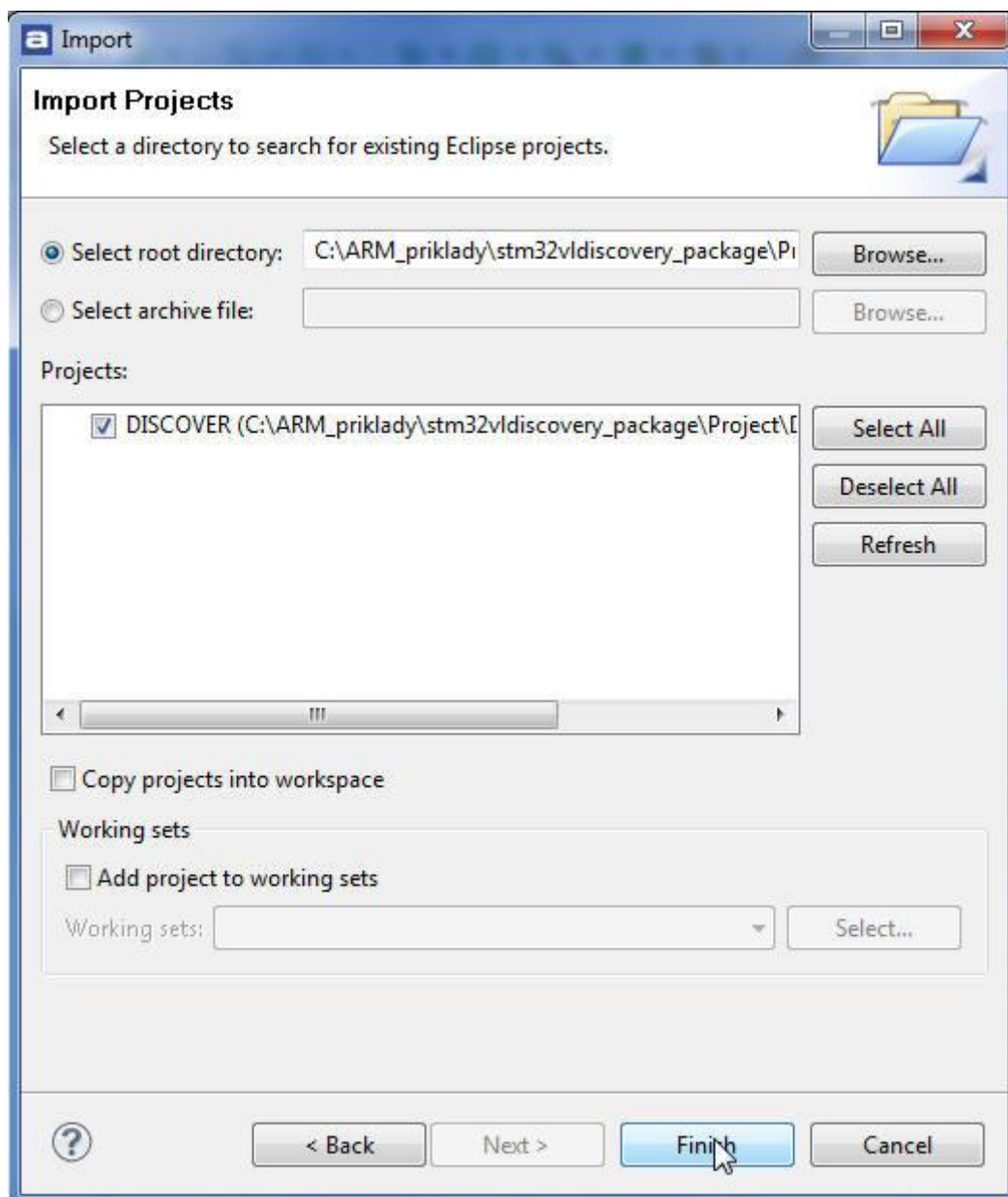
V **General** vybereme **Existing Projects into Workspace**, objeví se



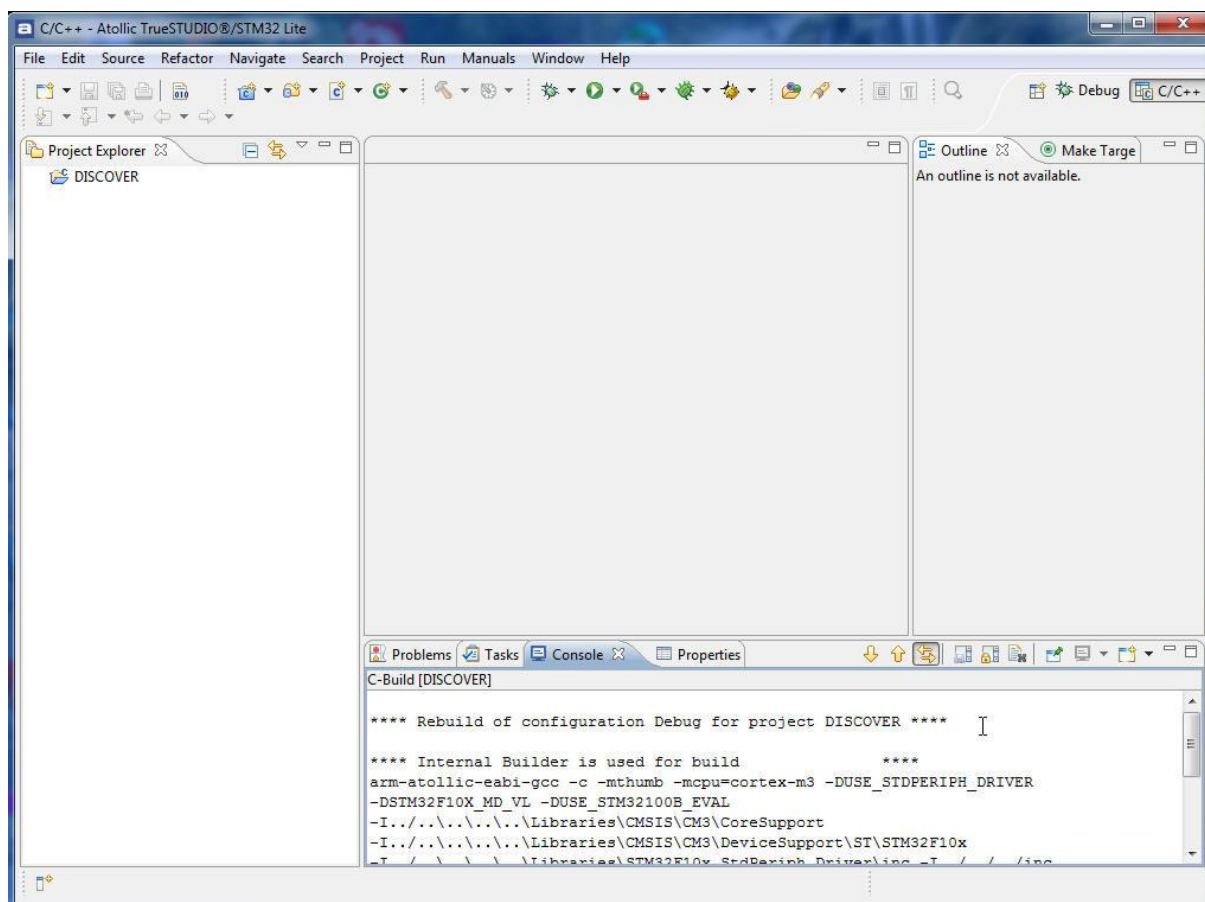
Klikneme na **Browse**. Máme



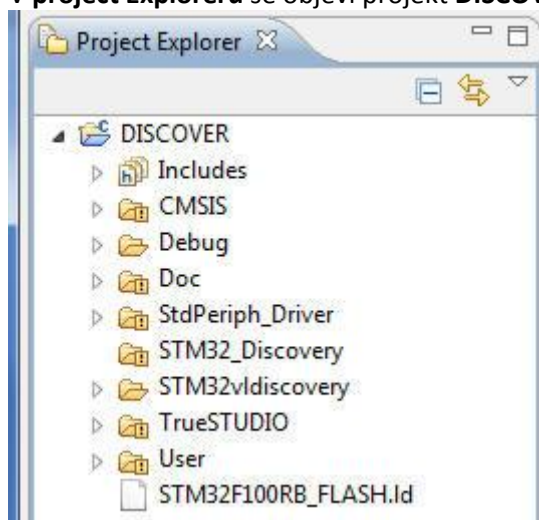
Klikneme na **OK**

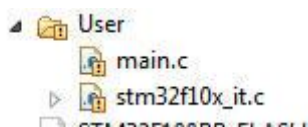
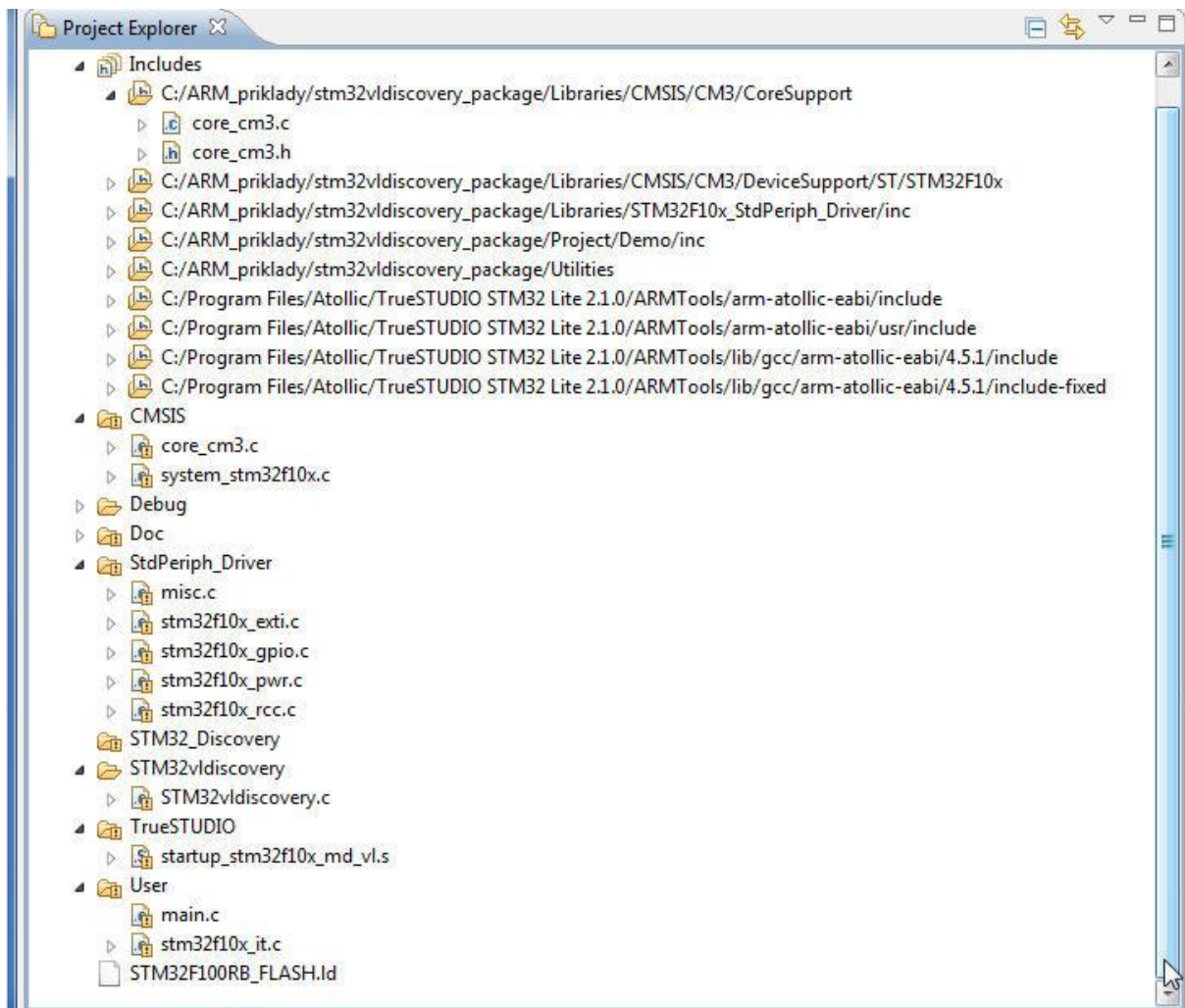


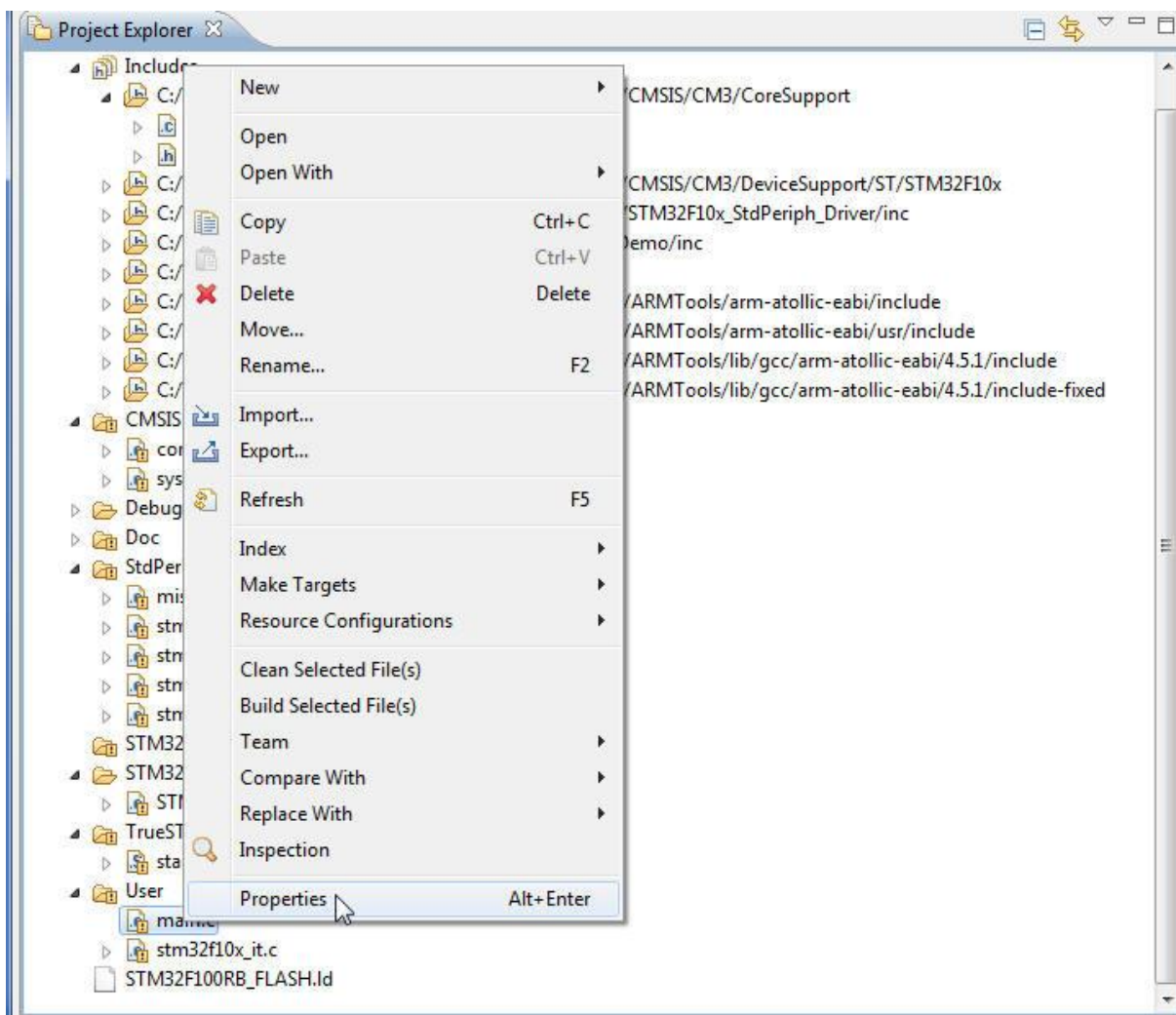
Zkontrolujeme, popř. opravíme dle obr. A klikneme na **Finish**. Dostaneme



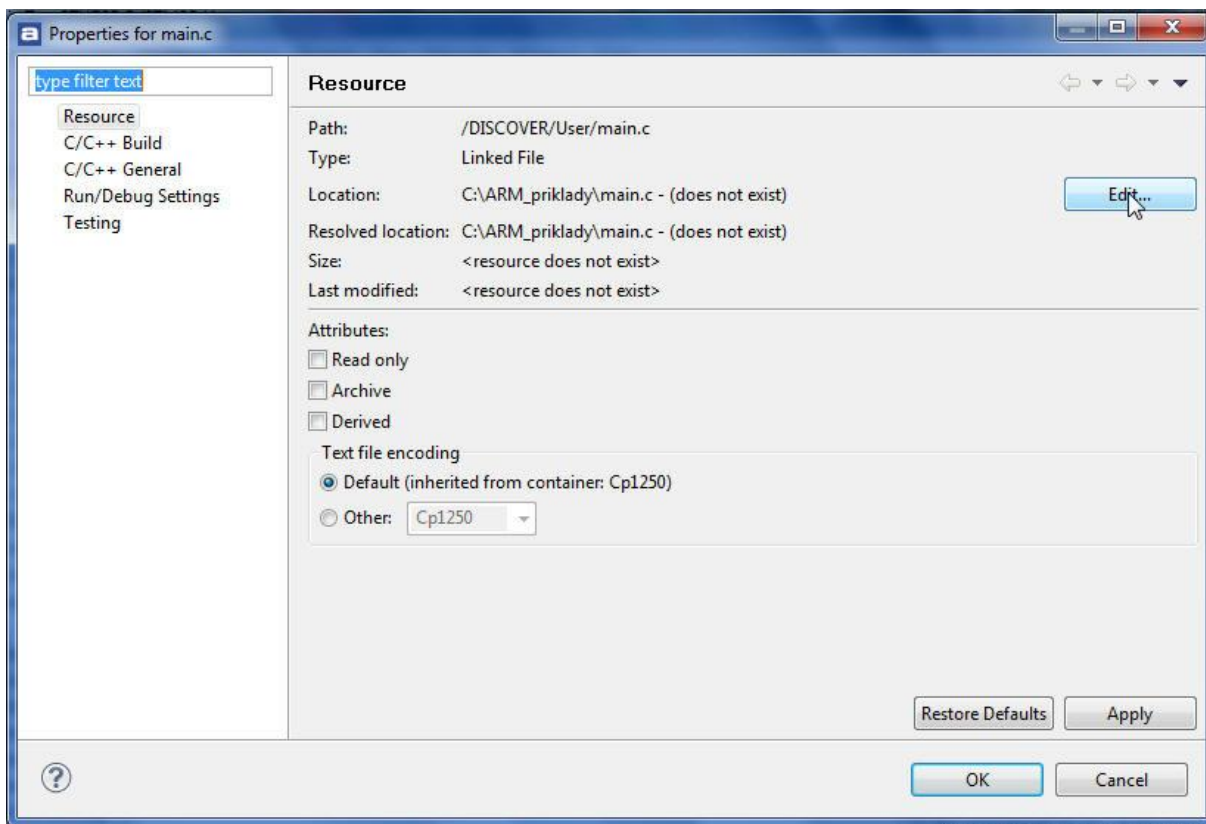
V project Exploreru se objeví projekt **DISCOVER**



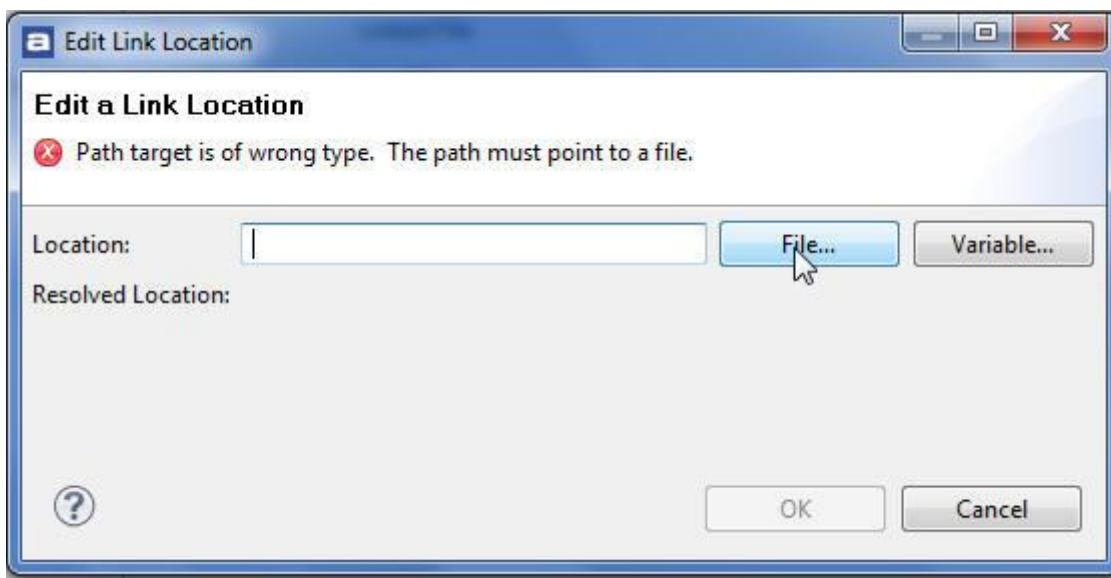




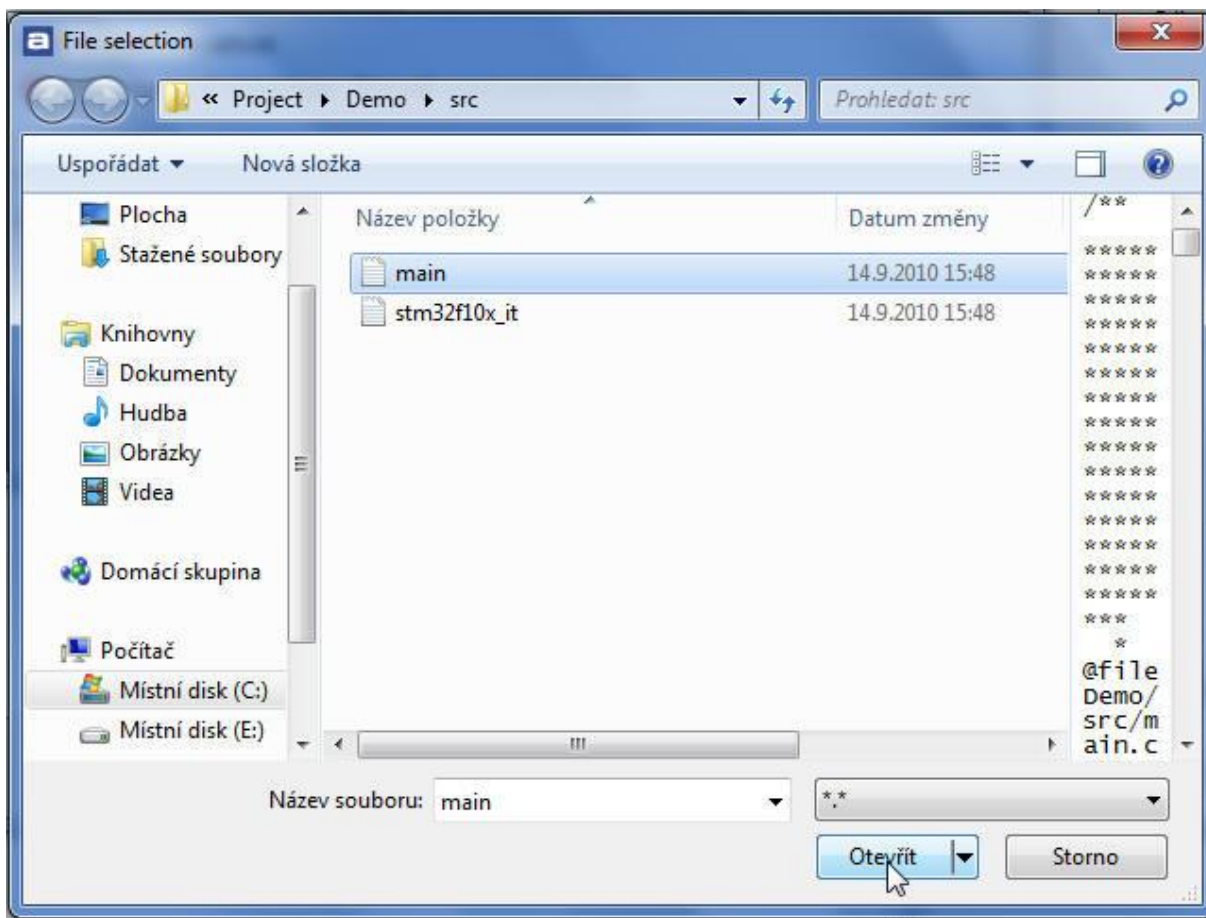
Nastavíme **Properties** pomocí místní nabídky



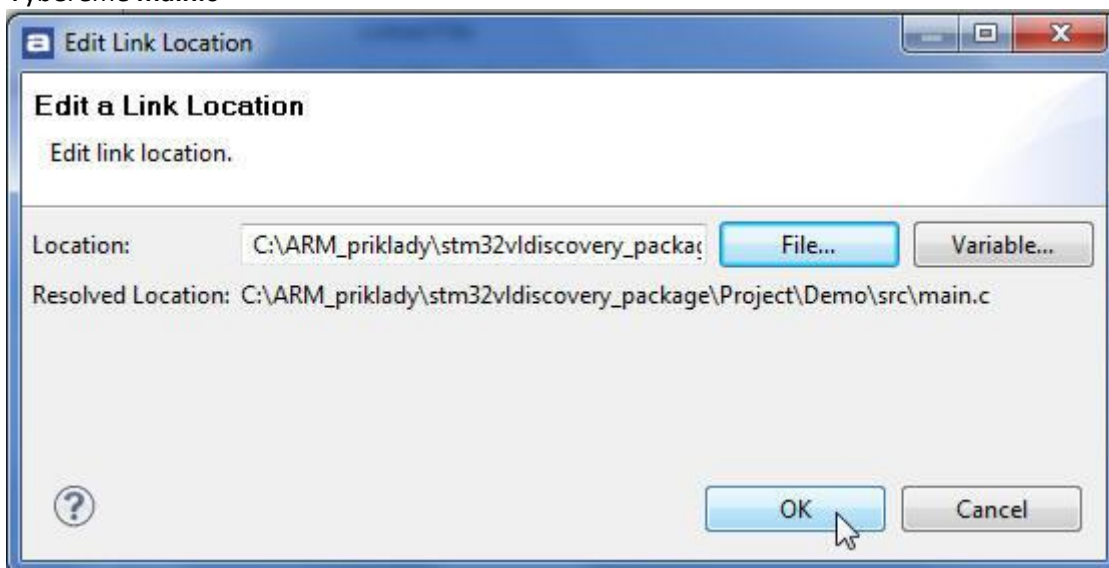
V **Resource** klikneme na **Edit**



Vybereme **Location** kliknutím na tlačítko **File**



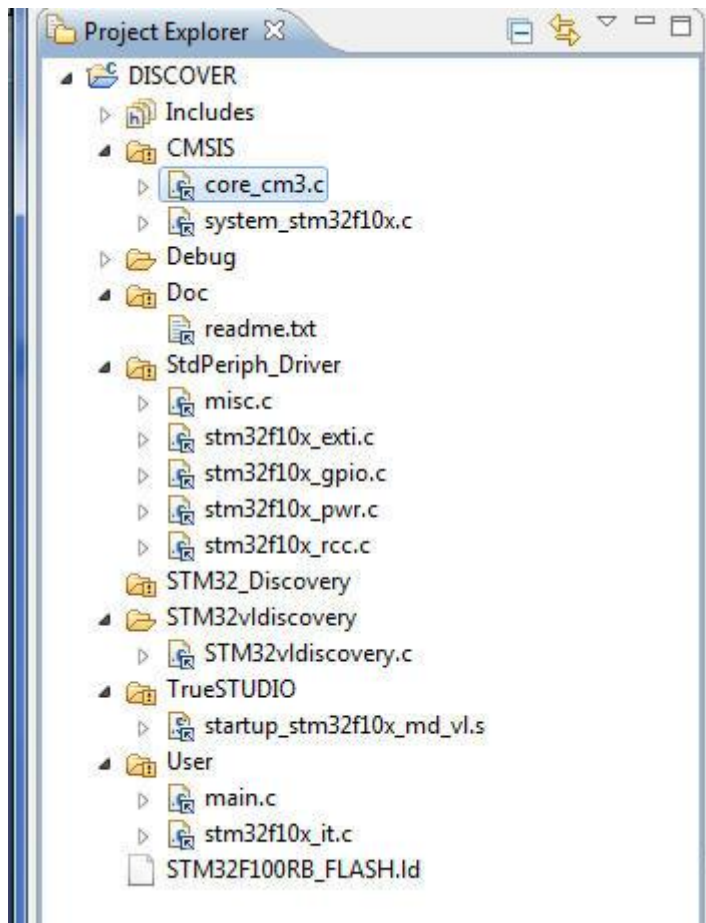
Vybereme **main.c**

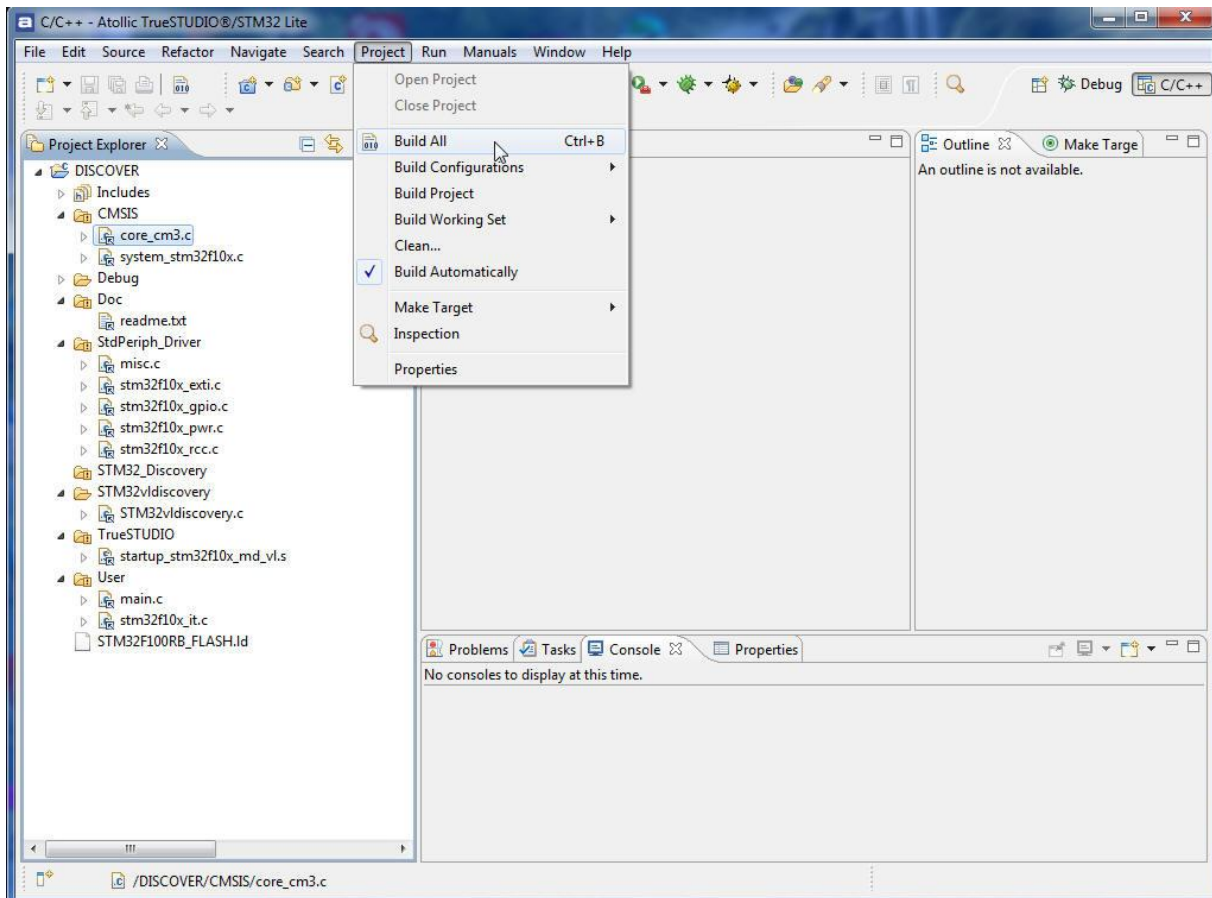


Klikneme na **OK**

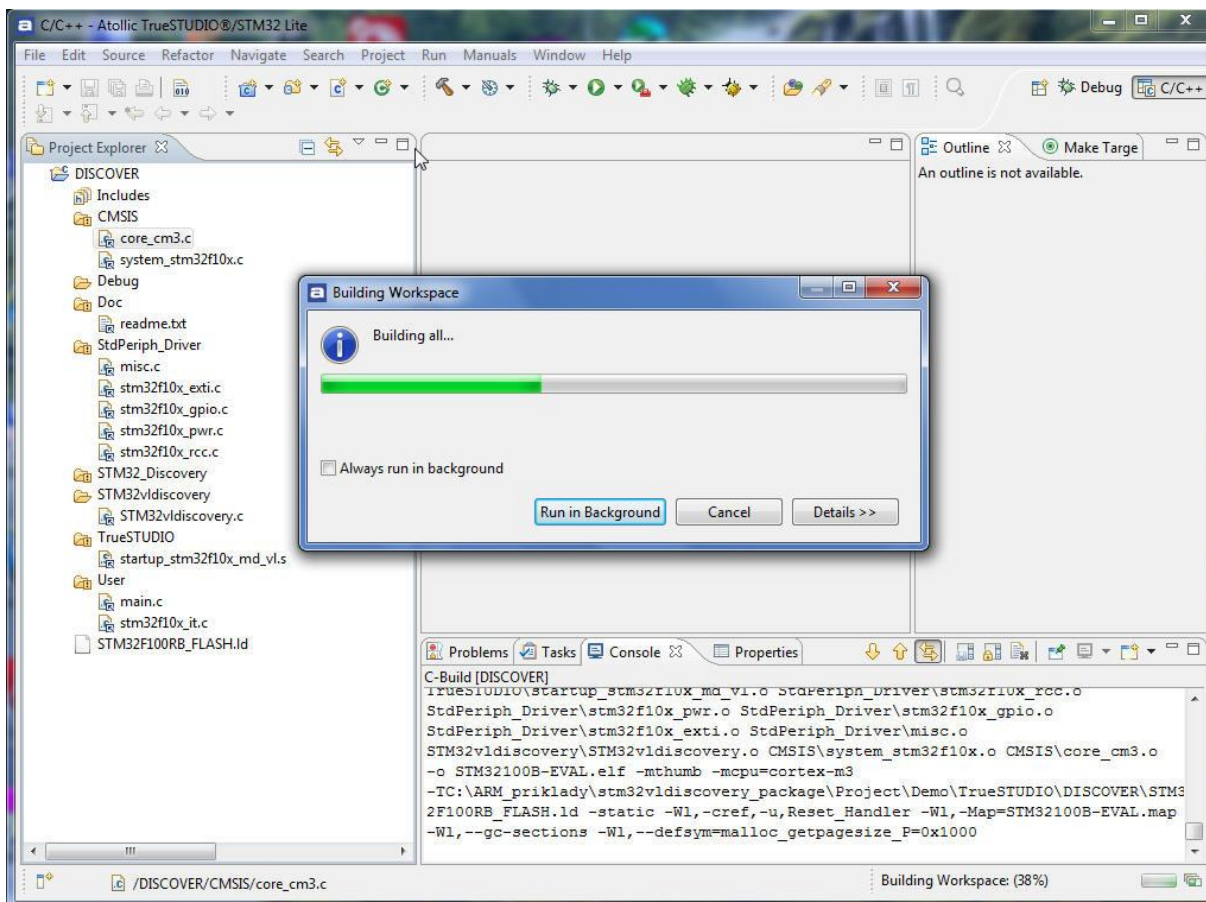


Obdobně s dalšími soubory

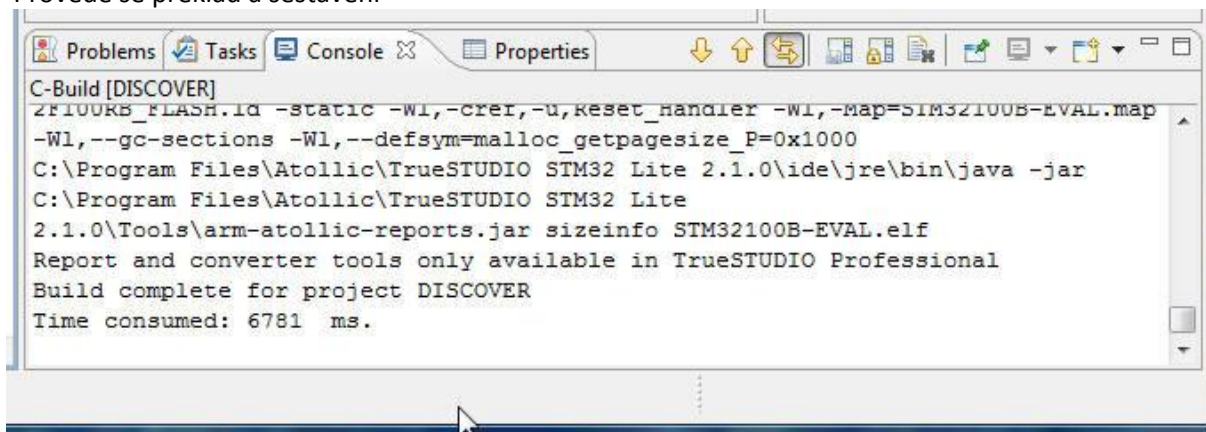




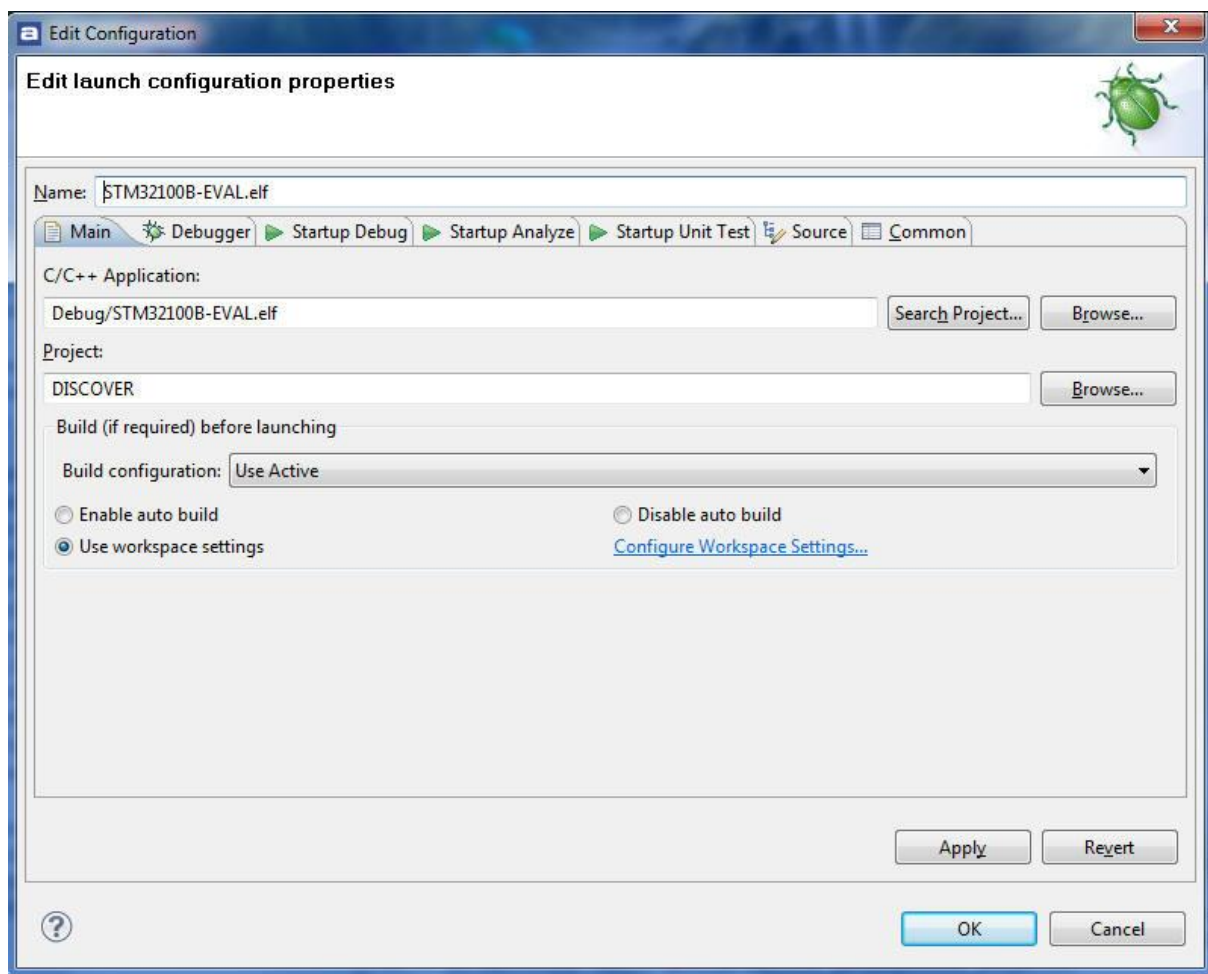
Spustíme **Project – Rebuil All**



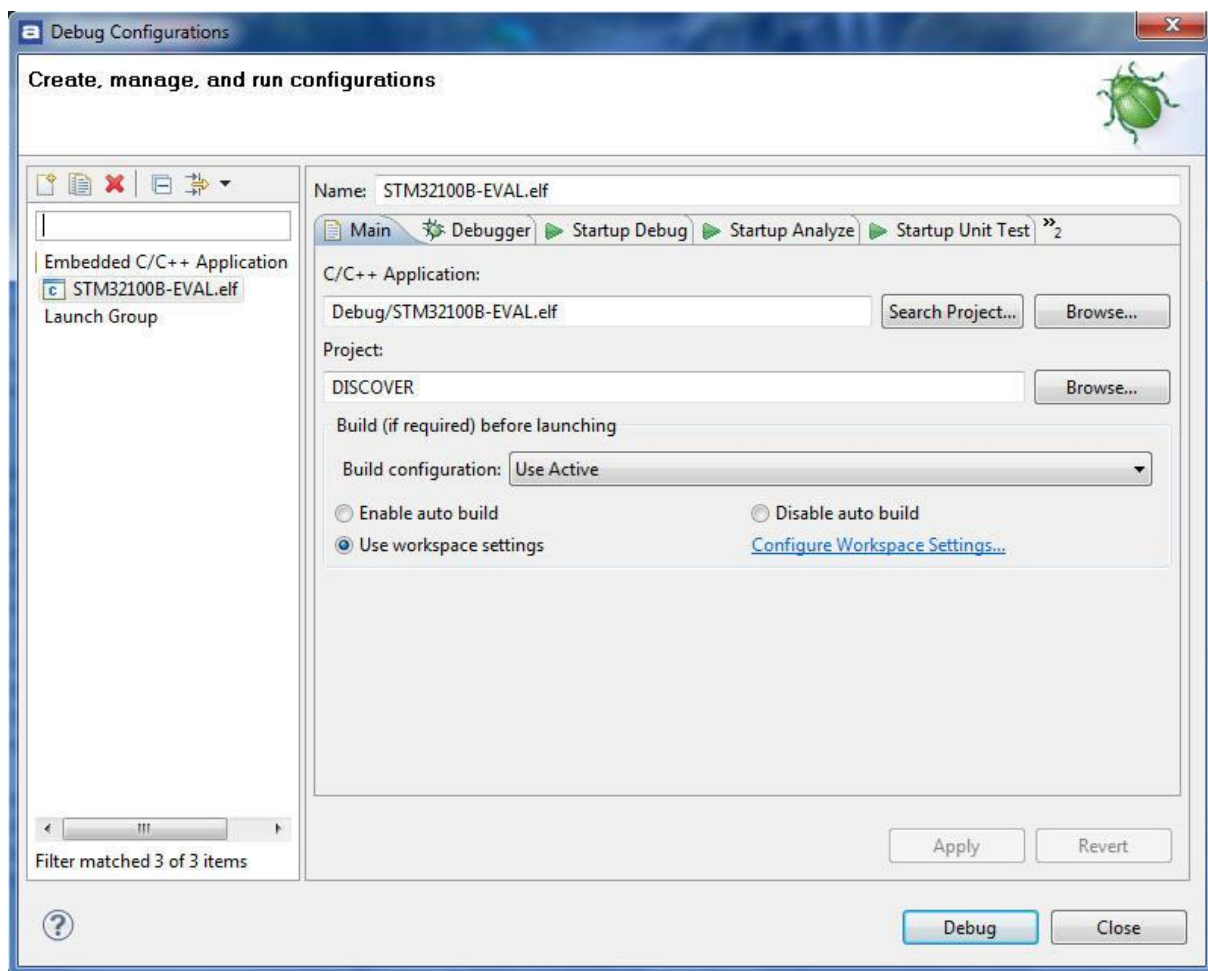
Provede se překlad a sestavení



Zbývá program (ze souboru *.elf) umístit do flash startkitu
Klikneme na **F11** (nebo na ikonku zeleného brouka), objeví se

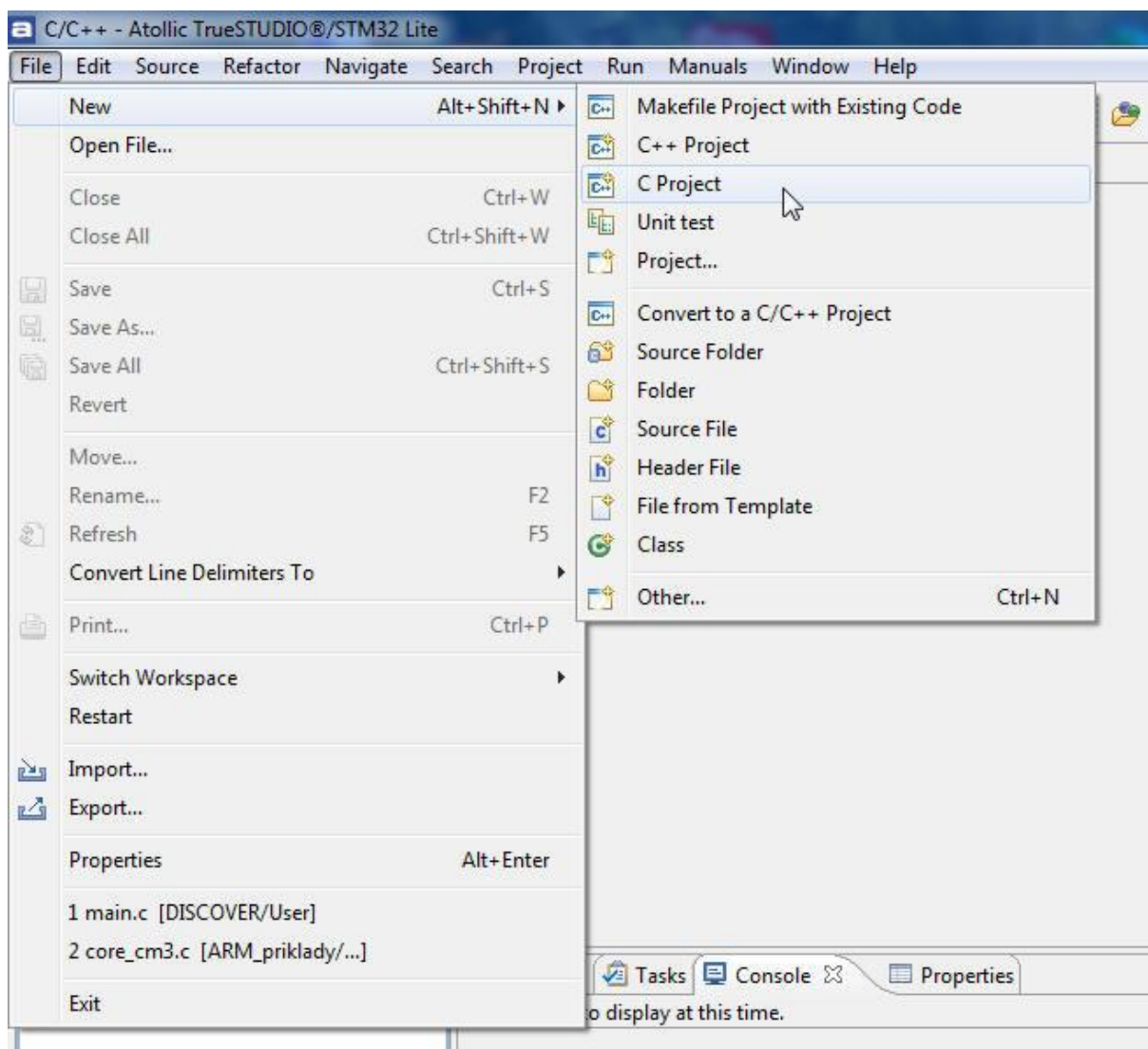


Klikneme na **OK**

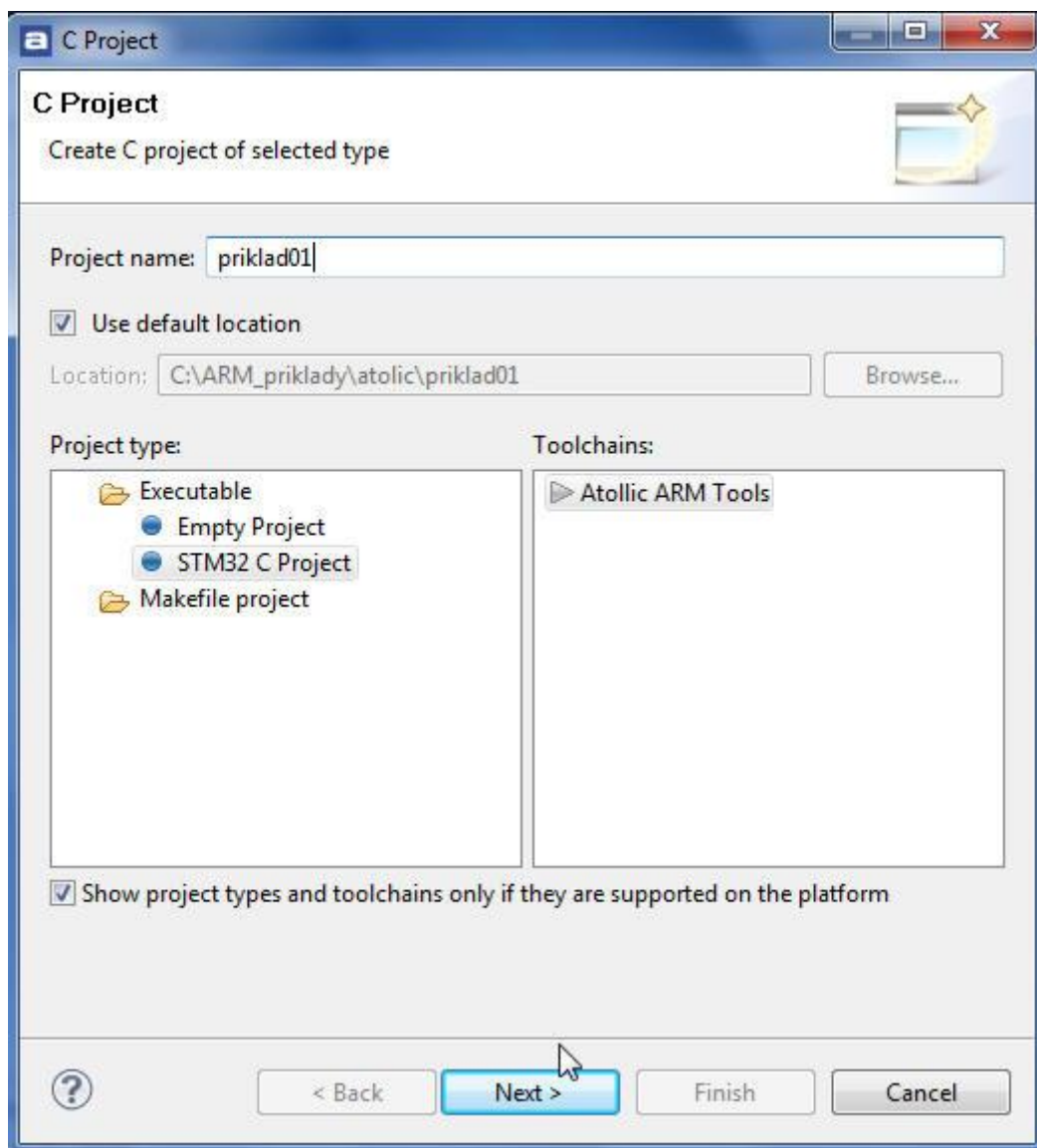


4.3 První vlastní program v TrueStudio

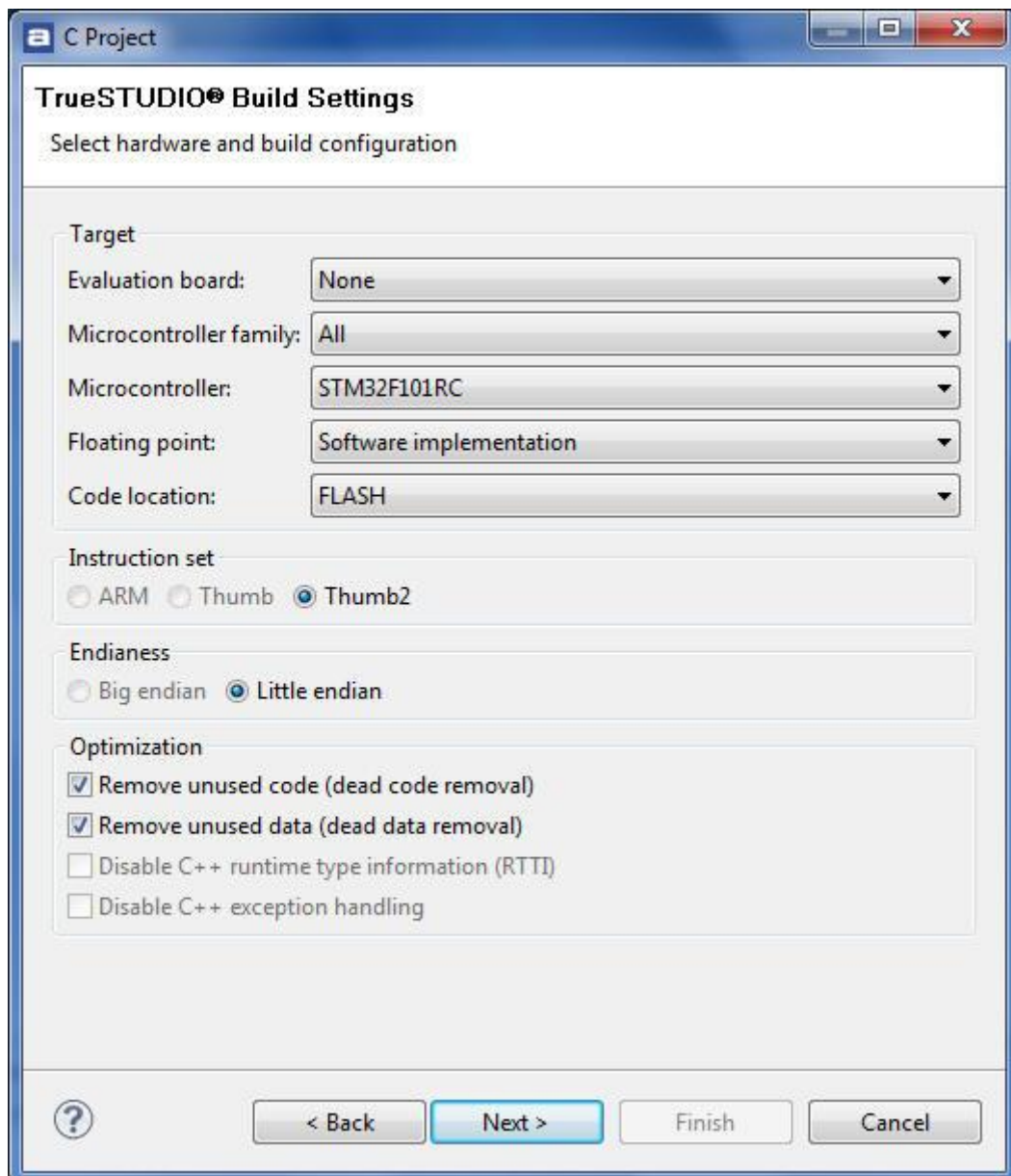
Založíme si první projekt začneme tím, že z menu vybereme **File**, pak **New** a nakonec **C Project**.



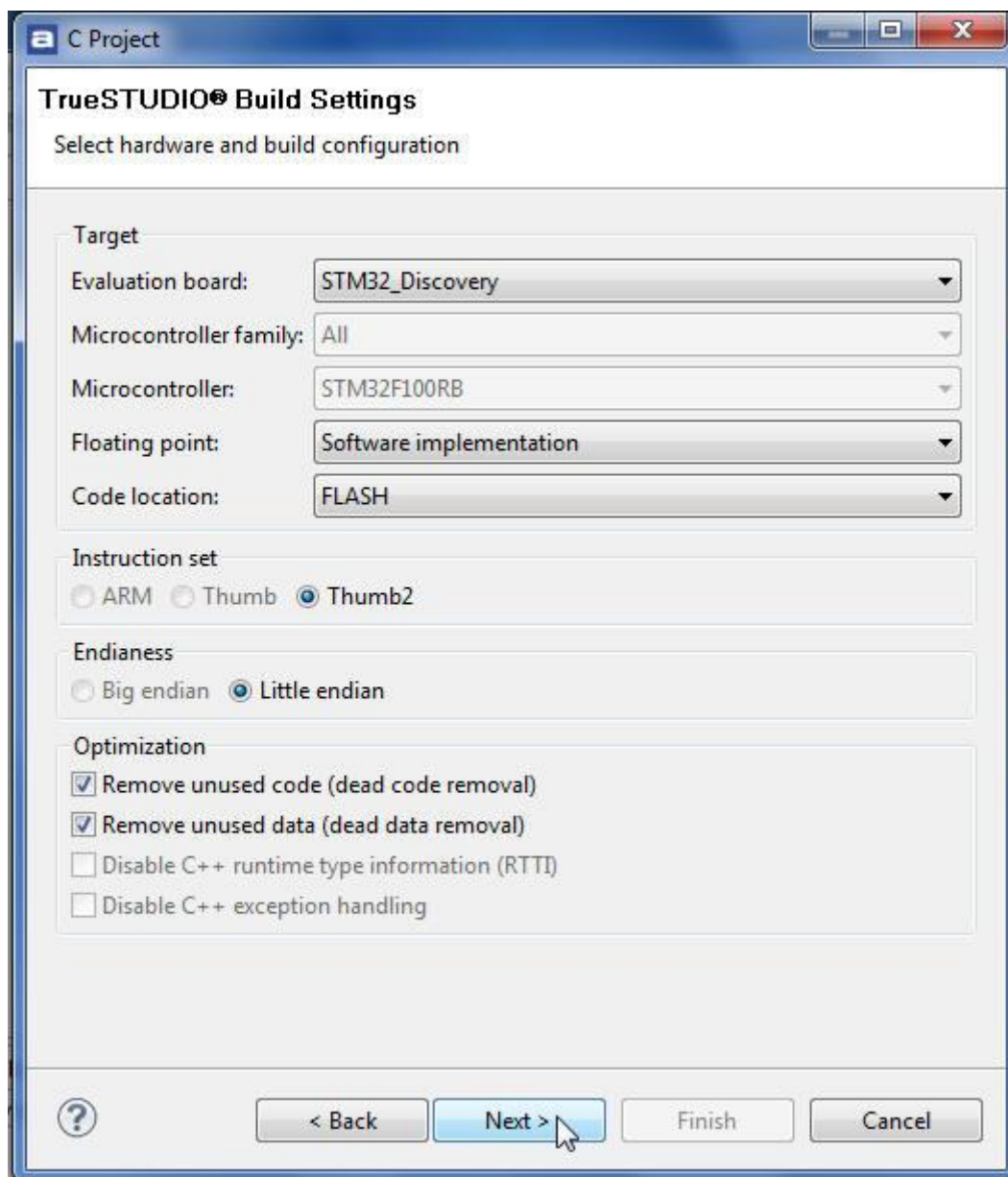
(předtím ovšem zavřeme všechny již existující projekty, např. DISCOVER)



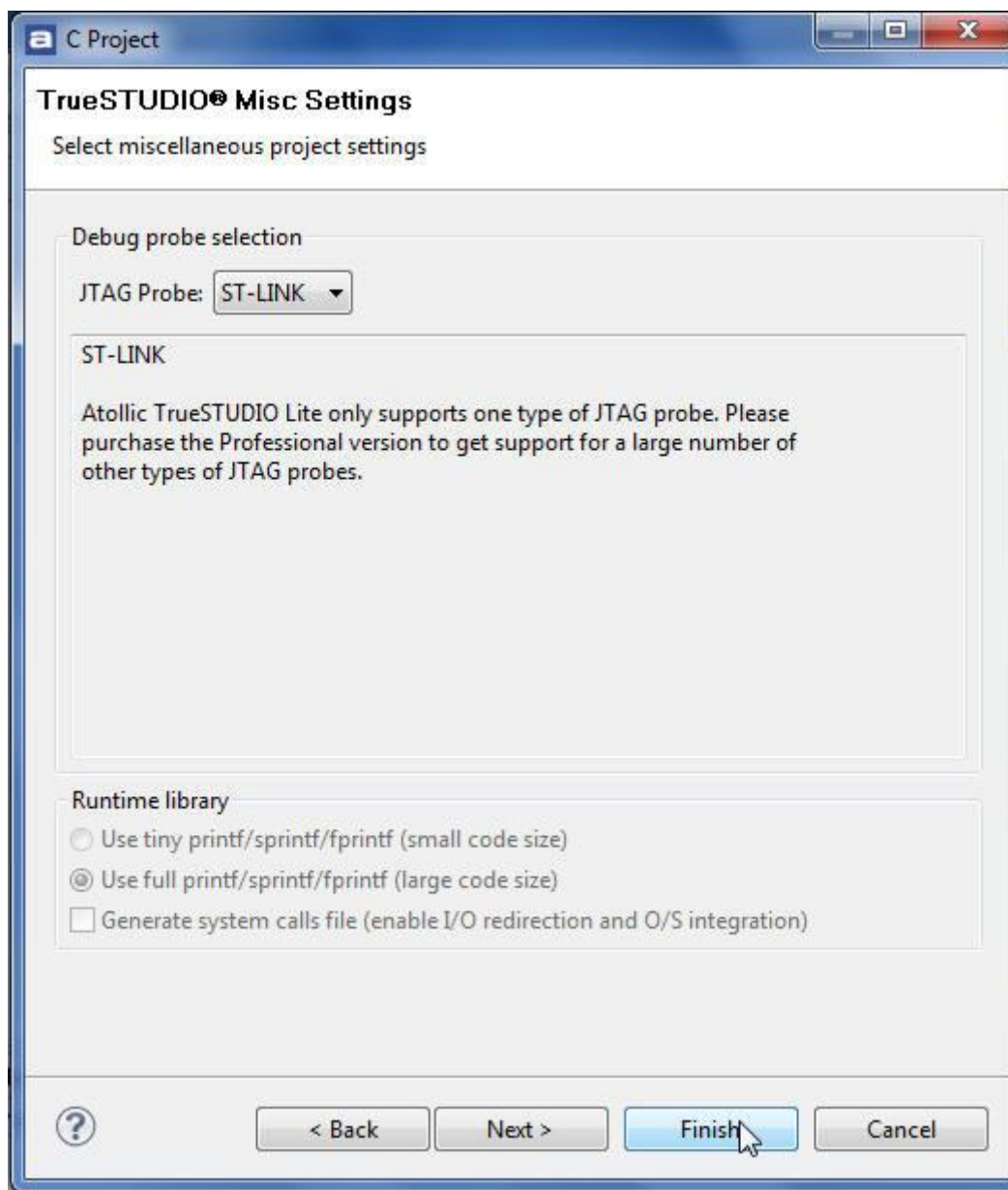
C Projekt nastavíme dle obr. a klikneme na **Next**. Objeví se



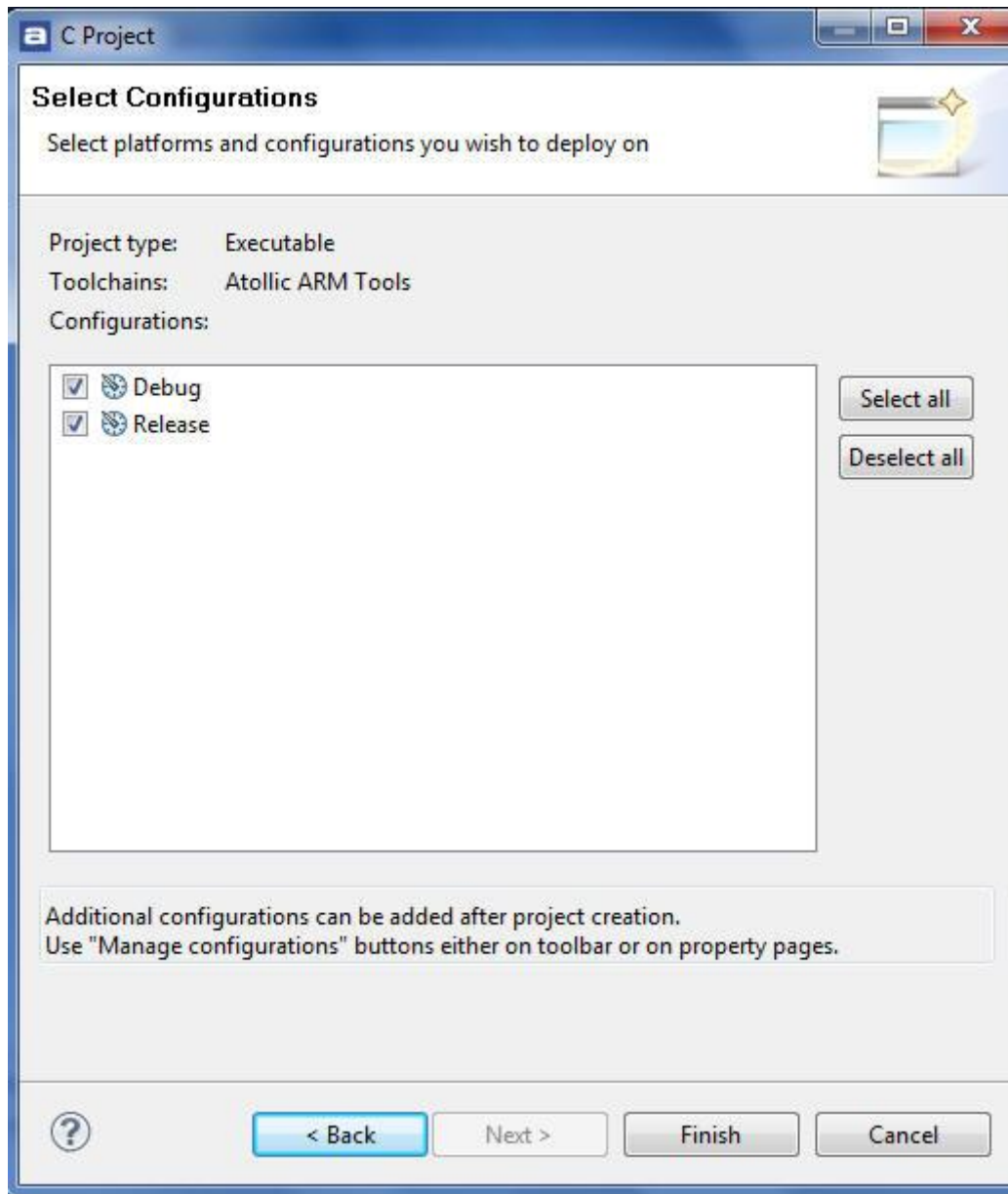
Nastavíme hardwarovou konfiguraci podle obrázku (viz níže)



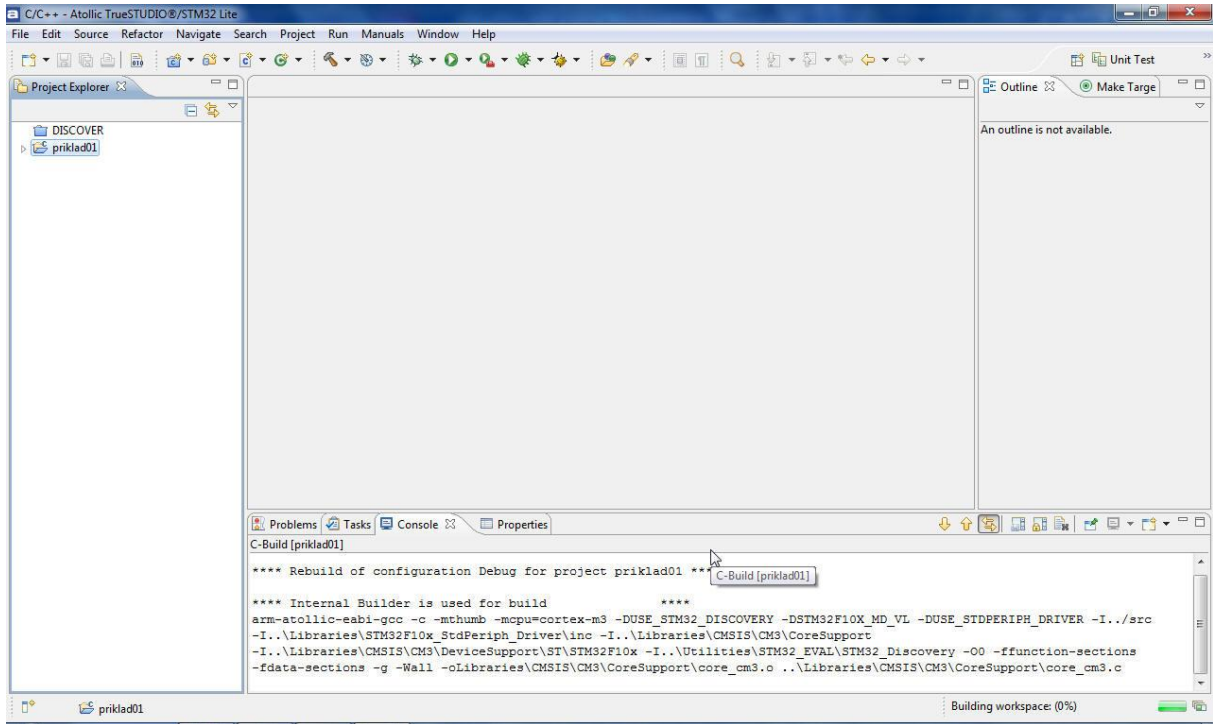
Klineme na **Next**



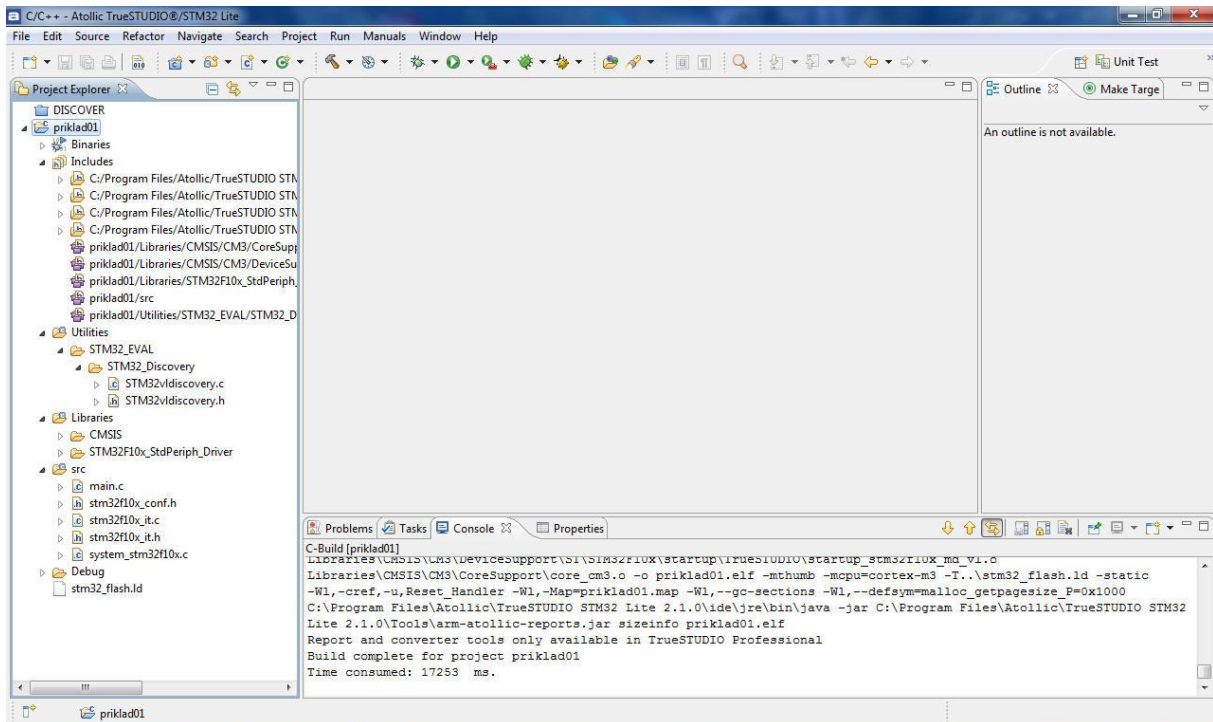
Nastavíme **ST_link** a klikneme na **Finish**



Klikneme na **Finish**



Rozvinu priklad01 v **Project Explorer**. V okně **Console** také vidíme, že proběhl překlad a sestavení projektu.



Kliknu na **main.c**

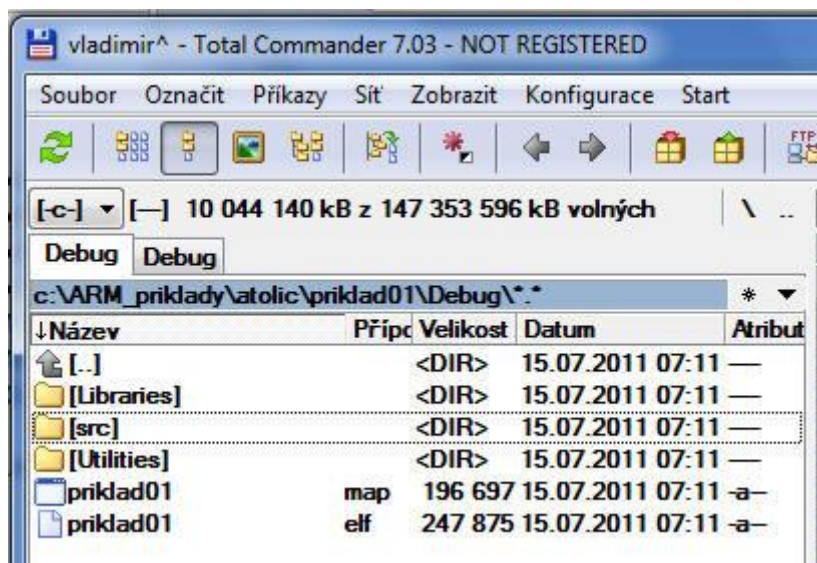
```

1  /**
2  ****
3  **
4  ** File      : main.c
5  **
6  ** Abstract  : main function.
7  **
8  ** Functions : main
9  **
10 ** Environment : Atollic TrueSTUDIO/STM32
11 **              STMicroelectronics STM32F10x Standard Peripherals Library
12 **
13 ** Distribution: The file is distributed "as is," without any warranty
14 **              of any kind.
15 **
16 ** (c)Copyright Atollic AB.
17 ** You may use this file as-is or modify it according to the needs of your
18 ** project. Distribution of this file (unmodified or modified) is not
19 ** permitted. Atollic AB permit registered Atollic TrueSTUDIO(R) users the
20 ** rights to distribute the assembled, compiled & linked contents of this
21 ** file as part of an application binary file, provided that it is built
22 ** using the Atollic TrueSTUDIO(R) toolchain.
23 **
24 **
25 ****
26 */
27
28 /* Includes */
29 #include <stddef.h>
30 #include "stm32f10x.h"

```

Podíváme se, co nám wizard vytvořil.

| Název | Přípc | Velikost | Datum | Atribut |
|-------------|-------|----------|------------------|---------|
| [.] | | <DIR> | 15.07.2011 07:11 | — |
| [.settings] | | <DIR> | 15.07.2011 07:11 | — |
| [Debug] | | <DIR> | 15.07.2011 07:11 | — |
| [Libraries] | | <DIR> | 15.07.2011 07:11 | — |
| [src] | | <DIR> | 15.07.2011 07:11 | — |
| [Utilities] | | <DIR> | 15.07.2011 07:11 | — |
| stm32_flash | ld | 4 789 | 15.07.2011 07:11 | a- |
| .project | | 2 148 | 15.07.2011 07:11 | a- |
| .cproject | | 31 212 | 15.07.2011 07:11 | a- |



Příklad01.elf je výsledný soubor který použijeme jako zdroj k naprogramování flash startkitu. Soubory bin a hex vytváří až **TrueSTUDIO Professional**.

Poznámka.

Z <http://www.jukie.net/~bart/utills/elf2hex> lze stáhnout utilitu pro převod elf na hex. Utilita je sice určená pro Linux, lze však stáhnout i její zdrojový kód `elf2hex.c`

Našel jsem i na

http://www.stm32.eu/index.php?option=com_content&view=article&id=174:konwersja-plikow-elf2hex-i-elf2bin&catid=35:porady&Itemid=57

Konverze elf na hex

ARM-GCC (Sourcery G++ Lite):

```
arm-none-eabi-objcopy -O ihex plik_zrodlowy.elf plik_wynikowy.hex
```

ARM-GCC (Yagarto):

```
arm-elf-objcopy -O ihex plik_zrodlowy.elf plik_wynikowy.hex
```

Konverze elf na bin

ARM-GCC (Sourcery G++ Lite):

```
arm-none-eabi-objcopy -O binary plik_zrodlowy.elf plik_wynikowy.bin
```

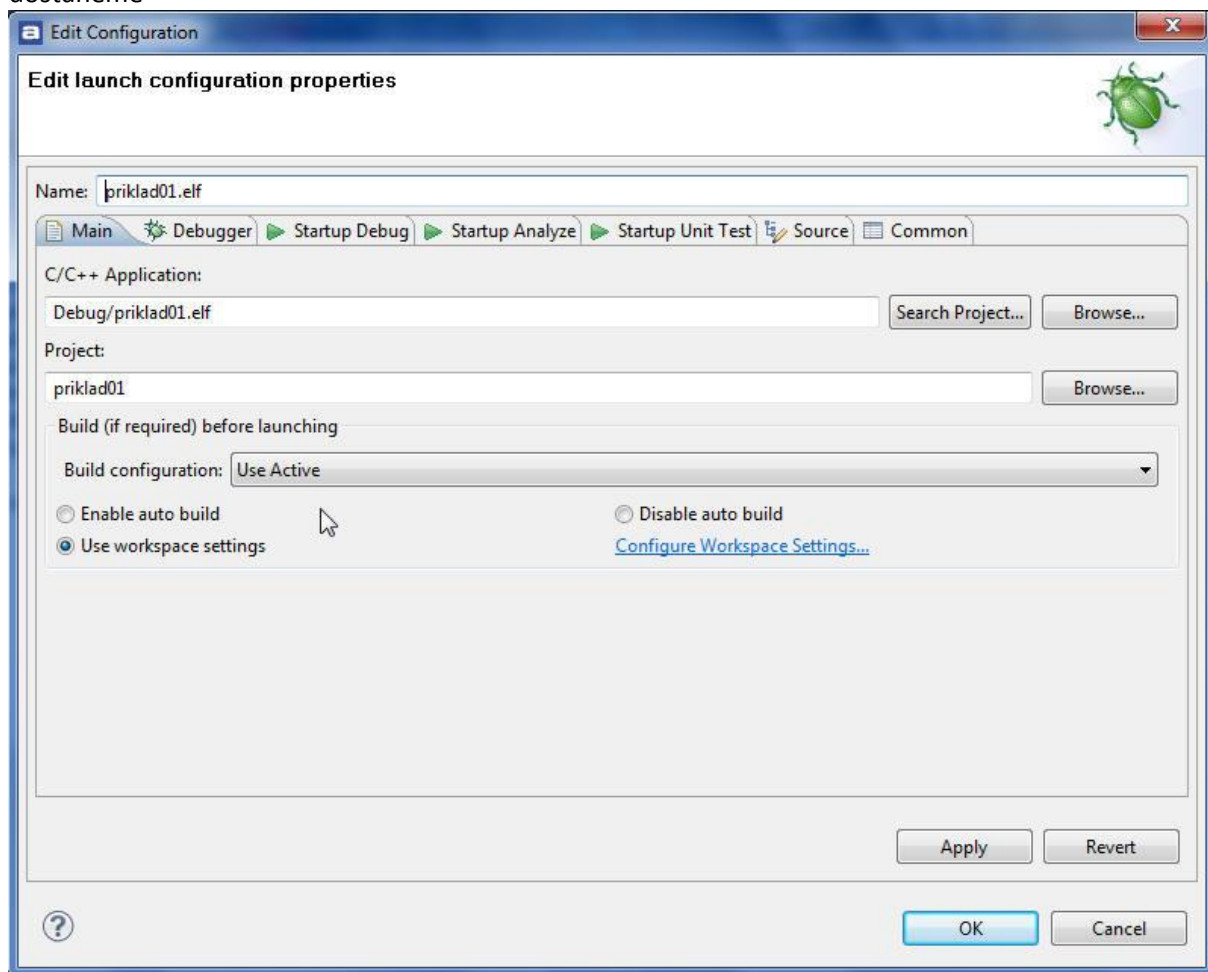
ARM-GCC (Yagarto):

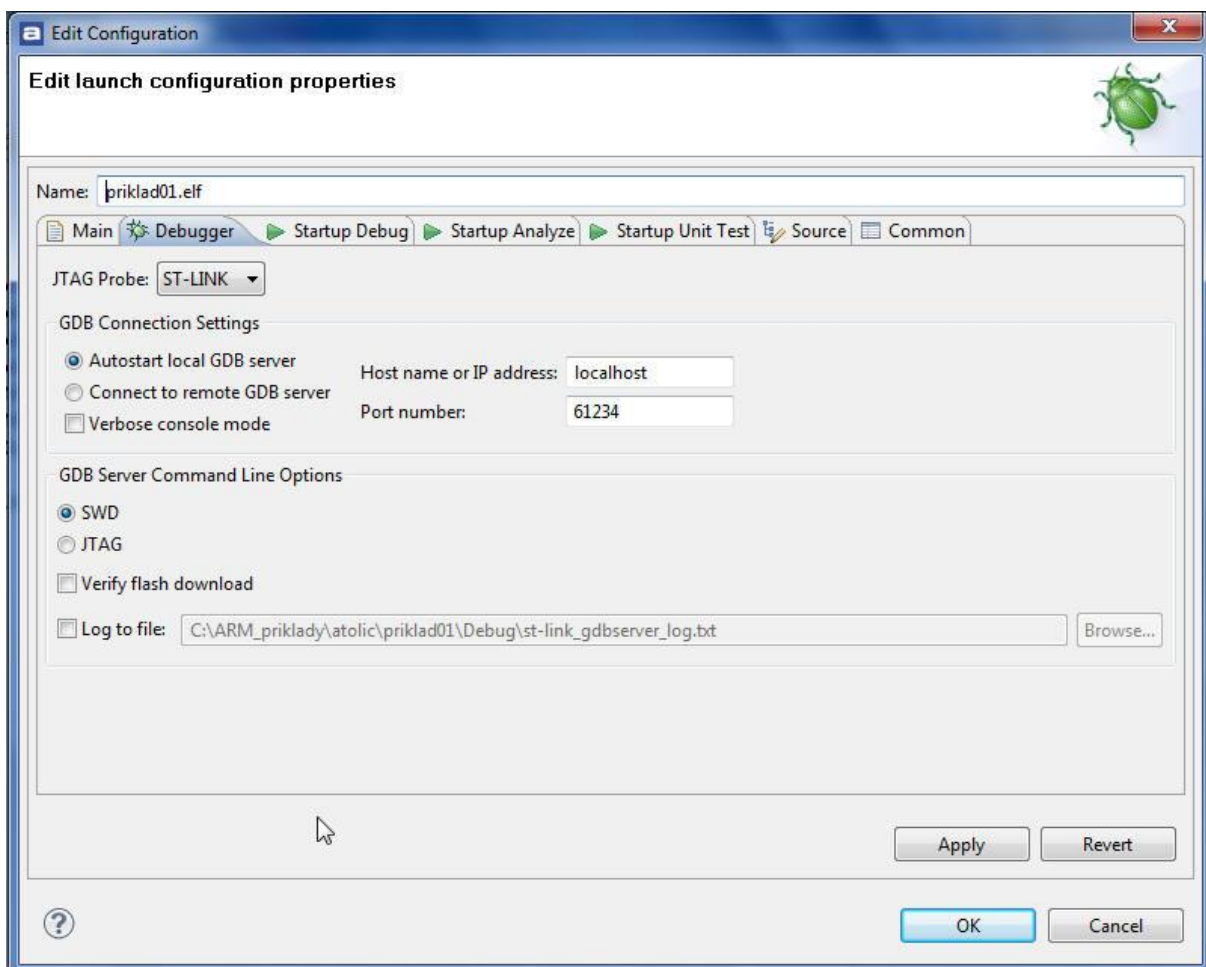
```
arm-elf-objcopy -O binary plik_zrodlowy.elf plik_wynikowy.bin
```

Klikneme na ikonu zeleného brouka (Debug)

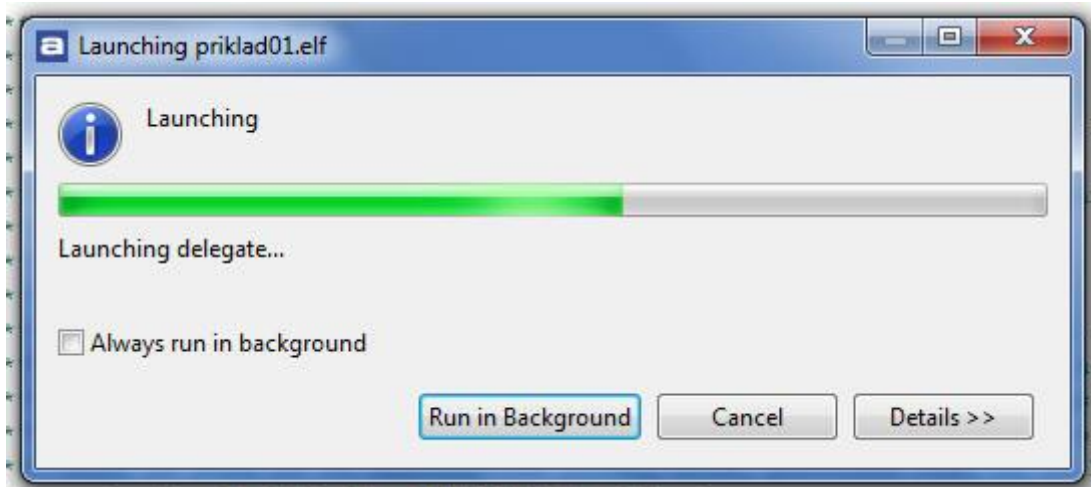


dostaneme

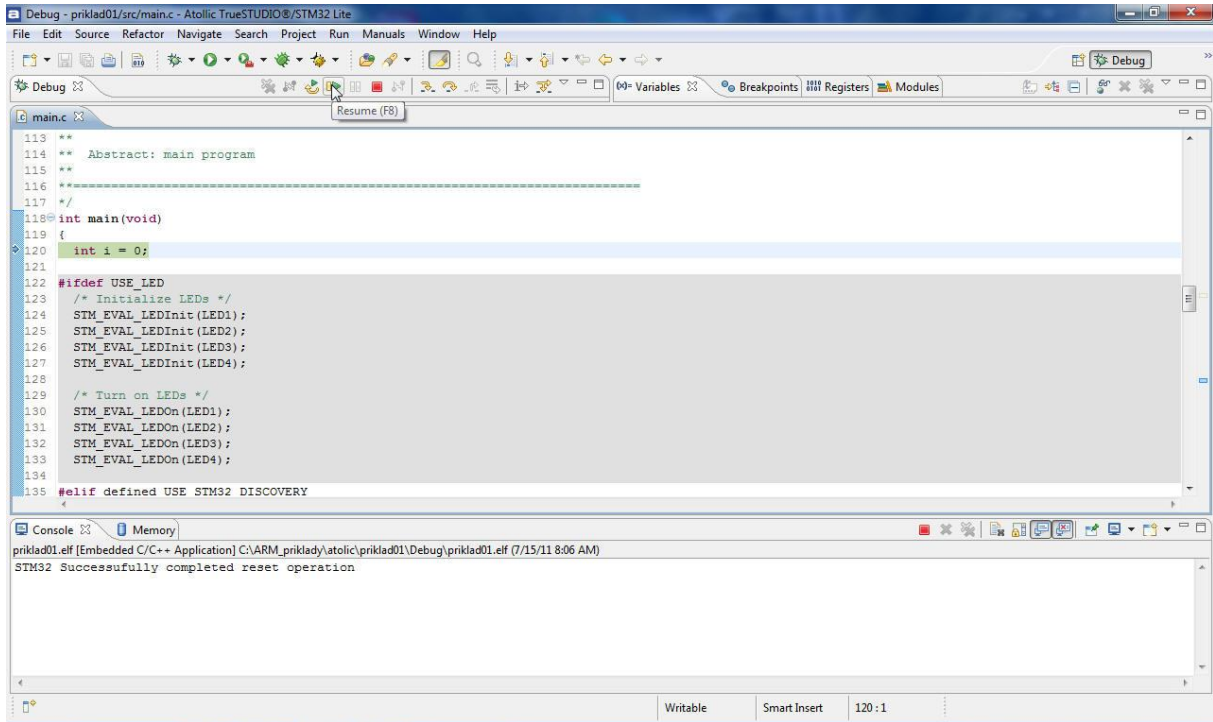




GDB Server musí být nastaven na SWD. Dále musíme na firewalu Windowsu povolit příslušný port - v mém případě 61234. Poté klikneme **OK**. Objeví se



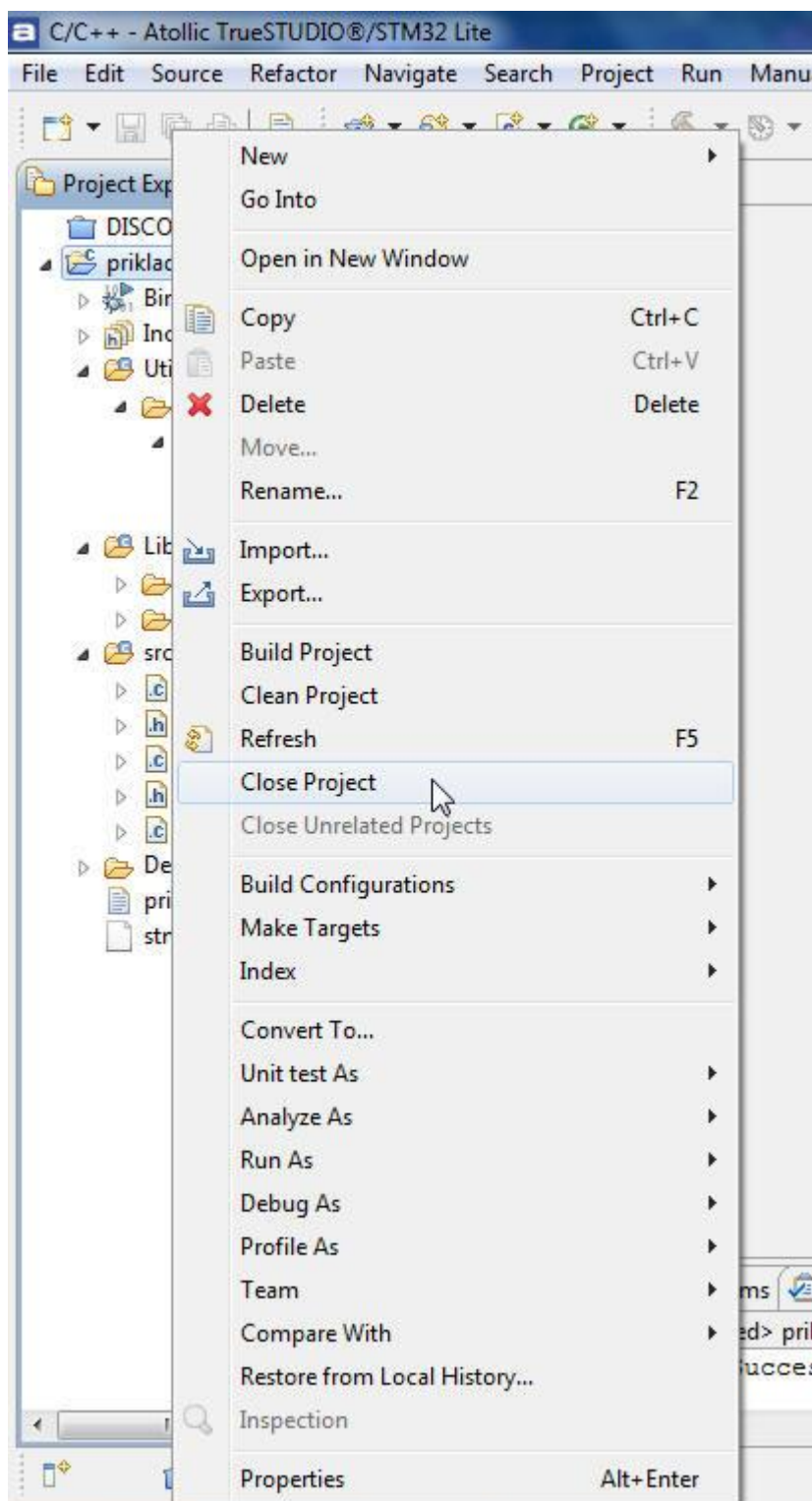
A poté již



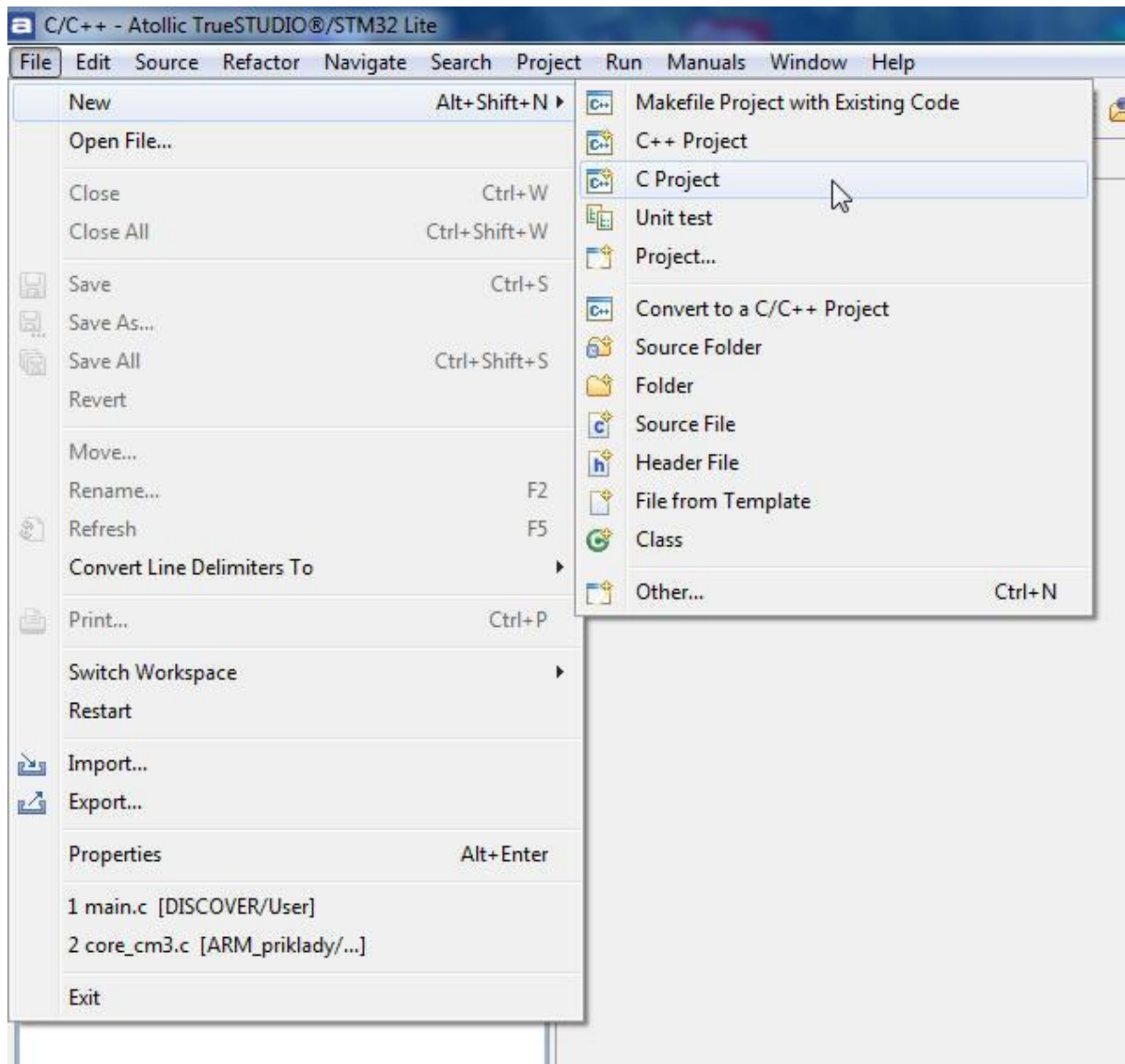
Klikneme na ikonku Resume (F8) . Tím se spustí program v startkitu. Svítí zelená LED. Po stisknutí tlačítka USB zhasne a rozsvítí se modrá LED.

4.4 Druhý vlastní program v TrueStudio

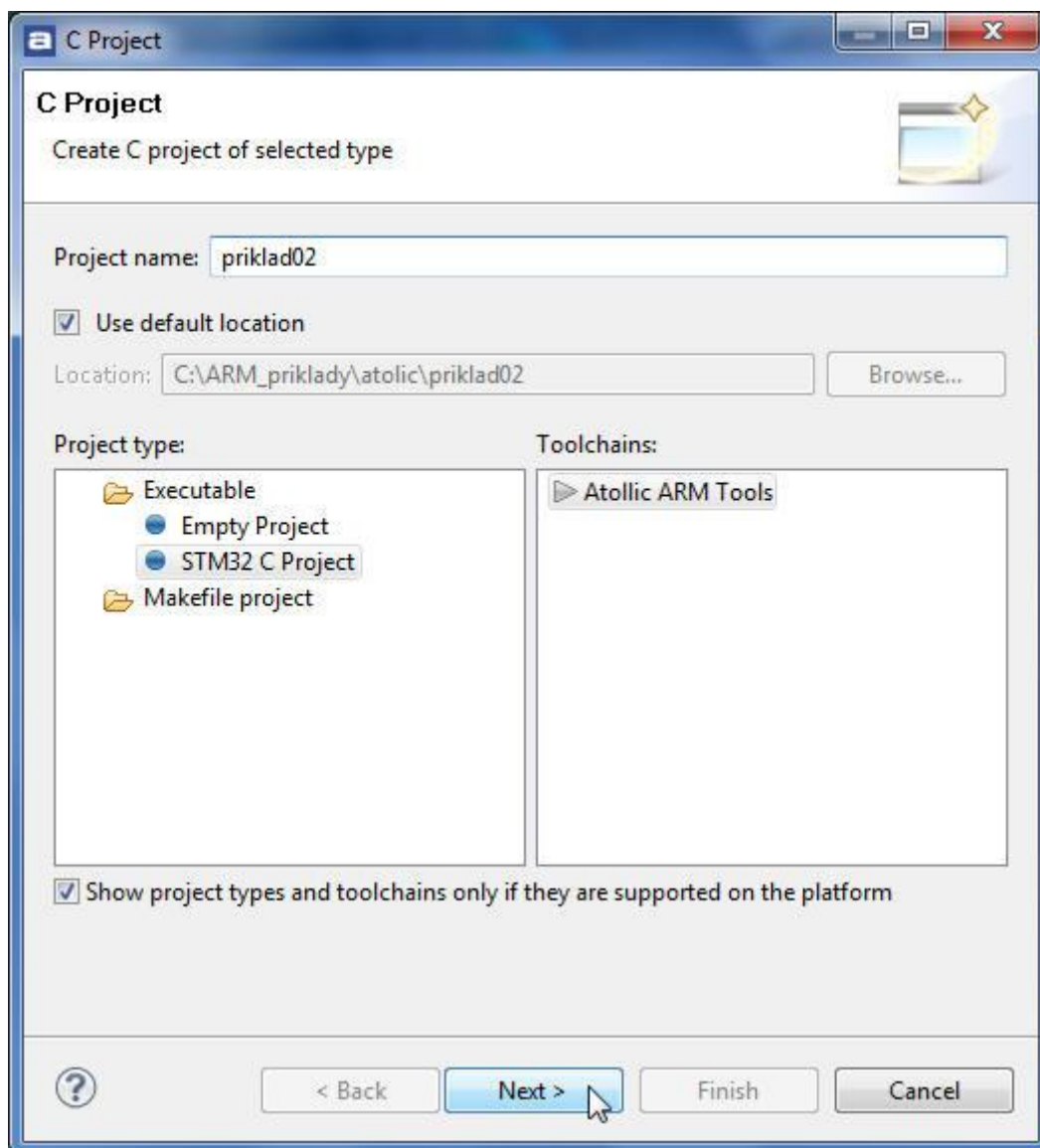
V předchozím, prvním, „vlastním“ programu jsme vlastně jen použili wizard **TrueStudio**, který vytvořil i zdrojový kód `main.c`, takže jsme žádný kód sami nepsali. Proto nyní znovu a lépe. Zavríme `příklad01`.



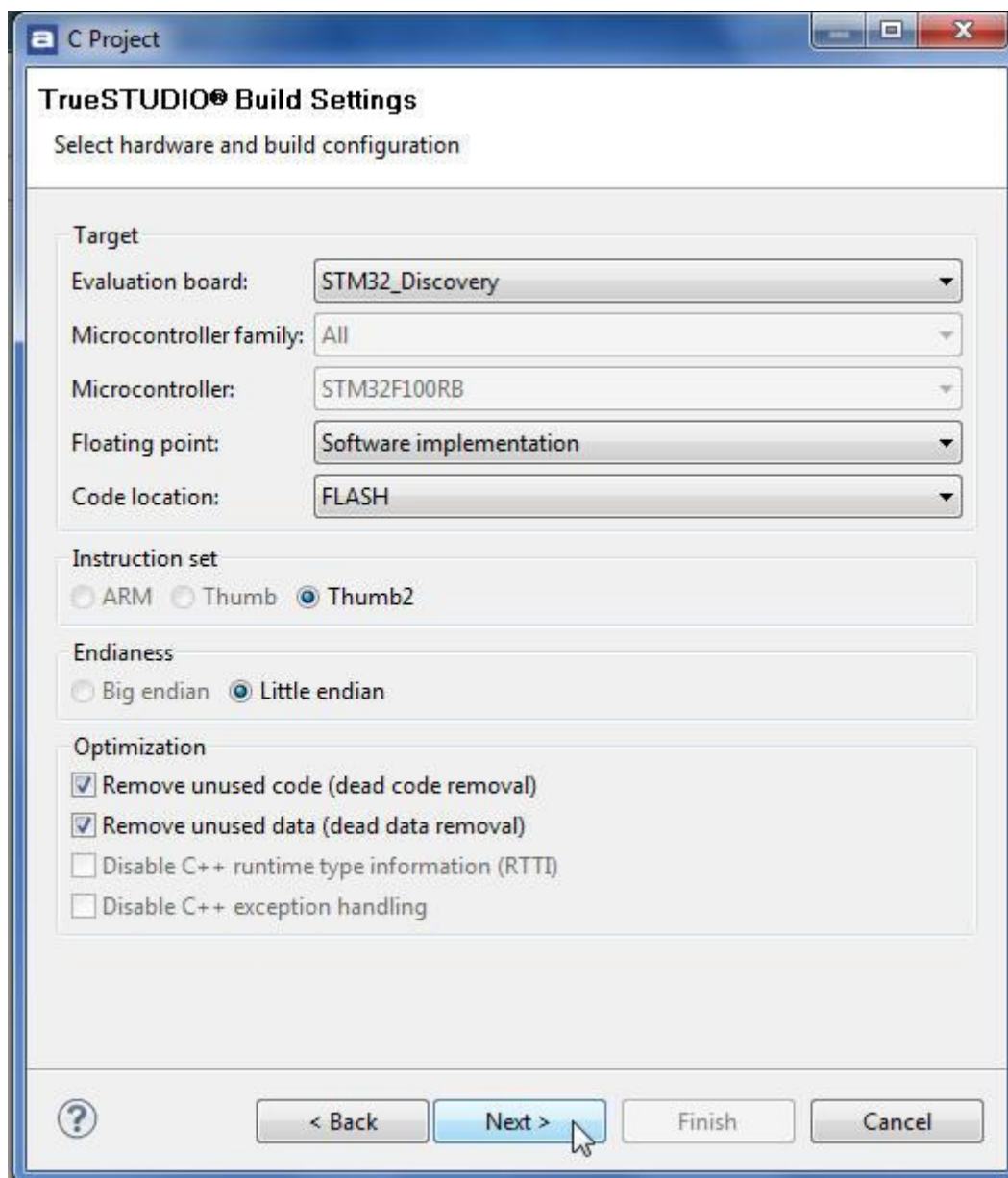
Nyní založíme druhý projekt. Z menu vybereme **File**, pak **New** a nakonec **C Project**.



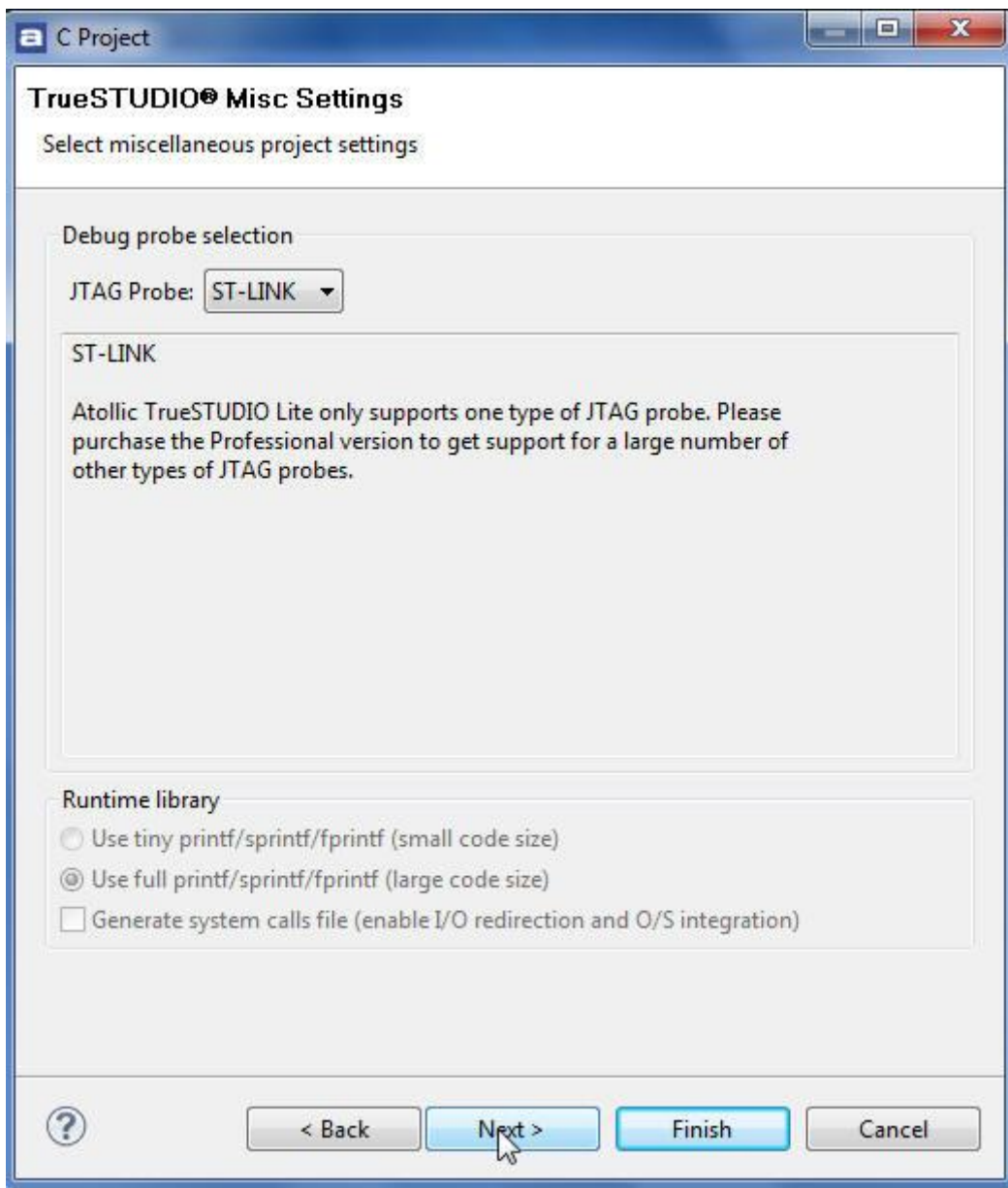
Objeví se okno **C project**



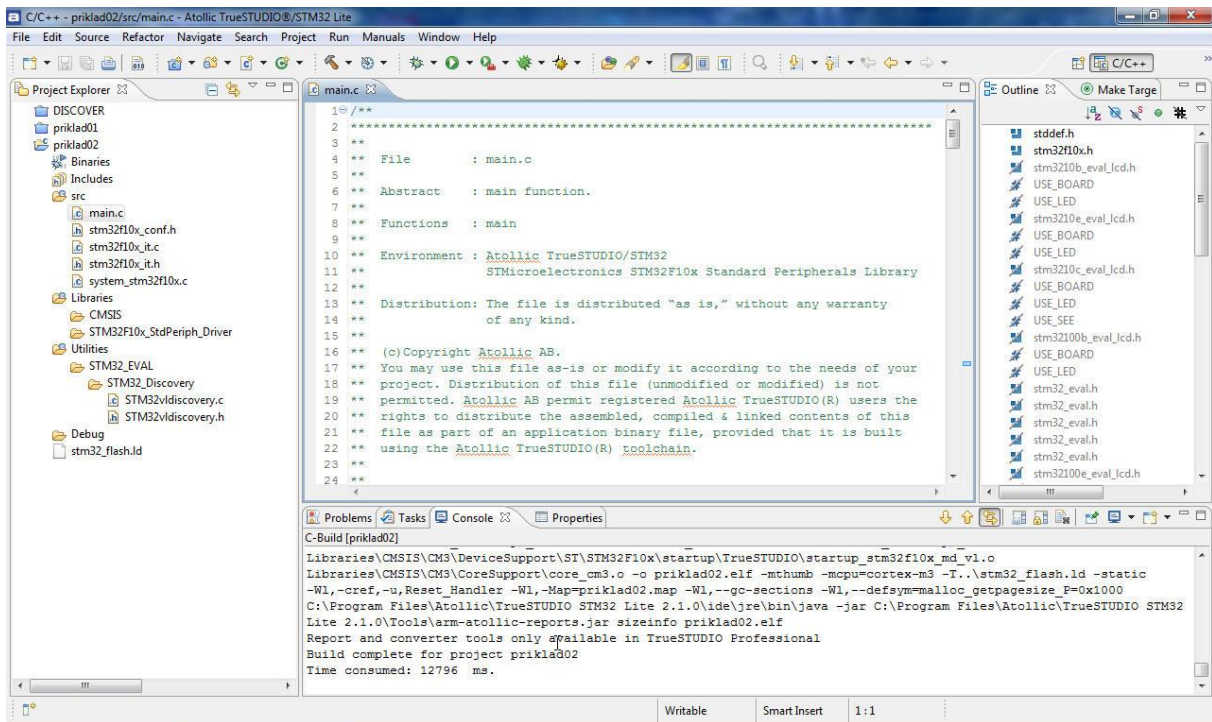
Vyplníme a klikneme na **Next**



Nastavíme dle obr. A opět **Next**



ST-Link a Next



Obsah `main.c` vymažeme a nahradíme novým kódem.

```

/* prikklad02 ----- */
#include <stddef.h>
#include "stm32f10x.h"
#include "STM32vldiscovery.h"

void GPIO_Inicializace(void); /* nastaveni vstupne/vystupnich pinu na
kitu*/
void Delay(__IO uint32_t nTick);

int main(void)
{
    GPIO_Inicializace();
    while (1)
    {
        STM32vldiscovery_LEDOff(LED4); // vypnem LED4 - modra
        Delay(0x55555);
        STM32vldiscovery_LEDToggle(LED4); // zapnem LED4 - modra
        Delay(0x55555);

        if(0 != STM32vldiscovery_PBGetState(BUTTON_USER))
        {
            // tj. když je stisknuto tlacitko
            STM32vldiscovery_LEDToggle(LED3); // zapnem LED3 - zelena
        }

        else
    }
}

```

```

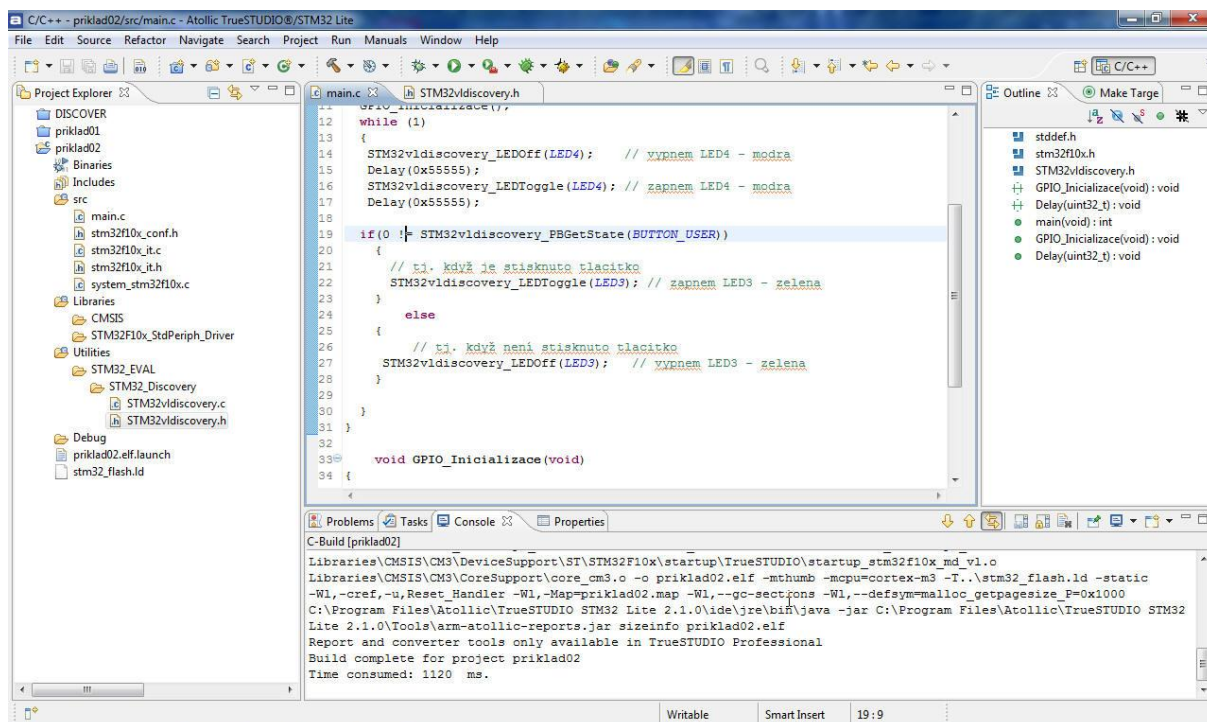
{
    // tj. když není stisknuto tlačítko
    STM32vldiscovery_LEDOff(LED3); // vypnem LED3 - zelena
}

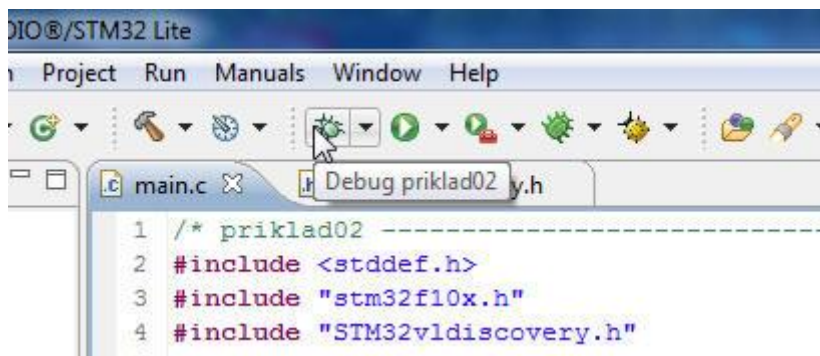
}

void GPIO_Inicializace(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);
}

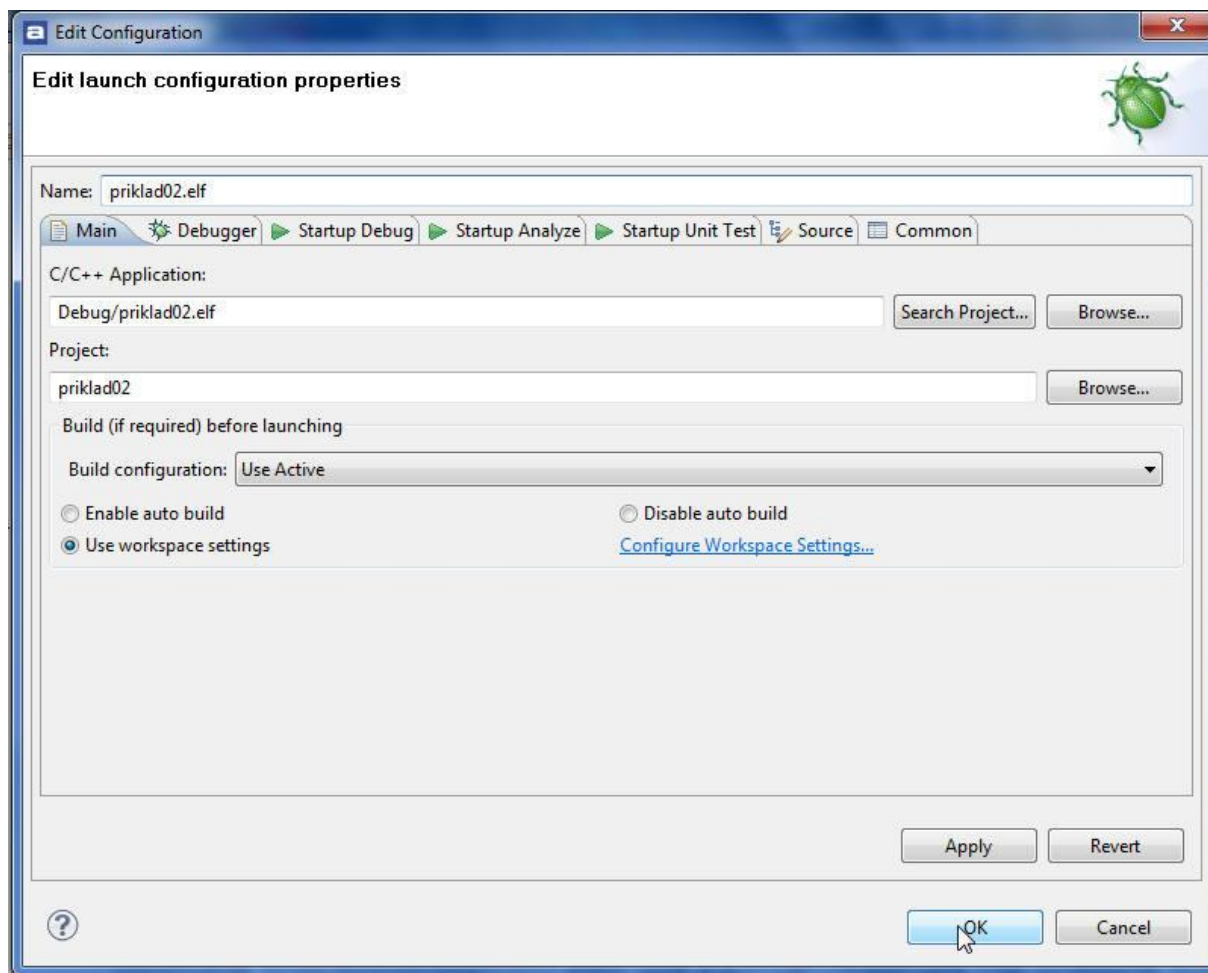
void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

```

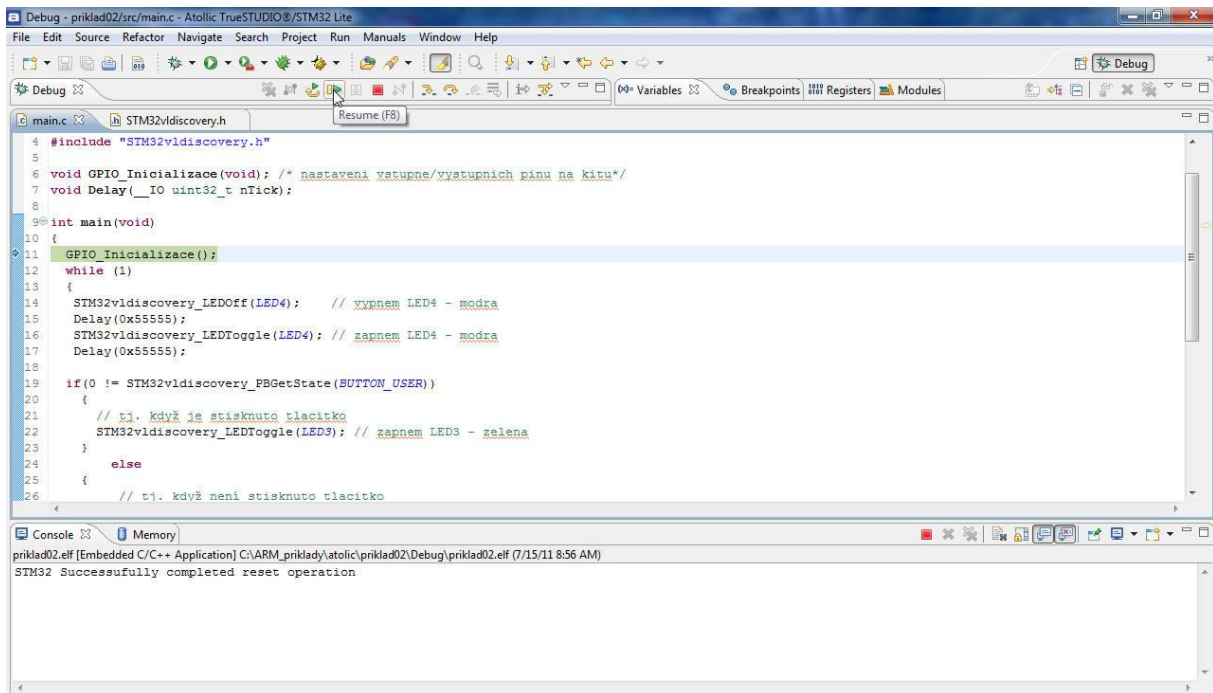




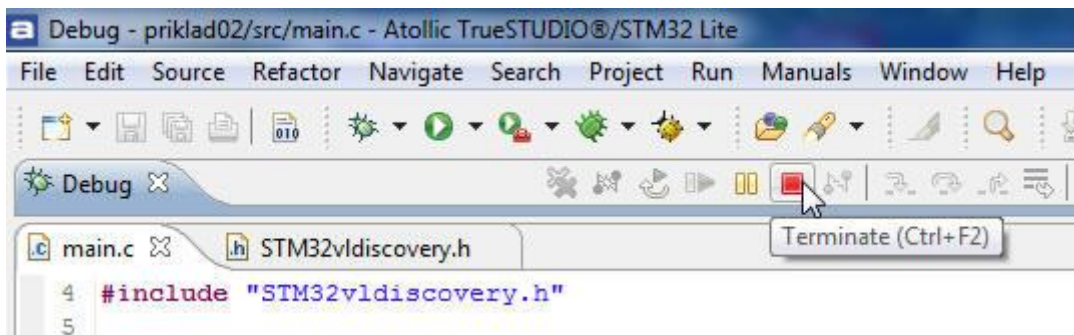
Spustíme **Debug** (ikonka zeleného brouka) , máme



Klikneme na **OK**



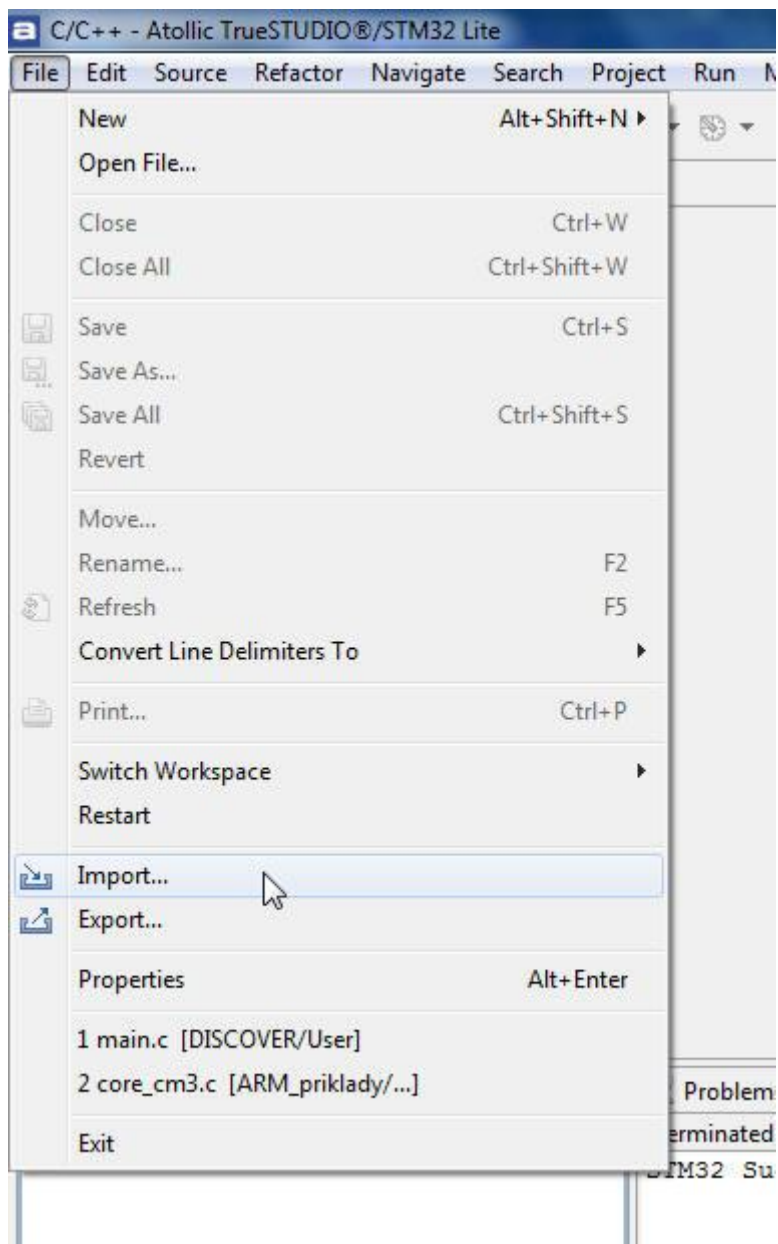
Klikneme na Resume (F8)

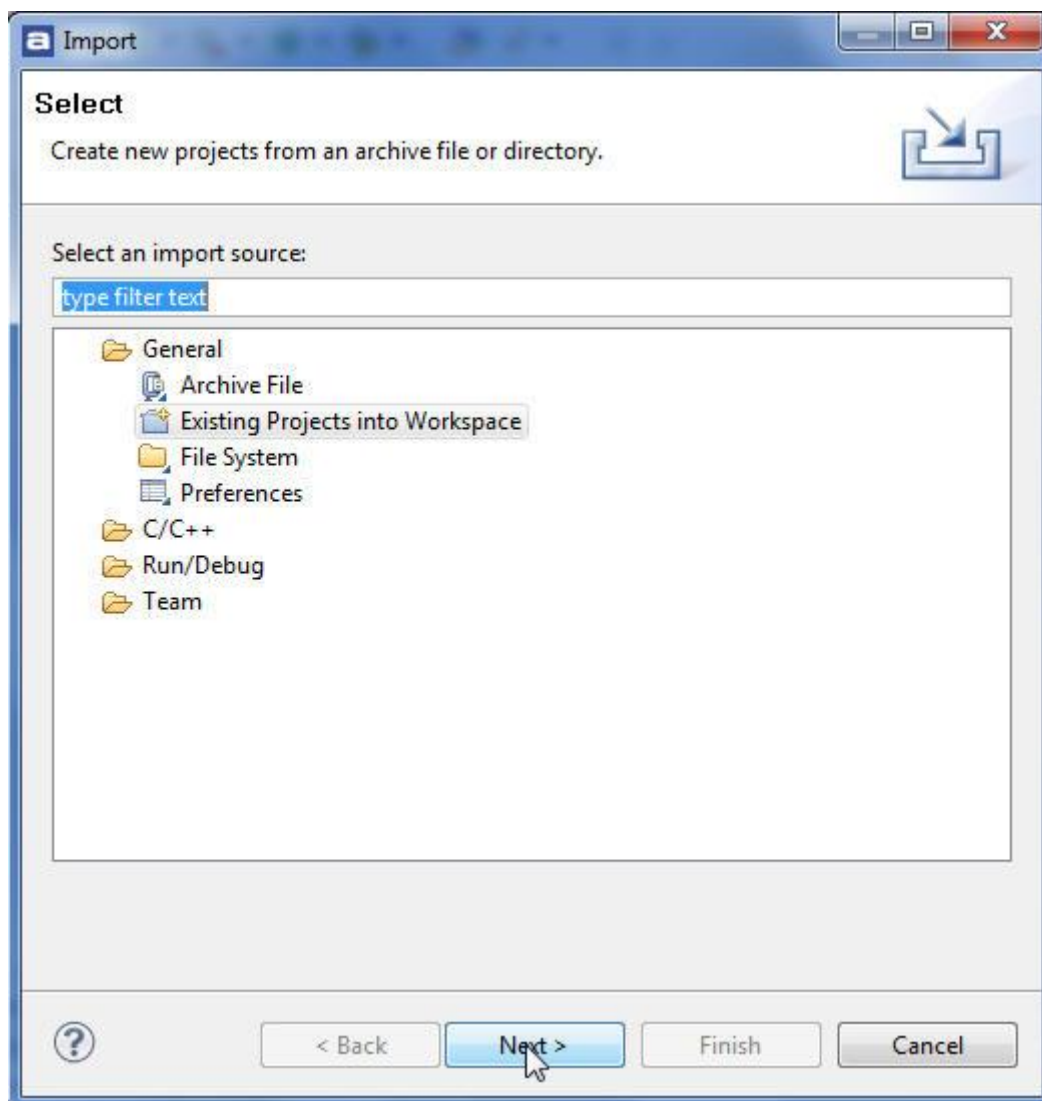


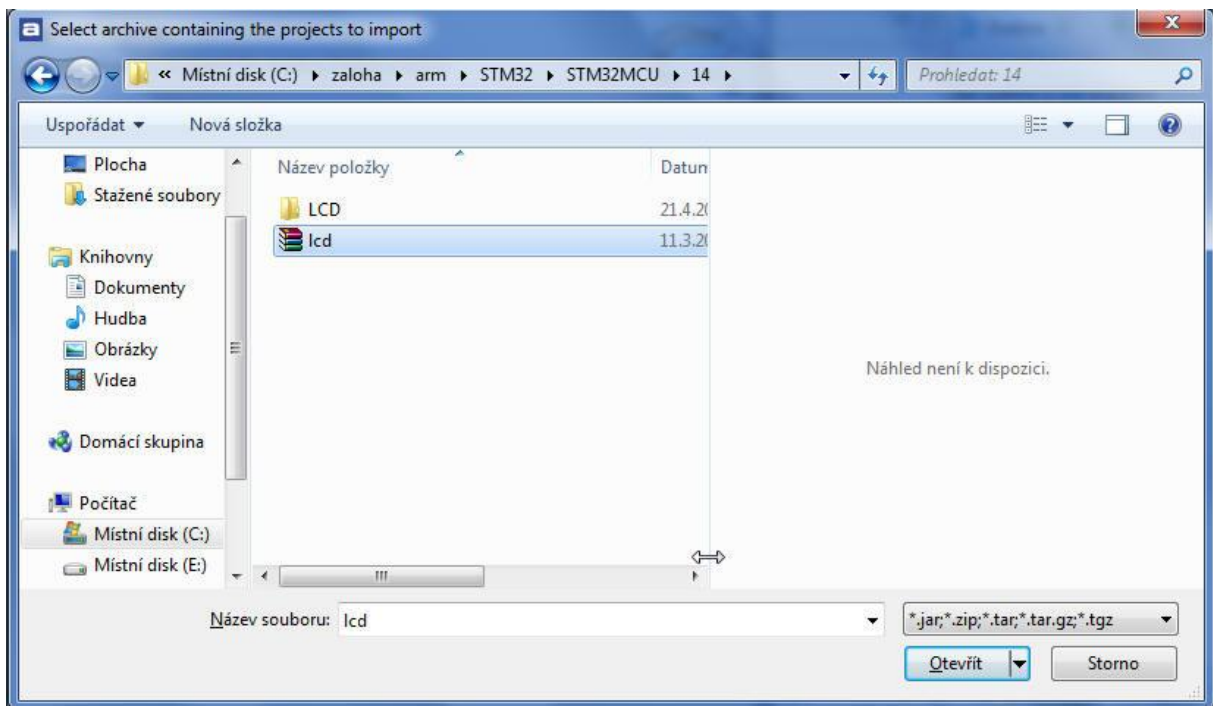
Program nyní běží – bliká modrá dioda. Pokud stisknu a držím tlačítko USB bude blikat i zelená dioda. Protože **Atollic TrueSTUDIO** je využíván v seriálu na **MCU serveru**, můžeme další programy převzít z tohoto seriálu a v těchto skriptech se jim již nebudeme příliš zabývat. Jen si ještě ukážeme, jak importovat do TrueSTUDIA projekty v souborech zip. Předtím ovšem nezapomeneme zavřít projekt příklad02.

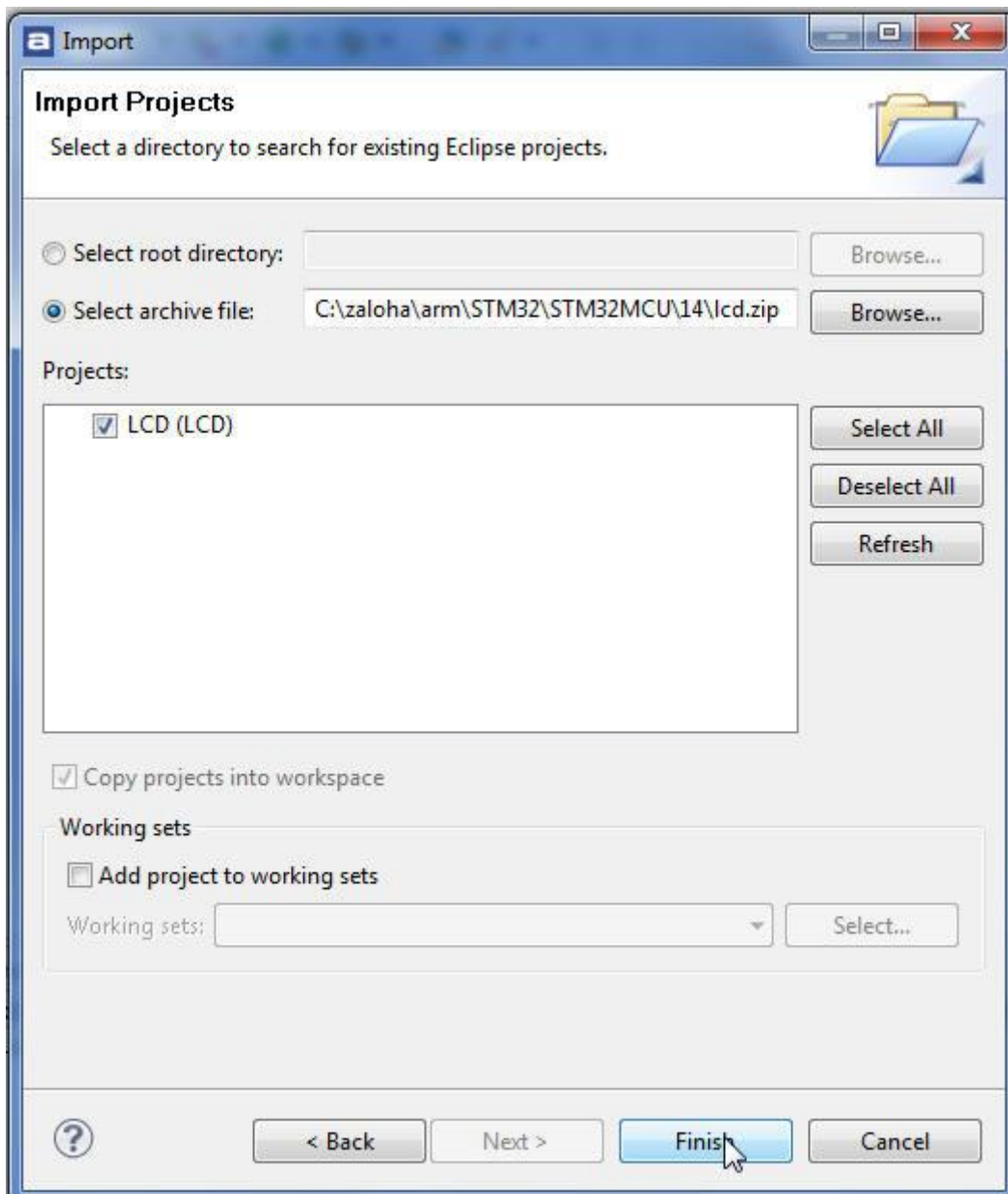
4.5 Import programu v TrueStudio

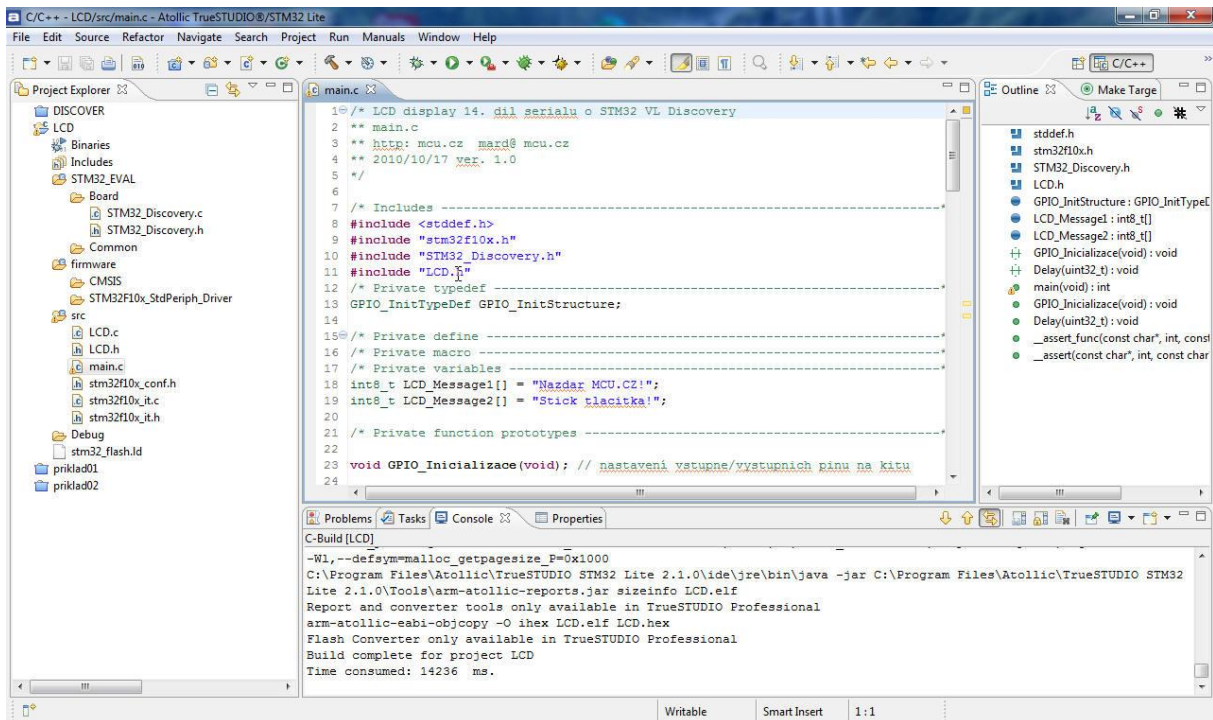
Použijeme **LCD.zip** ze 14 dílu seriálu z MCU serveru.











Projekt se přeložil. Poté jsem v Project Explorer rozvinul LCD a otevřel **main.c**.

Všimněme si, že v `STM32_EVAL` najdeme `STM32_Discovery`, kdežto pomocí wizardu jsme dostali `STM32vldiscovery`. To musíme mít na zřeteli, pokud budeme např. používat wizard a dále některé soubory z MCU seriálu a upravit příslušná místa v kódu

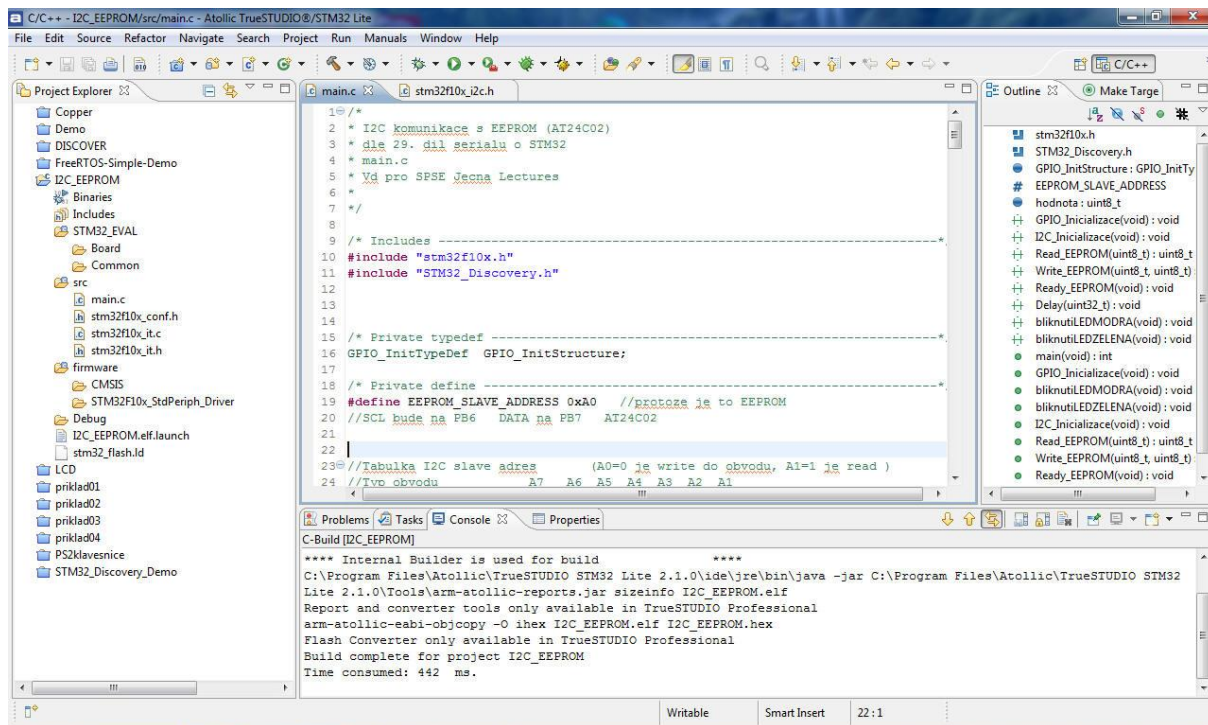
Pozn.

STM32VL Discovery má i/o signály na 3V logice, což může způsobit problémy při spolupráci s TTL obvody. U LCD řadiče HD44780 dále záleží na nastavení správných velikostí prodlev. To mohou být důvody, proč se mi nepodařilo zprovoznit LCD displej s **LCD.c** a **LCD.h** z MCU serveru. Proto, podobně jako u uVision4 Keil a IAR WorkBench používám tyto soubory ve verzi z Katedry měření ČVUT.

4.6 ukázka programu v TrueStudio s I2C

Jde o program z 29.dílu seriálu na MCU s tím, že program jsme doplnili o dvě funkce provádějící bliknutí zelené či modré LED a tyto funkce jsme umístili na několik míst v kódu `main.c`, čímž si při běhu programu ověříme, že provedl postupně všechny funkce a nikde se „nezasekl“. Úkolem programu je prostřednictvím I2C naprogramovat EEPROM AT24C02. Při komunikaci I2C provádí komunikující obvod (slave) potvrzování přijatých dat signálem Ack, STM32VL Discovery je v komunikaci I2C master. V jeho programu v `main.c` můžeme vidět několik smyček, čekajících právě na potvrzení Ack. **Není** zde např. naprogramováno, aby po určité době, do které nedojde potvrzení Ack program např. na LCD vypsál, že došlo k chybě.

Program je napsán tak, že v čekací smyčce je tak dlouho, dokud nedojde potvrzení Ack. Pokud nedojde, např. proto, že k STN32VL Discovery nemáme připojenou EEPROM, nebo sice máme, ale bez Pull Up odporů 4k7, tak program bude v nekonečné smyčce, Dojde k jeho uvíznutí (tzv. „zaseknutí“).



```

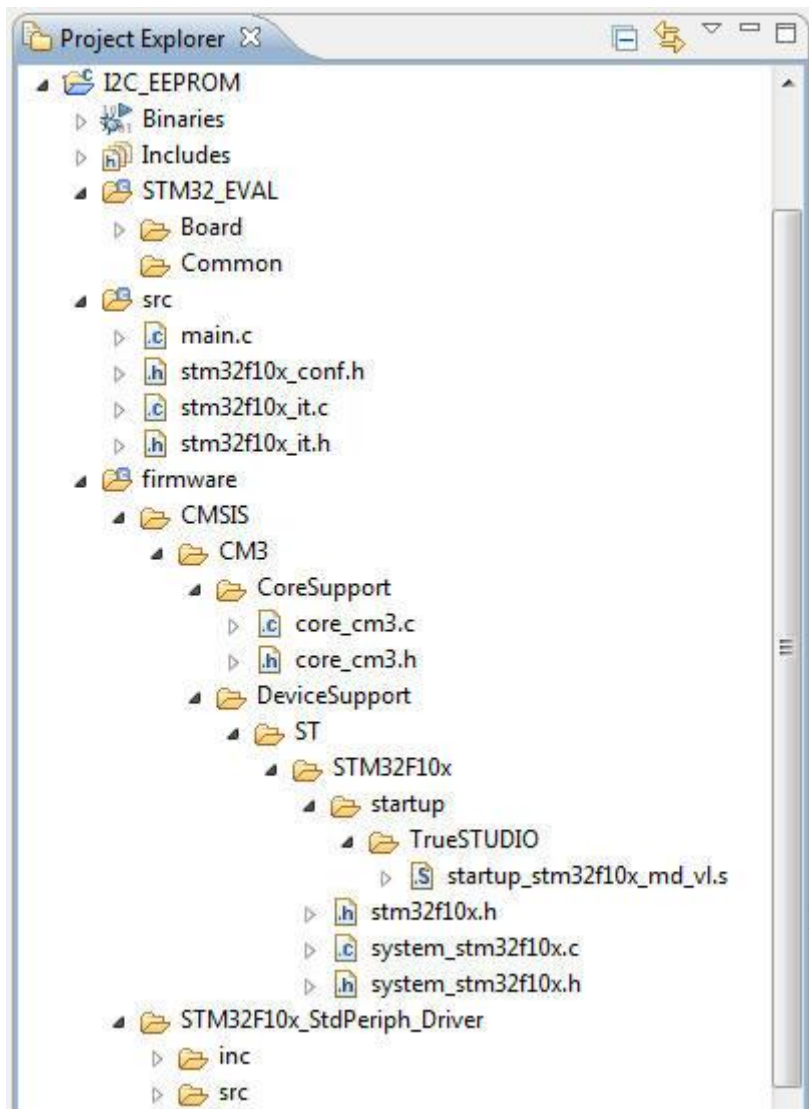
1  /*
2  * I2C komunikace s EEPROM (AT24C02)
3  * dle 29. díl seriálu o STM32
4  * main.c
5  * Vd pro SPSE Jecna Lectures
6  */
7  /*
8  */
9  /* Includes -----*/
10 #include "stm32f10x.h"
11 #include "STM32_Discovery.h"
12
13
14
15 /* Private typedef -----*/
16 GPIO_InitTypeDef  GPIO_InitStructure;
17
18 /* Private define -----*/
19 #define EEPROM_SLAVE_ADDRESS 0xA0 //protože je to EEPROM
20 //SCL bude na PB6  DATA na PB7  AT24C02
21
22
23 //Tabulka I2C slave adres (A0=0 je write do obvodu, A1=1 je read )
24 //Tvo obvodu      A7  A6  A5  A4  A3  A2  A1

```

```

C-Build [I2C_EEPROM]
**** Internal Builder is used for build ****
C:\Program Files\Atollic\TrueSTUDIO STM32 Lite 2.1.0\ide\jre\bin\java -jar C:\Program Files\Atollic\TrueSTUDIO STM32
Lite 2.1.0\Tools\arm-atollic-reports.jar sizeinfo I2C_EEPROM.elf
Report and converter tools only available in TrueSTUDIO Professional
arm-atollic-eabi-objcopy -O ihex I2C_EEPROM.elf I2C_EEPROM.hex
Flash Converter only available in TrueSTUDIO Professional
Build complete for project I2C_EEPROM
Time consumed: 442 ms.

```



```

/*
 * I2C komunikace s EEPROM (AT24C02)
 * dle 29. dil serialu o STM32
 * main.c
 * Vd pro SPSE Jecna Lectures
 *
 */

/* Includes -----*/
#include "stm32f10x.h"
#include "STM32_Discovery.h"

/* Private typedef -----*/

```

```

GPIO_InitTypeDef  GPIO_InitStructure;

/* Private define -----*/
#define EEPROM_SLAVE_ADDRESS 0xA0 //protoze je to EEPROM
//SCL bude na PB6  DATA na PB7  AT24C02

//Tabulka I2C slave adres      (A0=0 je write do obvodu, A1=1 je read )
//Typ obvodu                    A7      A6      A5      A4      A3      A2      A1
//8bitovy vstup/vystup          0      1      0      0      a      a      a
//napr. PCF8574
//EEPROM                        1      0      1      0      a      a      a
//napr. PCF8582
//Hodiny/kalendar              1      0      1      0      0      0      a
//napr. PCF8583
//8bitovy A/D a D/A            1      0      0      1      a      a      a
//napr. PCF8591
//PLL                           1      1      0      0      0      a      a
//napr. TSA5055

/* Private variables -----*/
uint8_t hodnota=0;

/* Private function prototypes -----*/

void GPIO_Inicializace(void);           // nastavení vstupne/vystupnich
pinu na kitu
void I2C_Inicializace(void);           // nastavení I2C
uint8_t Read_EEPROM(uint8_t adresa);   // cteni
void Write_EEPROM(uint8_t adresa, uint8_t data); // zapis
void Ready_EEPROM(void);               // polling potvrzeni
void Delay(__IO uint32_t nTick);       // cekani

void bliknutiLEDMODRA(void);           //
void bliknutiLEDZELENA(void);         //

int main(void)
{

  GPIO_Inicializace();
  I2C_Inicializace();

  // blikneme modrou jako priznak startu
  bliknutiLEDMODRA();

  /* pouzita EEPROM AT24C02 - 256 bajtu x 8 bitu */

  hodnota = Read_EEPROM(0x00);
  hodnota = Read_EEPROM(0x01);
  hodnota = Read_EEPROM(0x02);
  hodnota = Read_EEPROM(0x03);
  hodnota = Read_EEPROM(0x04);

```

```

Write_EEPROM(0x00, 0x01);
Ready_EEPROM();
bliknutiLEDZELENA();
Write_EEPROM(0x01, 0x12);
Ready_EEPROM();
bliknutiLEDMODRA();
Write_EEPROM(0x02, 0x34);
Ready_EEPROM();
bliknutiLEDZELENA();
Write_EEPROM(0x03, 0x56);
Ready_EEPROM();
bliknutiLEDMODRA();
hodnota = Read_EEPROM(0x00);
hodnota = Read_EEPROM(0x01);
hodnota = Read_EEPROM(0x02);
hodnota = Read_EEPROM(0x03);
hodnota = Read_EEPROM(0x04);

// blikneme zelenou jako priznak konce
bliknutiLEDZELENA();

while (1)
{
    bliknutiLEDMODRA();
    // tady uz jenom blikame modrou LED
}

void GPIO_Inicializace(void)
{
    STM32_Discovery_LEDInit(LED3);
    STM32_Discovery_LEDInit(LED4);
    STM32_Discovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32_Discovery_LEDOff(LED3);
    STM32_Discovery_LEDOff(LED4);
}

void bliknutiLEDMODRA(void)
{
    STM32_Discovery_LEDOn(LED4); // zapnem LED4 - modra
    Delay(0x5FFFF);
    STM32_Discovery_LEDToggle(LED4); // vypnem LED4 - modra
    Delay(0x5FFFF);
}

void bliknutiLEDZELENA(void)
{
    STM32_Discovery_LEDOn(LED3); // zapnem LED3 - zelena
    Delay(0x5FFFF);
    STM32_Discovery_LEDToggle(LED3); // vypnem LED3 - zelena
    Delay(0x5FFFF);
}

```



```

}

void I2C_Inicializace(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    I2C_InitTypeDef  I2C_InitStructure;
    I2C_DeInit(I2C1);

    /* I2C Periph clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* GPIO Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    /* Configure I2C pins: SCL and SDA */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD; //GPIO_Mode_AF_PP
    GPIO_Init(GPIOB, &GPIO_InitStructure); //I2C EEPROM pripojime k PB
    //SCL bude na PB6  DATA na PB7  AT24C02

    /* I2C configuration */
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = 0xA0; //EEPROM
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = 10000;

    /* I2C Peripheral Enable */
    I2C_Cmd(I2C1, ENABLE);
    /* Apply I2C configuration after enabling it */
    I2C_Init(I2C1, &I2C_InitStructure);
}

uint8_t  Read_EEPROM(uint8_t adresa)
{
    uint8_t Data;
    Data = 0;
    /* While the bus is busy */
    while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

    /* Send START condition */
    I2C_GenerateSTART(I2C1, ENABLE);

    /* Test on EV5 and clear it */
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

    /* Send EEPROM address for read */
    I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
    I2C_Direction_Transmitter);

    /* Test on EV6 and clear it */

```

```

while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

/* Clear EV6 by setting again the PE bit */
I2C_Cmd(I2C1, ENABLE);

/* Send the EEPROM's internal address to read from: MSB of the address
first */
I2C_SendData(I2C1, adresa);

/* Test on EV8 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

/* Send START condition a second time */
I2C_GenerateSTART(I2C1, ENABLE);

/* Test on EV5 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

/* Send EEPROM address for read */
I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Receiver);

/* Test on EV6 and clear it */
while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

/* Disable Acknowledgement */
I2C_AcknowledgeConfig(I2C1, DISABLE);

/* Send STOP Condition */
I2C_GenerateSTOP(I2C1, ENABLE);
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)){}

/* Read a byte from the EEPROM */
Data = I2C_ReceiveData(I2C1);

/* Enable Acknowledgement to be ready for another reception */
I2C_AcknowledgeConfig(I2C1, ENABLE);

return(Data);
}

void Write_EEPROM(uint8_t adresa, uint8_t data)
{
/* Send START condition */
I2C_GenerateSTART(I2C1, ENABLE);

/* Test on EV5 and clear it */
while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

/* Send EEPROM address for write */
I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);

```

```

        /* Test on EV6 and clear it */
        while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        /* Send the EEPROM's internal address to write to : MSB of the
address first */
        I2C_SendData(I2C1, adresa);

        /* Test on EV8 and clear it */
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        /* Send the byte to be written */
        I2C_SendData(I2C1, data);

        /* Test on EV8 and clear it */
        while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        /* Send STOP condition */
        I2C_GenerateSTOP(I2C1, ENABLE);
    }

void Ready_EEPROM(void)
{
    __IO uint16_t temp = 0;

    do
    {
        /* Send START condition */
        I2C_GenerateSTART(I2C1, ENABLE);

        /* Read I2C_EE SR1 register to clear pending flags */
        temp = I2C_ReadRegister(I2C1, I2C_Register_SR1);

        /* Send EEPROM address */
        I2C_Send7bitAddress(I2C1, EEPROM_SLAVE_ADDRESS,
I2C_Direction_Transmitter);

        }while(!(I2C_ReadRegister(I2C1, I2C_Register_SR1) & 0x0002));

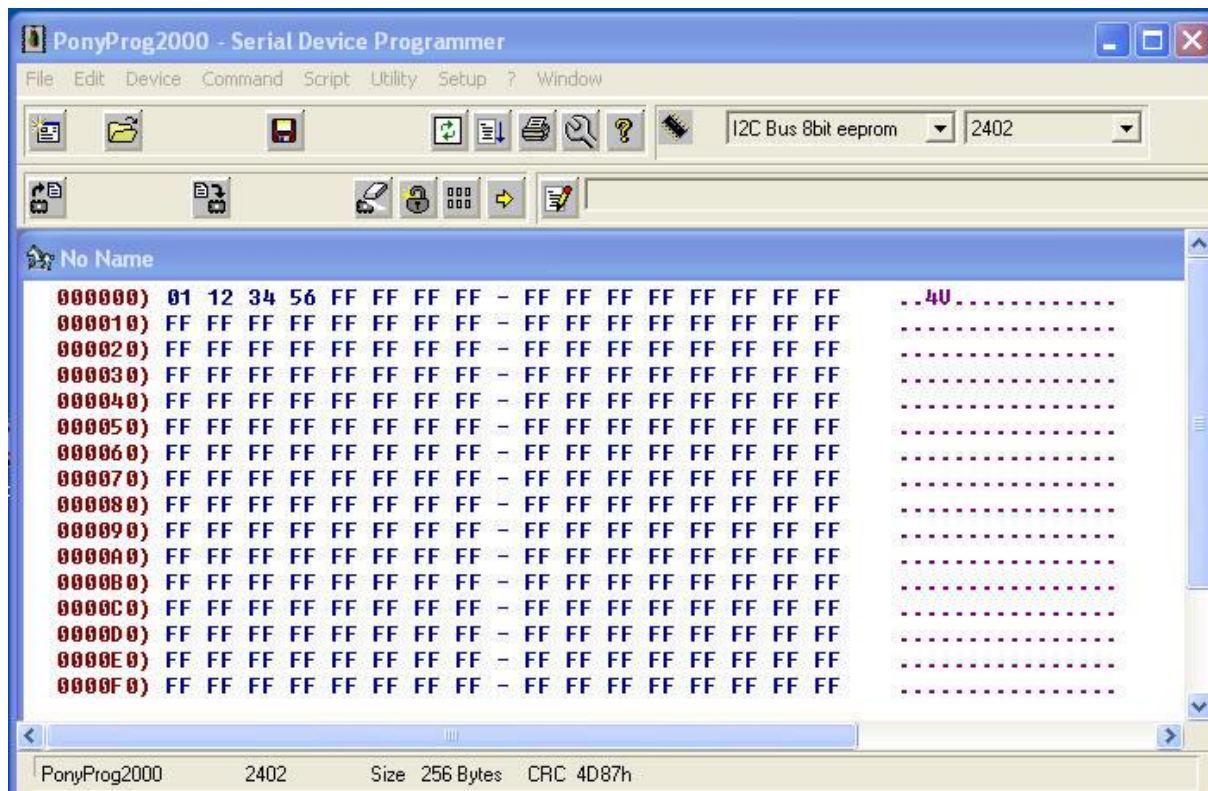
        /* Clear AF flag */
        I2C_ClearFlag(I2C1, I2C_FLAG_AF);

        /* STOP condition */
        I2C_GenerateSTOP(I2C1, ENABLE);
    }

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--)
    { ; }
}

```

Obsah naprogramované EEPROM můžeme např. zkontrolovat pomocí free programu PonyProg2000



Vidíme, že na první čtyři adresy se skutečně zapsalo 01, 12, 34, 56. Další programy pro TrueStudio najdeme v již zmiňovaném seriálu na MCU serveru.

5 Pokročilé programování, další projekty

5.1 RTOS (podle AT&P journal)

Operační systémy pro řízení v reálném čase RTOS (Real Time Operating System) lze rozdělit do několika skupin podle aplikační oblasti, uplatnění ve vestavěných systémech a využití standardních operačních systémů osobních počítačů.

Jako **první skupinu** lze uvést RT operační systémy s obecnějším uplatněním. Zde především patří tradiční oblast řízení technologických procesů, kde se jedná často o kritické úlohy reálného času, ale také o informační systémy pracující v reálném čase, zde se jedná i o nekritické systémy reálného času.

Druhou skupinu operačních systémů pro řízení jsou RTOS pro vestavěné (Embedded Systems) nebo jednodeskové systémy. U těchto aplikací lze konstatovat, že se jedná o nejrychlejší nárůst použití RTOS ze všech aplikačních oblastí. Aplikační oblasti použití vestavěných řídicích systémů s mikropočítači se neustále rozšiřují a v současné době je lze nalézt ve všech oblastech lidské činnosti. Nejrozšířenějšími aplikacemi jsou systémy v automobilní a dopravní technice. Mezi tyto RTOS patří i FreeRTOS, který budeme dále používat.

Třetí skupinu RT operačních systémů tvoří ty, které jsou orientovány pouze na skupinu procesorů Intel jako u PC, ale jsou použity pro vestavěné PC systémy.

Čtvrtou skupinu RTOS tvoří doplňující moduly jádra, které umožňují operačním systémům používaných u osobních počítačů PC, které tvoří nejrozšířenější skupinu výpočetní techniky, řešení úloh reálného času, často to však nejsou kritické úlohy reálného času.

5.2 FreeRTOS

Při přechodu z 8 nebo 16-bitových mikrokontrolérů hledají specialisté rychlejší, jednodušší a chybám méně náchylnou cestu vývoje, takže použití RTOS je důležitým krokem k využití plného výkonu mikrokontroléru při dodržení maximálního zjednodušení kódování.

RTOS je určen pro embedded aplikace v reálném čase, kde je efektivita provádění a kompaktnost kódu důležitými požadavky. RTOS se obvykle vyznačuje vysokou přenositelností, kompaktní velikostí kódu a hlavně, jeho architektura bývá optimalizovaná pro extrémně efektivní přepínání kontextu.

Obvyklé funkce RTOS:

- Efektivní a přenositelné preemptivní jádro.
- Statická architektura, vše je staticky přidělené již v době kompilace.
- Dynamické rozšíření, dynamické objekty jsou podporovány volitelnou vrstvou postavenou nad statickým jádrem.
- Bohatá sada primitiv: vlákna, virtuální časovače, semaforey, mutexy, stavové proměnné, zprávy, mailboxy a flagy událostí.
- Podpora algoritmu prioritního dědictví pro mutexy.
- HAL komponenty podporující různé abstraktní ovladače zařízení: Port, Serial, ADC, CAN, I2C, MAC, MMC, PWM, SPI, UART.
- Podpora pro externí komponenty UIP, lwIP, FatFs.
- Mnoho podporovaných architektur.
- Rozsáhlé testovací sady s benchmarky

HAL

Hardware Abstraction Layer je důležitou součástí přenositelnosti kódu a jeho rychlou tvorbou. HAL

Chibios/RT obsahuje:

- Configuration
- ADC Driver
- CAN Driver
- MMC over SPI Driver
- PAL Driver
- PWM Driver
- Serial Driver
- SPI Driver
- UART Driver

Drivery

Kromě použití HAL (což je doporučeno) obsahuje RTOS často i řadu driverů pro STM32 mcu. Jedná se o následující sadu:

- STM32 Initialization Support

- STM32 ADC Support
- STM32 CAN Support
- STM32 DMA Support
- STM32 GPIO Support
- STM32 PWM Support
- STM32 SPI Support
- STM32 USART Support (buffered)
- STM32 USART Support (unbuffered)

Pro ARM existuje řada RTOS pro ARM, např. ChibiOS/RT, ArmExe RTOS, eCOS či FreeRTOS. Zvolil jsem FreeRTOS.

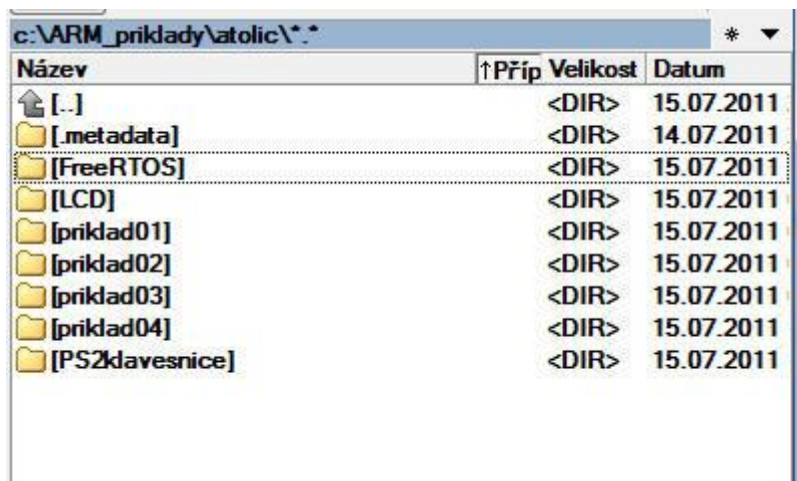
Pozn.

Použití RTOSu nám na jedné straně usnadní práci, na druhé straně je náročnější na prostředky MCU. Tyto prostředky sice máme, ale jsou využívány obvykle rozsáhlejším výsledným programem, než umožňují free education verze nástrojů uVision4 Keil a IAR Embedded Workbench. Toto omezení nemáme např. u Eclipse, včetně verzí na Eclipse postavených, jako je Atollic TrueSTUDIO. Proto pro FreeRTOS použijeme právě toto prostředí.

5.3 instalace FreeRTOS a demo program

Z <http://www.freertos.org/index.html> stáhneme FreeRTOSV7.0.1.exe , pak extrahuji adresář FreeRTOSV7.0.1 včetně obsahu.

Zkopíruji do workspace **atollic TrueStudio**

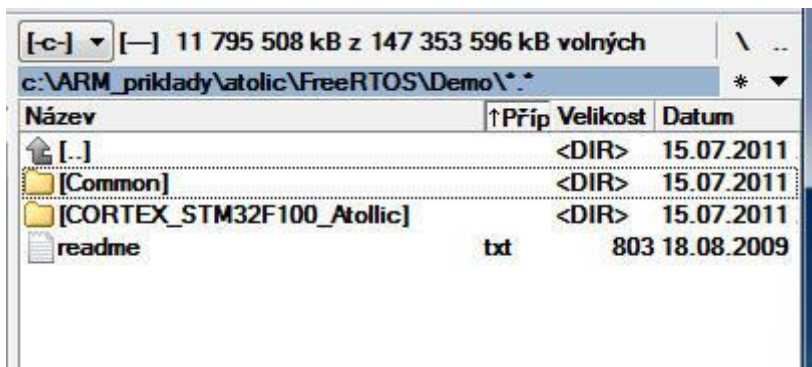


| Název | ↑ Příp | Velikost | Datum |
|------------|--------|----------|------------|
| [.] | | <DIR> | 15.07.2011 |
| [Demo] | | <DIR> | 15.07.2011 |
| [License] | | <DIR> | 15.07.2011 |
| [Source] | | <DIR> | 15.07.2011 |
| [TraceCon] | | <DIR> | 15.07.2011 |
| readme | | txt 873 | 18.08.2009 |

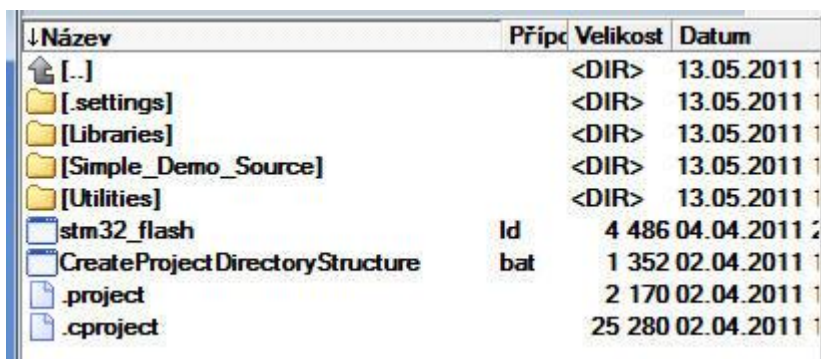
| Název | ↑ Příp | Velikost | Datum |
|-----------------------------------|--------|----------|----------|
| [.] | | <DIR> | 15.07.20 |
| [ARM7_AT91FR40008_GCC] | | <DIR> | 15.07.20 |
| [ARM7_AT91SAM7S64_IAR] | | <DIR> | 15.07.20 |
| [ARM7_AT91SAM7X256_Eclipse] | | <DIR> | 15.07.20 |
| [ARM7_LPC2106_GCC] | | <DIR> | 15.07.20 |
| [ARM7_LPC2129_IAR] | | <DIR> | 15.07.20 |
| [ARM7_LPC2129_Keil_RVDS] | | <DIR> | 15.07.20 |
| [ARM7_LPC2138_Rowley] | | <DIR> | 15.07.20 |
| [ARM7_LPC2368_Eclipse] | | <DIR> | 15.07.20 |
| [ARM7_LPC2368_Rowley] | | <DIR> | 15.07.20 |
| [ARM7_STR71x_IAR] | | <DIR> | 15.07.20 |
| [ARM7_STR75x_GCC] | | <DIR> | 15.07.20 |
| [ARM7_STR75x_IAR] | | <DIR> | 15.07.20 |
| [ARM9_AT91SAM9XE_IAR] | | <DIR> | 15.07.20 |
| [ARM9_STR91X_IAR] | | <DIR> | 15.07.20 |
| [AVR_ATMega323_IAR] | | <DIR> | 15.07.20 |
| [AVR_ATMega323_WinAVR] | | <DIR> | 15.07.20 |
| [AVR32_UC3] | | <DIR> | 15.07.20 |
| [ColdFire_MCF51CN128_CodeWarrior] | | <DIR> | 15.07.20 |
| [ColdFire_MCF52221_CodeWarrior] | | <DIR> | 15.07.20 |
| [ColdFire_MCF52233_Eclipse] | | <DIR> | 15.07.20 |
| [ColdFire_MCF52259_CodeWarrior] | | <DIR> | 15.07.20 |
| [ColdFire_MCF5282_Eclipse] | | <DIR> | 15.07.20 |
| [Common] | | <DIR> | 15.07.20 |

0 kB / 0 kB v 0 / 1 souborech a 0 / 101 složek

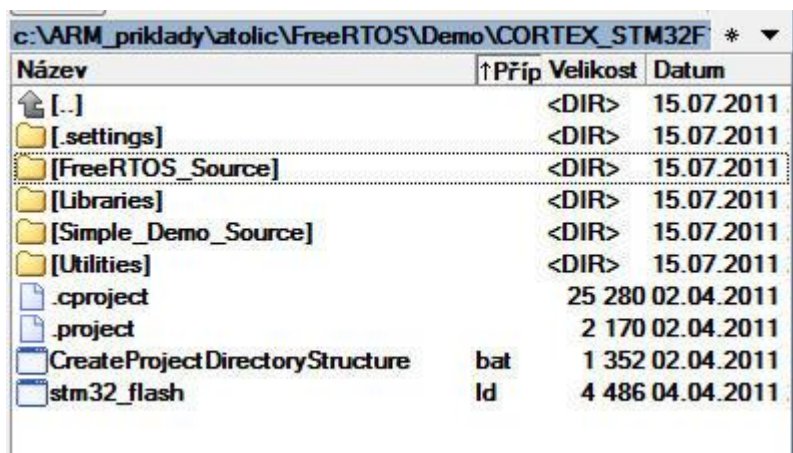
Vymažeme zbytečné podadresáře



Obsah adresáře CORTEX_STM32F100_Atollic je



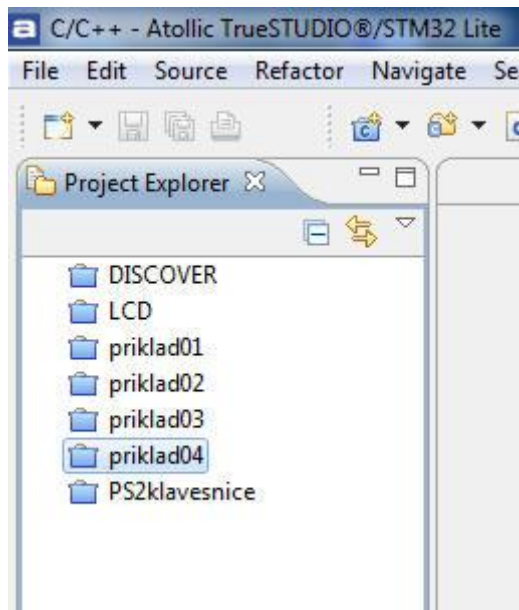
Předtím, než budeme pracovat s **TrueStudiem** spustíme **CreateProjectDirectoryStructure.bat**. Jeho název je plně vypovídající.



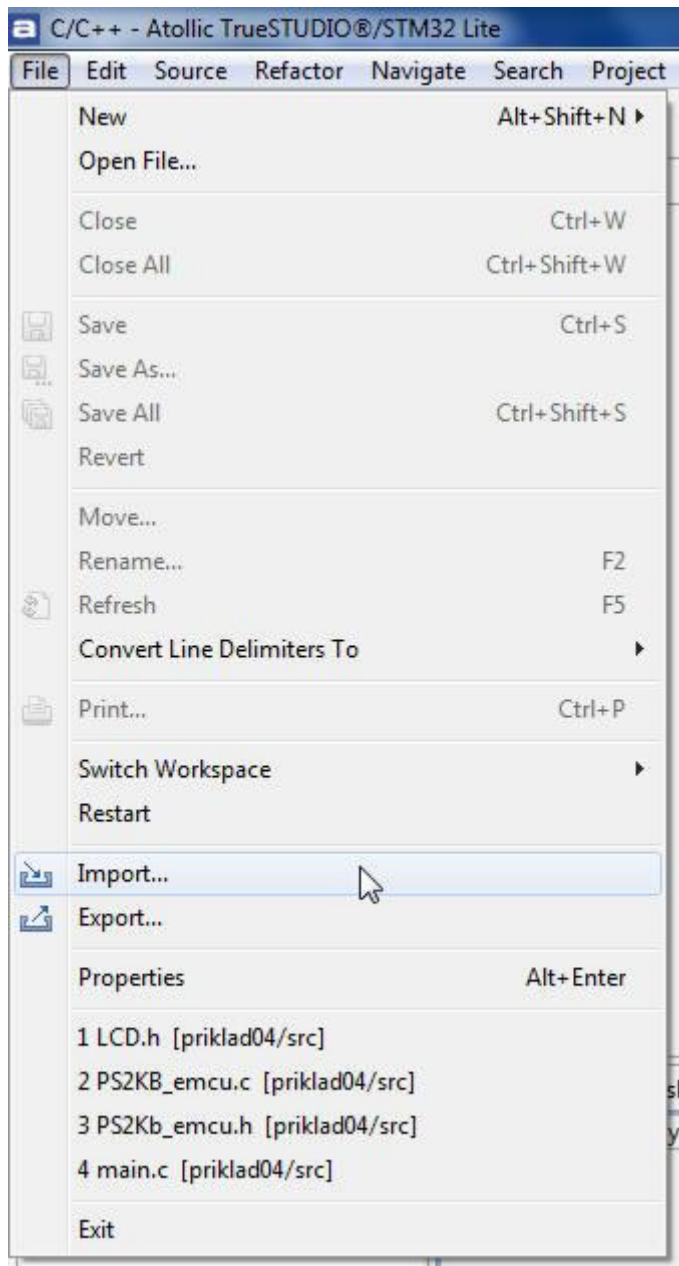
Vidíme, že se vytvořil **FreeRTOS_Source**

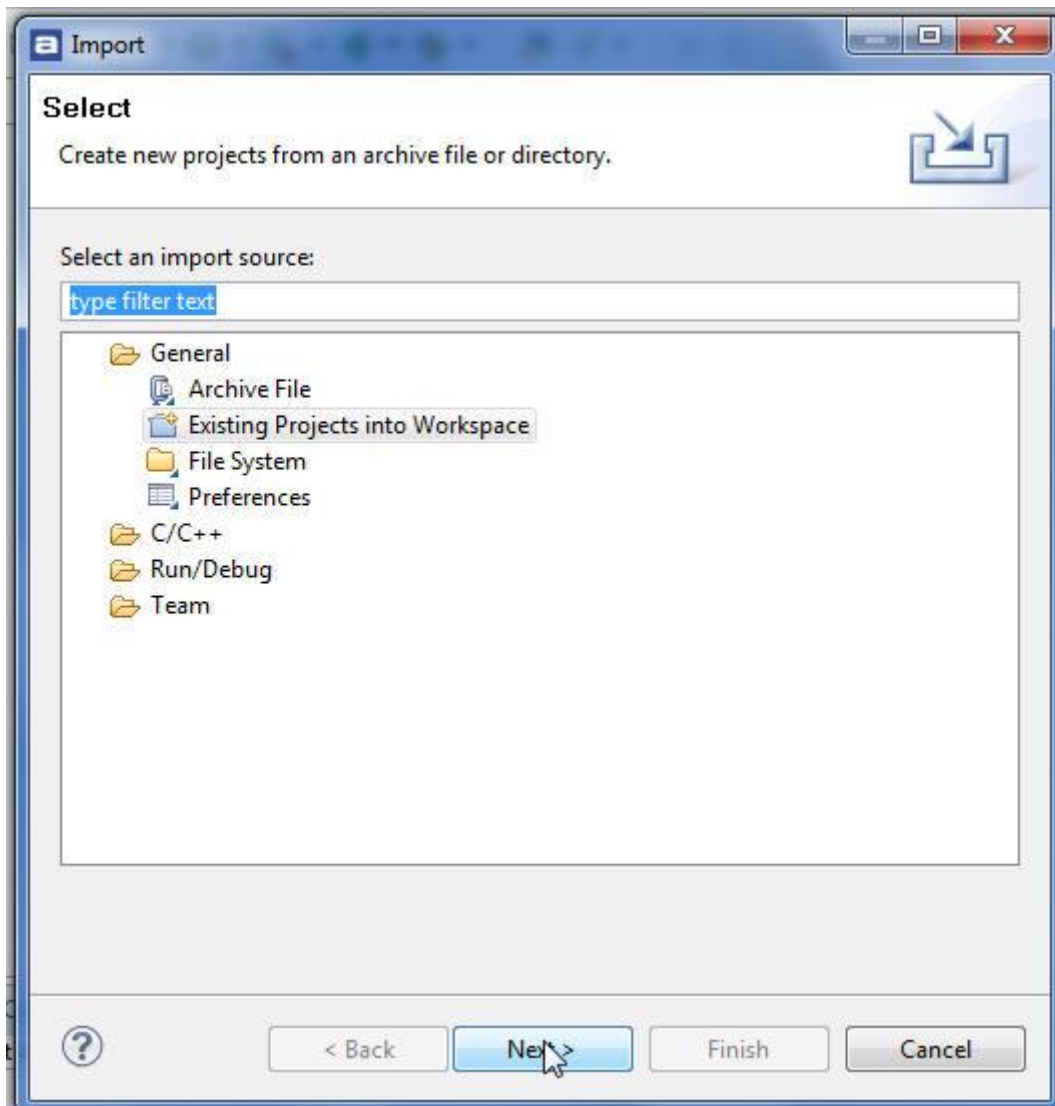
Nyní již spustíme **Atollic TreuSTUDIO**

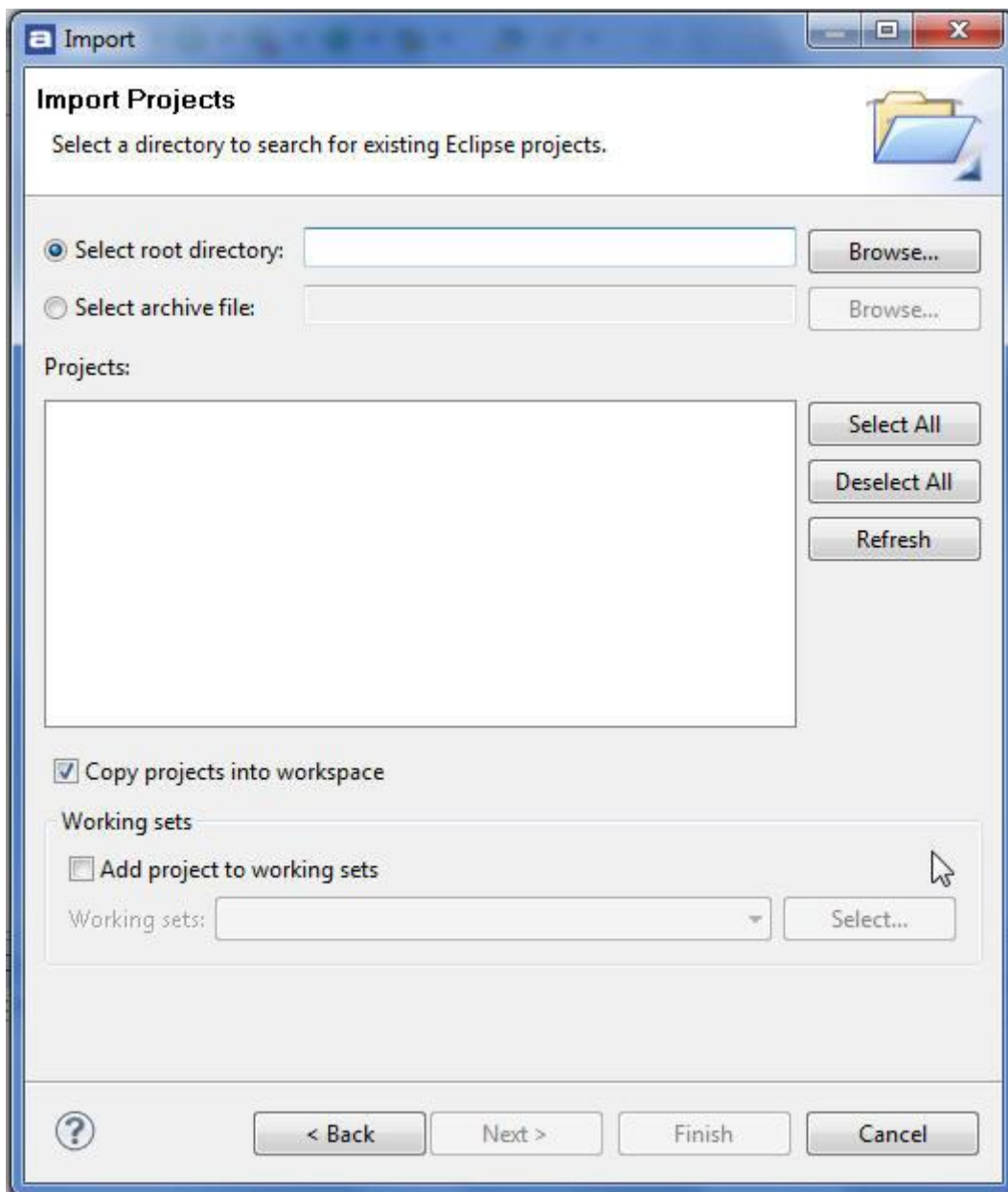
Nezapomeneme zavřít všechny projekty



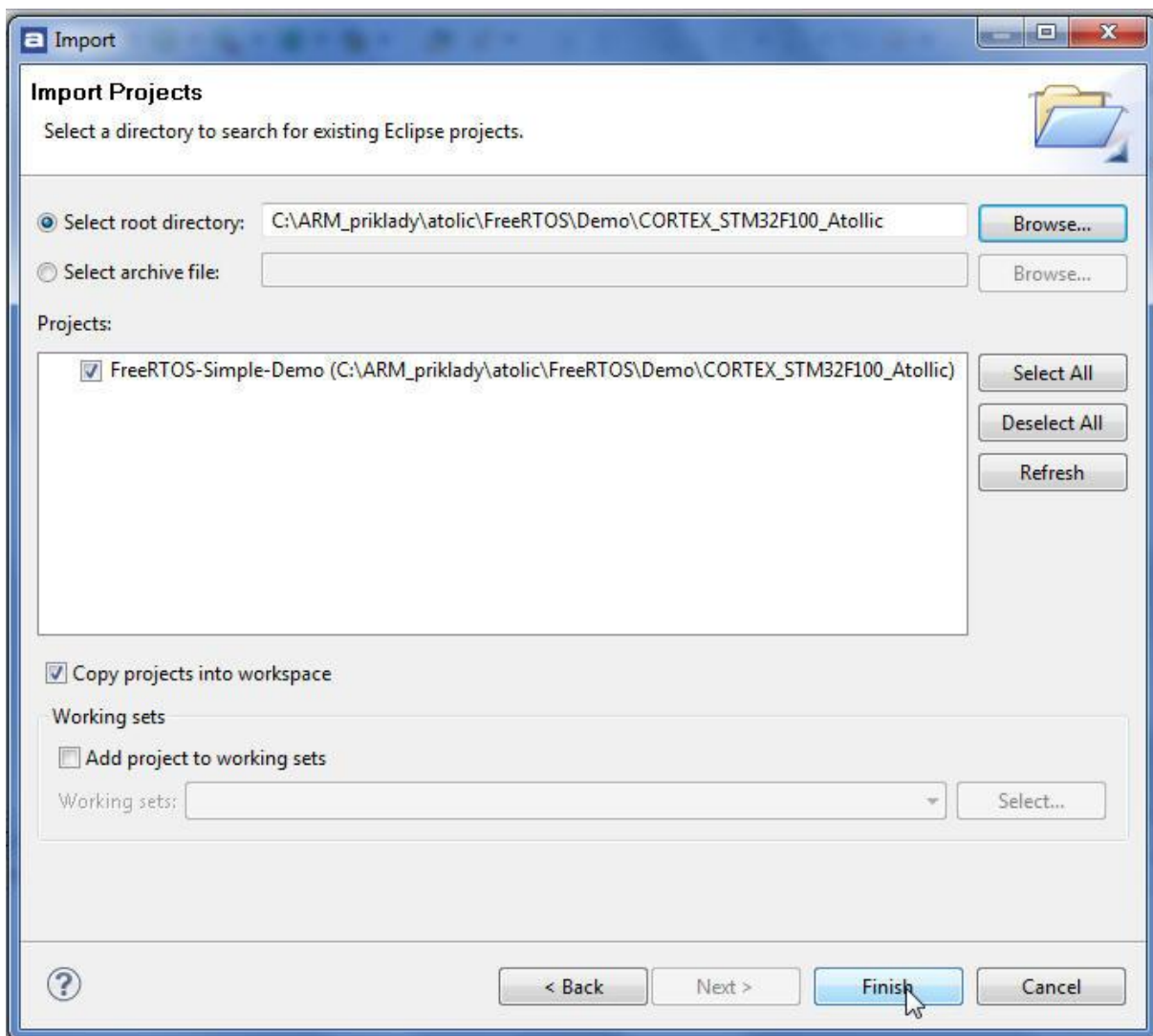
File --- Import ---- Existing Project Into Workspace

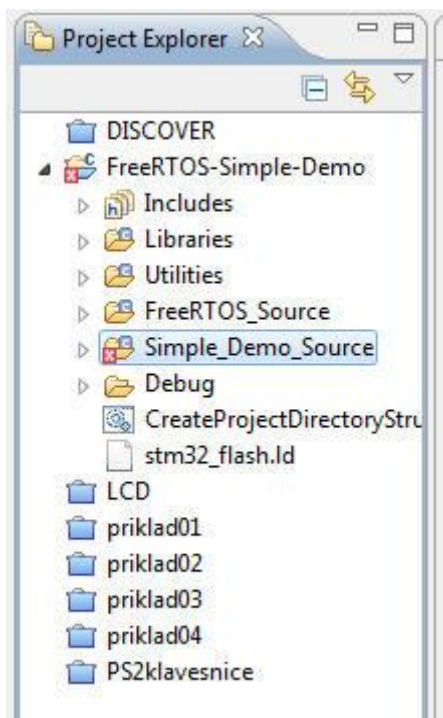






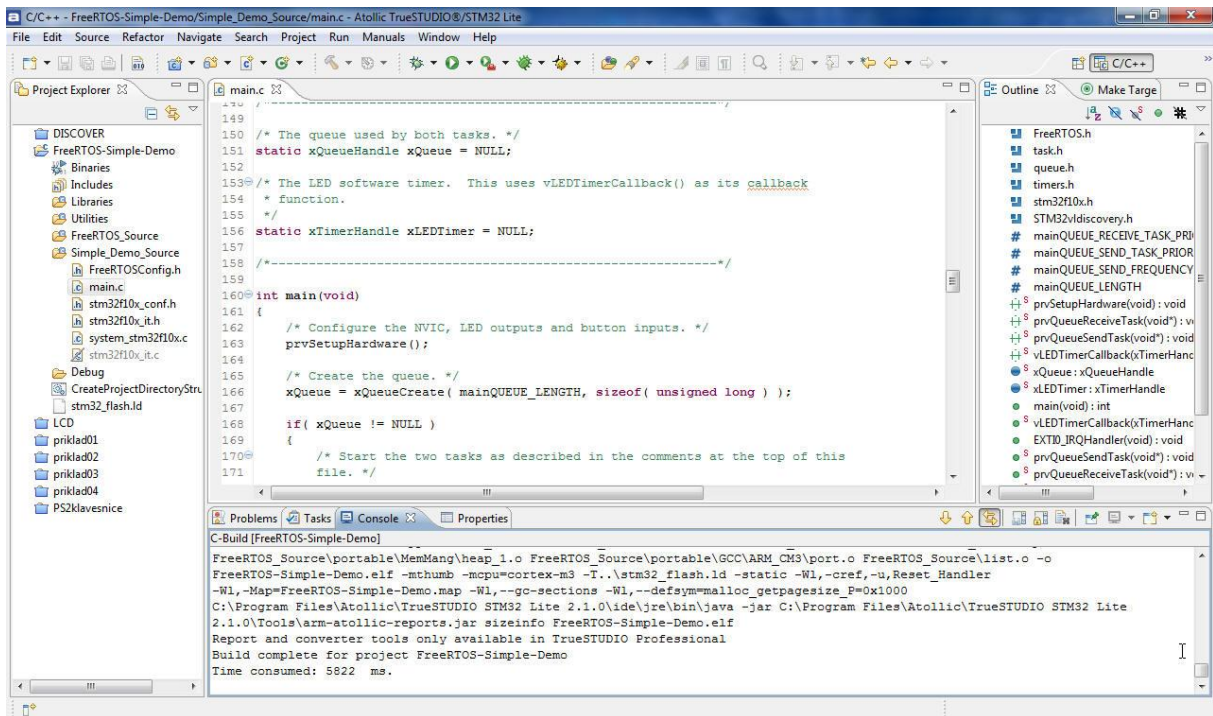
Do Select root directory dáme FreRTOS/Demo/CORTEX_STM32F100_Atollic



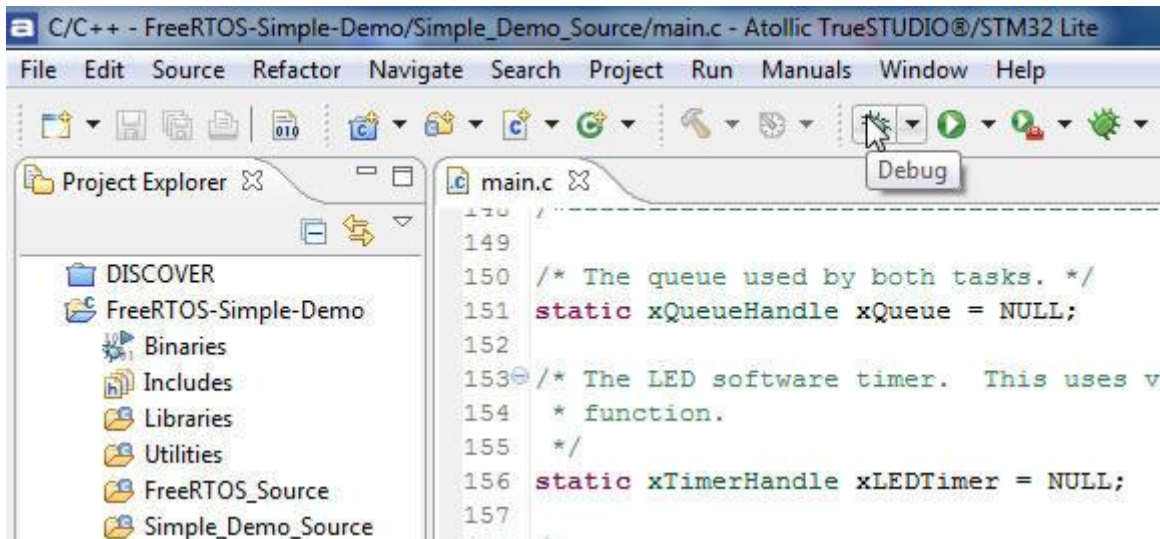


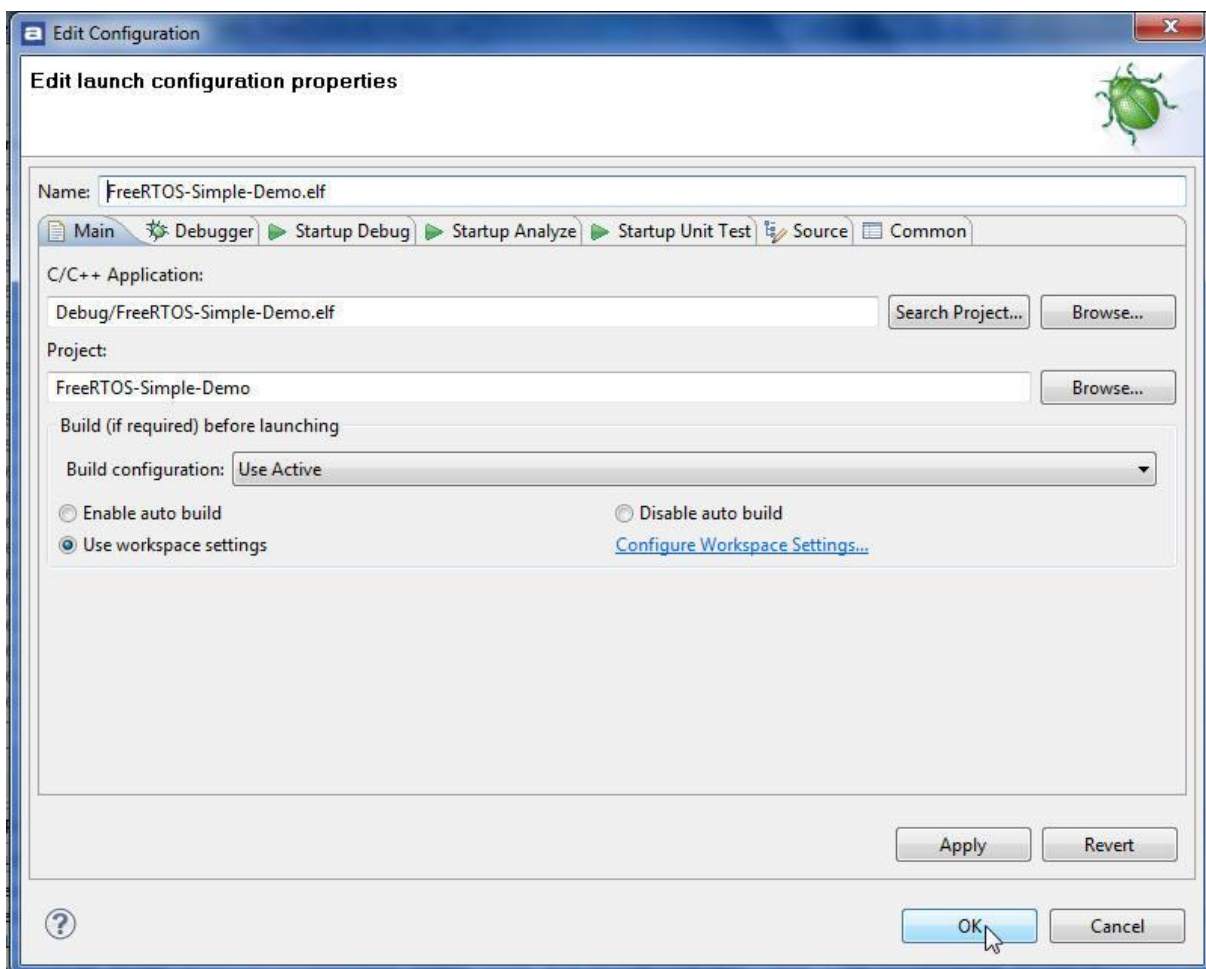
V souboru **FreeRTOSConfig.h** vymažeme řádku
`#error Ensure CreateProjectDirectoryStructure.bat has been executed before building. See comment immediately above.`

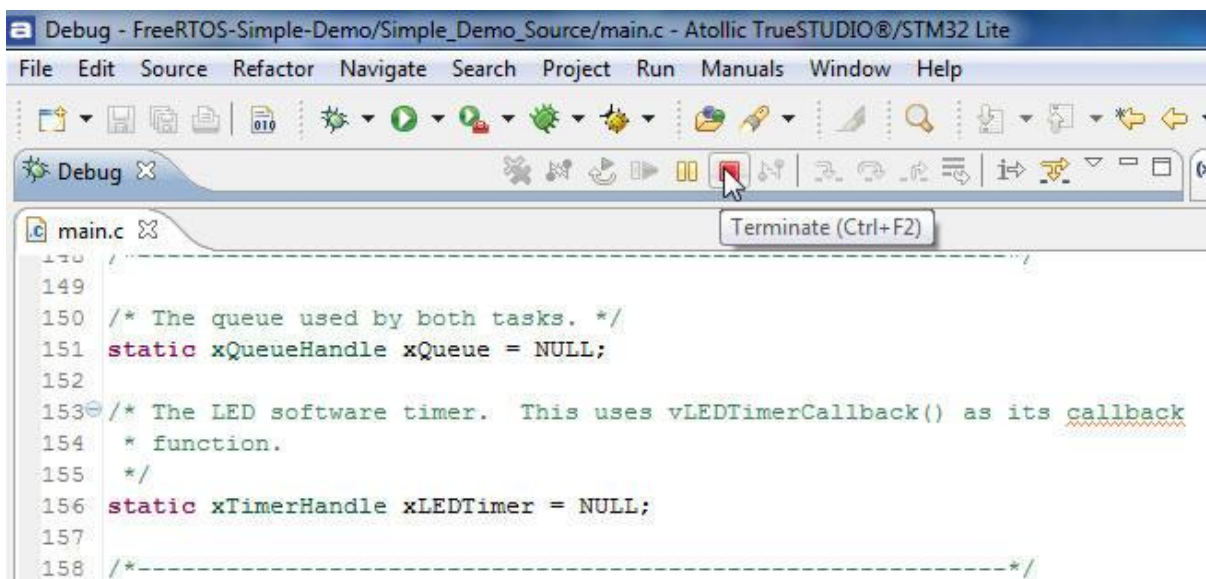
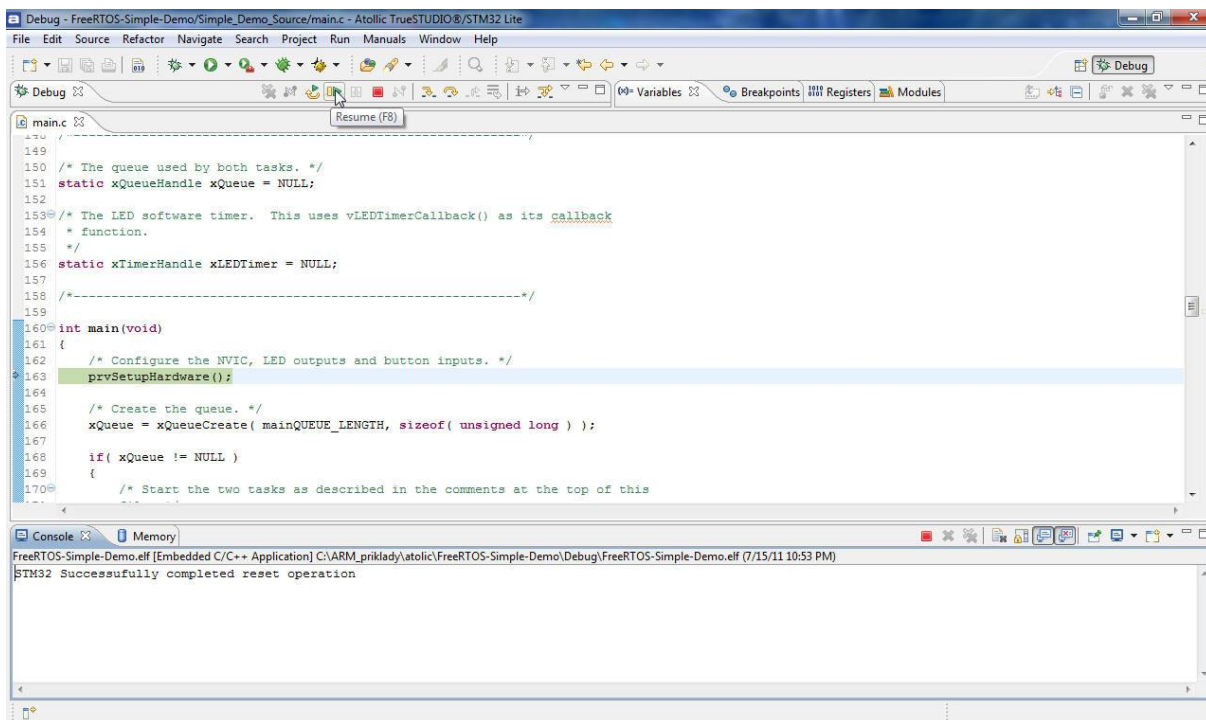
a projekt přeložíme



Naprogramujeme do startkitu







Program běží . bliká zelená LED. Pokud stiskneme USB tlačítko, rozsvítí se na cca 10s i modrá LED

5.4 programy pod FreeRTOS

Typická funkce `main()` mívá následující strukturu:

```
int main( void )
```

```

{
  /* Setup the microcontroller hardware for the demo. */
  prvSetupHardware();

  /* Create the common demo application tasks, for example: */
  vCreateFlashTasks();
  vCreatePollQTasks();
  vCreateComTestTasks();
  atd.

  /* Create any tasks defined within main.c itself, or otherwise specific to the
  demo being built. */
  xTaskCreate( vCheckTask, "check", STACK_SIZE, NULL, TASK_PRIORITY, NULL );
  Etc.

  /* Start the scheduler, this function should not return as it causes the execution
  context to change from main() to one of the created tasks. */
  vTaskStartScheduler();

  /* Should never get here! */
  return 0;
}

```

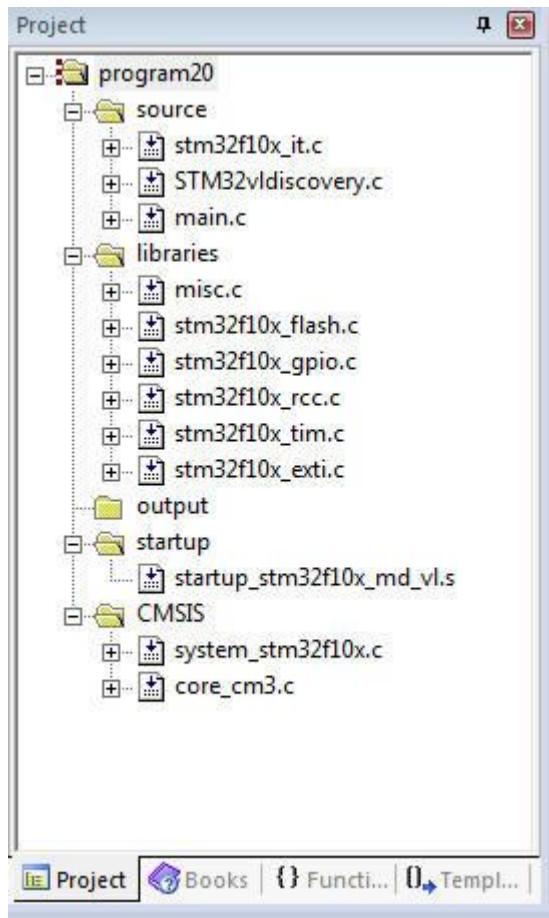
Partest.c je jméno souboru jehož název je dán historickými důvody 'parallel port test'
Další soubory:

| File | Features Demonstrated |
|-----------|---|
| main.c | <ul style="list-style-type: none"> Starting/Stopping the kernel Using the trace visualisation utility Allocation of priorities |
| dynamic.c | <ul style="list-style-type: none"> Passing parameters into a task Dynamically changing priorities Suspending tasks Suspending the scheduler |
| BlockQ.c | <ul style="list-style-type: none"> Inter-task communications Blocking on queue reads Blocking on queue writes Passing parameters into a task Pre-emption Creating tasks |
| ComTest.c | <ul style="list-style-type: none"> Serial communications Using queues from an ISR Using semaphores from an ISR Context switching from an ISR Creating tasks |

| | |
|-----------|---|
| CRFlash.c | <ul style="list-style-type: none"> • Creating co-routines • Using the index of a co-routine • Blocking on a queue from a co-routine • Communication between co-routines |
| CRHook.c | <ul style="list-style-type: none"> • Creating co-routines • Passing data from an ISR to a co-routine • Tick hook function • Co-routines blocking on queues |
| Death.c | <ul style="list-style-type: none"> • Dynamic creation of tasks (at run time) • Deleting tasks • Passing parameters to tasks |
| Flash.c | <ul style="list-style-type: none"> • Delaying • Passing parameters to tasks • Creating tasks |
| Flop.c | <ul style="list-style-type: none"> • Floating point math • Time slicing • Creating tasks |
| Integer.c | <ul style="list-style-type: none"> • Time slicing • Creating tasks |
| PollQ.c | <ul style="list-style-type: none"> • Inter-task communications • Manually yielding processor time • Polling a queue for space to write • Polling a queue for space to read • Pre-emption • Creating tasks |
| Print.c | <ul style="list-style-type: none"> • Queue usage |
| Semtest.c | <ul style="list-style-type: none"> • Binary semaphores • Mutual exclusion • Creating tasks |

5.5 Nastavení vysílače s ADF7012

V prvním díle tohoto výukového materiálu jsme uvedli základní údaje o integrovaném vysílačovém obvodu AD7012. Je použit např. v CanSAT stavebnici firmy PrattHobbies. Obvod lze získat od Analog Devices mj i jako free samples. Pomocí řídicích signálů clk, dat a le se nejprve tento obvod nakonfiguruje, tj sériovým signálem do něj pošleme čtyři 32bitová řídicí slova. V následujícím příkladu Program20 je ukázka inicializace tohoto obvodu. Řídicí signály generujeme na PC4, PC5 a PC6. Pro demonstrační účely jsem umístil inicializační kód do nekonečné smyčky, abychom ho mohli prozkoumat pomocí osciloskopu.



```

/* STM32 VL Discovery Library
 * program20 - inicializace FM vysilace s ADF7012
 * ver 1.0
 * Vd for SPSE Jecna 2011 lessons
 */

#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include <stdio.h>

#define uchar unsigned char //dodal jsem

uchar acc;
uchar dat_1[]={0x02,0x08,0x5E,0x10}; //R Register naplnime 0x02085E10
uchar dat_2[]={0x00,0x08,0xD4,0x01}; //N-Counter Latch naplnime 0x0008D401
uchar dat_3[]={0x00,0x00,0x37,0xE2}; //Modulation Register napl.0x000037E2
uchar dat_4[]={0x00,0x5A,0xA0,0x57}; //Function Register napl. 0x005AA057

void signal_clk(u8 vystup); //PC4
void signal_dat(u8 vystup); //PC5
void signal_le(u8 vystup); //PC6

```

```

void GPIO_Inicialization(void);
void Delay(__IO uint32_t nTick);
void delay_us(u8 Time);
void delay_ms(u8 Time);
void write_reg(uchar *dat1);

int main(void)
{
    GPIO_Inicialization();
    STM32vldiscovery_LEDOn(LED4); // LED4 - modra dioda
    STM32vldiscovery_LEDOn(LED3); // LED3 - zelena dioda pridat jsem

    /* inicializace ADF7012 */
    signal_le(1);
    signal_dat(1);
    signal_clk(0);
    /* inicializace ADF7012 */
    STM32vldiscovery_LEDOff(LED4); // LED4 - modra dioda
    STM32vldiscovery_LEDOff(LED3);
    Delay(0x2FFFFFF);

    while (1)
    {
        /* inicializace ADF7012 */

        write_reg(dat_1);
        delay_us(50);
        write_reg(dat_2);
        delay_us(50);
        write_reg(dat_3);
        delay_us(50);
        write_reg(dat_4);
        delay_us(50);

        Delay(0x0000FF);
    }
}

void GPIO_Inicialization(void)
{
    STM32vldiscovery_LEDInit(LED3);
    STM32vldiscovery_LEDInit(LED4);
    STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
    STM32vldiscovery_LEDOff(LED3);
    STM32vldiscovery_LEDOff(LED4);

    GPIOC->CRL &= ~(GPIO_CRL_MODE4 | GPIO_CRL_CNF4);
    GPIOC->CRL |= GPIO_CRL_MODE4_0;

    GPIOC->CRL &= ~(GPIO_CRL_MODE5 | GPIO_CRL_CNF5);
    GPIOC->CRL |= GPIO_CRL_MODE5_0;
}

```

```

        GPIOC->CRL &= ~(GPIO_CRL_MODE6 | GPIO_CRL_CNF6);
        GPIOC->CRL |= GPIO_CRL_MODE6_0;
    }

void signal_clk(u8 vystup)    //PC4
{
    if (vystup==0)
    {
        GPIOC->BSRR = GPIO_BSRR_BR4;
    }
    else
    {
        GPIOC->BSRR = GPIO_BSRR_BS4;
    }
}

void signal_dat(u8 vystup)    //PC5
{
    if (vystup==0)
    {
        GPIOC->BSRR = GPIO_BSRR_BR5;
    }
    else
    {
        GPIOC->BSRR = GPIO_BSRR_BS5;
    }
}

void signal_le(u8 vystup)    //PC6
{
    if (vystup==0)
    {
        GPIOC->BSRR = GPIO_BSRR_BR6;
    }
    else
    {
        GPIOC->BSRR = GPIO_BSRR_BS6;
    }
}

void Delay(__IO uint32_t nTick)
{
    for(; nTick != 0; nTick--);
}

void delay_us(u8 Time)
{
    if (Time > 0 )
        for(; Time != 0; Time--)
            {

```

```

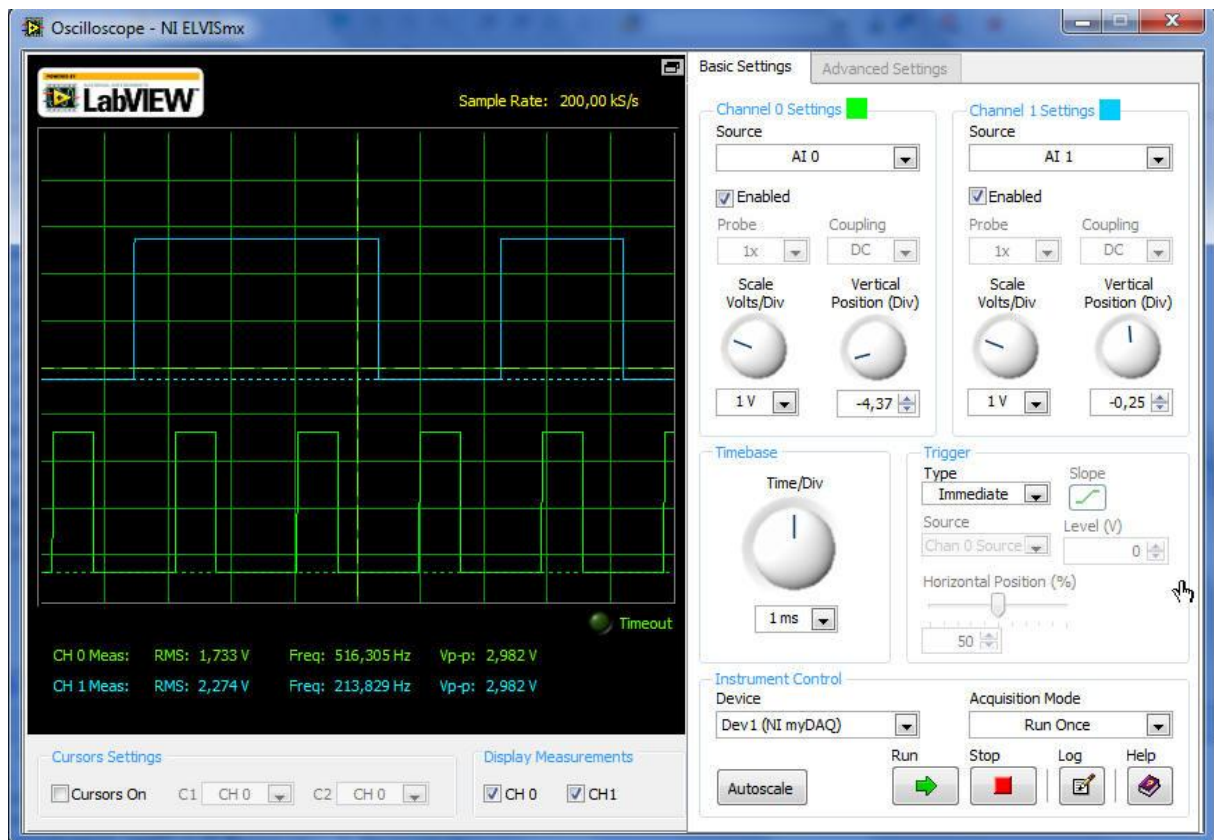
        Delay(0x00FF);
    }
};

void delay_ms(u8 Time)
{
    if (Time > 0 )
        for(; Time != 0; Time--)
        {
            delay_us(60);
        }
};

void write_reg(uchar *dat1)    //vysilame data pro registry ADF7012
{
    int i,j;
    signal_le(0);
    for(j=0;j<4;j++) // mame 4 x 8 = 32 bitova ridici slova obvodu ADF7012
    {
        acc=dat1[j]; //do acc zkopiruje j-tou osmici bitu z ridiciho slova
        for(i=0;i<8;i++) // vysilame po osmicich bitu
        {
            if (acc>127 ) signal_dat(1); //jestlize acc>127 tak v acc je 1xxxxxxx
            else //jestlize neni acc>127 tak v acc je 0xxxxxxx
            signal_dat(0);
            acc=acc<<1;
            delay_us(10); //na clk tj PC4 posleme postupne 0 1 0 a tim udelame
                //hodinovy pulz
            signal_clk(1);
            delay_us(10);
            signal_clk(0);
            delay_us(10);
        }
    }
}
}

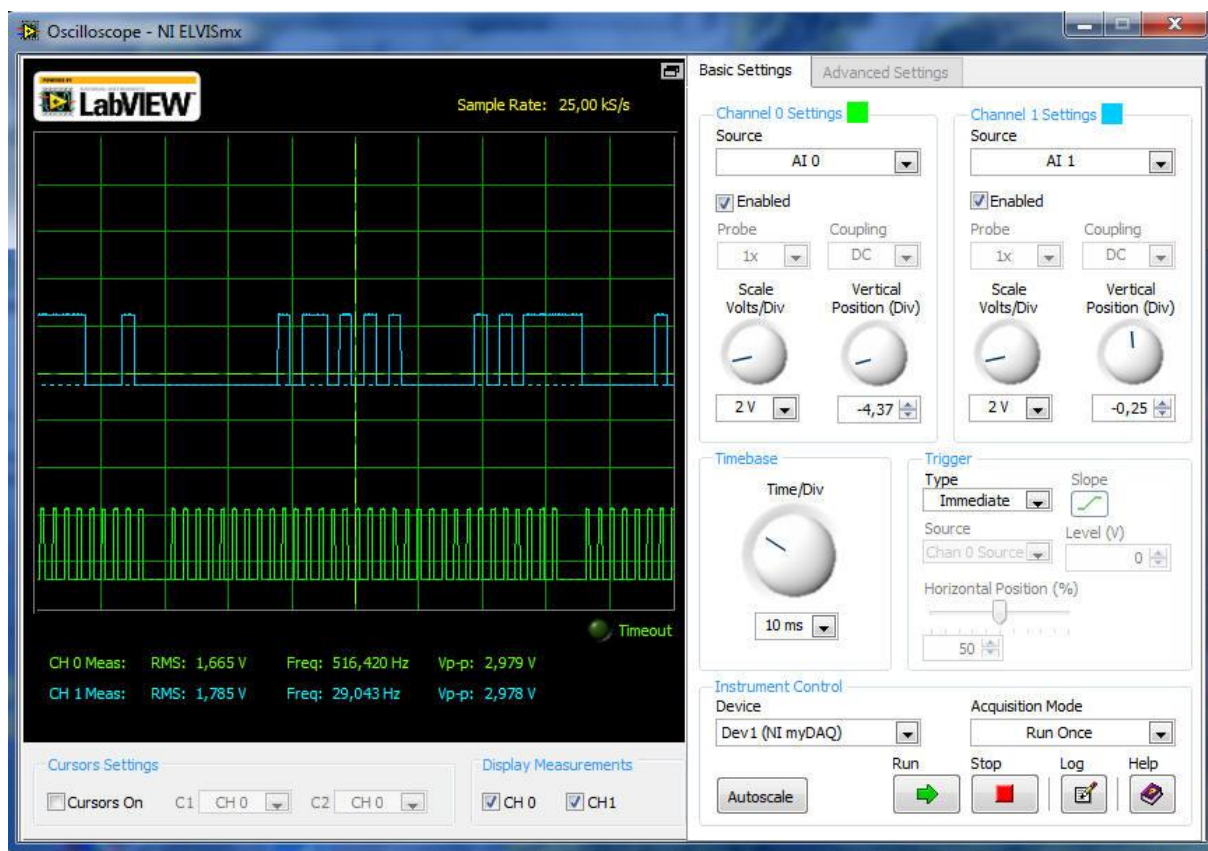
```

Na osciloskopu vidíme



Zobrazovaný úsek vysílaných dat je 011010 - horní modrý průběh ukazuje vysílaná data **dat**, dolní zelený průběh pak hodinový signál **clk**.

Na dalším obrázku si ukážeme na osciloskopu zobrazené celé 32bitové řídicí slovo



Na tomto obrázku vidíme na dat vyslaný signál 00000000 01011010 10100000 01010111 tj. 0x00 0x5A 0xA0 0x57 což je obsah Function Registru.

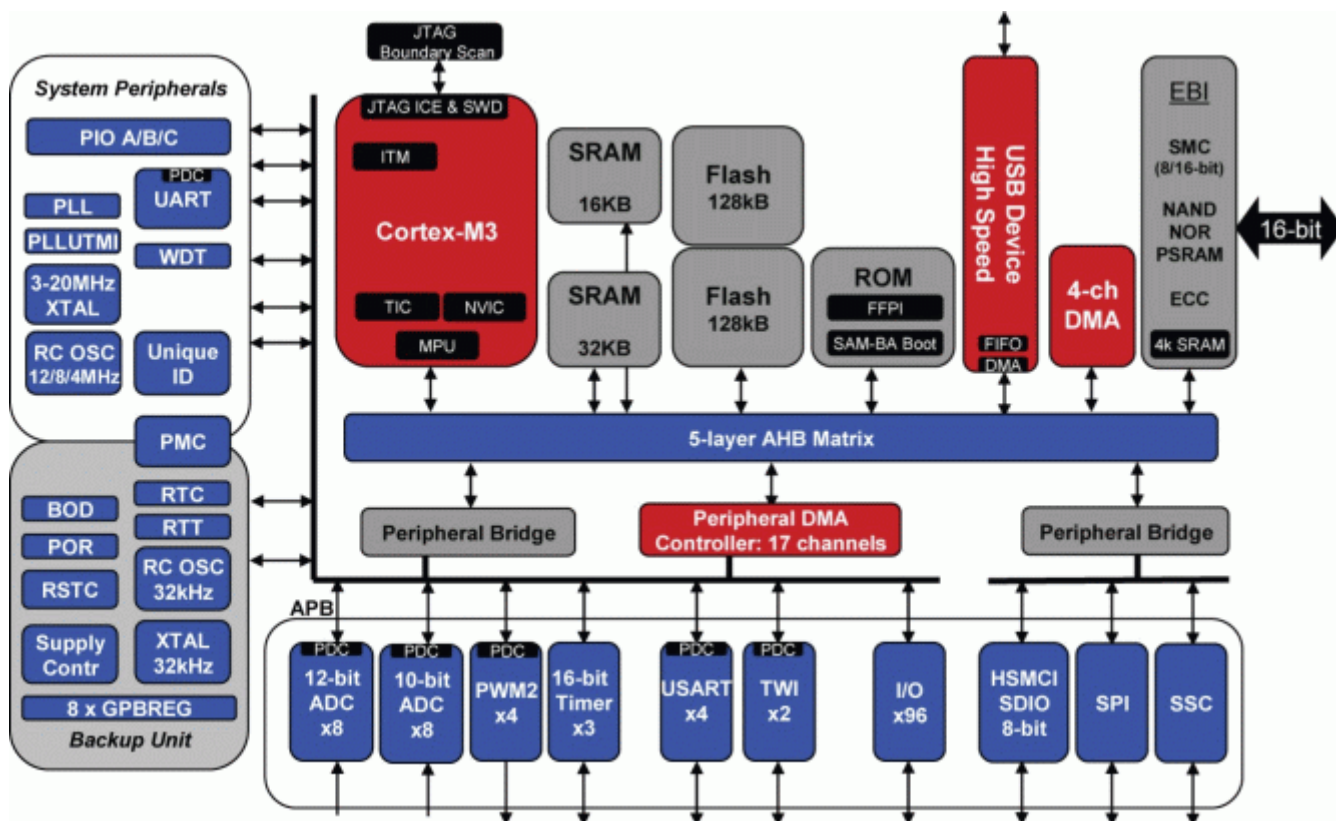
6.1 Příloha – Úvod do architektury cortex-m3 (převzato z pandatron.cz)

6.2.1 Úvod do architektury Cortex-M3 - díl. 1

Rodina procesorů založených na architektuře ARM Cortex poskytuje potřebné zdroje k řešení i náročných průmyslových, automobilových, bezdrátových a mnoha dalších technologií.

Rodina obvodů je složena ze tří hlavních typů архитектур: ARMv7 je typ pro velmi náročné aplikace, běžící na složitých operačních systémech. Typ R je určen pro použití v real-time systémech a typ M je naproti tomu optimalizován pro použití v levných embedded aplikacích.

Rodiny procesorů Cortex-M3 jsou první procesory architektury ARMv7-M, které byly navrženy s cílem dosáhnout vysokého výkonu systému a maximální efektivity levných vestavěných aplikací, jako jsou například průmyslové řídicí systémy, automobilová elektronika, drátové a bezdrátové telekomunikační systémy, systémy elektrického ovládání atd. K dosažení tohoto cíle však bylo potřeba provést v základní architektuře řadu zásadních změn. Zejména velmi zjednodušit proces tvorby softwarového balíku. Výsledkem je cenově efektivní využití procesorů architektury Cortex-M3 a to i v těch nejjednodušších aplikacích.



Obr. 1: Blokové schéma obvodu řady Atmel SAM3U

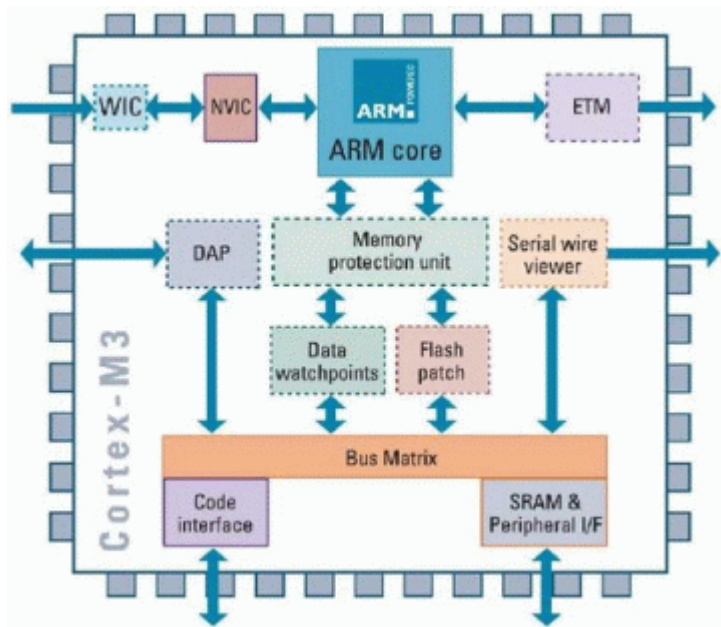
Chcete-li zvýšit výkon, musíme procesory donutit pracovat buď s vyšší taktovací frekvencí, nebo použít sofistikovanější metody zvyšování výkonnosti algoritmů. Zvýšením hodinové frekvence lze v zásadě zvyšovat produktivitu, ale zároveň tímto krokem dojde i ke zvýšení spotřeby energie a tím spojené složitosti celé aplikace, vycházející ze zajištění potřebného napájení. Naproti tomu zvýšení efektivity zpracování strojového kódu na nižší taktovací frekvenci je mnohem výhodnější. Ve svém jádru pracují procesory Cortex-M3 v závislosti na použité Harvardské technologii se třemi fázemi zpracování dat (pipeline), které nabízí řadu příležitostí. Množství jedno-cyklových operací přináší až 1,25 MIPS / MHz (v praktickém testu).

V procesorech Cortex-M3 je k dispozici nová sada příkazů s označením Thumb-2, která programátorovi umožňuje dosažení až o 70 % vyššího výkonu na jeden megahertz než klasické ARM procesory, založené na jádře ARM7TDMI a vybavené pouze klasickou Thumb instrukční sadou. Navíc je možné dosáhnout až o 35 % vyššího výkonu, než u shodných procesorů, které vykonávají základní sadu instrukcí ARM (testováno na shodném testu).

Zkrácení doby uvedení konečného produktu na trh a snížení nákladů na vývoj, jsou dnes ty nejdůležitější kritéria při výběru procesoru. Stejně jako schopnost rychle a snadno vytvořit strojový kód, jsou všechny uvedené kritéria klíčovým požadavkem aplikovaným v tomto směru.

Procesory založené na architektuře Cortex-M3, jejichž cílem je zajistit rychlé a jednoduché vytvoření efektivního strojového kódu bez použití vložek assembleru, netlačí na programátora zbytečně hluboké znalosti své architektury a to ani při vytváření průměrně náročných aplikací. Tyto procesory využívají zjednodušený programovací model zásobníku, ve kterém se podařilo velmi efektivně provázat ARM jádro s myšlenkami aplikovanými ve standardních 8- a 16-bitových mikrokontrolérech. Například tak

funkce řízení přerušení umožňuje produkovat velmi jednoduchý kód a to zcela bez použití assembleru nebo složité manipulace s registry.

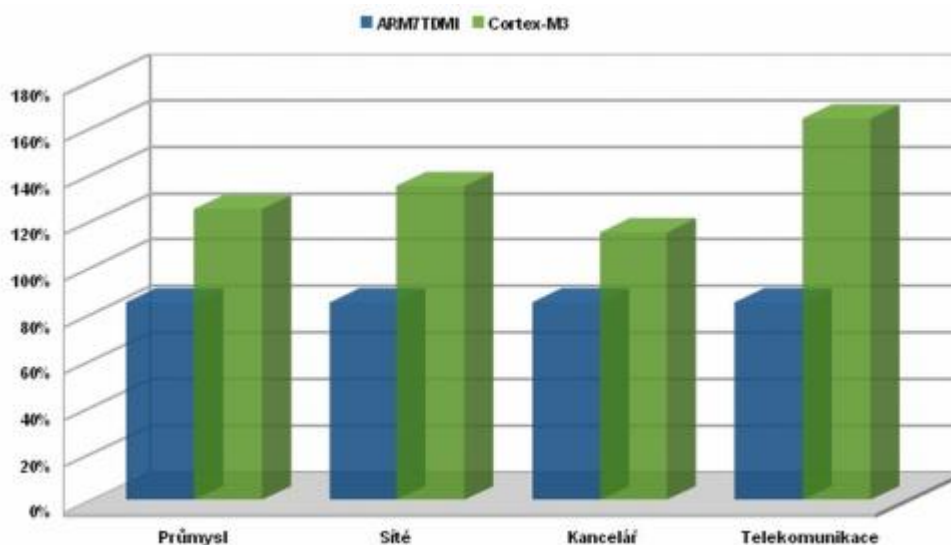


Obr. 2: Blokové schéma jádra Cortex-M3

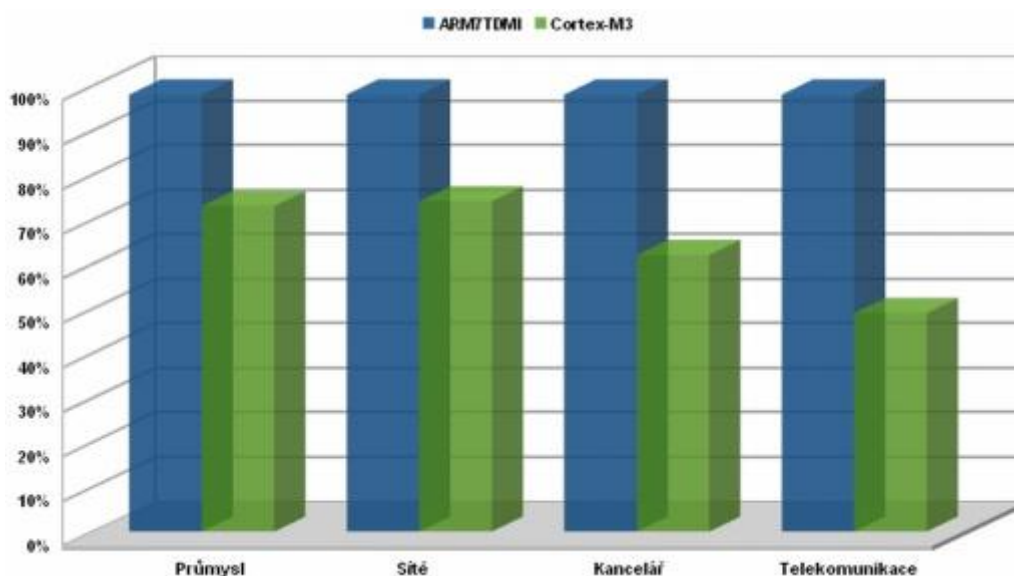
Zbrusu novým, klíčovým prvkem systému instrukcí Thumb-2 je účinnější kompilátor C s možností operací s bity a bitovými poli, hardwarovým tříděním a účinnějším prováděním strukturálních podmínek (jakými jsou kupříkladu if / then). Z pohledu programátora, umožňuje instrukční soubor Thumb-2 generovat strojový kód mnohem rychleji, jednodušeji a efektivněji. Navíc pomocí odpovídajícího kompilátoru není třeba řešit ani volbu mezi optimalizací kódu pro rychlost nebo objem - jednoduše je možné zvolit komplexní optimalizaci objemu a rychlosti. To dále zrychluje i samotný proces přípravy a generování kódu, neboť nyní vývojáři nemusí předkompilovat kritické části kódu a přilinkovávat je na hlavní program v podobě samostatných knihoven.

Hlavním omezujícím faktorem v masivním nasazování efektivnějších procesorů je jejich vyšší výrobní cena. Moderní technologie jsou drahé a proto je rozhodujícím faktorem při snižování nákladů na výrobu procesoru jediné velikost použitého křemíku. Z toho důvodu procesory vyrobené technologií Cortex-M3 obsahují nejmenší jádro typu ARM, které je k dnešnímu dni průmyslově aplikovatelné. Obsahuje pouze 33 tisíc tranzistorů, vyrobených technologií 0,18 mikronů a stejně tak i okolní periferie snížily počet svých prvků na rozumnou velikost. Další obměna proběhla i v přístupu jádra procesoru do paměťového prostoru a to zavedením technologie uchování pracovních dat, změnou práce bitových operací a systémových instrukcí Thumb-2.

Výsledkem těchto změn je o více než 25 % nižší počet přístupů do hlavního paměťového prostoru ve srovnání s klasickým jádrem ARM. Na následujících obrázcích jsou uvedeny grafy výkonu a objemu strojového kódu v různých oblastech využití procesorů Cortex-M3 a ARM7TDMI.



Obr. 3: Srovnání výkonu procesorů ARM7TDMI a Cortex-M3 v jednotlivých aplikacích



Obr. 4: Srovnání objemu strojového kódu procesorů ARM7TDMI a Cortex-M3 v jednotlivých aplikacích

Přestože procesory s jádrem ARM nejsou na trhu příliš dlouho, i tak již našly uplatnění v celé řadě embedded aplikací. Dnes jsou však nejprogresivnější obvody založené na architektuře Cortex-M3 a v blízké budoucnosti se zřejmě stanou nepoužívanější ARM architekturou. Především je tomu tak v důsledku vyšší produktivity, nižší složitosti programovacího modelu, lépe propracovaným systémem přerušení a v neposlední řadě i díky své nízké ceně. Základní výhody a srovnání obvodů založených na architekturách Cortex-M3 a ARM7TDMI jsou uvedeny v následující tabulce.”

| Parametr | Jádro | |
|----------|------------|-----------|
| | ARM7TDMI-S | Cortex-M3 |
| | | |

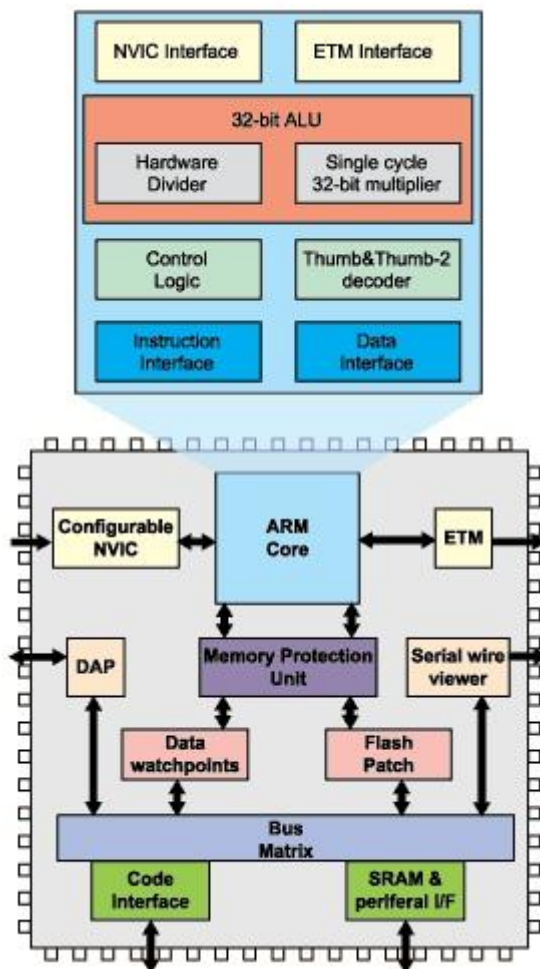
| Jádro | ARMv4T | ARMx7-M |
|-------------------------------|------------------------|--|
| Architektura | von Neumann | Harvard |
| Podporované instrukční sady | Thumb | Thumb / Thumb-2 |
| Přerušení | FIQ /IRQ | NMI + 1 až 240 fyz. přerušení |
| Čas přerušení | 24 .. 42 cyklů | 12 cyklů |
| Stepping režim | Ne | Integrované |
| Ochrana paměti | Ne | 8 regionálních jednotek |
| Výkon | 0,95 DMIPS / MHz (ARM) | 1,25 DMIPS / MHz |
| Příkon | 0,28 mW / MHz | 0,19 mW / MHz |
| Plocha čipu v mm ² | 0,62 (pouze jádro) | 0,86 (základní a standardní periferie) |

Tab. 1: Srovnání základních vlastností obvodů Cortex-M3 a ARM7TDMI

6.2.2 Úvod do architektury Cortex-M3 - díl. 2

Rodina procesorů založených na architektuře ARM Cortex poskytuje potřebné zdroje k řešení i náročných průmyslových, automobilových, bezdrátových a mnoha dalších technologií.

Procesory založené na architektuře Cortex-M3 jsou v podstatě hierarchickou skupinou. Obsahují základní CM3Core rozšířené periferie, obsahující mechanismy pro řízení přerušení, ochranu a přístup k paměti a mnoho dalších. Tento soubor je do značné míry volně konfigurovatelný, což umožňuje snadné použití procesoru pro skutečně širokou škálu úkolů, které většinou plně uspokojí veškeré požadavky. Jádro Cortex-M3 a jeho začlenění do zbylé architektury (obr. 1) jsou navrženy pro splnění všech požadavků a zároveň minimalizaci požadované velikosti paměti, spotřeby energie apod.

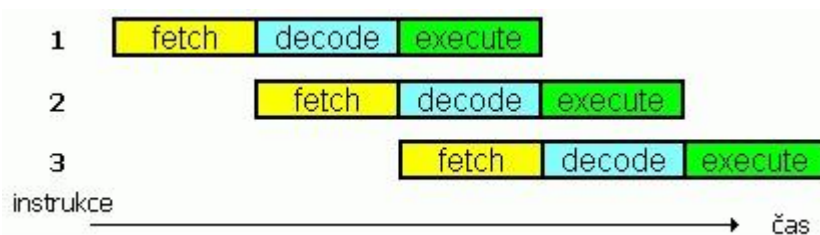


Obr. 1: Blokóvý diagram obvodu s jádrem ARM Cortex M3

Jádro Cortex-M3 je založeno na architektuře, vzniklé na Harvardově univerzitě. To je hlavní rozdíl od ostatních ARM procesorů typu von Neuman, který spočívá v oddělené instrukční a datové sběrnici. Při vykonávání pověľů čte procesor data i instrukce zároveň a díky zřetězení je schopen vykonávat i několik operací současně. Tím je samozřejmě dosaženo vyššího výkonu systému.

Tří-stupňový pipeline vykonává naráz v jediném hodinovém cyklu následující operace:

- **Fetch** - vyzvednutí z paměti instrukce č.3
- **Decode** - dekódování instrukce č.2
- **Execute** - zpracování nejstarší instrukce č.1



Uvedený způsob spolu se základní schopností predikce vede ke značnému zrychlení vykonávání instrukcí v porovnání s jinými architekturami. Je-li tak již při načítání instrukcí z paměti známo místo pokračování kódu, není potřeba čekat na vykonání skokové instrukce a do zásobníku je možné ihned načít čísla data z vyžadované oblasti. To dále snižuje zbytečné prostoje a urychluje přísun dat výkonnému jádru.

Procesorové jádro Cortex-M3 obsahuje dekodér pro standardní instrukce a instrukce Thumb-2. Ty přinášejí mnohem lepší využití dostupných prostředků, především paměti, řídicí logiky, rozhraní a ALU s podporou hardwarového násobení a dělení.

Použitá architektura procesoru Cortex-M3 je 32-bitová s 32-bitovou datovou sběrnicí, pomocnými registry a perifériemi a paměť. Jádro obsahuje celkem 13 registrů pro všeobecné použití, dva ukazatele zásobníku, registr vazby, programový čítač, stav rejstříku a sadu speciálních registrů.

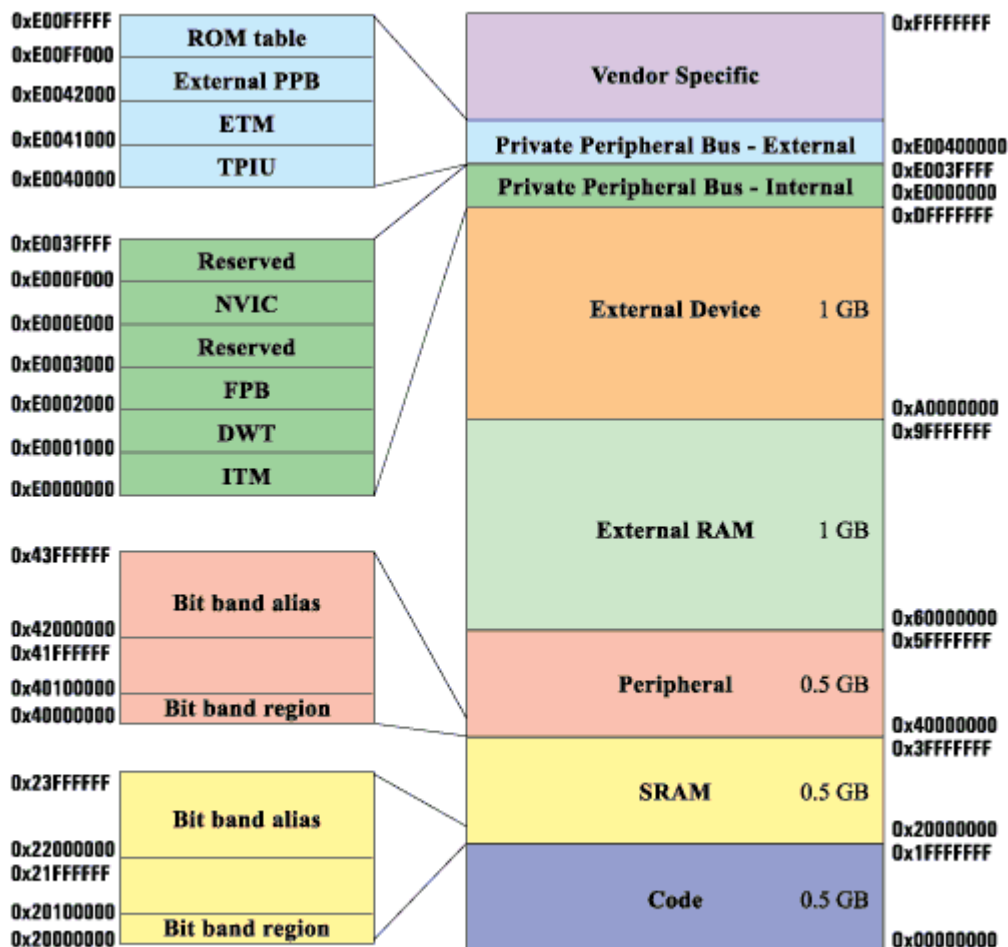
Z pohledu režimu provozu podporuje procesor Cortex-M3 dva pracovní režimy. Thread a Handler, stejně jako dvě úrovně přístupu ke kódu (privilegované a bez privilegií), umožňují snadno vytvářet i složité struktury a to bez ztráty kvality. Neprivilegovaná úroveň přístupu omezuje nebo zcela eliminuje přístup a použití některých zdrojů, jako jsou kupříkladu určité paměťové oblasti apod. Režim vlákna, tedy Thread je standardní pracovní režim, podporující distribuci úrovní přístupu. Režim Handler je využíván pouze omezeně, v opodstatněných případech, kdy celý kód získá privilegovanou úroveň přístupu.

6.2.3 Úvod do architektury Cortex-M3 - díl. 3

Rodina procesorů založených na architektuře ARM Cortex poskytuje potřebné zdroje k řešení i náročných průmyslových, automobilových, bezdrátových a mnoha dalších technologií.

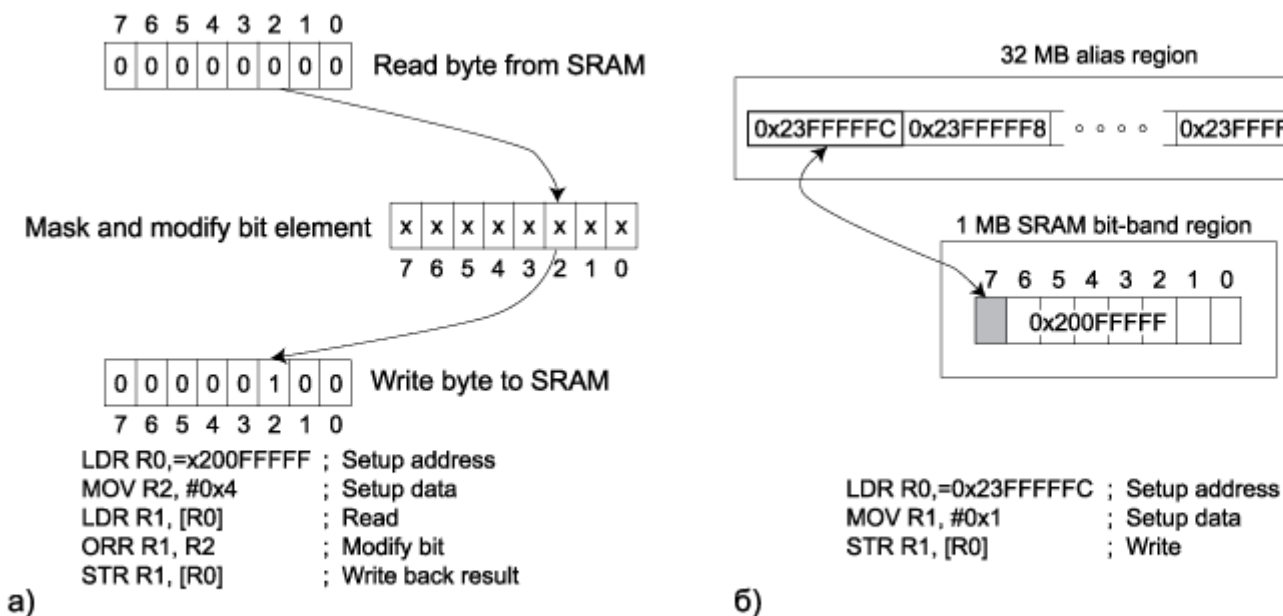
V předchozích dvou dílech úvodu do architektury Cortex-M3 jsme se zaměřili na popsání hlavních principů jádra a obecné struktury. Dále byly uvedeny hlavní rozdíly mezi jádrem Cortex-M3 a tradičním jádrem architektury ARM7 a popsány základní principy interakce mezi organizačními jednotkami procesorů vystavěných na architektuře Cortex-M3.

Jak bylo uvedeno dříve, jádro Cortex-M3 obsahuje dekodér pro obě tradiční instrukční sady a pro novou generaci Thumb-2. Rozšířené ALU rovněž podporuje hardwarové násobení a dělení, stejně jako obsahuje blok řídicí logiky a rozhraní s ostatními komponentami systému procesoru. Procesor postavený na architektuře jádra Cortex-M3 má jednoduchou, pevnou paměťovou plochu s maximálním adresovatelným prostorem až do 4 GB. Její součástí je pevně daný adresní prostor pro programovou paměť Flash, pracovní paměť RAM, externí paměť, periférie a další (obr. 1). Jak je z obrázku patrné, součástí adresovatelného paměťového prostoru jsou rovněž adresy, které jsou vyhrazeny výrobcem.



Obr. 1: Adresovatelný paměťový prostor obvodů architektury Cortex-M3

Procesory postavené na jádře architektury Cortex-M3 poskytují rovněž přímý přístup k datovým bitům a to pomocí jednoduchého mechanismu (obr. 2). Všimněte si, že uvedené operace jsou základní a nemohou být přerušeny dalšími operacemi a přerušením.



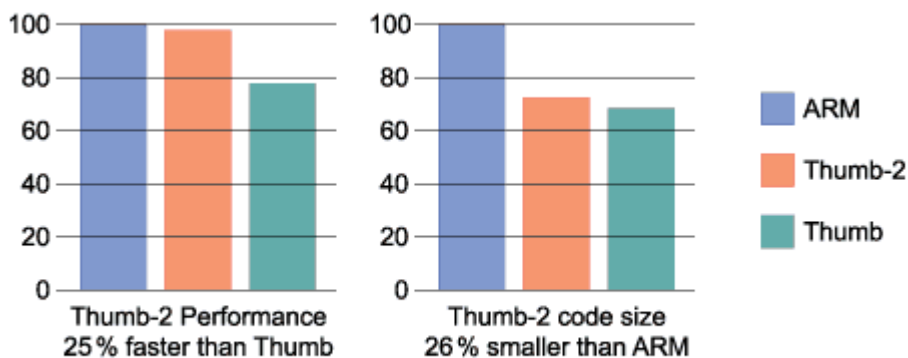
Obr. 2: Srovnání mechanismu bitových operací v tradiční architektuře ARM7 (a) a Cortex-M3 (b)

Tradiční procesory založené na architektuře ARM7 podporují pouze přístup k celým datovým slovům, což znamená, že veškeré datové práce musí být zarovnané na hranice 32 bitových slov. Zpracování stejných instrukcí na architektuře Cortex-M3 umožňuje přístup k nezarovnaným datům, což výrazně minimalizuje časové prodlevy spojené s přístupem k takovým datům. V případě, že je požadován přístup k nezarovnaným datům, je možné provést jejich rozdělení do několika paralelních vláken se zarovnanými daty, takový proces je transparentní i pro uživatele programu, jako je tomu automaticky uvnitř jádra.

Kromě toho, architektura obvodů Cortex-M3 podporuje 32-bit násobení v jediném cyklu, stejně jako práci se signed a unsigned daty, což ve svém důsledku vyžaduje 2 až 12 hodinových taktů v závislosti na velikosti operandů. Operátor dělení je samozřejmě rychlejší v případě, že dělitel a dělenec jsou malé. Takové zlepšení v provádění matematických operací, poskytované architekturou obvodů CortexM3, je významnou výhodou oproti procesorům založeným na standardu architektury ARM7 a to především pro řešení problémů, které vyžadují rozsáhlé matematické výpočty.

Mikroprocesory architektury ARMv7-M jsou rozšířením architektury ARM7, které se od předchozích ARM architektur liší podporou systému instrukcí Thumb-2. Thumb-2 je technologie 16 - a 32-bit instrukční sady, poskytující kapacitu 32-bit ARM instrukcí a velice malou velikost výsledného kódu. Je třeba rovněž poznamenat, že systém instrukcí Thumb-2 je zpětně kompatibilní s 16-bit instrukcemi Thumb. Obr. 3 je uveden výsledek testu jednotlivých instrukčních sad, což svědčí o výkonnosti systému Thumb-2 ve srovnání s jinými instrukčními systémy obvodů ARM. V předchozích verzích procesorů architektury ARM7 bylo pro efektivní kód, maximální výkon a celkově kompaktní kód potřeba vytvořit různé části programu buď pomocí ARM instrukční sady a nebo sady Thumb. Současně procesory architektury CortexM3 nevyžaduje tak složité manipulace, stejně jako jsou maximálně tolerantní k 16-bit a 32-bit instrukcím, vykonávaným ve stejném režimu. Vzhledem k tomu, že systém Thumb-2 je vytvořen z 16-bit instrukční sady Thumb, je možné na architektuře

Cortex-M3 spustit i programy napsané pomocí instrukční sady Thumb stanovené pro starší verze procesorů. Stejně tak jsou obvody Cortex-M3 kompatibilní i s kódem dřívějších verzí sérií ARM7.

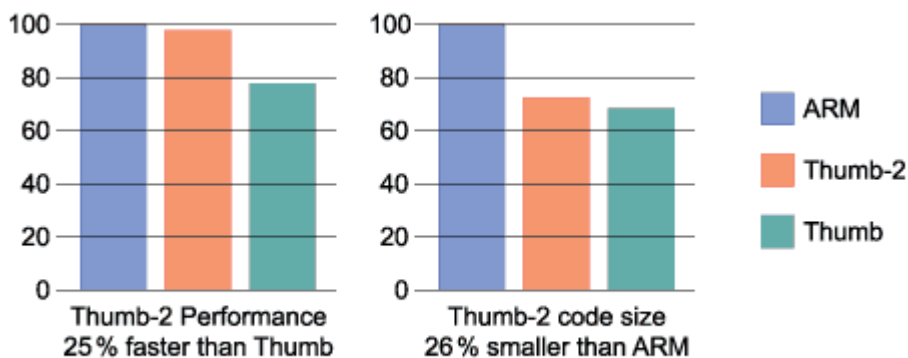


Obr. 3: Srovnání jednotlivých instrukčních sad

6.2.4 Úvod do architektury Cortex-M3 - díl. 4

Rodina procesorů založených na architektuře ARM Cortex poskytuje potřebné zdroje k řešení i náročných průmyslových, automobilových, bezdrátových a mnoha dalších technologií.

V předchozích dílech Úvodu do architektury Cortex-M3 jsme se zaměřili na popsání hlavních principů jádra a obecné struktury. Dále byly uvedeny hlavní rozdíly mezi jádrem Cortex-M3 a tradičním jádrem architektury ARM7 a popsány základní principy interakce mezi organizačními jednotkami procesorů vystavěných na architektuře Cortex-M3 i postup přístupu procesoru k datům.



Obr. 1: Připomenutí srovnání jednotlivých instrukčních sad

Instrukční sada Thumb-2 obsahuje povely, které umožňují zjednodušit a snížit množství kódu ve spoustě aplikací. To znamená, že kupříkladu instrukce BFA a BFC, které jsou určeny pro práci s bitovými poli, jsou často používány i k řešení problémům zpracování, jako jsou datové pakety sítí v telekomunikačních aplikacích. Instrukce SBFX a UBFX zjednodušují práci s bity v registrech, což je

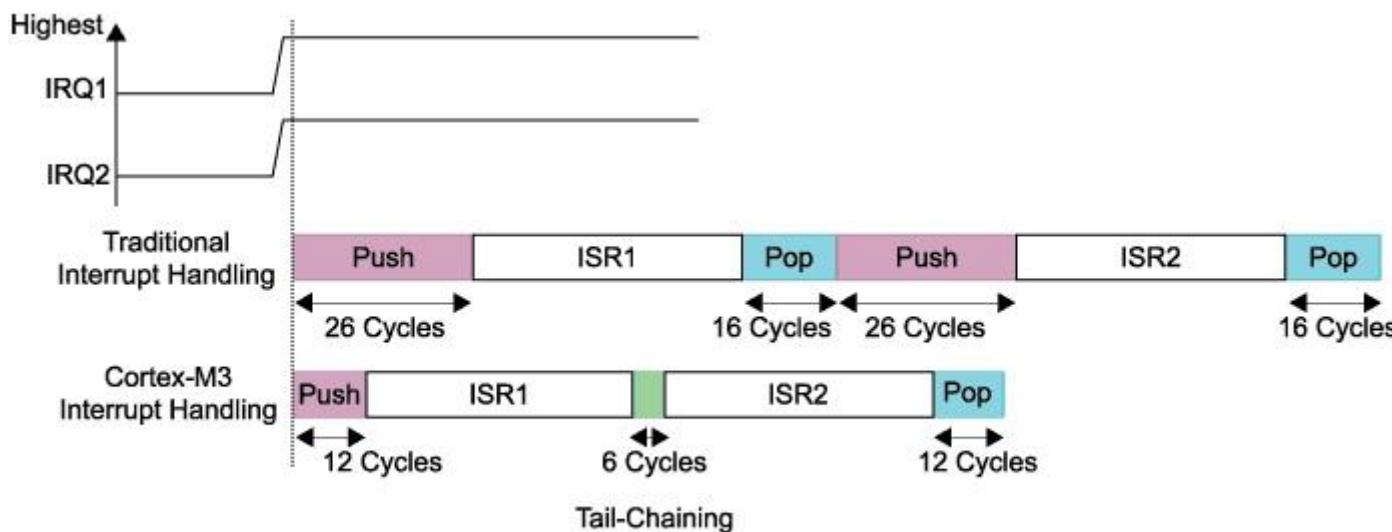
velmi užitečné v automobilovém průmyslu. Instrukce RBIT přeskupuje bity v datovém slově a v důsledku toho se velmi často využívá při provádění DSP algoritmů, jako jsou Fourierovy transformace. Instrukce pro práci s tabulkami - TBB a TBH naproti tomu umožňují dosažení rovnováhy mezi kompaktností kódu a vysokým výkonem. V neposlední řadě je rovněž možné zmínit novou implementaci instrukcí IF / THEN v instrukční sadě Thumb-2, která umožňuje předpovídání pořadí provedení až do čtvrté podmínky.

Nedílnou součástí architektury Cortex-M3 procesorů je konfigurovatelný blok NVIC, nabízející vynikající podmínky pro obsluhu přerušení. Ve výchozím nastavení generuje nemaskované přerušení a 32 fyzických utilit s 8 úrovněmi priority. Zároveň však může být nakonfigurován i tak, aby vygeneroval libovolný počet 1 až 240 fyzických přerušení s až 256 úrovněmi priority a to při velice jednoduché konfiguraci.

Architektura procesorů CortexM3 obsahuje handler tabulky vektorů přerušení, která obsahuje adresy funkcí, vykonávaných při daném požadavku. V případě přerušení činnosti procesoru je tato adresa vektoru využita s pomocí rozhraní instrukční sběrnice k přesunu v programové paměti. Tabulka vektorů probíhajících přerušení se výchozím stavu nachází na nulové hodnotě, ale umožňuje přesun s předprogramovanými kontrolními registry.

V případě, že nastane situace, kdy je voláno jedno přerušení po druhém, opakují procesory s tradiční architekturou celý cyklus zálohování dat a obnovení vždy dvakrát (při vstupu a ukončení přerušení), s výstupem ze zpracovávaného přerušení, při vstupu do dalšího. To má však za následek značné zpoždění a ve svém důsledku i výrazný pokles celkového výkonu systému. Architektura procesorů CortexM3 zjednodušuje přechod z aktivního do nového přerušení uplatněním moderních technologií tzv. "dokování přerušení" a hardwarovou implementaci kontroly NVIC.

Technologií dokování přerušení lze dosáhnout mnohem menších časových prodlev tím, že nahradí sekvenční obnovu a ochranu dat, která obvykle trvá zhruba 30 cyklů, jednoduchým mechanismem, který celkem vyžaduje pouhých 6 cyklů (Obr. 2). Stavové registry procesoru jsou automaticky uloženy při vstupu do prvního přerušení a obnoví se až po odchodu s využitím jen několika cyklů, což je mnohem rychlejší než plně softwarová implementace obsluhy přerušení, známá z klasických procesorových architektur. Takové zdokonalení umožňuje získat velmi vysoký výkon, což je obzvláště důležité v situacích, kdy je na standardní programový kód vyžadováno zpracování velkého počtu přerušení.



Obr. 2: Mechanismus pro manipulaci s přerušáním v procesoru architektury Cortex-M3

Jednotka NVIC je rovněž zodpovědná za obsluhu napájení systému a podporu pro úsporné režimy. NVIC obsahuje integrovaný 24-bitový systém a časovač, který se používá pro získání přerušování. Pravidelné časové intervaly jsou nepostradatelné při běhu operačních systémů a real-time funkcí. MPU (jednotka ochrany paměti) je volitelná součást procesorové architektury Cortex-M3, která však může výrazně zvýšit systémovou spolehlivost či ochranou kritických částí kódu. Nejčastěji je využívána pro ochranu operačních systémů, rozdělených na procesy, ve kterých zabraňuje nepovolenému přístupu k některým částem pracovní paměti. Zároveň umožňuje vymezení určitých oblastí paměti jako "read only" a detekci nežádoucích účinků přístupu do paměti, které by mohly narušit celý systém.

MPU tak aplikacím umožňuje pohodlné rozložení do souboru procesů. Každý takový proces má přidělenou svou vlastní oblast paměťového prostoru (včetně přidělené paměti programu, datové paměti, zásobníku atd.) a určených periférií, přičemž zároveň může odkazovat i do společné paměťové oblasti a společných periférií. MPU tak zajišťuje privilegovaný přístup dle stanovených úrovní. K těm patří i spuštění kódu s odpovídajícími preferencemi a práci s vyhrazenou pamětí a perifériemi. Zároveň s možností přístupu do sdíleného paměťového prostoru je možné i zde aplikovat ochranu, aby se zabránilo neoprávněnému přístupu. MPU podporuje až 8 takových domén, z nichž každá může být rozdělena do 8 suboblastí. Minimální velikost pole je 32 bytů a zvyšuje se v krocích dělitelných dvěma, maximálně však do adresovatelné paměti 4 GB. Přístup k paměti není zahrnut v oblastech určených dle MPU a nepovolený přístup do takové paměti povede k vyvolání odpovídající chyby.

Ochrana oblastí paměti před neautorizovaným přístupem je prováděna dle pravidel, která jsou naopak založena na typech operací (čtení, zápis, spuštění procesu) a stupni prioritní oblasti programového kódu, který provádí přístup. Každá oblast má sadu bitů, které reprezentují povolené typy činností oblasti, které jsou odpovědné za druhy akcí.

Mezi další výhody MPU patří rovněž i podpora překrývajících se oblastí paměťového prostoru. Tato vlastnost poskytuje velmi výrazný přínos k ochraně uložených informací. Vzhledem k tomu, že velikost oblastí je násobkem dvou, je zde příležitost plného vstupu do jedné nebo více oblastí i v rámci dalších



Inovace ve vzdělávání na území hl.m. Prahy 2011, proj.č.2046

oblastí paměti, realizaci speed multi-level struktury ochrany či řízení přístupu, stejně jako provádění vnořeného překrývání paměťových oblastí.

Literatura a další zdroje:

- [1] Váňa V.: ARM pro začátečníky. BEN, Praha 2009, ISBN 978-80-7300-246-6
- [2] <http://www.st.com>
- [3] <http://www.st.com/internet/evalboard/product/250863.jsp>
- [4] <http://cz.mouser.com/stm32discovery>
- [5] Pokorný V.: Vývojový modul s 32bitovým procesorem typu ARM. Bakalářská práce VUT Brno 2009
- [6] Jůn L.: Vývojový modul s 32bitovým procesorem typu ARM. Diplomová práce VUT Brno 2009
- [7] serial na <http://www.mcu.cz/>
- [8] [http://pandatron.cz/?1252&uvod_do_architektury_cortex-m3 - dil. 1](http://pandatron.cz/?1252&uvod_do_architektury_cortex-m3_-_dil._1)
- [9] [http://pandatron.cz/?1281&uvod do architektury cortex-m3 - dil. 2](http://pandatron.cz/?1281&uvod_do_architektury_cortex-m3_-_dil._2)
- [10] [http://pandatron.cz/?1389&uvod do architektury cortex-m3 - dil. 3](http://pandatron.cz/?1389&uvod_do_architektury_cortex-m3_-_dil._3)
- [11] [http://pandatron.cz/?1465&uvod do architektury cortex-m3 - dil. 4](http://pandatron.cz/?1465&uvod_do_architektury_cortex-m3_-_dil._4)
- [12] <http://measure.feld.cvut.cz/vyuka/predmety/A4M38AVS>
- [13] <http://neuron.feld.cvut.cz/micro/stm32/index.html>
- [14] <http://www.emcu.it/STM32.html>
- [15] <http://embeddednewbie.blogspot.com/2011/01/free-toolchains-for-stm32vldiscovery.html>
- [16] Günther Pregartner a Oliver Reisky : Sensordatenerfassungsmodul, Höheren technischen Bundes-, Lehr-, und Versuchsanstalt\Bulme Graz-Göting Abteilung Elektronik