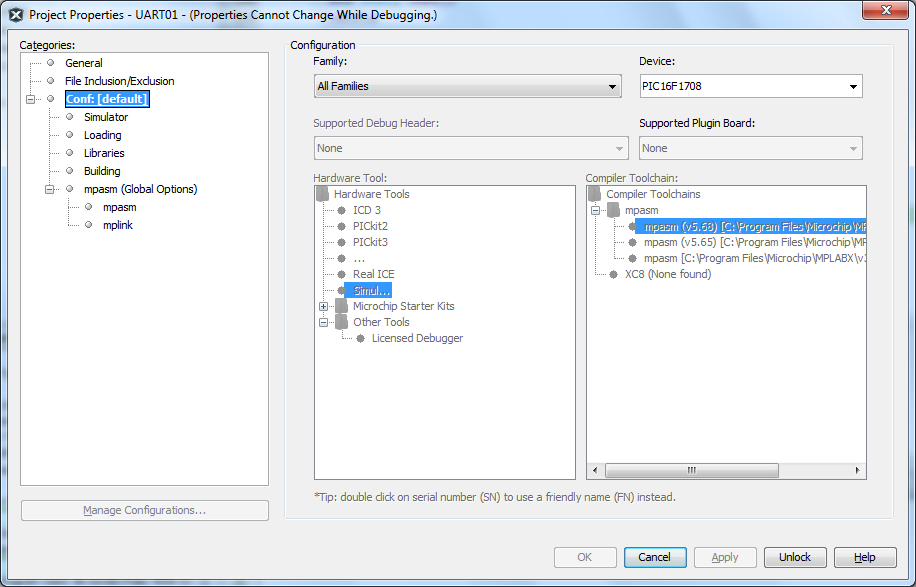
**V tomto dokumentu naleznete popis práce se simulátorem UART a vysvětlení příkladů uart01.asm, uart02.asm, uart03.asm**

**UART – simulátor**

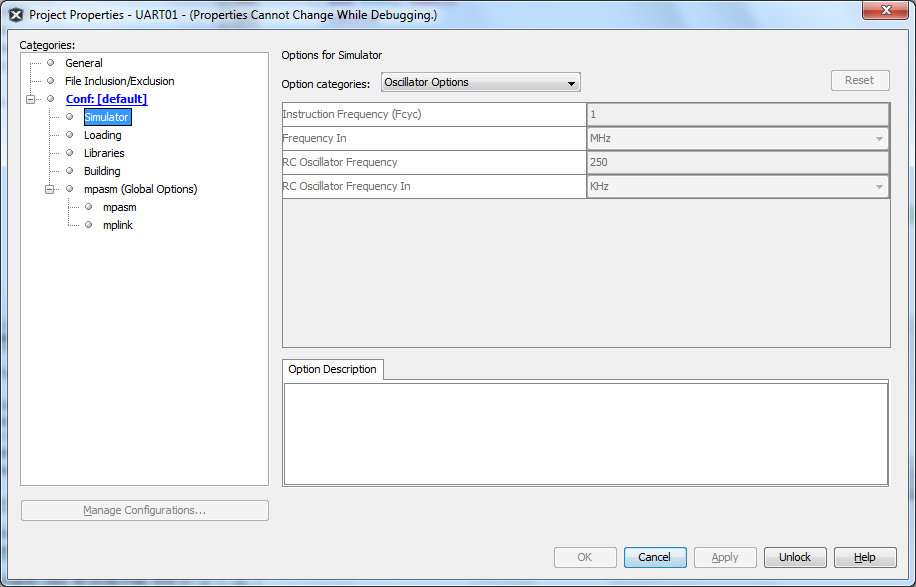
Simulaci UARTu si musíme nejdříve zapnout.

Cvakneme pravým tlačítkem na název projektu, dole properties

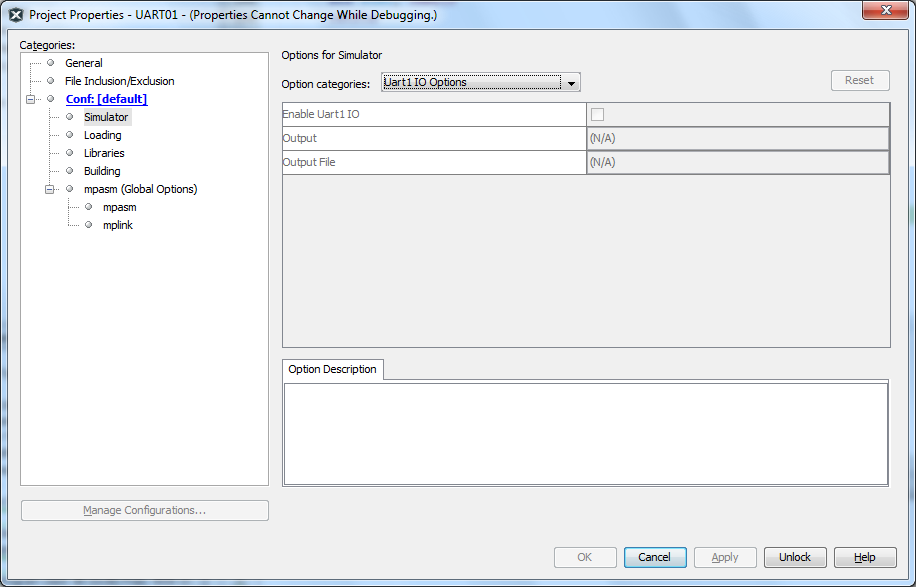
Dále cvakáme na Simulator



Objeví se:



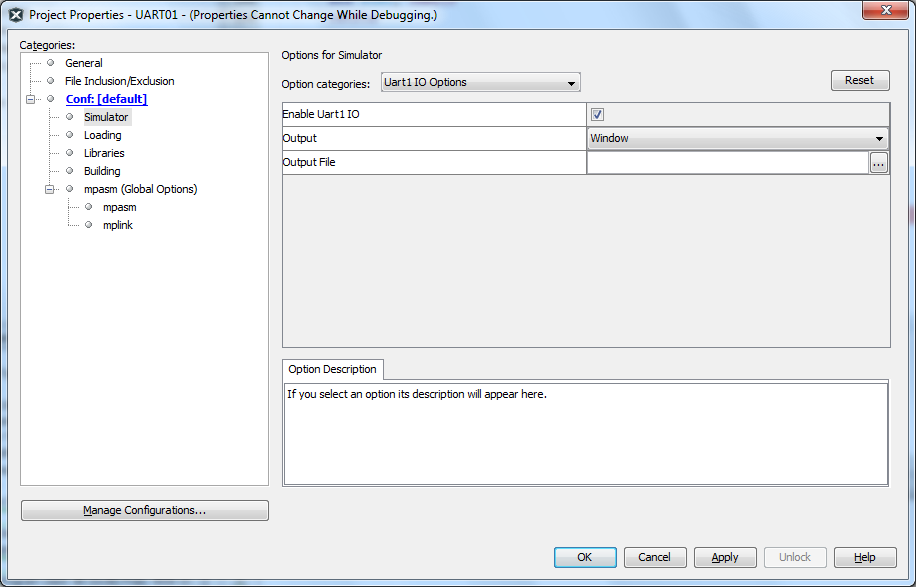
Nahoře v Option Categories zvolíme UART1IO Option



Dole Unlock, jinak nebude nic fungovat !

Zde dáme Enable Uart IO

A output dáme Window

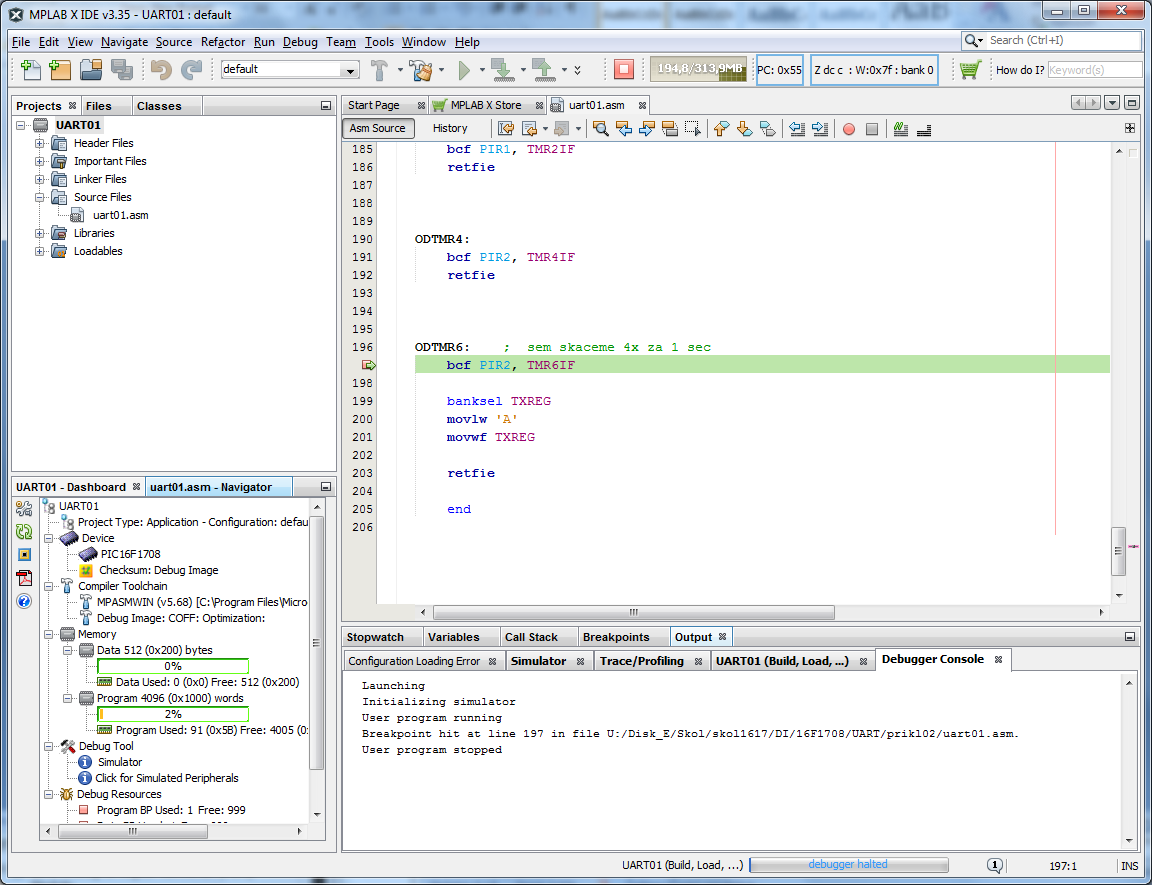


Dole Apply a OK

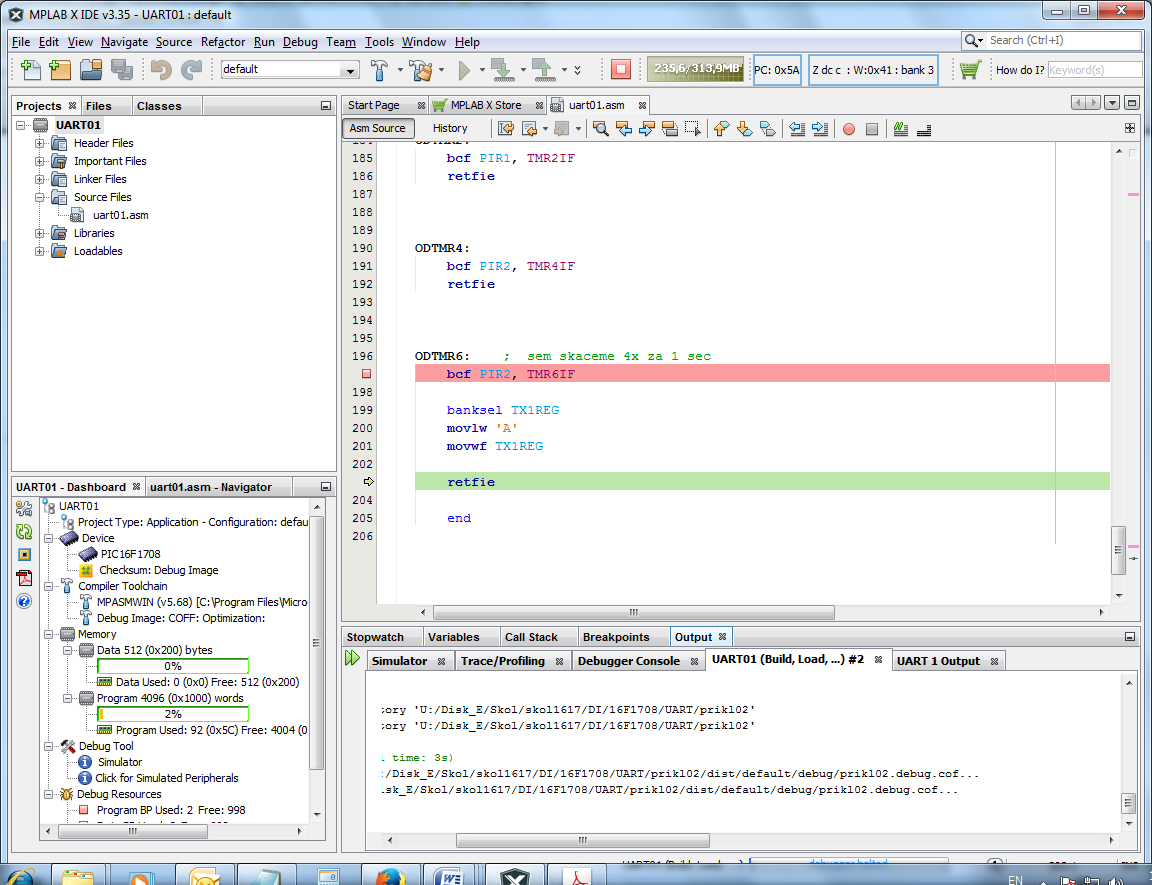
Tím jsme nastavili simulaci výstupu,.

Znovu pustíme debugger – Nahoře v liště Debug – Debug Main project

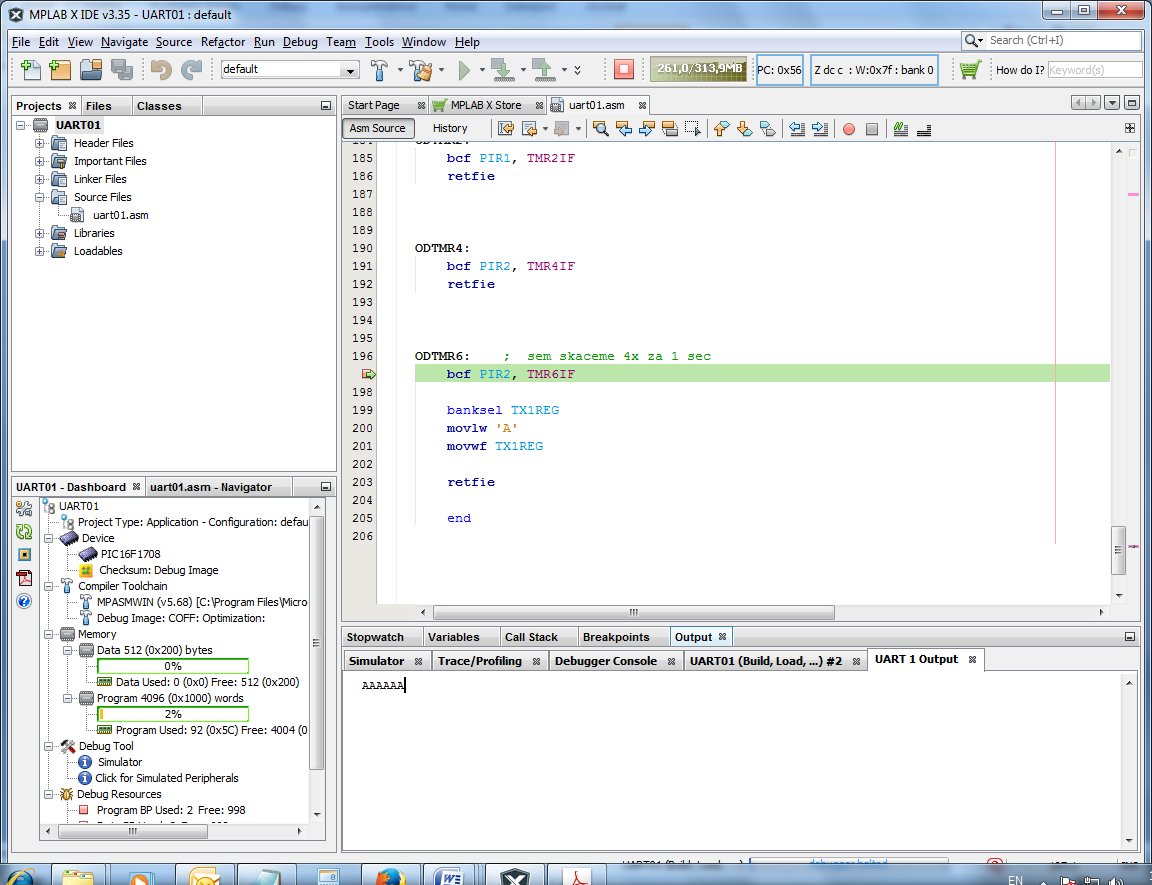
Dole nám přibylo okénko UART01



No a teď už mačkáme F5, máme samozřejmě nastavený breakpoint, a díváme se. Po odvysílání prvního znaku (tedy po prvním zápisu do TX1REG) se objeví další okénko dole v liště – UART1OUTPUT



A když na něj cvakneme myší, vidíme v okně, co nám UART vysílá



Nejdříve musíme mít zobrazené okno UART 1 OUTPUT, a teprve potom lze zobrazit znaky, které UART vysílá

Na procvičení máte v adresáři program uart01.asm

Tento program vysílá dokolečka jedno jediné písemko A , čtyřikrát za 1 sec. V programu si dejte breakpoint do podrogramu ODTMR6: , někam za instrukci movwf TX1REG – to je instrukce, která odvysílá jeden znak .

**Použití přerušení pro UART**

V programu uart02.asm máme příklad použití přerušení pro UART.

V podprogramu ODTMR6 je tentokrát intsrukce bsf PIE1,TXIE , která povolí přerušení od vysílače UARTu. UART následně odvysílá celou abecedu od A do Z, to je zařízeno v přerušení ODUARTUTX.

!!!!!! Nedivte se, že v tomto přerušení není nulování požadavkového bitu – něco jako bcf PIR1, TXIF - Požadavkový bit UARTu se nedá softwarově nulovat.

V přerušení ODUARTUTX: se postupně vysílá hodnota, která je uložena v buňce PISMENKO. Až buňka dosáhne hodnoty písmene Z, podprogram zakáže přerušení od UARTu a hodnotu buňky PISMENKO opět vrátí na A. Po zakázání přerušení čeká program, až opět přeteče čítač TMR6, v jehož přerušení se povolí přerušení pro vysílač UARTu.

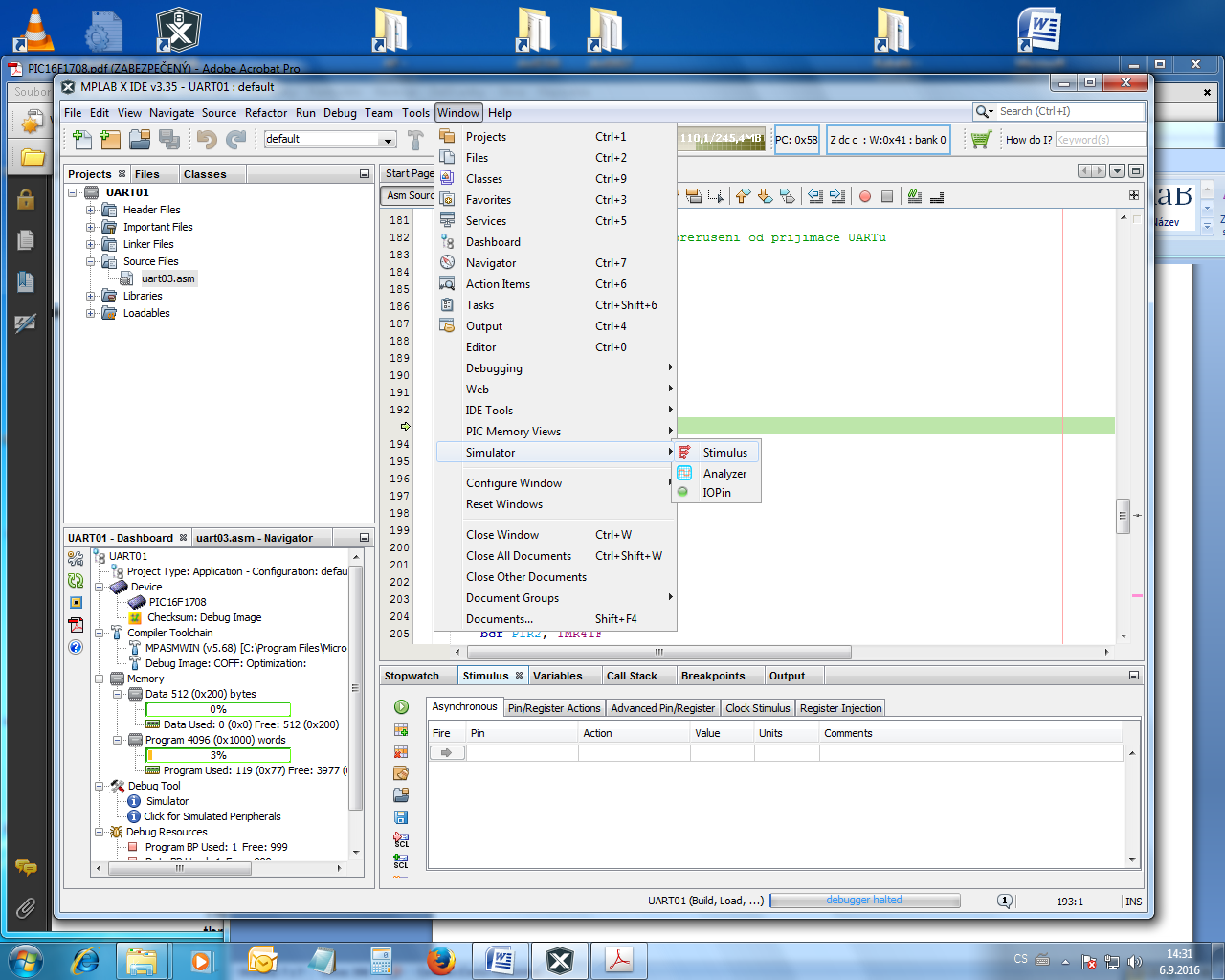
No a zase si to celé pusťte, breakpoint někam do ODTMR6: , ve výstupním okně UARTu vidíte celou abecedu. Pak si dáte breakpoint někam do ODUARTUTX: a ujasníte si, jak program pozná, že už odvysílal písmenko Z, jak odvysílá další písmenko, jak se po písmenku Z nastaví písmenko A . Prostě to celé úplně dokonale pochopíte.

**UART příjem**

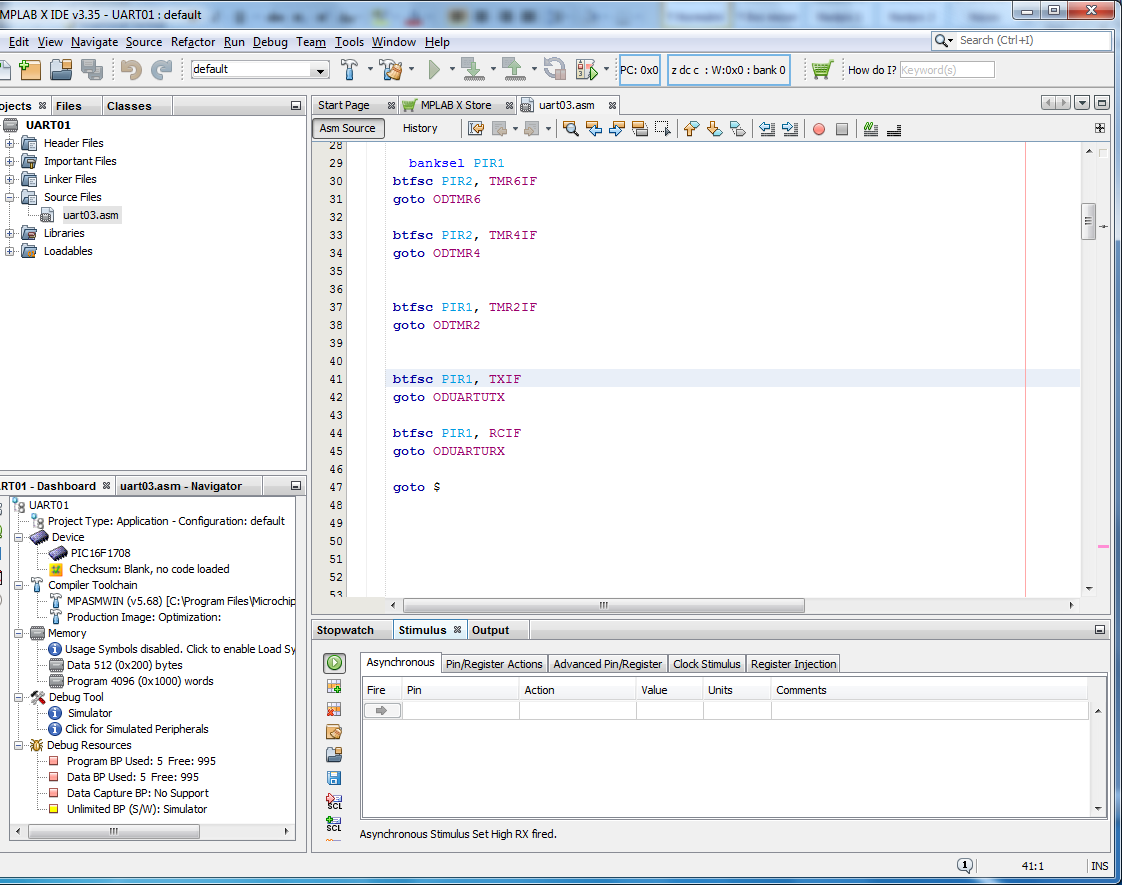
Program na demonstraci simulace přijímaných znaků je uart03.asm. Je udělán z uart02.asm, vysílání je zablokováno, ale většina kódu tam zůstala. Přidali jsme příjem - k  uart02.asm jsme přidali přerušení ODUARTURX: . Dejte si do něj breakpoint a pusťte simulátor. No a program běží a běží- Proč by také neběžel, vždyť jsme mu neposlali žádný znak.

Běh programu zastavíme – Pause

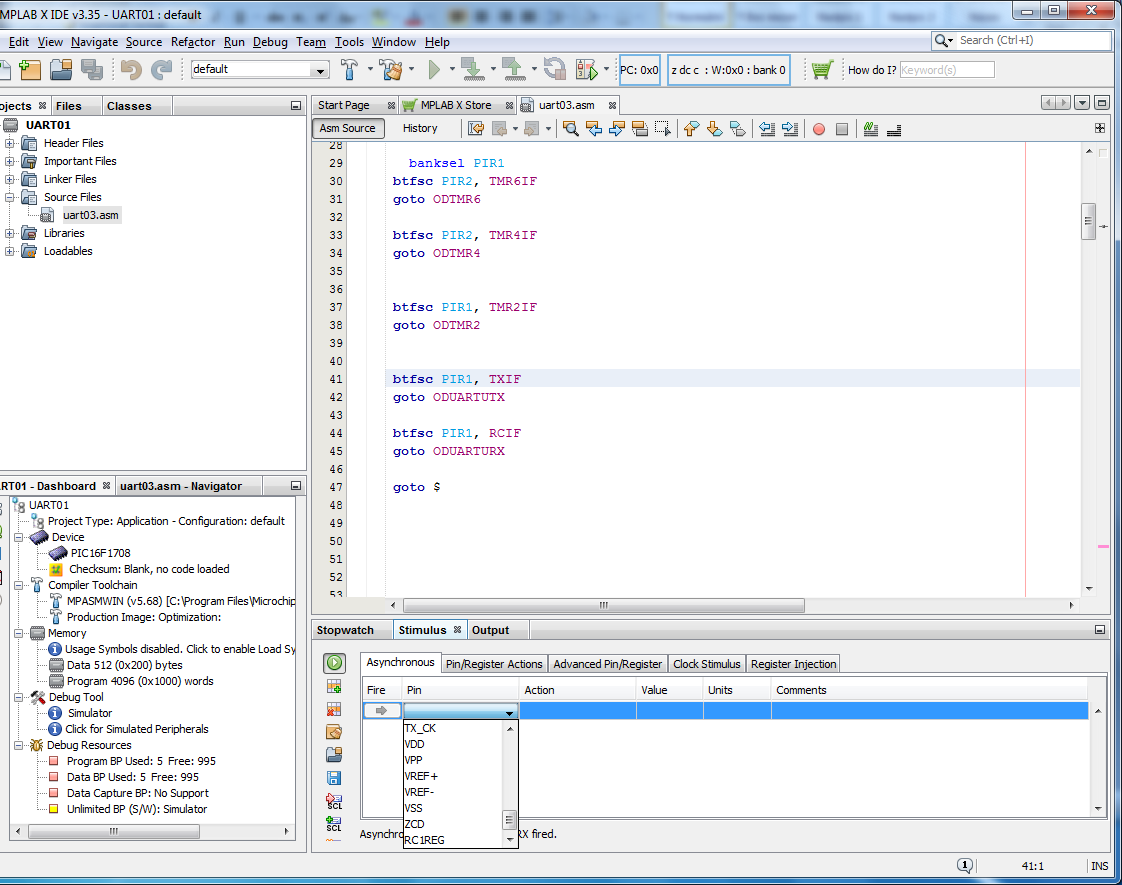
Vysílání znaku do UARTu nasimulujeme pomocí Stimulus (Window – Simulator-Stimulus)



Dole se objeví okno Stimulus. To, co potřebujeme, je Asynchronous



V nám již známém okně jsou řádky. Cvakneme do sloupečku pin, a v rozbalovacím menu úplně dole najdeme RC1REG

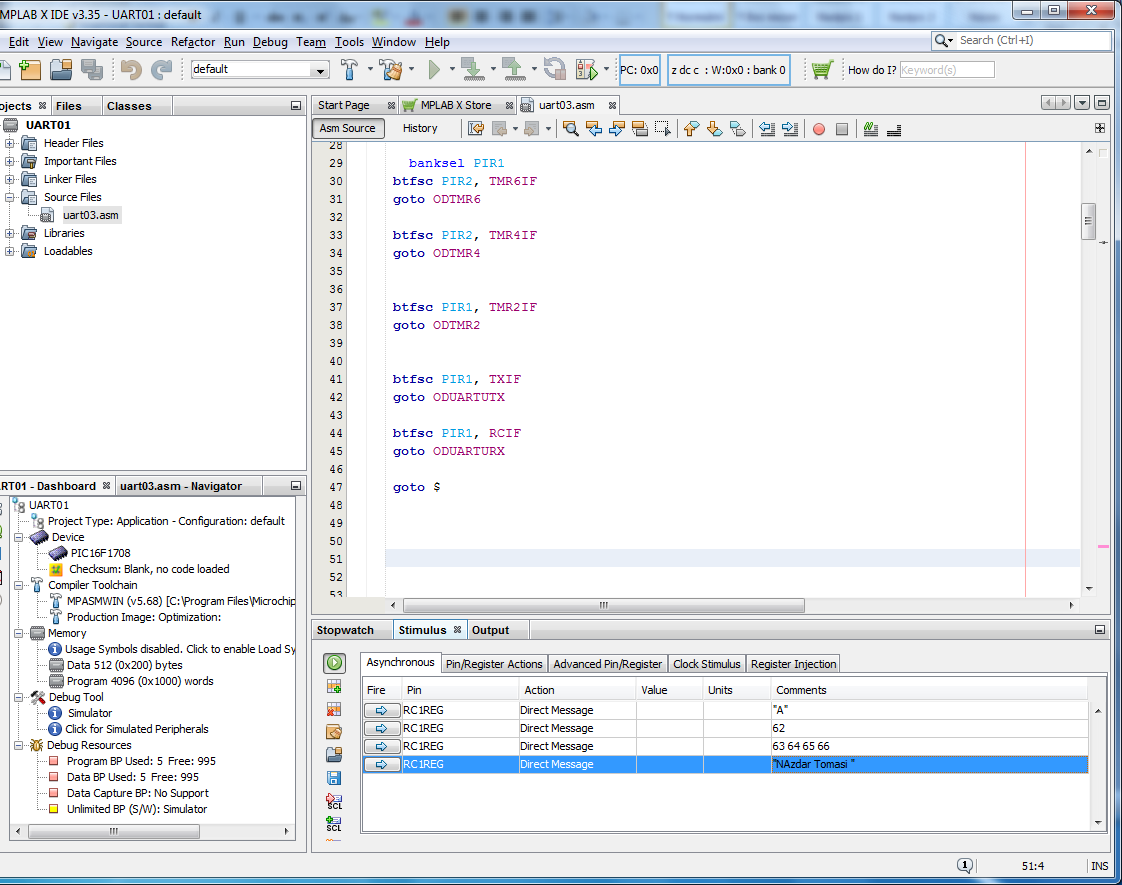


Zvolíme ho.

Můžeme si udělat několik řádek s týmž registrem – RC1REG

Ve sloupečku Action nastavíme Ditect Message . Ve sloupco Commenst máte napsáno, co máte dělat. Bohužel dost nečitelně. – Type sting or hex code . Sting je samozřejmě v uvozovkách, jak známe z jazyka C. Hexadecimální čísla jsou ve tvaru 56 fa 23 89 11 ff 1f atd, pokud jich dáme víc na řádek, jsou oddělena mezerami. V jednom řádku může být jeden znak nebo více znaků. Zadáme je dvojitým kliknutím na Comments.

Příklad máte zde:



No a dál je práce naprosto stejná, jako byla u nastavování bitů portu. Cvakneme na Fire, a v ten okamžik se nastavená hodnota pošle zvenku do UARTu. Samozřejmě, pokud máme v řádku několik znaků, posílají se postupně a procesor tedy generuje několik přerušení.

Pustíme si uart03.asm, breakpoint máme na retfie podprogramu ODUARTURX:

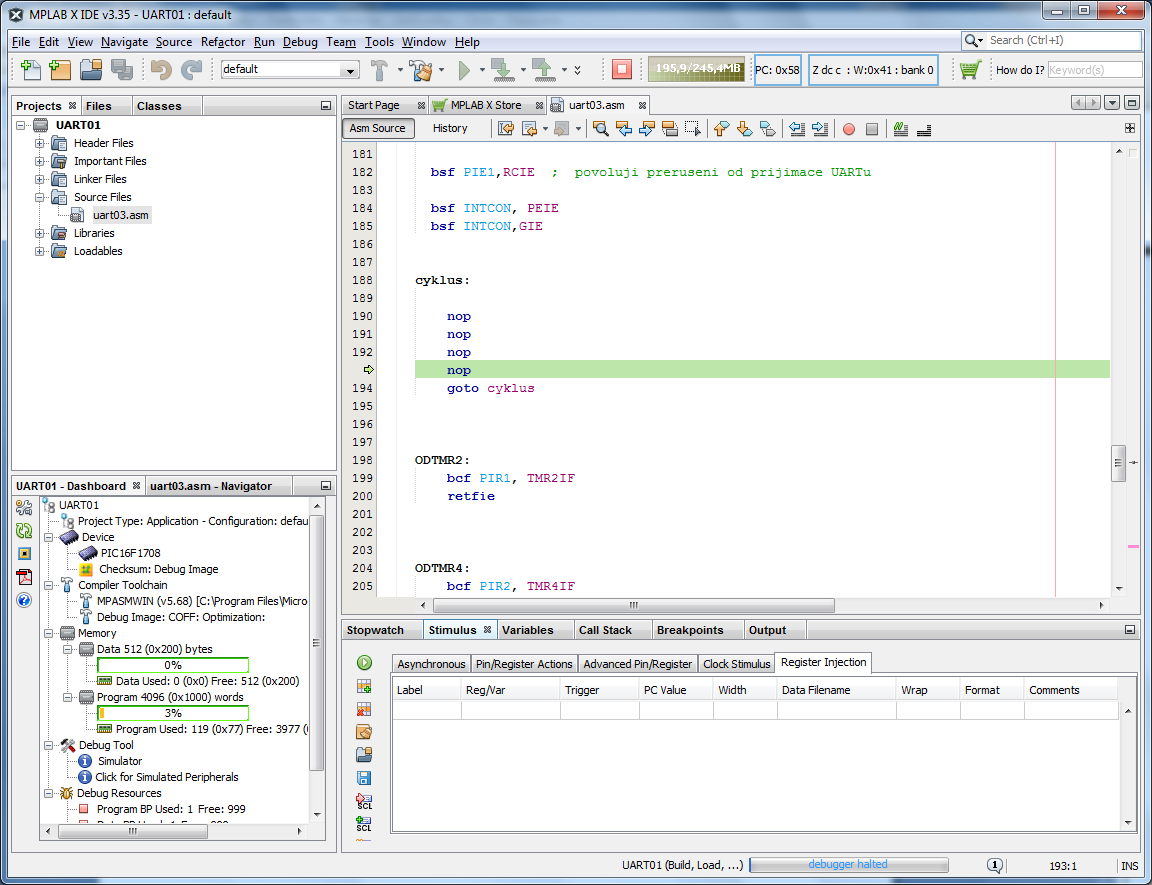
No a pomocí fire posíláme znaky. Debuuger přitom běží !! Vždycky, když cvaknete na Fire, pošlete znak (nebo několik znaků) a program se zastaví na breakpointu. Ve Variables se podíváme na WREG nebo na buňku PRIJATE, a tam vidíme, co jsme přijali

Pokud si „nacvakáte“ pomoví fire mnoho znaků do bufferu, pamatujte, že s nimi simulátor stále počítá a chce je přijímat. Zbavíme se jich tak, že uděláme „close debugger section“ a potom znovu „Debug main project“

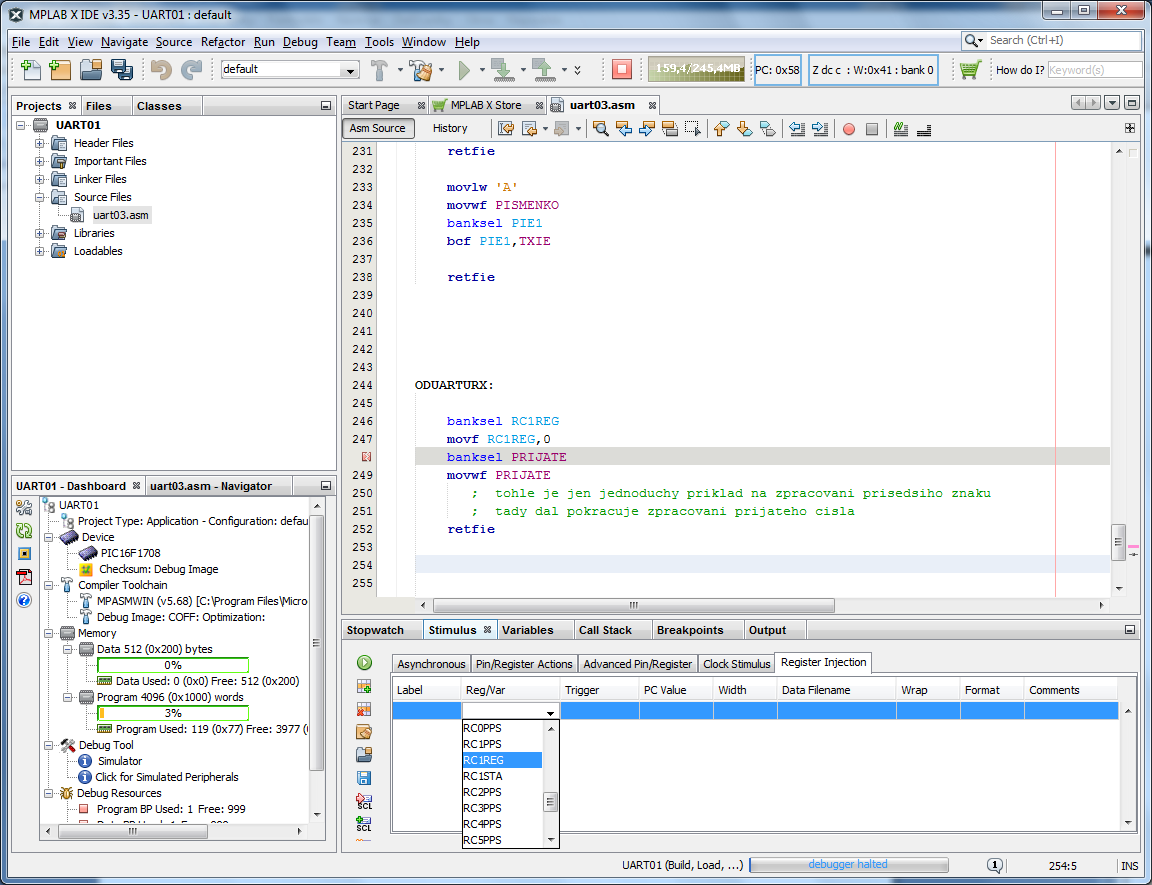
Další možnost je použít „Register injection“. Je podstatně krásnější, ale také komplikovanější.

Pouštíme Stimulus

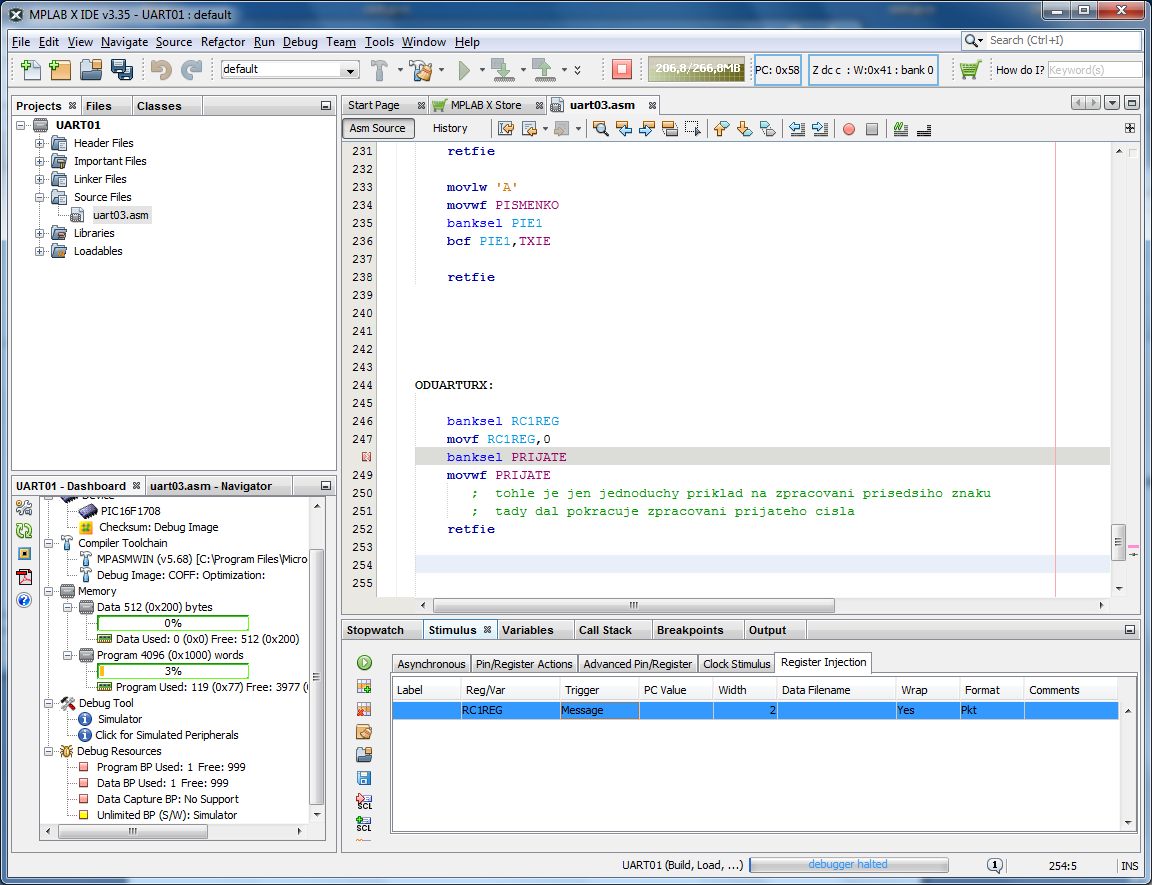
Dole se objeví okno Stimulus. To, co potřebujeme, je Register injection.



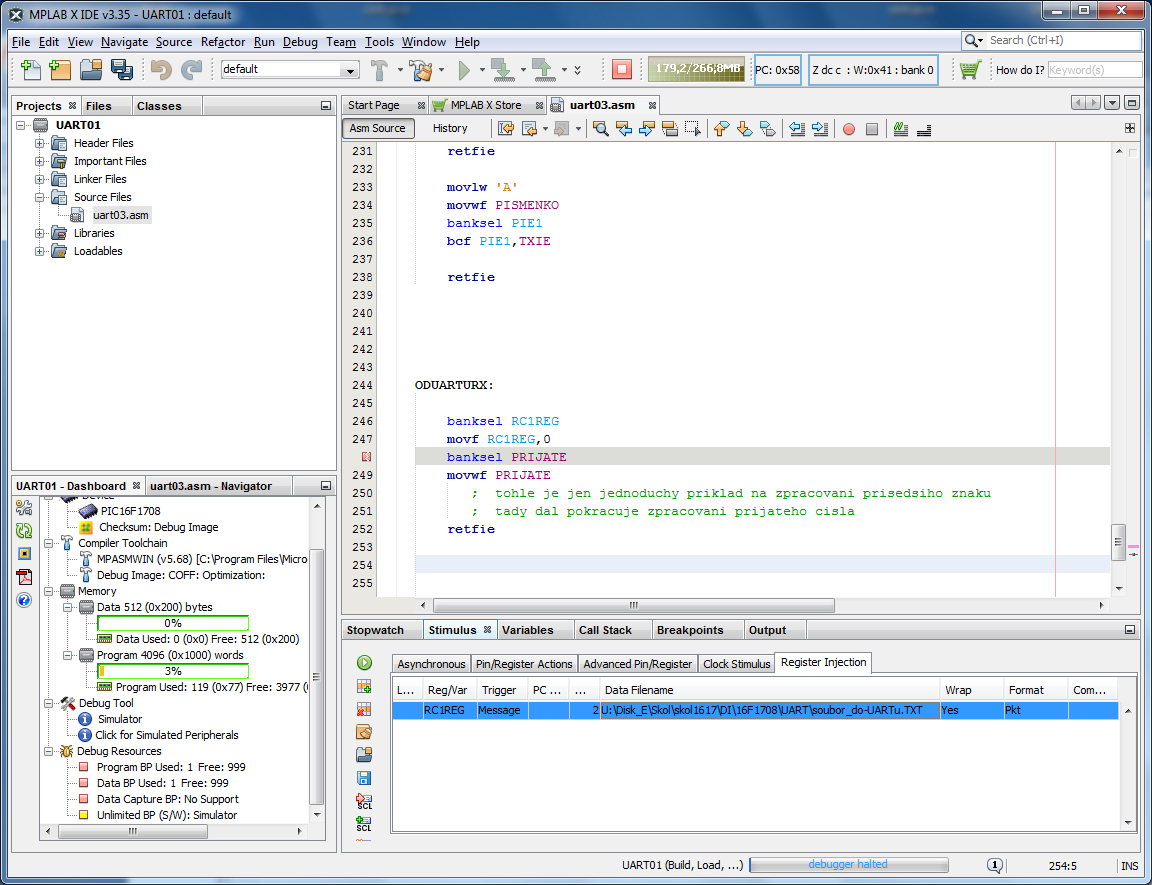
Cvakneme na sloupeček Reg/Var a tam najdeme náš přijímací registr RC1REG.



Ve sloupečku trigger nastavíme message

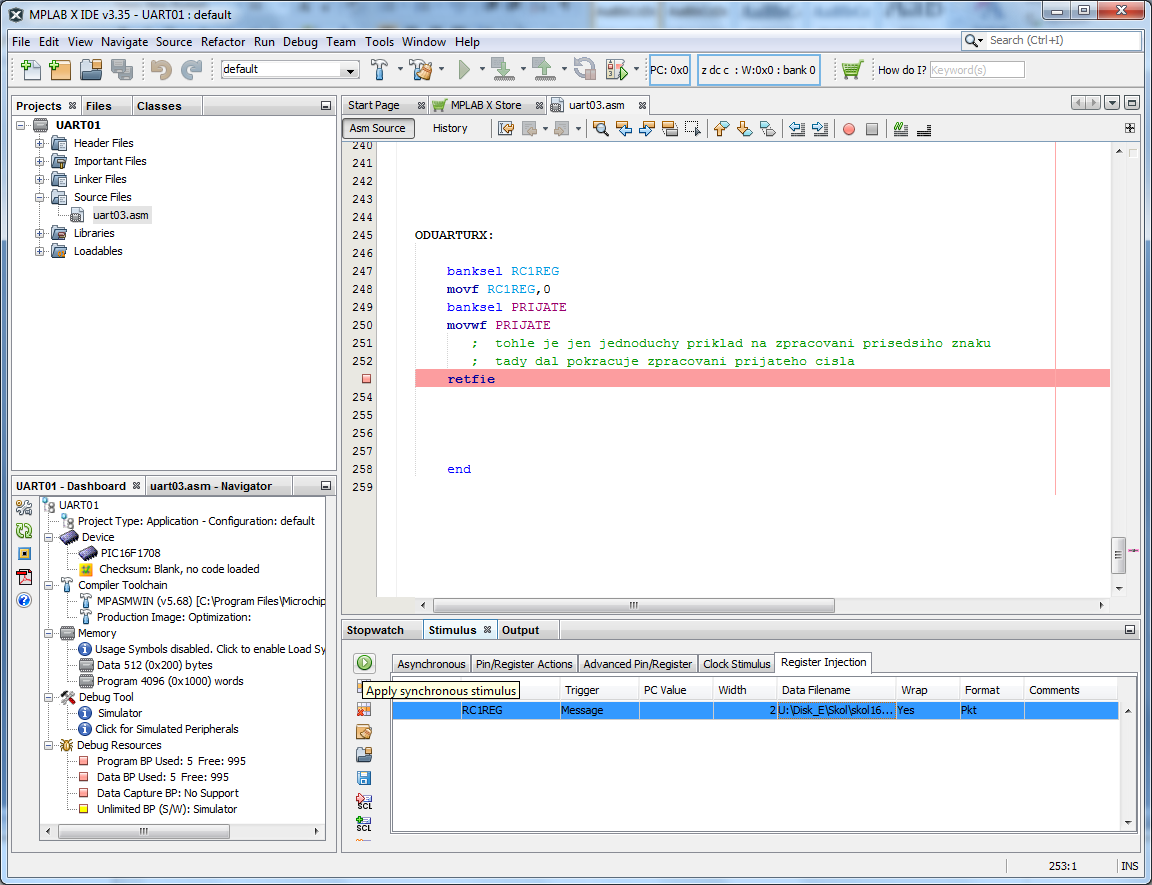


A do „Data filename“ zadáme cestu k file, ve kterém máme příkazy pro UART – tedy hlavně data, která chceme poslat do UARTu. Příklad syntaxe file je v  soubor\_do\_UARTu.txt



Dále cvakáme na zelenou šipku vlevo u horního okraje okna stimulus

„apply synchronous stimulus“



Pustíme si také stopwatch, abychom viděli, jak často se volá přerušení.

No a F5. Breakpoint máme stále na retfie podprogramu ODUARTURX

A pak je to hračka. Zmáčkneme F5, program se „kousne“ na breakpointu, podíváme se do WREG, jaký znak přišel, a do stopwatch, za jaký čas. Stejně jako v minulých příkladech si ve Variables nechte zobrazit i sloupec „char“ , abyste mohli znaky pohodlně číst.

File soubor\_do-UARTu.TXT je self-explanatory, podívejte se do něj a vše musí být jasné.

Pokud jsme ve sloupci Wrap nastavili YES, file soubor\_do-UARTu.TXT se poté, co program dosáhne jeho konce, začne provádět opět od začátku, stále dokolečka.