Cvičení 3

Práce s datasheetem
Přerušení
Příklad C

Práce s datasheetem

MCU jsou obecně velmi složitá zařízení se kterými se často pracuje na té nejnižší úrovni – tedy zadáváním instrukcí přímo procesoru. Komunikace s okolím, nastavení se řeší skrze desítky registrů. Vzhledem k tomu, že terminologie se mezi výrobci liší, že je registrů velké množství a že se výrobci snaží implementovávat různé nové technologie a funkce, je téměř nemožné si všechno pamatovat. Jako referenční příručky slouží takzvané datasheety [= datový list], ve kterých jsou vysvětleny a zapsány všechny funkce MCU. Datasheety se dají stáhnout ze stránek výrobce, v našem případě z adresy:

http://www.st.com/stonline/products/literature/ds/11381.pdf.

Podívejme se nyní na jeho obsah[1]:

1. **Introduction** – úvod;
2. **Pin description** – popis pinů, vývodů MCU;
3. **Register & Memory map** – popis základních registrů (CC, A, X,..) a rozdělení paměti;
4. **FLASH programm memory** – programová paměť typu FLASH, její popis, metody ukládání programu,..;
5. **Data EEPROM** – datová paměť typu EEPROM;
6. **Central processing unit** (CPU) – popis CPU a základních registrů a jejich funkčnosti;
7. **Supply, reset and clock management** – popis možností získávání hodinového signálu, funkčnost resetu a možnosti napájení;
8. **Interrupts** – popisuje možnosti přerušení;
9. **Power saving modes** – jak jsme již zmiňovali, mají dnešní MCU několik režimů pro nižší spotřebu. Tato část popisuje jejich možnosti.
10. **I/O ports** – popisuje porty A a B;
11. **On-chip peripherals** – popisuje periferie zabudované v čipu MCU, jako například A/D převodník.
12. **Instruction set** – popisuje způsoby programování MCU, typy adresování apod..
13. **Electrical chraracteristics** – definuje napájení ve vztahu ke garantované funkčnosti MCU a frekvenci jádra. Také obsahuje informace o napětí a proudech na výstupech a další...
14. **Package characteristics** – popisuje pouzdro MCU;
15. **Device configuration** – nejde o konfiguraci MCU, ale o možnosti dodávky, typy MCU, objednávku a další obchodní informace;
16. **Known limitations** – obsahuje informace o známých omezeních;
17. **Revision history** – tabulka s označenými verzemi datasheetu a popisem změn mezi jednotlivými verzemi.

Přerušení

Přerušení (interrupt) se obecně používá v případě, že je nutné přerušit chod hlavního programu z důvodu nutnosti vykonat určité instrukce v čase, kdy nastanou. Tedy například při normálním chodu MCU je stlačeno tlačítko, které má uložit stav nějaké proměnné do paměti. Anebo když je třeba

1 V této sekci vědomě porušíme zvyklost v uvádění překladů z slovníku. Budeme za pomlčkou uvádět jen vysvětlení, nikoliv překlad. Toho se dopouštíme s vědomím, že většina pojmů je již známa a jejich zdlouhavé vysvětlování by bylo jen na škodu čtivosti textu.

vzbudit MCU ze stavu nižší spotřeby energie.
ST7FLITE39 má poměrně širokou nabídku přerušení, jak ukazuje následující tabulka:

Table 6. Interrupt Mapping

N°	Source Block	Description	Register Label	Priority Order	Exit from HALT	Address Vector
	RESET	Reset	N/A	Highest Priority	yes	FFFEh-FFFFh
	TRAP	Software Interrupt			no	FFFCh-FFFDh
0	AWU	7 Interrupt	AWUCSR		yes[1)]	FFFAh-FFFBh
1	ei0	External Interrupt 0	N/A		yes	FFF8h-FFF9h
2	ei1	External Interrupt 1				FFF6h-FFF7h
3	ei2	External Interrupt 2				FFF4h-FFF5h
4	ei3	External Interrupt 3				FFF2h-FFF3h
5	LITE TIMER	LITE TIMER RTC2 interrupt	LTCSR2		no	FFF0h-FFF1h
6	LINSCI	LINSCI Interrupt	SCICR1/ SCICR2		no	FFEEh-FFEFh
7	SI	AVD interrupt	SICSR		no	FFECh-FFEDh
8	AT TIMER	AT TIMER Output Compare Interrupt or Input Capture Interrupt	PWMxCSR or ATCSR		no	FFEAh-FFEBh
9		AT TIMER Overflow Interrupt	ATCSR		yes[2)]	FFE8h-FFE9h
10	LITE TIMER	LITE TIMER Input Capture Interrupt	LTCSR		no	FFE6h-FFE7h
11		LITE TIMER RTC1 Interrupt	LTCSR		yes[2)]	FFE4h-FFE5h
12	SPI	SPI Peripheral Interrupts	SPICSR	Lowest Priority	yes	FFE2h-FFE3h
13	AT TIMER	AT TIMER Overflow Interrupt 2	ATCSR2		no	FFE0h-FFE1h

Note 1: This interrupt exits the MCU from "Auto Wake-up from Halt" mode only.
Note 2: These interrupts exit the MCU from "ACTIVE-HALT" mode only.

Tabulka 1: Tabluka přerušení (datasheet ST7LITE3, strana 36)

Tabulka je řazena dle priorit (tedy v případě, že při vyřizování jednoho přerušení nastane další a má vyšší prioritu (je výše v tabulce), bude vykonáno okamžitě). Nejvyšší prioritu má RESET, následuje softwarové přerušení nejvyšší priority TRAP, AWU (auto wake up, automatické probuzení) přerušení obsluhující přechod z režimu nízké spotřeby. Další jsou vnější přerušení (= external interrupt), přerušení vyvolaná vnějšími událostmi (např. stlačením tlačítka). A těmi se budeme v příkladě C zabývat.

Příklad C

Zadání

Napište program, který obslouží jednoduché přerušení a:

1. *bude načítat počet přerušení do proměnné*
2. *bude upozorňovat na přerušení bliknutím diod*

Jako zdroj přerušení použijte tlačítko na PB.

Rozbor

K vyřešení tohoto problému budeme muset nahlédnout do datasheetu, abychom zjistili jakým způsobem nastavíme MCU, aby přijímalo přerušení. Strana 37 nám vysvětluje existenci registrů EICR [= External interrupt control register = vnější přerušení ovládací registr] a EISR [= External interrupt selection register = vnější přerušení vybírací registr]. Tlačítka umístěná na REVA kitu jsou na PB5 a 6, což dle schématu níže odpovídá ei2:

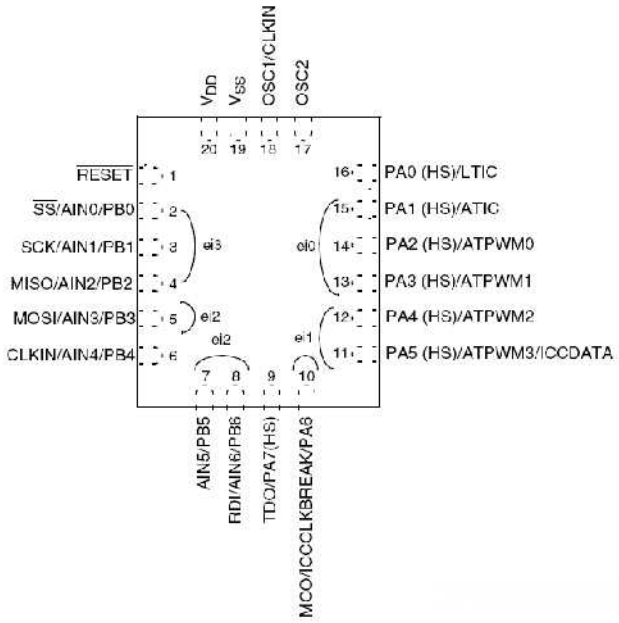


Schéma 1: Rozdělení přerušení na vývodech z pouzdra MCU (Datasheet ST7LITE3 strana 6)

To znamená, že chceme aktivovat ei2 a propojit ho na PB 5 a 6. Nejdříve se budeme zabývat EISR a nastavíme ei2 na PB5. Dle tabulky:

ei21	ei20	I/O Pin
0	0	No interrupt
0	1	PB3
1	0	PB5
1	1	PB6

Tabulka 2: Nastavení EISR

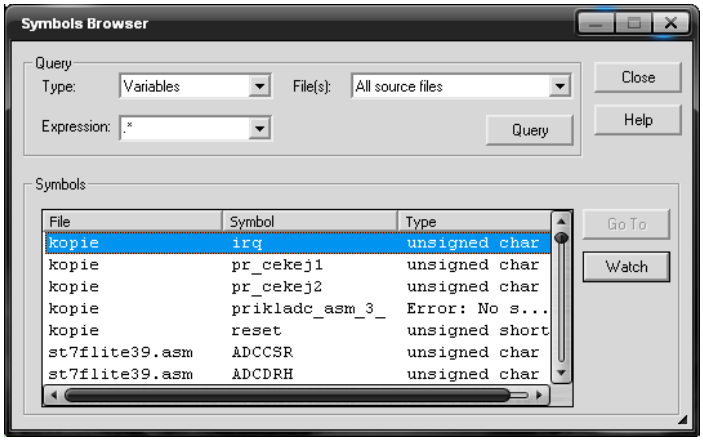
Nastavíme tedy EISR registr na hodnotu 00100000.
Registr EICR definuje na jakou změnu má přerušení reagovat, jestli na vzestupnou (Rising edge), nebo sestupnou (Falling edge) hranu. Zvolíme sestupnou hranu stejným způsobem z tabulky, jako tomu bylo výše. Hodnota EICR vyjde: 00100000. Je důležité nastavit sestupnou hranu i u tlačítka na REVA kitu, pomocí switche umístěného vedle tlačítek.
Nyní se můžeme pustit do vlastního programu.

Verze 1

```
;-------------- definice -------------------------
PADDR_1     equ %10011111      ; pro ladění, zachovaná komunikace
PBDDR_1     equ %00000000
PBOR_1      equ %11111111
EISR_1      equ %00100000
EICR_1      equ %00100000
;-------------- blok paměti RAM -------------------------
      BYTES
      segment 'ram0'
pr_cekej1  DS.B
pr_cekej2  DS.B
irq  DS.B
```

```
;-------------- vlastni program -------------------------
      WORDS
      segment 'rom'
.main       LD A,#PADDR_1
      LD PADDR,A              ; nastaveni portu A.0 jako výstup
      LD A,#PBOR_1
      LD PBOR,A               ; nastaveni portu B pro čtení přepínačů a tlačítek
      LD A,#PBDDR_1
      LD PBDDR,A              ; nastaveni portu B pro čtení přepínačů a tlačítek
      LD A,#EISR_1
      LD EISR,A               ; nastaveni EISR pro čtení přerušení z PB 5
      LD A,#EICR_1
      LD EICR,A               ; nastavení EICR pro detekci sestupné hrany
      LD Y,#1
      LD irq,Y                ; vloží číslo 1 do proměnné „irq“, kterou budeme
                              ; používat pro načítání počtu stisknutí tlačítka
.normal
      WFI                     ; WFI [= wait for interrupt = čekej na přerušení],
                              ; příkaz jádru, který uvede MCU do režimu „standby“,
                              ; tedy zastaví CPU a čeká na přerušení, které by MCU
                              ; „probudilo“. Také nastavuje flag I CC registru na 0 a
                              ; tím aktivuje přerušení
      JP normal               ; zacyklení
;-------------- podprogram čekání -----------------------
(...)                         ; známe, neuvádím
;------------- obsluha přerušení --------------------------
.dummy   iret                 ; prázdný podprogram, obsluha přerušení, která
                              ; nechceme vyřizovat
.tlacitko                     ; podprogram obsluhující naše přerušení
      INC irq                 ; INC [= increment = zvětši o jedna, inkrementuj], přičte
                              ; jedničku k proměnné irq
      IRET                    ; ukončí obsluhu přerušení a navrátí se do hlavního
                              ; programu
;------------- vektory přerušení --------------------------
            segment 'vectit'
      DC.W    dummy     ; 13 přerušení časování
      DC.W    dummy     ; 12 přerušení komunikace po SPI
      DC.W    dummy     ; 11 přerušení časování
      DC.W    dummy     ; 10 přerušení časování
      DC.W    dummy     ; 9 přerušení časování
      DC.W    dummy     ; 8 přerušení časování
      DC.W    dummy     ; 7 přerušení detekce napájení
      DC.W    dummy     ; 6 přerušení komunikace po LINSCI
      DC.W    dummy     ; 5 přerušení časování
      DC.W    dummy     ; 4  EI3
      DC.W    tlacitko  ; 3  EI2 – nasměrované na obslužný podprogram „tlacitko“
      DC.W    dummy     ; 2  EI1
      DC.W    dummy     ; 1  EI0
      DC.W    dummy     ; 0 přerušení AWU
      DC.W    dummy     ; T přerušení TRAP
.reset DC.W    main      ; R přerušení RESET
      END
```

Pokud takový program zadáme, nebude MCU na přerušení z tlačítka reagovat jinak, než přičtením 1 k hodnotě „irq“. A tu musíme nějak sledovat. Proto spustíme debugger, z menu „view“ vybereme „Symbols browser“. V okně jako „Type“ zvolíme „Variables“ a zadáme „Query“ [= dotaz].



Screenshot 1: Symbols browser

Vybereme naši proměnnou „irq“ a klikneme na „watch“. V hlavním okně se nám otevře tabulka „watch“, kde bude proměnná irq a její hodnota (pokud se tabulka neobjeví, vyvoláme ji z menu „view“>“watch“). Potom ještě v menu pravého tlačítka myši zvolíme „Display all“ > „Decimal“ abychom počet stisknutí viděli jako desítkové číslo. Spustíme-li program, několikrát zmáčkneme tlačítko BT6[2], program pozastavíme, uvidíme v tabulce počet stisknutí + 1 (Vzpomínáte? Zadali jsme 1 jako výchozí hodnotu.). Přerušení funguje.

Verze 2 – Příklad C

Přidat do stávajícího programu ještě signalizaci pomocí LED bude už velmi jednoduché. Stačí si nadefinovat ještě dvě konstanty:

```
blik        equ %11110101      ; připravená konstanta pro bliknutí po stisknutí tlačítka
nic         equ %11111111      ; připravená konstanta pro zhasnutí LEDek
```

a poněkud rozšířit obslužný program přerušení:

```
.tlacitko                      ; podprogram obsluhující naše přerušení
      INC irq                  ; INC [= increment = zvětši o jedna, inkrementuj], přičte
                               ; jedničku k proměnné irq
      PUSH A                   ; při práci s podprogramem nevíme, co je uloženo v
                               ; registrech A, X a Y, proto je důležité (samozřejmě ve
                               ; složitějších programech, v našem případě to nehraje
                               ; velkou roli) použít příkaz PUSH, který uloží hodnotu
      PUSH Y                   ; registru do paměti
      LD A,#blik
      LD PADR,A                ; vložíme „blik“ do PADR, rozsvítíme LEDky
      CALL wait                ; počkáme (jinak by bylo bliknutí jen velmi krátké,
                               ; neznatelné)
      LD A,#nic
      LD PADR,A                ; zhasneme LEDky
      POP A                    ; vyvoláme obsahy A a Y z paměti
      POP Y
      IRET                     ; ukončí obsluhu přerušení a navrátí se do hlavního
                               ; programu
```

Přerušení pracuje velmi rychle a přesně, při správném nedomáčknutí tlačítka je vidět, že přerušení

2 Jistě vám přijde zvláštní, proč celou dobu mluvíme o nadefinování PB5 jako vstupu pro tlačítko a najednou máme mačkat tlačítko PT6. To je způsobeno pravděpodobně konstrukcí REVA kitu.

zareaguje třeba několikrát.