MSP430SBC

Jednodeskový počítač

(Single Board Computer)

s mikrokontrolérem

MSP430F149

autor dokumentu: Hynek Sladký
verze dokumentu: 1.0
poslední úpravy: 11.3.2013

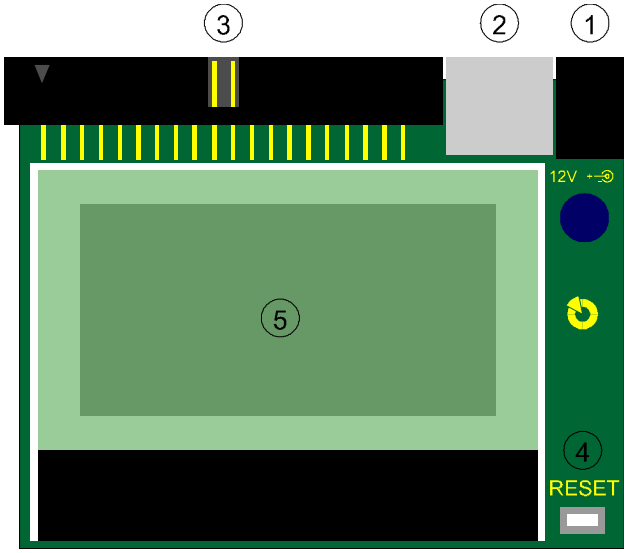
Obsah

1. Začínáme

Než poprvé zapojíte a spustíte výukový počítač MSP430SBC, přečtěte si tento návod, který vám pomůže počítač lépe používat.

1.1. Seznámení s počítačem

Počítač MSP430SBC je dodáván spolu s napájecím zdrojem a návodem, který právě držíte v rukou. Počítač obsahuje LCD displej (5), konektor pro připojení napájecího zdroje (1), konektor pro připojení PS/2 klávesnice (2) (ta není součástí balení) a konektor pro připojení vašich vlastních obvodů (3), které můžete pomocí počítače řídit a ovládat. Pokud někdy dojde k zaseknutí počítače, můžete počítač restartovat stisknutím tlačítka RESET (4).



Ilustrace 1: Hlavní části počítače

1.2. První zapnutí

Než začneme počítač používat, připojíme klávesnici a napájecí zdroj. Teprve nyní můžeme zasunout napájecí zdroj do zásuvky. Na displeji se zobrazí úvodní text:

```
MSP430SBC BASIC v1.0
(c) Hynek Sladky 2013
Free: 1024 RAM, 32768 Flash
>
```

Na poslední řádce je zobrazený znak >, kterým počítač hlásí, že je připravený zpracovat naše příkazy. Můžeme třeba napsat:

```
>PRINT 1+1
```

a po stisknutí klávesy Enter počítač napíše výsledek

```
2
```

1.3. Seznámení s jazykem BASIC

Pokud si slovo BASIC vyhledáme v anglicko-českém slovníku a dozvíme se, že to znamená „základní“. Můžeme se také podívat do roku 1964, kdy tento jazyk vznikl, a zjistíme, že název **BASIC** je vlastně zkratka pro základní vlastnost tohoto jazyka: **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode, a to si můžeme volně přeložit jako „víceúčelový programovací jazyk pro začátečníky“. Při vzniku tohoto jazyka byl hlavní důraz kladený právě na jednoduchost použití; vznikl totiž pro studenty netechnických oborů,

aby se mohli i bez technických znalostí naučit programovat. V 70. a 80. letech 20. století se BASIC používal jako hlavní programovací jazyk v domácích počítačích a i dnes se tento jazyk stále používá, i když s daleko většími možnostmi (např. VisualBasic).

Teď se seznámíme s několika základními příkazy, abychom mohli vyzkoušet, co počítač umí. V předchozí kapitole jsme si už vyzkoušeli, že počítač umí počítat. Příkaz **PRINT** slouží k vypisování údajů na displeji. Uvedeme si teď pár příkladů:

```
>PRINT 1
1
>PRINT 2*3
6
>PRINT (3+11)/2
7
>PRINT 3*3,4*4
9     16
>PRINT "vysledek=";5*5
vysledek=25
```

Příkaz PRINT může vypsat víc údajů za sebou. Údaje oddělujeme zpravidla **čárkou**, tím je mezi jednotlivými údaji nechána mezera, aby byly lépe čitelné (předposlední příklad). K oddělení údajů můžeme použít také **středník**, pak údaje následují hned za sebou (poslední příklad). Jako údaj může být buď číslo, matematické operace nebo i matematické funkce, ale také text v **uvozovkách**.

Kromě čísel můžeme také používat proměnné, které se používají pro uložení číselných hodnot a ve výpočtech.

```
>I=11:K=5:L=K*I:PRINT L
55
```

Programovací jazyk v počítači MSP430SBC umí počítat pouze s **celými čísly**, a to v rozsahu **-2147483648** až **2147483647**. Proto nemůžeme ve výpočtech používat větší čísla než je uvedený rozsah. Pokud při výpočtu dostaneme výsledek mimo tento rozsah, dojde k přetečení a výsledek je znehodnocený — je to výsledek, který naprosto neodpovídá matematickým pravidlům.

V příkladu také vidíme, že pokud chceme na jednom řádku zapsat víc příkazů za sebou, musíme je od sebe oddělit dvojtečkou.

```
>PRINT 100000*100000
1410065408
```

Také je důležité si uvědomit, že i při dělení dostáváme jako výsledek celá čísla, proto další výpočty s takovým výsledkem jsou více či méně zkreslené. Zkusíme si to ukázat s pomocí jednoduchého programu:

```
>FOR A=1 TO 9:PRINT A,A/3,(A/3)*3:NEXT
1     0     0
2     0     0
3     1     3
4     1     3
5     1     3
6     2     6
7     2     6
8     2     6
9     3     9
```

Na příkladu tak vidíme, jak funguje na počítači celočíselné dělení, proto si tuto skutečnost musíme při programování uvědomovat a brát ji v úvahu.

V posledním příkladu vidíme nový příkaz: cyklus **FOR** – **NEXT**, který počítá v proměnné (zde A) od jedné meze do druhé (zde od 1 do 9) a postupně provádí příkazy uvedené v těle cyklu (mezi příkazy FOR a NEXT). Cyklus FOR — NEXT je vhodné používat tam, kde dopředu víme, kolikrát má daný

cyklus proběhnout nebo tam, kde potřebujeme proměnnou v každém cyklu zvětšit o 1.

BASIC má k dispozici ještě další dva typy cyklů, které už neumí automaticky pracovat s proměnnou. Jde o cyklus WHILE — WEND a REPEAT — UNTIL.

Cyklus **WHILE** — **END** je tzv. cyklus s podmínkou na začátku. Dokud je hodnota výrazu za slovem WHILE nenulová, tělo cyklu se vykonává. Pokud je výraz nulový už před prvním cyklem, cyklus se nevykoná ani jednou. Zkusíme teď napsat cyklus, který se chová stejně jako FOR:

```
>A=1:WHILE A<=9:PRINT A:A=A+1:WEND
1
2
3
4
5
6
7
8
9
```

Jak vidíme, proměnná A nabývá postupně hodnot od 1 do 9, stejně jako v předchozím příkladě. Akorát jsme to museli popsat trochu složitěji...

Cyklu **REPEAT** — **UNTIL** se také říká cyklus s podmínkou na konci. Nejprve se jednou vykoná tělo cyklu, teprve pak se poprvé vyhodnotí výraz za slovem UNTIL. Cyklus se opakuje, dokud je hodnota výrazu nulová; je to opačná podmínka než u cyklu WHILE!

```
>A=1:REPEAT:PRINT A:A=A+1:UNTIL A>9
1
2
3
4
5
6
7
8
9
```

Teď už umíme použít všechny cykly. Jak vidíme, můžeme tady dostat stejný výsledek s použitím kteréhokoli cyklu. To ale neplatí vždy: u složitějších programů můžeme využít výhody konkrétního typu cyklu.

Vyzkoušíme si teď příkaz **INPUT** pro načítání hodnot do proměnných. Tento příkaz se používá tehdy, když je potřeba hodnotu proměnné zadat podle aktuální potřeby uživatele.

```
>INPUT "Zadej hodnotu:",A
Zadej hodnotu:
```

Teď musíme zadat číslo a stisknout klávesu Enter. Zadané číslo se uloží do proměnné A. Za příkazem INPUT může následovat víc proměnných i textů:

```
>INPUT A,B,C
>INPUT "A:",A,"B:",B,"jeste C:",C
```

Tímto způsobem musíme do příkazu zapsat název každé proměnné, kterou chceme zadat. Pokud potřebujeme zpracovat řadu čísel, můžeme k tomu použít pole hodnot. To je podobné proměnným, ale pod jedním názvem se ukrývá větší počet proměnných, ke kterým přistupujeme pomocí „indexu“. Před prvním použitím musíme každé pole připravit — „alokovat“ — příkazem **DIM**.

V příkladu si připravíme pole H () pro 4 hodnoty:

```
>DIM H(4)
>H(0)=10
```

```
>H(1)=11
>H(2)=20
>H(3)=33
```

Hodnoty v poli můžeme zpracovat jednoduše v cyklu:

```
>FOR I=0 TO 3:H(I)=H(I)*10:NEXT
```

Dokonce si hodnoty pomocí cyklu můžeme nechat zadat včetně velikosti pole:

```
>INPUT "Pocet prvku:",N:DIM N(N):N=N-1:FOR I=0 TO N:INPUT
"Hodnota:",N:NEXT
```

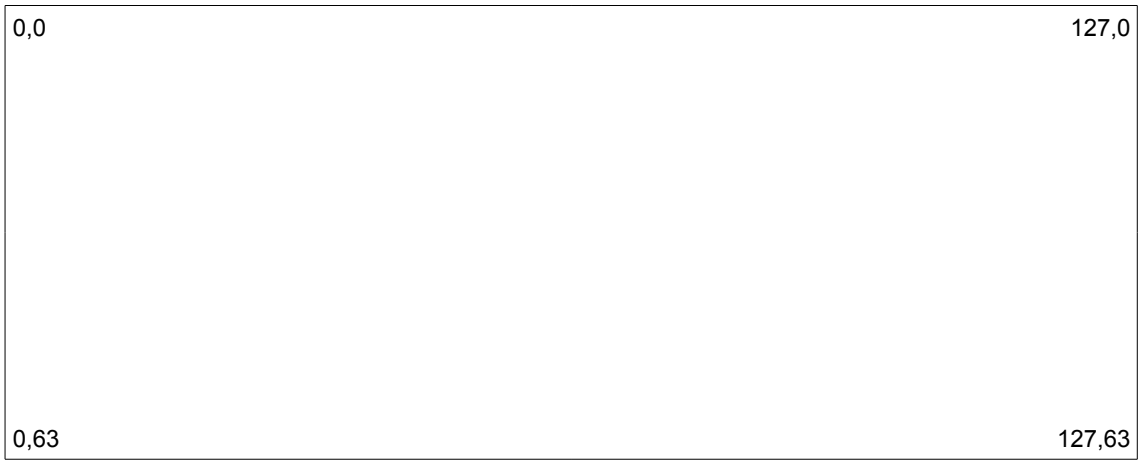
Občas potřebujeme zjistit, zda hodnota vyhovuje dalšímu výpočtu. Pak se nám bude hodit podmíněný příkaz IF.

```
>A=0:IF A=0 PRINT "A je nula."
A je nula.
>A=1:IF A=0 PRINT "A je nula."
```

Počítač MSP430SBC umí také kreslit body a úsečky:

```
>DOT 50,50
>LINE 0,0,127,63
```

První příkaz **DOT** nakreslí bod na souřadnicích 50,50. Příkaz **LINE** nakreslí úsečku z bodu 0,0 do bodu 127,63 — z levého horního rohu do pravého dolního rohu. Platné souřadnice jsou tedy od 0 do 127 pro šířku a 0 až 63 pro výšku:



Zkusíme si vykreslit víc čar na displeji s využitím cyklu:

```
>BPUT12:F.I=0TO31:LIN.I*4,0,127-I*4,63:NEXT:
F.I=0TO15:LIN.0,I*4,127,63-I*4:NEXT
```

Protože je na příkazové řádce hodně příkazů, musíme používat zkratky, aby se nám tam všechny příkazy vešly. Co přesně se vykreslí uvidíte na displeji.

1.4. První program

Zatím jsme si ukázali, jak zadávat příkazy na příkazovou řádku. Příkazy takto zadané jsou ihned vykonány. Pokud chceme něco rychle vyzkoušet, pak je toto nejjednodušší cesta, jak vidět ihned výsledek. Pro složitější operace však není moc praktické zadávat všechny příkazy znovu a znovu. Proto si teď ukážeme, jak začít psát skutečný program.

Programy v jazyku BASIC na počítači MSP430SBC mají číslované programové řádky. Pokud píšeme program, musí tedy každý programový řádek začínat číslem. Je dobrým zvykem číslovat řádky třeba po 10, abychom mohli později doplňovat a upravovat program (např. mezi řádky 50 a 60 lze dopsat dalších 9 řádků od 51 do 59). Zkusíme si teď napsat první program:

```
>10 PRINT "Ahoj"
```

Po stisku klávesy Enter se ale nic nestalo. Co s tím? Zadáme teď příkaz **RUN** a náš první program se spustí. Teprve pak na displeji uvidíme výstup:

```
Ahoj
```

Zkusíme si teď napsat krátký program na výpočet obsahu obdélníku:

```
>10 INPUT "Vyska:",A,"Sirka:",B
>20 IF A<=0 OR B<=0 GOTO 50
>30 PRINT "Obsah=";A*B
>40 END
>50 PRINT "Chyba zadani!"
RUN
Vyska:4
Sirka:5
Obsah=20
```

Na řádku 20 vidíme použití podmíněného příkazu pro kontrolu zadaných čísel. Funkce OR znamená „nebo“, takže příkaz za podmínkou je vykonaný pokud první nebo druhá část výrazu je platná. Pokud jsou obě čísla A i B větší než nula, není podmínka splněná a program pokračuje řádkem 30.

Na řádku 20 vidíme také nový příkaz: **GOTO**. Jako parametr má číslo řádku, na kterém má vykonávání programu pokračovat.

Další nový příkaz najdeme na řádku 40. Příkaz **END** ukončí vykonávání programu. Jiný způsob ukončení programu je po řádce 50, protože to je poslední řádek programu a žádné další příkazy už nenásledují.

Pokud si chceme program znovu prohlédnout, použijeme příkaz LIST.

```
>LIST
10 INPUT "Vyska:",A,"Sirka:",B
20 IF A<=0 OR B<=0 GOTO 50
30 PRINT "Obsah=";A*B
40 END
50 PRINT "Chyba zadani!"
>
```

Pokud máme dlouhý program, tak uvidíme na displeji jen posledních 7 řádků. Příkaz **LIST** můžeme proto použít s jedním nebo dvěma parametry, které říkají, jaké řádky chceme zobrazit.

```
>LIST 20,40
20 IF A<=0 OR B<=0 GOTO 50
30 PRINT "Obsah=";A*B
40 END
>
```

Pokud potřebujeme příkazy některého řádku opravit nebo změnit, můžeme řádek napsat celý znovu se stejným číslem nebo použijeme příkaz **EDIT** a napíšeme číslo řádku, který chceme změnit. Na příkazové řádce se nám ukáže zvolený řádek. Šipkami **vlevo** a **vpravo** můžeme posunovat kurzor, klávesami **Delete** nebo **Backspace** můžme mazat znaky, psaní textu znaky přidává na pozici kurzoru. Po stisku klávesy **Enter** je změněný řádek uložený do paměti.

Než začneme psát další program, musíme ten předchozí smazat z paměti příkazem **NEW**.

Pokud potřebujeme opakovaně vykonávat některou část programu, můžeme použít cyklus, jak jsme si ukázali dříve. Někdy to ale není vhodné, tak můžeme použít „podprogram“ — ten se dá „zavolat“ z libovolného místa programu. Opět budeme počítat obsah obdélníku, ale tentokrát přidáme i obsah čtverce.

```
>NEW
```

```
>10 INPUT "Vyska:",A,"Sirka:",B
>20 GOSUB 100
>30 INPUT "Strana:",A:B=A
>40 GOSUB 100
>50 END
>100 IF A<=0 OR B<=0 THEN 150
>110 PRINT "Obsah=";A*B
>120 RETURN
>150 PRINT "Chyba zadani!"
>160 END
```

Na řádcích 10 a 30 jsou příkazy pro načtení hodnot pro výpočet, řádky 20 a 40 obsahují volání podprogramu na řádku 100. Pro oba výpočty tak máme použitou stejnou část programu.

Podprogram je tedy společný pro všechny výpočty obsahu, které potřebujeme v našem programu provádět. Podprogramy tak šetří čas při psaní programu — společnou část píšeme jen jednou — a také šetří místo v paměti programu, takže pak můžeme psát delší a složitější programy.

Na řádku 100 vidíme alternativní zápis podmíněného příkazu **IF** – **THEN** a číslem řádku. Tento zápis je shodný s podmíněným příkazem **IF** – **GOTO**.

Teď si náš program ještě trochu vylepšíme, protože máme počítat obsahy různých obdélníků za domácí úkol. Abychom se nespletli při opisování rozměrů, napíšeme si je rovnou do programu. Tak je můžeme lépe zkontrolovat a případně včas opravit.

```
>10 REPEAT
>20 READ A,B
>30 GOSUB 100
>40 UNTIL FALSE
>50 END
>100 IF A<=0 OR B<=0 THEN 150
>110 IF A=B PRINT "Ctverec ";A;
>120 IF A<>B PRINT "Obdelnik ";A;"x";B;
>130 PRINT "Obsah=";A*B
>140 RETURN
>150 PRINT "Chyba zadani!"
>160 END
>200 DATA 5,4,13,11,6,6,7,9
>201 DATA 134,167,50,50
>202 DATA 0,0
```

Teď už máme program docela dlouhý a složitý, ale postupně jsme si vysvětlili, jak použité příkazy fungují. Zde nám přibyly příkazy READ a DATA.

Příkaz **DATA** slouží pro uložení číselných hodnot v programu. Příkaz **READ** pak tato data čte do zadaných proměnných.

K těmto dvěma příkazům patří také příkaz **RESTORE**, který nastaví pozici čtení dat příkazem READ.

Poslední příkaz, který si vyzkoušíme, je TRACE. Pokud potřebujeme vědět, jak je program nebo jeho část vykonávána, použijeme příkaz **TRACE ON** pro zapnutí „trasovacího“ výstupu a **TRACE OFF** pro jeho vypnutí. Napíšeme si krátký program a zkusíme, jak příkaz TRACE funguje.

```
>1 REM Pokusny program
>10 PRINT "TRACE TEST"
>20 TRACE ON 'zapnuti trace
>30 GOSUB 100
>40 GOTO 90
>90 TRACE OFF 'vypnuti trace
>99 END
>100 PRINT "PODPROGRAM"
```

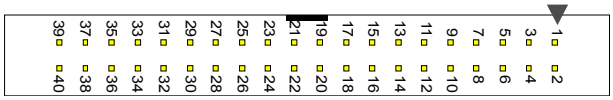
```
>110 RETURN
>RUN
[30][100]PODPROGRAM
[110][30][40][90]
>
```

V hranatých závorkách se vypisují čísla řádků, jak je program postupně vykonáván. Protože se k tomu přidávají také výstupy příkazů PRINT, je potřeba používat tento příkaz s rozmyslem na místech, která chceme ladit, protože jinak budeme mít na displeji promíchaná čísla řádků a výstupy a nakonec se v tom ani nevyznáme...

V příkladu si ještě můžeme všimnout na řádku 1 příkazu **REM**, nebo také na řádcích 20 a 90 a znaku „**apostrof**“. Obě formy příkazu se používají pro zápis poznámek k programu. Pokud píšeme delší a složitější programy, je vhodné si do programu psát komentáře, abychom i po čase pochopili, k čemu program slouží a jak funguje.

1.5. Připojení vlastních obvodů

Pokud chceme řídit nějaký vlastní obvod, můžeme ho připojit na konektor (3). Máme zde k dispozici 31 vývodů procesoru, které mohou plnit různé funkce podle nastavení příslušných registrů procesoru.



Ilustrace 2: Konektor pro připojení vlastních zařízení

Piny 1, 11, 21 a 31 poskytují napájení 3.3V, na piny 10, 20, 30 a 40 je připojený druhý pól napájení, signál GND. Piny 2 až 9 jsou připojené na port P6, piny 12 až 19 na port P1, piny 22 až 29 na port P3 a piny 32 až 39 na port P4. Přehledná tabulka rozmístění pinů a jejich funkcí je v kapitole 4.1.

V dalších programech budeme často používat čísla zapsaná v šestnáctkové (hexadecimální) soustavě, proto si teď ukážeme, jak vypadají a jak se používají.

Počítače pracují s čísly ve **dvojkové** soustavě. Takové číslo má pouze dvě číslice 0 a 1. Jedné dvojkové číslici říkáme bit. Počítač zpravidla pracuje s delšími čísly: binární číslo dlouhé 8 bitů — tomu říkáme **bajt** (anglicky se píše byte). Delší binární čísla označujeme jako **slova**. Podle toho také dělíme počítače na 8-bitové, 16-bitové, 32-bitové, 64-bitové. 32-bitový počítač umí jednou instrukcí zpracovat slovo dlouhé 32 bitů. MSP430SBC je 16-bitový počítač, umí tedy zpracovávat slova dlouhá 16-bitů. Pokud chceme po 16-bitovém počítači zpracovávat delší slova, tak k tomu potřebuje víc instrukcí.

Když bychom chtěli binárně zapsat 16-bitové číslo, mělo by 16 číslic: např. 1001101011101101. To není moc přehledné. Proto se při programování často používá zápis čísel v **šestnáctkové** (hexadecimální) soustavě. To je výhodné hlavně z toho důvodu, že se jednoduše převádí zápis mezi dvojkovou a šestnáctkovou soustavou. Každá čtveřice bitů se převede na jednu „hexa-číslici“. Té se odborně říká **nibl**. Protože šestnáctková číslice musí mít 16 hodnot, používáme pro zápis číslice 0 až 9 a pokračujeme písmeny A až F. Dohromady tedy máme 16 hodnot.

Binární	Hexadecimální	Dekadická	Binární	Hexadecimální	Dekadická
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

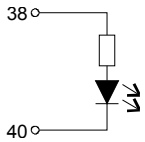
Vidíme, že do hodnoty 9 jsou hexadecimální i dekadická hodnota stejné. Teď si ukážeme několik příkladů binárního, hexadecimálního a „normálního“ dekadického zápisu větších čísel. Pro zjednodušení budeme používat zatím 8 bitů.

Binární	Hexadecimální	Dekadická	Binární	Hexadecimální	Dekadická
0001 0000	10	16	0001 0010	12	18
0010 0000	20	32	0010 0101	25	37
1000 0000	80	128	1000 1100	8C	140
1111 0000	F0	240	1111 1111	FF	255

Jak můžeme vidět v tabulce, převod mezi binárním a hexadecimálním tvarem čísla není složitý. Stačí si pamatovat, že 4 bity v niblu mají hodnotu 1, 2, 4 a 8. Pak už dokážeme napsat jakékoli číslo v binárním tvaru jako hexadecimální a opačně. Ještě si musíme říci, že bity se číslují od 0, takže 8-bitová hodnota má bity 0 až 7, 16-bitová hodnota má bity 0 až 15.

V následujících kapitolách budeme často zapisovat do registrů procesoru. Protože většinou budeme nastavovat jen některé bity, budeme pro to používat hexadecimální čísla. Počítač MSP430SBC potřebuje poradit, které číslo je hexadecimální: k tomu používáme znak **$**.

1.5.1. Připojení LED



Nejdřív zkusíme připojit třeba diodu LED. Zapojíme ji přes vhodný rezistor anodou na pin 38 a katodou na pin 40.

Všechny piny jsou po resetu nastaveny jako vstupní, proto musíme nejdřív použitý pin nastavit do výstupního režimu. Protože pin 38 je na portu P4 bit 6, používáme k ovládání hexadecimální konstantu $40.

```
>P4DIR=$40
```

Teď už můžeme měnit stav výstupu na pinu 39 zápisem

```
>P4OUT=$40
>P4OUT=$00
```

Pokud jsme připojili LED správně, vidíme, že se nejprve rozsvítí a pak opět zhasne. Teď si můžeme napsat program pro jednoduchý blikač:

```
>10 REM Blikac LED
>20 REM Nastaveni portu
>30 P4DIR=$40
>40 A=TICK
>50 REPEAT
>60 A=A+50
>70 WHILE A>TICK:WEND
>80 P4OUT=P4OUT XOR $40
>90 UNTIL FALSE
```

Funkce **TICK** vrací hodnotu tiků systémového časovače, která je 122-krát za sekundu zvětšena o 1. Program pak čeká, dokud se hodnota systémového časovače nezvětší o 50 (1s/122*50=409 milisekund). Pak změní stav bitu P4.6 a vše se opakuje znovu od začátku.

V dalším příkladu se naučíme řídit úroveň svitu LED pomocí pulzně-šířkové modulace (PWM) na časovači TimerB.

```
>P4DIR=$40
>P4SEL=$40
>TBCCTL6=$1C0
>TBCCR6=$8000
```

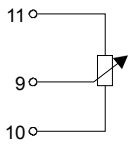
Nejprve se nastaví pin P4.6 jako výstupní, pak se povolí speciální funkce pro P4.6, kterou je podle tabulky 4 signál TB6 časovače TimerB. Další příkaz nastaví parametry signálu TB6 a hodnotu, při které

se dochází k překlopení výstupu PWM. Časovač TB počítá velkou rychlostí od 0 do $FFFF a stále dokola — to stihne 122x za sekundu. Při dosažení hodnoty 0 se nastaví výstup PWM do stavu 1 a při dosažení hodnoty nastavené v registru TBCCR6 se výstup PWM opět vynuluje. Proto v tomto příkladu svítí LED polovičním svitem než v předchozích příkladech, protože hodnota $8000 je v polovině mezi hodnotami 0 a $FFFF — polovinu doby LED svítí, polovinu doby je LED zhasnutá.

Zkusíme teď měnit registr TBCCR6 a budeme pozorovat, jak to ovlivňuje svícení LED:

```
>TBCCR6=$1000
>TBCCR6=$F000
```

1.5.2. Měření napětí



Připojíme si teď potenciometr 1k až 10k na piny 9, 10 a 11 a budeme měřit analogové napětí.

Pin 9 konektoru je připojený na port P6.7, který může mít také funkci vstupu 7 analogově–digitálního převodníku (ADC).

```
>ADC12CTL0=$8810
>ADC12CTL1=$0238
>ADC12MCTL0=$87
>P6SEL=$80
>ADC12CTL0=$8813
>PRINT ADC12MEM0
```

Uvedenými příkazy nastavíme převodník pro měření napětí 0V až 3.3V na kanálu 7. Pin P6.7 je nastavený na funkci ADC vstupu A7. Hodiny pro ADC jsou nastaveny na 4MHz, doba měření analogového signálu je 64µs. Pro lepší pochopení můžeme nahlédnout do kapitoly 4.3.5, kde jsou funkce jednotlivých bitů řídících registrů ADC popsány. Můžeme si zkusit měnit registr ADC12MCTL0 a vybrat jiný vstupní kanál INCHx pro měření nebo jiné referenční napětí SREFx. Pokud chceme použít vnitřní referenci, musíme ji zapnout a nastavit v registru ADC12CTL0.

Na displeji uvidíme číslo mezi 0 a 4095, které odpovídá nastavení potenciometru. Hodnotě 0 odpovídá napětí 0V a hodnotě 4095 odpovídá napájecí napětí procesoru, tedy 3.3V. Můžeme si teď napsat program, který bude fungovat jako voltmetr s rozsahem 0V až 3.3V:

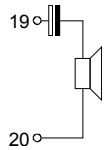
```
>10 REM nastaveni ADC
>20 ADC12CTL0=$8810
>30 ADC12CTL1=$0238
>40 ADC12MCTL0=$87
>50 P6SEL=$80
>60 REPEAT
>70 REM spusteni prevodu
>80 ADC12CTL0=$8813
>90 REM cekani na dokonceni prevodu
>100 WHILE ADC12CTL0 AND 1
>110 WEND
>120 V=ADC12MEM0/1241 ' cela cast hodnoty napeti
>130 U=ADC12MEM0 % 1241 ' desetinna cast hodnoty napeti
>140 U=(U*1000)/1241 ' prevod na hodnoty 0 az 999
>150 PRINT "Napeti:";V;"."; ' tisk cele casti
>160 IF U>=100 PRINT U ' tisk desetinne casti
>170 IF U>=10 PRINT "0";U
>180 IF U<10 PRINT "00";U
>190 UNTIL FALSE
```

Když už umíme řídit svit LED diody a měřit napětí na analogovém vstupu, můžeme obě funkce propojit a řídit svit LED diody podle změřeného napětí:

```
>10 REM Nastaveni PWM pro LED
```

```
>20 TBCCTL6=$1C0
>30 P4SEL=$40
>40 P4DIR=$40
>50 REM Nastaveni ADC
>60 ADC12CTL0=$8810
>70 ADC12CTL1=$0238
>80 ADC12MCTL0=$87
>90 P6SEL=$80
>100 REPEAT
>110 REM spusteni prevodu
>120 ADC12CTL0=$8813
>130 REM cekani na dokonceni prevodu
>140 WHILE ADC12CTL0 AND 1
>150 WEND
>160 TBCCR6=ADC12MEM0*16
>170 UNTIL FALSE
```

1.5.3. Generování tónů



V dalším příkladu si připojíme reproduktor a budeme generovat tóny. Na pinu 19 máme signál P1.7, což je zároveň signál TA2 časovače.

```
>P1SEL=$80
>P1DIR=$80
>TACTL=$2D0 ' hodiny 1MHz
>TACCTL2=$40
>TACCR0=2272 ' vystup 440Hz
>TACCR2=TACCR0/2
```

Pro časovač TA nastavíme hodiny 1MHz. Signál TA2 nastavíme pro PWM s periodou podle TACCR0 a střídou TACCR2, pomocí které můžeme regulovat hlasitost tónu.

Reproduktor můžeme připojit také přímo na signál TA0. Nastavování je jednodušší, ale nemůžeme regulovat hlasitost:

```
>P1SEL=$20
>P1DIR=$20
>TACTL=$2D0
>TACCTL0=$80
>TACCR0=1135 ' vystup 440Hz
```

Generovanou frekvenci si můžeme zkontrolovat čítačem nebo osciloskopem.

1.5.4. Graf napětí

Funkci měření napětí si rozšíříme o kreslení grafu. Nejprve budeme měřit maximální rychlostí, kterou nám program měření umožní.

```
>10 REM nastaveni ADC
>20 ADC12CTL0=$8810
>30 ADC12CTL1=$0238
>40 ADC12MCTL0=$87
>50 P6SEL=$80
>60 REM pamet pro vzorky
>70 DIM V(128)
>100 REPEAT
>110 FOR I=0 TO 127
>120 REM spusteni prevodu
>130 ADC12CTL0=$8813
```

```
>140 REM cekani na dokonceni prevodu
>150 WHILE ADC12CTL0 AND 1
>160 WEND
>170 REM ulozeni vysledku
>180 V(I)=63-ADC12MEM0/64
>190 NEXT
>200 REM zobrazeni vysledku
>210 BPUT 12
>220 FOR I=1 TO 127
>230 X=I-1
>240 LINE X,V(X),I,V(I)
>250 NEXT
>290 UNTIL FALSE
```

Po spuštění programu vidíme na displeji nějaké průběhy napětí. Pokud připojíme signál z generátoru, můžeme změřit, jakou maximální frekvenci dokáže takový „osciloskop“ zobrazit.

Příklad můžeme upravit třeba pro měření napětí např. každou sekundu, pokud potřebujeme zobrazovat pomalejší průběhy. Abychom nemuseli čekat tak dlouho na zobrazení jednoho průběhu, musíme upravit zobrazování tak, aby se průběh posunoval s každým změřeným a přidaným bodem.

```
>10 REM nastaveni ADC
>20 ADC12CTL0=$8810
>30 ADC12CTL1=$0238
>40 ADC12MCTL0=$87
>50 P6SEL=$80
>60 REM pamet pro vzorky
>70 DIM V(128)
>80 T=TIME
>90 I=0
>100 REPEAT
>110 REM posunuti všech vzorku
>110 FOR I=1 TO 127
>120 V(I-1)=V(I)
>130 NEXT
>140 REM cekani na další sekundu
>150 T=T+1
>160 WHILE T<TIME:WEND
>170 REM spusteni prevodu
>180 ADC12CTL0=$8813
>190 REM cekani na dokonceni prevodu
>200 WHILE ADC12CTL0 AND 1:WEND
>210 REM ulozeni vysledku
>220 V(127)=63-ADC12MEM0/64
>230 REM zobrazeni vysledku
>240 BPUT 12
>250 FOR I=1 TO 127
>260 X=I-1
>270 LINE X,V(X),I,V(I)
>280 NEXT
>290 UNTIL FALSE
```

Tento poslední příklad funguje jako hodně pomalý osciloskop. Můžeme ho ale použít třeba na měření napětí akumulátoru při nabíjení nebo teploty nebo jiných veličin, které se mění pomalu. Funkce **TIME** vrací počet sekund od zapnutí nebo resetu počítače. Pomocí této funkce můžeme časovat dlouhé časy přehledněji než s funkcí TICK, protože vidíme hodnotu času v sekundách.

2. Programovací jazyk BASIC

Předlohou pro BASIC v počítači MSP430SBC je „Butterfly Basic“ Paula Curtise, který sám vychází z BASICu na počítačích Acorn Atom. Více informací včetně zdrojového kódu lze najít zde: http://www.rowley.co.uk/msp430/basic.htm

V dalším textu si nejprve řekneme o datech, která mohou mýt programem v BASICu zpracovávaná. Pak si projdeme všechny příkazy, kterým interpretr BASICu rozumí. Ty můžeme rozdělit do 3 skupin:

- příkazy, které lze používat pouze v příkazové řádce programu
- příkazy, které jdou použít jak na příkazové řádce tak ve vlastním programu
- funkce vracející číselnou hodnotu

2.1. Poznámky k formátu následujících kapitol

Nyní se seznámíme s některými pojmy, které budeme v dalším textu používat.

Parametry příkazů a funkcí jsou popisované textem v ostrých závorkách: **<**parametr**>**

Parametry, které jsou volitelné (mohou nebo nemusejí být použité), jsou uváděny v hranatých závorkách: **[**<volitelný_parametr>**]**

Pokud je jako parametr očekávaná **<proměnná>**, nebo číselný **<výraz>**, případně **<text>**, je to uvedeno u popisu každé jednotlivé funkce.

Některé parametry jsou pro pochopení popsány textem, např. **<kanál>**, **<adresa>**, **<řádek>**... Program v těchto místech vždy očekává výraz, pro který vypočítá číselnou hodnotu. Hodnoty <adresa> jsou v rozsahu 0 až 65535, hexadecimálně $0 až $FFFF. Hodnoty parametru <řádek> mohou být od 1 do 65536. Hodnoty parametru <kanál> jsou shrnuty v tabulce.

<kanál>	**Význam**
1	LCD pro výstup, PS/2 klávesnice pro vstup
2	BSL piny P1.1 a P2.2 (115.2kBd)
3	UART0
3	UART1

Tabulka 1: Komunikační kanály

V popisech cyklů se používá označení **<tělo cyklu>** pro příkazy, které se během opakování cyklů budou vykonávat.

Pokud je někdy výběr z více možností, jsou od sebe odděleny znakem **|**: **ON|OFF** nebo **[HEX|BIN]**.

2.2. Datové typy

Tato kapitola vás seznámí s tím, jaká data umí program v BASICu zpracovávat.

2.2.1. Čísla

BASIC v počítači MSP430SBC pracuje s 32-bitovými celými čísly, která mohou nabývat hodnot v rozsahu -2147483648 až 2147483647.

Hodnoty je možné zapsat také v šestnáctkové (hexadecimální) soustavě. Takto zapsané číslo začíná znakem $ a obsahuje kromě číslic 0 až 9 také písmena A až F: např. $12ABCDEF. Rozsah hexadecimálních čísel je od $0 fo $FFFFFFFF. Hexadecimální čísla jsou výhodná pro nastavování registrů procesoru a pro práci se strojovým kódem.

Použít můžeme také zápis znaku uzavřeného v apostrofech: 'K'. Takto zapsaný výraz se převede na číslo podle ASCII kódu znaku.

2.2.2. Text

Příkazy PRINT a INPUT umožňují použití textů. Text je uzavřený v uvozovkách a může obsahovat libovolné znaky, např. "toto je text".

2.2.3. Proměnné

Pro uložení pracovních hodnot v programu slouží 26 proměnných, označených písmeny A až Z.

2.2.4. Pole proměnných

V programu můžeme také používat pole proměnných, pro ty jsou vyhrazeny identifikátory A() až Z(). V závorkách je uvedený index prvku pole. Před použitím se musí pole alokovat příkazem DIM; parametrem příkazu je počet prvků pole. Pokud má pole A() například 4 prvky, pak index může být od 0 do 3. Index mimo alokovaný rozsah způsobí vypsání chybové hlášky. Každý prvek pole se chová jako samostatná proměnná s 32-bitovým rozsahem hodnot.

2.2.5. Vektory bytů a slov

Pokud potřebujeme uložit větší množství bytů do paměti, můžeme použít bytový vektor. Pro přístup do bytového vektoru potřebujeme znát adresu začátku vektoru a index. Inicializace bytového vektoru může vypadat například takto:

```
>DIM V(4)
>A=VARPTR V(0)
```

Nyní máme 4-prvkové pole 32-bitových hodnot V() a jeho adresu uloženou v proměnné A. Pole V() má velikost 4x4 byty, tj. 16 bytů. K jednotlivým bytům teď můžeme přistupovat pomocí vektoru:

```
>PRINT A?0
0
>A?1=33
>A?2=A?3+A?5
>FOR I=0 TO 15:A?I=0:NEXT
```

Vektor 32-bitových slov se odlišuje znakem (a samozřejmě velikostí prvku):

```
>A!0=123456
>A!1=A!2
```

Vektory jsou podobné funkcím PEEK, POKE apod. Rozdíl je v tom, že vektory používají adresu prvního prvku a index, kdežto u funkcí typu PEEK, POKE se musí adresa každého prvku vypočítat.

2.2.6. Výrazy

Výrazy jsou čísla, proměnné, aritmetické operace nebo funkce, jejichž výsledkem je vždy číselná hodnota. V následující tabulce jsou vypsány všechny operace, které BASIC nabízí. Jsou seřazeny podle priority zpracování od nejnižší po nejvyšší. Jednotlivé skupiny priorit jsou označeny číslem od nejnižší 10 až po nejvyšší prioritu 60. Při výpočtu jsou nejprve vypočítány operace, jejichž priorita je vyšší. Výrazy s operacemi stejné priority jsou počítány zleva doprava.

Výsledek operací porovnání jsou hodnoty TRUE nebo FALSE. Jejich číselná hodnota je -1 a 0 (viz další kapitola).

Operace	Priorita	Popis	Příklad
AND	10	Bitový součin dvou hodnot; kde jsou oba bity nastavené, tam je i výsledný bit nastavený	PRINT BIN 7, BIN 5, BIN 7 AND 5
BIC		Nulování bitů z první hodnoty maskou z druhé hodnoty; kde má maska nastavené bity, tam jsou bity výsledku nulové	PRINT BIN 7 BIC 5, BIN 5 BIC 7
OR		Bitový součet dvou hodnot; pokud je aspoň jeden z bitů nastavený	PRINT BIN 5 OR 6

Operace	Priorita	Popis	Příklad
XOR		Bitový exkluzivní součet; pokud jsou bity rozdílné, jsou bity výsledku nastavené	PRINT BIN 7 XOR 5
BIT	15	Pokud je zadaný bit nastavený, vrací hodnotu TRUE, jinak vrací FALSE	PRINT 7 BIT 2, 7 BIT 3
<=	20	Porovnání dvou hodnot; pokud je první hodnota menší nebo rovná druhé, vrací hodnotu TRUE, jinak vrací FALSE	PRINT 7<=5,5<=7
<>		Pokud nejsou hodnoty stejné, vrací TRUE	PRINT 7<>5,7<>7
<		Pokud je první hodnota menší, vrací TRUE	PRINT 7<5,5<7
>=		Pokud je první hodnota větší nebo rovna druhé, vrací TRUE	PRINT 7>=5,5>=7
>		Pokud je první hodnota větší, vrací TRUE	PRINT 7>5,5>7
=		Pokud jsou hodnoty stejné, vrací TRUE	PRINT 7=5,5=5
+	30	Součet dvou hodnot	PRINT 5+7
-		Rozdíl dvou hodnot	PRINT 5-7
*	40	Součin dvou hodnot	PRINT 5*7
/		Podíl dvou hodnot	PRINT 33/6
%		Zbytek po dělení	PRINT 33 % 6
funkce	50	Viz další kapitola	
( )	60	Závorky	PRINT 1+2*3,(1+2)*3

Tabulka 2: Priority funkcí

2.2.7. Funkce

Funkce jsou příkazy, které vracejí hodnotu; lze je proto použít ve výrazech. Některé funkce jsou bez parametru, jiné vyžadují parametr, kterým může být hodnota nebo výraz.

Formát	Popis
ABS <výraz>	Vrátí absolutní hodnotu čísla.
BGET #<kanál>	Vrátí hodnotu bytu přečteného ze vstupního kanálu
!<adresa>	Vrátí 32-bitovou hodnotu přečtenou z dané adresy
BIT <výraz> <výraz1> BIT <výraz2>	Vrátí bitovou masku pro zadaný bit Vrátí hodnotu TRUE nebo FALSE podle hodnoty bitu <výraz2> v hodnotě <výraz1>
FALSE	Vrátí hodnotu 0
NOT <výraz>	Vrátí bitově invertovanou hodnotu
PEEK <adresa> ?<adresa>	Vrátí byte přečtený ze zadané adresy
DPEEK <adresa> @<adresa>	Vrátí 16-bitové slovo přečtené ze zadané adresy
RND [(<výraz>)]	Vrátí hodnotu z generátoru pseudonáhodných čísel Pokud je použitý parametr, je výsledkem číslo menší, než hodnota parametru (tj. RND % <výraz>)
SGN <výraz>	Vrátí hodnotu znaménka: 1 pro kladnou hodnotu výrazu, -1 pro zápornou a 0, pokud je hodnota výrazu 0
TICK	Vrátí počet „tiků“ od zapnutí počítače. Každou sekundu se zvětší počet tiků o 122
TIME	Vrátí počet sekund od zapnutí počítače. Dá stejný výsledek jako výraz TICK/122
TOP	Vrátí adresu první volné pozice v paměti RAM

Formát	Popis
TRUE	Vrátí hodnotu -1
VARPTR <proměnná>	Vrátí adresu zadané proměnné

2.3. Příkazová řádka

Příkazová řádka umožňuje jednoduchou editaci zadávaného textu. Pro pohyb kurzoru používáme šipky vlevo a vpravo. Kombinace CTRL a šipky vlevo nebo vpravo přesouvá kurzor po jednotlivých slovech. Klávesy Home a End přesunou kurzor na začátek nebo na konec řádku.

Příkazy jdou zkracovat zapsáním pouze části klíčového slova ukončeného tečkou. Např. PRINT jde zkrátit zápisem P. Zkratky jsou uvedeny v popisu každého příkazu.

Následující povely se dají používat pouze na příkazové řádce. Jejich použití v programu není možné.

2.3.1. AUTO

A. AUTO [<start>[,<krok>]]

Příkaz slouží k urychlení zápisu programu tím, že připravuje číslo programového řádku. Parametr start udává první použité číslo. Každé další číslo řádku je zvětšené o hodnotu krok.

Ukončení režimu automatického číslování řádků je buď stiskem klávesy ESC nebo zadáním prázdné programové řádky (bez příkazů).

```
>AUTO 20,5
>20 REM automaticke cislovani radku
>25 REM cisla radku jsou automaticky pripravena
>30 REM takze staci jen psat prikazy programu
>35
```

2.3.2. CONTINUE

C. CONTINUE

Příkaz obnoví provádění programu od místa přerušení po provedení příkazu STOP nebo stisku klávesy CTRL+C (BREAK).

```
>10 PRINT "Priklad preruseni"
>20 STOP
>30 PRINT "Pokracovani..."
>RUN
Priklad preruseni
*** BREAK IN LINE 20
>CONTINUE
Pokracovani...
```

2.3.3. EDIT

E. EDIT <řádek>

Příkaz načte zadaný programový řádek a umožní jeho úpravu (editaci) v řádkovém editoru. Pokud tedy chceme řádek opravit nebo doplnit, použijeme tento příkaz, abychom nemuseli celý řádek psát znovu.

2.3.4. LIST

L. LIST [#<kanál>,][<řádek1>][,][<řádek2>]

Příkaz vypíše zadané řádky uloženého programu. Můžeme použít tyto formy:

```
>LIST
>LIST 10
>LIST 100,
```

```
>LIST ,200
>LIST 100,200
```

První příklad vypíše celý program, druhý pouze řádek číslo 10. Třetí vypíše všechny řádky od čísla 100 do konce programu, čtvrtý zas řádky od začátku do řádku 200. Poslední příklad vypíše řádky s číslem od 100 do 200.

Pokud chceme poslat výpis programu na jiný než právě používaný komunikační kanál, musíme ho zadat s prefixem # jako první parametr příkazu.

2.3.5. MEMORY

ME. MEMORY

Příkaz vypíše využití paměti Flash (uložení programu) a RAM (uložení dat).

```
>MEMORY
1024 RAM bytes free, 35762 FLASH bytes free
```

2.3.6. MONITOR

M. MONITOR

Příkaz ukončí činnost jazyka BASIC a přejde do monitoru pro práci se strojovým kódem. Popis činnosti monitoru najdete v kapitole 3.

2.3.7. NEW

N. NEW

Příkaz smaže celý program BASICu z paměti Flash.

2.3.8. RUN

R. RUN [<řádek>]

Příkaz spustí program od začátku nebo od zadané řádky. Před spuštěním programu je proveden příkaz CLEAR.

2.4. Programové příkazy

Programové příkazy se (skoro všechny) dají používat v programu i na příkazové řádce.

2.4.1. BPUT

BP. BPUT [#<kanál>,]<výraz>|<text>[,<výraz>|<text>[...]]

Příkaz slouží k odesílání jednotlivých znaků nebo textu na aktuální nebo vybrané výstupní zařízení.

```
>BPUT 65,72,79,74,13
AHOJ
>BPUT #3,$8F
```

Poslední příkaz odešle hodnotu $8F na TXD1 signál procesoru. Podmínkou správného fungování je, aby byl tento port nastavený a povolený.

LCD stejně jako některé sériové terminály umožňuje ovládání pomocí řídících kódů. Pomocí nich je možné displej vymazat, nastavit kurzor na požadované místo apod. Tyto kódy je možné používat právě pomocí příkazu BPUT.

Sekvence	Funkce
`BPUT 12`	Smazání displeje
`BPUT 13`	Návrat kurzoru na začátek řádku
`BPUT 27,"[3;10H"`	Nastavení pozice kurzoru (3. řádek, 10. znak na řádku)
`BPUT 27,"[4m"`	Podtržení znaků

Sekvence	Funkce
`BPUT 27,"[7m"`	Inverzní zobrazení znaků
`BPUT 27,"[0m"`	Normální zobrazení znaků
`BPUT 27,"[7l"`	Vypnutí automatického přechodu na nový řádek (dlouhý řádek se nevytiskne celý)
`BPUT 27,"[7h"`	Zapnutí automatického přechodu na nový řádek (dlouhý řádek se zobrazí na více řádcích)
`BPUT 27,"[s"`	Uložení pozice kurzoru
`BPUT 27,"[u"`	Obnovení pozice kurzoru

2.4.2. BREAK

ON B. ON BREAK <příkaz>

Pokud dojde k pokusu přerušit program stiskem CTRL+C, provede se zadaný příkaz. Zpravidla se používá příkaz GOTO, který spustí speciální kód k ukončení nebo uklizení přerušeného algoritmu. Můžeme použít libovolný příkaz, ale pak nepůjde běžící program přerušit!

```
>10 ON BREAK PRINT "Uz me neprerusis :-)"
>20 GOTO 20
>RUN
```

Tento příklad trvale zablokuje program tak, že nejde přerušit. Jediný způsob, jak se dostat na příkazový řádek, je stisk tlačítka RESET. Bezpečnější řešení může být třeba toto:

```
>10 ON BREAK GOTO 100
>20 GOTO 20
>100 PRINT "Program se prerusi!"
>110 REM Tady muze byt třeba nulovani promennych...
>120 END
```

2.4.3. CLEAR

CL. CLEAR

Příkaz vynuluje proměnné A až Z a uvolní paměť alokovanou příkazy DIM.

2.4.4. DATA

DA. DATA <výraz>[,<výraz>[...]]

Slouží k uložení dat načítaných příkazem READ. Příklad použití naleznete v kapitole 1.4.

2.4.5. DIM

DIM <proměnná>(<velikost>)

Příkaz alokuje pro zadanou proměnnou pole se zadaným počtem členů, pokud je v paměti dostatek místa. Pole pak obsahuje členy s indexem 0 až <velikost>-1.

```
>DIM D(4)
```

Alokuje se pole proměnných A() se 4 prvky: D(0), D(1), D(2) a D(3).

2.4.6. DOT

DOT <X>,<Y>

Nakreslí bod na zadaných souřadnicích.

2.4.7. END

END

Ukončí provádění programu zobrazí příkazovou řádku. Vykonávání programu končí buď příkazem END nebo vykonáním posledního řádku programu.

2.4.8. EXIT

```
EXIT WHILE
EXIT FOR
EXIT REPEAT
```

Příkaz pro předčasné ukončení cyklu FOR, WHILE nebo REPEAT. Příklady jsou uvedené u jednotlivých cyklů.

2.4.9. FOR — NEXT

```
F.      FOR <proměnná>=<výraz_od> TO <výraz_do>
        <tělo cyklu>
NEX.    NEXT
```

Cyklus FOR je vykonáván opakovaně se zadaným počtem opakování.

Na začátku je do proměnné přiřazena hodnota <výraz_od>. Na konci cyklu je hodnota proměnné zvětšena o 1 a dokud její hodnota nepřekročí hodnotu <výraz_do>, opakuje se vykonávání příkazů v těle cyklu. Po ukončení cyklu má tedy proměnná hodnotu <výraz_do>+1.

Hodnota výrazů se vyhodnocuje na začátku cyklu. Pokud <výraz_do> obsahuje nějakou proměnnou, která se uvnitř cyklu mění, její změněná hodnota se nijak neprojeví na počtu cyklů, protože se použila její hodnota před prvním provedením cyklu.

Cyklus je možné předčasně ukončit příkazem EXIT FOR (cyklus se opouští v místě příkazu, takže následující příkazy z těla cyklu už nejsou vykonány) nebo také nastavením proměnné na hodnotu větší než <výraz_do>, kdy je dokončeno provádění celého těla cyklu, ale další cyklus se už nespustí.

```
>10 INPUT "Zadej cislo:",A ' Cislo od 1 do 99
>20 B=1
>30 FOR I=2 TO 50
>40 IF (A % I)=0 THEN B=I
>50 IF I>=(A/2) THEN EXIT FOR ' Staci hledat jen do poloviny
>60 NEXT
>70 PRINT "Nejvetsi spolecny delitel:";B
```

2.4.10. GOTO

```
G.      GOTO <řádek>
```

Příkaz přesune provádění programu na zadanou řádku programu.

```
>10 REM Program GOTO
>20 GOTO 60
>30 GOTO 40
>40 PRINT "Uz me to nebavi."
>50 END
>60 GOTO 30
```

2.4.11. GOSUB

```
GOS.    GOSUB <řádek>
```

Příkaz zavolá podprogram. Provedením příkazu RETURN se pak vrátí vykonávání zpět do hlavního programu za příkaz GOSUB.

```
>10 GOSUB 50
>20 GOSUB 60
>30 END
>50 PRINT "Podprogram 1"
>60 PRINT "Podprogram 1a"
>70 GOSUB 90
>80 RETURN
>90 PRINT "Podprogram 2"
>100 RETURN
```

2.4.12. IF … [THEN]

IF <výraz> [THEN] <příkaz>
IF <výraz> THEN <řádek>

Pokud je <výraz> nenulový, je provedený <příkaz> nebo skok na <řádek>. Jinak pokračuje program příkazem následujícím za IF.

2.4.13. INPUT

INPUT [#<kanál>,][<text>,]<proměnná>[,[<text>,]<proměnná>[...]]

Příkaz načítá hodnoty ze standardního nebo vybraného vstupu do zadaných proměnných.

2.4.14. LINE

LINE <X1>,<Y1>,<X2>,<Y2>

Příkaz nakreslí úsečku z bodu X1,Y1 do bodu X2,Y2.

2.4.15. MODE

MODE <xor>[,<or>]

Režim zobrazování textů a grafiky.

V režimu zobrazování textu maska <xor> invertuje zobrazovaná data, maska <or> nastavuje body podle masky.

```
>MODE $FF ' invertované znaky
>MODE 0,$80 ' podtrzene znaky
```

Při kreslení bodů a úseček je význam jiný:

```
>MODE 0 ' kresli cernou caru
>MODE 1,0 ' invertuje body na displeji
>MODE 1,1 ' kresli bilou caru = maze body
```

Při kreslení bodů a úseček program pouze kontroluje, zda je hodnota parametrů <xor> a <or> nulová nebo nenulová.

2.4.16. ON … GOTO

ON G. **ON <výraz> GOTO <řádek1>[,<řádek2>[...]][ELSE <řádek0>]**

Pokud se výraz rovná 1, pak se skočí na <řádek1>, při hodnotě 2 se skočí na řádek 2, podobně pro další hodnoty. Pokud je hodnota mimo zadaný rozsah, pokračuje se buď následujícím příkazem programu nebo se zadá část ELSE, která provede skok na <řádek0>.

2.4.17. POKE, DPOKE

POKE <adresa>,<data>
DPOKE <adresa>,<data>
?<adresa> = <data>
@<adresa> = <data>
!<adresa> = <data>

Příkaz zapíše byte (POKE) nebo 16-bitové slovo (DPOKE) na zadanou <adresu>. Stejně se dají použít funkce ? (byte), @ (16-bitové slovo) a ! (32-bitové slovo).

2.4.18. PRINT

P. **PRINT [#<kanál>,][HEX|BIN][<výraz>|<text>][,|;][...]**

Příkaz slouží pro vypisování textů a hodnot.

Pro vypisování hodnot je možné použít funkci HEX a BIN pro vypsání hexadecimální nebo binární reprezentace čísla. Jednotlivé parametry se oddělují buď středníkem nebo čárkou. Při použití středníku je další parametr vypsán ihned za předchozí parametr. Při použití čárky je tisk zarovnaný na násobek 8 znaků (zarovnání na tabulátor).

Pokud příkaz nekončí čárkou nebo středníkem, je kurzor přesunutý na nový řádek.

```
>PRINT

>PRINT A,:PRINT B
0       0
>PRINT:PRINT A

0
>PRINT "$";HEX 10,BIN 10
$A      1010
```

2.4.19. RANDOMIZE

RA. RANDOMIZE <výraz>

Inicializuje generátor pseudonáhodných čísel pomocí zadaného výrazu.

2.4.20. READ

READ <proměnná>[,<proměnná>[...]]

Čte postupně hodnoty uvedené za příkazem DATA a ukládá je do zadaných proměnných. Při dosažení konce dat začíná číst opět od začátku.

2.4.21. REM

REM <poznámka>
' <poznámka>

Příkaz slouží k umístění poznámek do programu. Může také sloužit k dočasnému zablokování části programu. V poznámce může být použit libovolný znak. Při provádění programu je zbytek řádku od příkazu přeskočený a vykonávání pokračuje dalším řádkem.

2.4.22. REPEAT — UNTIL

REP. REPEAT
<tělo cyklu>
U. UNTIL <výraz>

Provádí příkazy uvedené v těle cyklu, dokud je hodnota výrazu nulová.

Cyklus je možné předčasně ukončit příkazem EXIT REPEAT.

2.4.23. RESET

RES. RESET

Provede reset celého počítače, jako by došlo k zapnutí napájení nebo stisku tlačítka RESET.

2.4.24. RESTORE

REST. RESTORE [<řádek>]

Nastaví načítání dat příkazem READ od začátku programu nebo od zadaného řádku.

2.4.25. STOP

S. STOP

Zastaví provádění programu, jako by došlo ke stisku CTRL+C.

Provádění programu může být obnoveno příkazem CONTINUE.

2.4.26. TRACE

T. TRACE ON|OFF

Povolí nebo zakáže vypisování čísla řádku prováděného programu. Umožňuje tak sledovat, jaké řádky program vykoná. Čísla řádků jsou vypisována v hranatých závorkách.

Je možné takto sledovat jen část programu (v příkladu mezi řádky 50 až 100):

```
50 TRACE ON
   <sledovaný program>
100 TRACE OFF
```

2.4.27. WHILE — WEND

W. **WHILE <výraz>**
<tělo cyklu>
WE. **WEND**

Provádí příkazy uvedené v těle cyklu, dokud je hodnota výrazu nenulová.

Cyklus je možné předčasně ukončit příkazem EXIT WHILE.

```
>10 REM
>20 A=1
>30 WHILE A<10
>40 A=A+1
>50 IF A=8 EXIT WHILE
>60 WEND
```

Uvedený příklad vykonává tělo cyklu dokud je A<10. Příkaz EXIT WHILE však ukončuje cyklus již při dosažení hodnoty A=8.

2.5. Strojový kód

Z programů v BASICu je možné spouštět programy ve strojovém kódu, které si připravíme v monitoru (viz kapitola 3). Také je možné číst a zapisovat do celé paměti procesoru pomocí příkazů PEEK a POKE. Abychom tím však nezpůsobili nějakou nevratnou škodu, potřebujeme pro to už velkou znalost procesoru MSP430F149.

2.5.1. USR

USR <adresa>[,<parametr>]

Spustí program ve strojovém kódu od zadané <adresy>. Pokud je zadaný <parametr>, je uložený do registrů R15:R14, do nichž může být uložena i návratová hodnota.

Tuto funkci si můžeme vyzkoušet na volání některých systémových funkcí popsaných v kapitole 3.4.

Při psaní vlastních podprogramů v assembleru je potřeba zajistit, aby obsah registrů R4 až R11 zůstal nezměněný (např. uložení hodnot na zásobník a před návratem jejich obnovení).

2.6. Automatické spuštění programu po zapnutí

Pokud chceme, aby po zapnutí napájení počítač spustil zapsaný program, můžeme na první řádek programu zapsat tento příkaz:

```
>10 '!RUN
```

Příkaz RUN je po spuštění počítače automaticky přepsán na příkazovou řádku a vykonán. Takto můžeme zapsat jakýkoli programový příkaz nebo i víc příkazů oddělených dvojtečkou.

2.7. Tipy pro efektivnější programy

Zde je několik tipů pro psaní efektivních programů:

- místo konstant používejte proměnné; k načtení konstanty je v programu potřeba více času než k načtení proměnné
- použití cyklů je vhodnější než používání příkazů GOTO

- cyklus FOR je rychlejší než cykly WHILE a REPEAT, protože u nich se podmínka vyhodnocuje každý cyklus znovu

2.8. Chyby v programech

Při vykonávání programu může program skončit vypsáním chybové hlášky. V tabulce jsou popsány všechny chyby včetně možných příčin a způsobu nápravy.

Text chyby	Popis
DIVIDE BY ZERO ERROR	Dělení nulou není povolené
CAN'T CONTINUE	Není možné použít příkaz CONTINUE; program byl zastavený jinak než příkazem STOP nebo stiskem CTRL+C nebo byl program restartován
OUT OF DATA	V programu není příkaz DATA
OUT OF MEMORY	Není dostatek volné paměti pro příkaz DIM
ARRAY NOT DIMENSIONED	Pole nebylo před použitím alokováno příkazem DIM
INDEX OUT OF BOUNDS	Index prvku pole je mimo alokovaný rozsah
SYNTAX ERROR	Chyba syntaxe příkazu
VALUE ERROR	Chyba zadané hodnoty v příkazu INPUT
NO SUCH LINE	V programu není řádek s požadovaným číslem
MISMATCHED '('	Při vyhodnocení výrazu nesouhlasí počet závorek
MISMATCHED '"'	Při tisku textu nenalezené ukončovací uvozovky
EXIT NOT WITHIN LOOP	Příkaz EXIT není uvnitř cyklu
END WHILE WITHOUT MATCHING WHILE	K příkazu WEND neexistuje párový WHILE
UNTIL WITHOUT MATCHING REPEAT	K příkazu UNTIL neexistuje párový REPEAT
RETURN WITHOUT MATCHING GOSUB	Příkaz RETURN není umístěn v podprogramu
NEXT WITHOUT MATCHING FOR	K příkazu NEXT neexistuje párový FOR
WHILE WITHOUT MATCHING END WHILE	K příkazu WHILE neexistuje párový WEND
BAD CHANNEL ERROR	Chybné číslo komunikačního kanálu
BREAK	Zastavení programu příkazem STOP nebo stiskem CTRL+C
BAD INPUT	Chyba zadané hodnoty v příkazu INPUT
STACK UNDERFLOW	Podtečení zásobníku
STACK OVERFLOW	Přetečení zásobníku
COMMAND NOT ALLOWED	Použití příkazu pro příkazovou řádku není v programu povoleno.
ILLEGAL LINE NUMBER	Chybně zadané číslo řádku programu
NO ROOM IN FLASH	Paměť Flash je plná, není už žádné místo pro další řádky programu

Tabulka 3: Chybové zprávy

Pokud dojde k chybě v příkazovém režimu, vypadá výpis chyby takto:

```
>PRINT 1/0
*** DIVIDE BY ZERO ERROR
```

Stejná chyba v programovém režimu:

```
>10 PRINT 1/0
>RUN
*** DIVIDE BY ZERO ERROR IN LINE 10
```

Text chyby je doplněný informací o řádku, na kterém k chybě došlo.

3. Monitor

Monitor slouží pro práci se strojovým kódem procesoru. Umožňuje prohlížet paměť, zapisovat do paměti instrukce a také spouštět a krokovat programy. Monitor vypisuje všechny hodnoty v hexadecimálním tvaru. Zadávané hodnoty mohou být dekadické nebo hexadecimální podobně jako v BASICu: hexadecimální hodnoty musí začínat znakem $. Příkazový řádek zde začíná znakem „**dvojtečka**“, aby se odlišil od příkazové řádky BASICu.

Nejprve si vyzkoušíme funkci prohlížení paměti a instrukcí. K tomu nám poslouží příkazy **MEM** a LST.

```
:MEMB $0200
0200 00 00 00 00 00 00 00 00
0208 00 00 00 00 00 00 00 00
0210 00 00 00 00 00 00 00 00
0218 00 00 00 00 00 00 00 00
0220 00 00 00 00 00 00 00 00
0228 00 00 00 00 00 00 00 00
0230 00 00 00 00 00 00 00 00
0238 00 00 00 00 00 00 00 00
```

Na příkladu vidíme obsah paměti RAM od adresy 0200. V příkazu si můžeme vybrat, zda chceme zobrazit byly (**MEMB**), 16-bitová slova (**MEMW**) nebo text (**MEMT**).

Ještě zkusíme, jak vypadají instrukce assembleru, a to pomocí příkazu **LST**:

```
:LST $0C04
0C04 MOV #0200,SP
0C08 CLR R11
0C0A CLR.B F60A(PC)
0C0E DINT
0C10 BIC.B #0032,&IE1
0C16 MOV #5480,&WDTCTL
0C1C MOV.B #0085,&BCSCTL1
0C22 MOV.B #0080,&DCOCTL
```

Význam jednotlivých instrukcí můžeme najít v kapitole 4.2.3.

Teď zkusíme napsat jednoduchý program, který sečte dvě čísla:

```
:$0200 ADD R4,R5
:$0202 JMP 0200
:$0204
:APP $0200
0000  0000  0000  0000
0000  0000  0000  0000
0000  0000  0000  0000
05FE->1A02  0002  0002
0008: - - - - - I - - -
0200 ADD R4,R5
0202 JMP 0200
0204 DW 0000
```

Nastavíme hodnotu registrů R4 a R5:

```
:R4 $0011
:R5 $0033
```

Teď můžeme klávesou **F8** krokovat program po jednotlivých instrukcích a uvidíme, jak instrukce pracují. První přičte hodnotu R4 do registru R5, druhá vykoná skok na adresu $0200, takže program se opakuje od začátku, ale se změněným registrem R5.

První parametr instrukce je zdrojový operand (src), druhý parametr je cílový operand (dst). Přehled

instrukcí procesoru můžeme najít v kapitole 4.2.3.

3.1. Příkazy

Nyní se seznámíme se všemi příkazy monitoru.

3.1.1. APP

APP [<adresa>]

Příkaz zobrazí stav procesoru (registry, zásobník, instrukce).

Pokud napíšeme i adresu, inicializuje příkaz stav procesoru pro nové spuštění programu od zadané adresy. Tj. nastaví registr PC podle zadané adresy, nastaví SP na konec paměti RAM a vynuluje všechny registry.

```
:APP
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
05FE->1A02 0000 0000
0008: - - - - - I - - -
1B14 CLR R15
1B16 CALL #1B08
1B1A NOP
```

První tři řádky zobrazují obsah registrů R4 až R15. Další řádek ukazuje stav registru SP a první tři položky uložené na zásobníku. Na pátém řádku vidíme stav registru SR: nejdřív jeho číselnou hodnotu, pak také jednotlivé bity. Poslední tři řádky zobrazují aktuální a dvě následující instrukce, které bude procesor vykonávat. V tabulce jsou uvedeny funkce kláves, které můžeme pro ladění programu využívat.

Klávesa	Popis Funkce
F9	Spuštění programu; program se ukončí buď provedením instrukce RET nebo stiskem CTRL+C
F8	Krokování programu po jednotlivých instrukcích
F7	Spuštění programu a průběžné vypisování hodnot registrů
CTRL+C	Přerušení běžícího programu

Teď si projdeme všechny příkazy monitoru.

3.1.2. BAS

BAS

Vrátí řízení zpět do BASICu.

3.1.3. ERA

ERA <adresa>

Smaže segment paměti na zadané adrese. Povolí smazat pouze oblast, kterou monitor může používat. Program v BASICu tak zůstane nedotčený.

Pokud je adresa $FE00 až $FFFF, tak se smaže celá paměť Flash vyhrazená pro monitor

3.1.4. INFO

INFO

Vypíše informaci o volné Flash v procesoru. Paměť z udaného rozsahu je možné použít pro uložení vlastního programu ve strojovém kódu. Paměť je výhodné zaplňovat od nejvyšších bloků k nižším. Nejprve tedy zapisujeme od adresy $FC00 do adresy $FDFF. Pak použijeme blok $FA00 až $FBFF. Pokud potřebujeme uložit delší úsek dat než je blok Flash paměti, tak samozřejmě začneme od takové vhodné adresy, aby se nám celý program vešel do volných bloků.

```
:INFO
Free Flash: 7200 - FDFF
```

3.1.5. LST

LST [<adresa>]

Výpis instrukcí od posledně použité adresy nebo od zadané adresy.

3.1.6. MEM, MEMB, MEMW, MEMT

MEM[B|W|T] [<adresa>]

Výpis paměti od naposledy použité adresy nebo od zadané adresy. Režim zobrazení je buď naposledy použitý (příkaz MEM), nebo se nastaví podle posledního písmene příkazu:

B - zobrazení po bytech

W - zobrazení po 16-bitových slovech

T - zobrazení textu

3.1.7. RST

RST

Provede reset procesoru. Počítač se tedy znovu spustí v režimu BASIC.

3.1.8. R#, PC, SP, SR

R<číslo>|PC|SP|SR <hodnota>

Uloží do vybraného registru novou hodnotu a zobrazí stav procesoru (jako příkaz APP).

3.1.9. Assembler

<adresa> <instrukce> [<parametr>[,<parametr>]]

Seznam všech instrukcí můžeme najít v kapitole 4.2.3.

Kromě těchto instrukcí je možné použít ještě pseudoinstrukce DB pro zápis textu, znaků, bytů nebo DW pro zápis 16-bitových slov do paměti.

Instrukce je možné zapisovat do paměti RAM nebo Flash využitelné pro monitor (viz mapa paměti).

Před zápisem do paměti Flash je potřeba smazat použité bloky příkazem ERA.

3.2. Příklady programů v assembleru

3.2.1. Volání systémových funkcí

Nejprve si vykoušíme volání systémových funkcí uvedených v kapitole 3.4.4. Začneme funkcí **PUTC**:

```
:$0200 MOV.B #12,R15
:$0204 CALL &PUTC
:$0208 JMP $0208
:APP $0200
:<F9>
```

Po spuštění programu klávesou **F9** se obsah displeje vymaže a program zůstane v nekonečné smyčce na adrese $0208. Stiskem kláves **CTRL+C** je možné program přerušit.

Instrukce, které mají příponu **.B**, jsou bytové instrukce. Používají pouze dolních 8 bitů registru, horních 8 bitů je nulových. Parametr začínající znakem **#** je konstanta. Takový parametr může být pouze jako zdrojový (src). Instrukce **MOV** uloží data získaná z prvního parametru do umístění daného druhým parametrem. V příkladu se do registru R15 procesoru uloží 8-bitová konstanta s hodnotou 12.

Parametr, který začíná znakem **&**, načte hodnotu pro výpočet z uvedené adresy nebo na tuto adresu

zapíše výsledek operace. Instrukce **CALL** si z paměti na uvedené adrese (zde je adresa zapsána pomocí konstanty CPUTC, která má hodnotu $1004) načte slovo, kterou použije jako adresu funkce, která je následně spuštěna.

Program končí instrukcí skoku **JMP** na stejnou adresu, kde je instrukce uložená. Tomu se říká nekonečná smyčka a program tak nikdy neskončí. V monitoru můžeme tuto nekonečnou smyčku přerušit stiskem CTRL+C. Pokud bychom program ukončili instrukcí **RET**, program by se vrátil do monitoru automaticky, ale displej by se přepsal aktuálním obsahem registrů, takže bychom neviděli, jestli nám program funguje.

Teď si napíšeme program, který vypíše na displej zadaný text funkcí **PUTS** a před návratem do monitoru bude čekat na stisk libovolné klávesy ve funkci **GETC**.

```
:$0200 MOV #$020E,R15
:$0204 CALL &PUTS
:$0208 CALL &GETC
:$020C RET
:$020E DB 12,"To je muj text",0
:APP $0200
```

Po spuštění programu se do Registru R15 uloží adresa dat, která jsou následně vypsána na displej. Data jsou do paměti vložena pomocí pseudoinstrukce **DB**, která assembleru říká, že následující hodnoty, znaky nebo text má do paměti uložit jako 8-bitové konstanty. První vypsaný znak má kód 12, který provede jako v prvním příkladu smazání displeje. Další znaky už obsahují vlastní text, který se na displeji vypíše. Pak program čeká na stisk klávesy. Po stisku klávesy se vrátí program do monitoru. To nám ukazuje i text „EXIT“ vypsaný místo instrukcí:

```
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0600->0000 0000 0000
0008: - - - - - I - - -
1C80 -- EXIT --
```

Další program, který si vyzkoušíme, bude psát všechny znaky, dokud nezadáme třeba tečku.

```
:$0200 MOV.B #12,R15  ;smazani displeje
:$0204 ;tady zacina cyklus
:$0204 CALL &PUTC
:$0208 CALL &GETC     ;cteni znaku
:$020C CMP.B #'.',R15 ;porovnani se znakem tecka
:$0210 ;dokud není zadana tecka, cyklus se opakuje
:$0210 JNZ $0204
:$0212 RET
```

Instrukce **CMP** provádí porovnání dvou hodnot. Používá k tomu funkci odčítání, výsledek nikam neuloží, nastaví pouze stavové bity procesoru, které jsou následně použity pro rozhodování podmíněných skoků. Zde používáme instrukci **JNZ**, která provede skok, je-li hodnota stavového bitu **Z** nulová, tedy pokud hodnota registru R15 nebyla při provedení instrukce CMP shodná se znakem tečka.

Na příkladu také vidíme způsob zápisu komentářů v assembleru. Všechno, co je zapsané za **středníkem**, je assemblerem ignorováno a nijak nezasahuje do našeho programu. Je to stejné jako příkaz REM v BASICu. Komentáře také můžeme psát od začátku řádku místo instrukce, adresa se pak nemění a na dalším řádku můžeme napsat další instrukci. Na rozdíl od jazyku BASIC se komentáře v assembleru neukládají do paměti. U velkých počítačů jsou programy v assembleru uložené v souborech společně s komentáři, ale po překladu do strojového kódu se komentáře vypouštějí. U počítače MSP430SBC máme možnost zadávat instrukce pouze z klávesnice, proto je zbytečné opisovat programy i s komentáři. Je však správné si komentáře psát do svých programů, které máme zapsané třeba na papíru, protože nám pomáhají se v programu lépe orientovat.

Program spustíme a můžeme psát na displeji texty. Jakmile stiskneme tečku, program skončí. Pokud zkusíme stisknout klávesu Enter, vidíme, že kurzor se přesunul na začátek prvního řádku, ale ne na další řádek. Klávesa Enter má kód **13** nebo také **$0D** a tento kód se používá pro nastavení kurzoru na začátek právě používaného řádku. Pro přechod na nový řádek se používá kód **10** (**$0A**). Z klávesnice ho můžeme zadat stisknutím **CTRL+J**. Abychom to nemuseli dělat tak složitě, upravíme si svůj program tak, aby to dělal za nás.

```
:$0200 MOV.B #12,R15 ;smazani displeje
:$0204 CALL &PUTC
:$0208 CALL &GETC    ;cekani na klavesu
:$020C CMP.B #$0D,R15
:$0210 JNZ $0216
:$0212 MOV.B #$0A,R15;misto $0D se pouzije $0A
:$0216 CMP.B #'.',R15
:$021A JNZ $0204
:$021C RET
```

Pro zadávání textu můžeme také použít funkci **EDITSTR**. Tato funkce umožní editovat zadaný text stejně, jak to můžeme dělat na příkazové řádce BASICu nebo monitoru. Funkce končí, pokud stiskneme klávesu Enter nebo CTRL+C.

V dalším programu si tedy ukážeme, jak lze tuto funkci používat.

```
:$0200 MOV.B #12,R15
:$0204 CALL &PUTC
:$0208 MOV #$0230,R15 ;adresa bufferu
:$020C CLR 0(R15)     ;smazani bufferu
:$0210 MOV #40,R14    ;max.delka
:$0214 CALL &EDITSTR  ;editace radku
:$0218 TST R15
:$021A JZ $0222       ;nebyl stisknuty Enter: konec
:$021C TST &$230
:$0220 JNZ $0208      ;nebyl zadany prazdny radek: opakuj
:$0222 RET
```

Instrukce **CLR** nuluje daný bajt nebo slovo (lze zapsat také jako MOV #0,<dst>), instrukce **TST** testuje, zda je hodnota nulová (provádí instrukci CMP #0,<dst>).

Každý řádek je možné do stisknutí klávesy Enter upravovat. Dokud bude na zadaném řádku aspoň jeden znak, program pokračuje. Jakmile stiskneme Enter bez předchozího zadání jakéhokoli znaku, program končí.

Podíváme se teď na funkci **PRINTVAL** pro tisk čísel. Tato funkce tiskne na displej čísla v zadaném formátu. Teď budeme používat formát dekadický.

```
:$0200 MOV.B #12,R15
:$0204 CALL &PUTC     ;smazani displeje
:$0208 MOV #-1,R15    ;vyssi slovo cisla
:$020A MOV #-1,R14    ;nizsi slovo cisla
:$020C MOV #0,R13     ;rezim tisku
:$020E CALL &PRINTVAL ;tisk cisla
:$0212 JMP $0212
```

Po spuštění programu se na displeji objeví číslo 4294967295, což je maximální číslo, které se vejde do 32-bitové proměnné. Pro výpis v hexadecimálním tvaru změníme řádek:

```
:$020C MOV #1,R13
```

Na displeji vidíme totéž číslo, ale v hexadecimálním tvaru: FFFFFFFF.

Teď si vyzkoušíme funkci **GETKEY**, která umí zpracovat nejen textové znaky, ale také řídící klávesy (Fn, Insert, Delete, PageUp, PageDown, kurzorové klávesy aj.). Program bude zobrazovat hodnoty

kláves v hexadecimálním režimu. Program skončí po stisku mezerníku.

```
:$0200 MOV.B #12,R15
:$0204 CALL &PUTC
:$0208 CALL &GETKEY
:$020C CMP #' ',R15
:$0210 JZ $0222
:$0212 MOV R15,R14
:$0214 CLR R15
:$0216 MOV #1,R13
:$0218 CALL &PRINTVAL
:$021C MOV #10,R15
:$0220 JMP $0204
:$0222 RET
```

Nyní využijeme funkci **GETVAL** a napíšeme program, který bude sčítat zadaná čísla. Zadání mezery nuluje sčítací proměnnou, zadání prázdného řádku ukončuje program.

```
:$0200 MOV.B #12,R15
:$0204 CALL &PUTC     ;smazani displeje
:$0208 CLR R4         ;vynulovani akumulatoru
:$020A CLR R5
:$020C MOV.B #'=',R15
:$0210 CALL &PUTC     ;tisk znaku
:$0214 MOV R4,R14
:$0216 MOV R5,R15
:$0218 MOV #0,R13
:$021A CALL &PRINTVAL ;tisk cisla
:$021E MOV.B #10,R15
:$0222 CALL &PUTC     ;novy radek
:$0226 MOV.B #'+'
:$022A CALL &PUTC     ;tisk znaku
:$022E MOV #$280,R15
:$0232 CLR.B 0(R15)
:$0236 MOV #40,R14
:$023A CALL &EDITSTR  ;cteni cisla
:$023E TST R15
:$0240 JZ $0258      ;konec
:$0242 CMP.B #' ',&$280
:$0248 JZ $0208      ;jdi na nulovani
:$024A MOV #$280,R15
:$024E CALL &GETVAL  ;prevod cisla
:$0252 ADD R14,R4    ;pricteni cisla
:$0254 ADDC R15,R5
:$0256 JMP $020C
:$0258 RET
```

Instrukce **ADD** přičítá obsah <src> k <dst>, instrukce **ADC** k tomu přičítá ještě 1, pokud je bit **C** nastavený. To se používá při sčítání větších než 16-bitových čísel (zde máme 32-bitová čísla).

Obě funkce z minulého příkladu teď použijeme v programu, který nám převede číslo zadané hexadecimálně na dekadické a dekadicky zadané číslo na hexadecimální.

```
:$0200 MOV.B #12,R15
:$0204 CALL &PUTC    ;smazani displeje
:$0208 MOV.B #'?',R15
:$020C CALL &PUTC    ;tisk znaku
:$0210 MOV #$280,R15
:$0214 CLR.B 0(R15)
:$0218 MOV #40,R14
```

```
:$021C CALL &EDITSTR ;cteni cisla
:$0220 TST R15
:$0222 JZ $025A      ;konec
:$0224 MOV.B #'=',R15
:$0228 CALL &PUTC    ;tisk znaku
:$022C CMP.B #'$',&$280
:$0232 JZ $023C
:$0234 MOV.B #'$',R15
:$0238 CALL &PUTC
:$023C MOV #$280,R15
:$0240 CALL &GETVAL  ;prevod cisla
:$0244 CLR R13       ;DEC do HEX
:$0246 CMP.B #'$',&$280
:$024C JZ $0250
:$024E MOV #1,R13    ;HEX do DEC
:$0250 CALL &PRINTVAL;tisk cisla
:$0254 MOV.B #10,R15
:$0258 JMP $0204
:$025A RET
```

Po spuštění program smaže displej a zobrazí otazník: čeká na zadání dekadického nebo hexadecimálního čísla. Po zadání zjistí, zda číslo začíná znakem $ a vytiskne číslo v dekadickém tvaru. Jinak vytiskne číslo v hexadecimálním tvaru.

3.2.2. Hledání textu

Další příklad nám poslouží k rychlejšímu zkoumání „vnitřností“ systému počítače. Pomůže nám vyhledávat zadaný text v paměti Flash počítače.

```
:$0200 MOV.B #12,R15
:$0204 CALL &PUTC    ;smazani displeje
:$0208 MOV.B #'?',R15
:$020C CALL &PUTC
:$0210 MOV #$280,R15
:$0214 CLR.B 0(R15)
:$0218 MOV #40,R14
:$021C CALL &EDITSTR ;zadani textu
:$0220 TST R15
:$0222 JZ $025E      ;konec
:$0224 TST.B &$280
:$0228 JZ $025E      ;konec
:$022A MOV #$280,R4  ;adresa hledaneho textu
:$022E MOV #$0FFF,R5 ;adresa flash - 1
:$0232 INC R5
:$0234 JZ $0208      ;konec flash
:$0236 MOV R5,R15
:$0238 MOV R4,R14    ;adresa hledaneho textu
:$023A TST.B 0(R14)
:$023E JZ $024A      ;nalezeno
:$0240 CMP.B @R14+,0(R15)
:$0244 JNZ $0232     ;hledej znovu
:$0246 INC R15
:$0248 JMP $023A     ;cyklus
:$024A CLR R15
:$024C MOV R5,R14
:$024E MOV #1,R13
:$0250 CALL &PRINTVAL;tisk adresy
:$0254 MOV.B #10,R15
:$0258 CALL &PUTC
```

```
:$025C JMP $0232     ;hledej znovu
:$025E CALL &GETC
:$0260 RET
```

Po spuštění programu se vypíše na displeji otazník a program čeká na zadání textu. Po stisknutí klávesy Enter hledá zadaný text v paměti Flash a vypisuje adresy, kde tento text nalezl. Po ukončení hledání začíná program opět od začátku a je možné zadat další text k hledání. Program se ukončí zadáním prázdného řádku (žádný znak, pouze klávesa Enter).

3.2.3. Sériový terminál

Pro přípojení k dalším zařízením se nám může hodit sériový terminál. Znaky z klávesnice budou odesílány signálem TXD do připojeného zařízení, znaky přicházející signálem RXD budou zobrazovány na displeji. Ukážeme si zde také použití přerušení.

V příkladu použijeme nastavení sériového portu na rychlost přenosu 19200Bd, 8 bitů bez parity, 1 STOP bit.

```
:$0200 ;nastaveni portu
:$0200 BIS.B #$10,&P3OUT
:$0206 BIS.B #$10,&P3DIR
:$020C ;nastaveni UARTu
:$020C BIS.B #$C0,&ME1 ;povoleni UARTu
:$0212 MOV.B #$11,&U0CTL
:$0218 MOV.B #$20,&U0TCTL
:$021E CLR.B &U0RCTL
:$0222 MOV.B #$A0,&U0BR0 ;prenosova rychlost
:$0228 MOV.B #$01,&U0BR1
:$022C MOV.B #$6D,&U0MCTL
:$0232 BIS.B #$30,&P3SEL
:$0238 ;spusteni UARTu
:$0238 BIC.B #1,&U0CTL
:$023C ;nastaveni vektoru preruseni
:$023C MOV #$02A6,&$760
:$0242 MOV #$02CE,&$762
:$0248 ;smazani bufferu
:$0248 CLR &$02FC
:$024C CLR &$02FE
:$0250 ;povoleni preruseni
:$0250 BIS.B #$40,&IE1
:$0256 ;==================
:$0256 ;hlavni smycka
:$0256 ;cteni klaves
:$0256 CALL &KEY
:$025A TST R15
:$025C JZ $0288
:$025E CALL &GETC
:$0262 ;predani klavesy do bufferu pro odeslani
:$0262 MOV.B &$02FC,R14
:$0266 MOV.B R15,$0300(R14)
:$026A INC R14
:$026C AND #$000F,R14
:$0270 MOV.B R14,&$02FC
:$0274 BIT.B #$80,&IE1
:$027A JNZ $0288
:$027C BIS.B #$80,&IFG1
:$0282 BIS.B #$80,&IE1
:$0288 ;zobrazeni znaku
:$0288 MOV.B &$02FE,R14
```

```
:$028C CMP.B R14,&$02FF
:$0290 JZ $02A4 ;muze byt i $0260
:$0292 ;zobrazeni znaku
:$0292 MOV.B $0310(R14),R15
:$0296 INC R14
:$0298 AND #$003F,R14
:$029C MOV.B R14,&$02FE
:$02A0 CALL &PUTC
:$02A4 ;konec hlavni smycky
:$02A4 JMP $0256
:$02A6 ;================
:$02A6 ;preruseni pro vysilani znaku
:$02A6 PUSH R14
:$02A8 MOV.B &$02FD,R14
:$02AC CMP.B R14,&$02FC
:$02B0 JZ $02C4
:$02B2 ;odeslani dalsiho znaku
:$02B2 MOV.B $0300(R14),&U0TXBUF
:$02B8 INC R14
:$02BA AND #$000F,R14
:$02BE MOV.B R14,&$02FD
:$02C2 JMP $02CA
:$02C4 ;zakazani preruseni
:$02C4 BIC #$80,&IE1
:$02CA POP R14
:$02CC RETI
:$02CE ;================
:$02CE ;preruseni pro prijem znaku
:$02CE PUSH R14
:$02D0 MOV.B &$02FF,R14
:$02D4 MOV.B &U0RXBUF,$0310(R14)
:$02DA INC R14
:$02DC AND #$003F,R14
:$02E0 MOV.B R14,&$02FF
:$02E4 POP R14
:$02E6 RETI
```

Program nejprve nastaví parametry sériového portu UART0. Pak nastaví vektory přerušení pro vysílání i příjem a povolí je.

V hlavní smyčce se kontroluje, zda je k dispozici znak z klávesnice. Pokud ano, uloží se do vysílacího bufferu a povolí se vysílací přerušení. Dále se kontroluje, zda je k dispozici přijatý znak. Každý přijatý znak je vypsán na displej.

Poslední část programu obsahuje kód pro přerušení vysílací a přijímací části UARTu.

Dále si uvedeme program v BASICu, který uloží náš strojový program do paměti:

```
>10 READ D
>20 A = $200
>30 WHILE D <> $FFFFFFFF
>50 READ D
>40 !A = D:A = A + 4
>60 WEND
>90 END
>100 DATA $0010D0F2,$D0F20019,$001A0010,$00C0D0F2
>101 DATA $40F20004,$00700011,$002040F2,$43C20071
>102 DATA $40F20072,$007400A0,$007543D2,$006D40F2
>104 DATA $40B20760,$076202CE,$02FC4382,$02FE4382
```

```
>103 DATA $D0F20073,$001B0030,$0070C3D2,$02A640B2
>105 DATA $0040D0F2,$12920000,$930F100A,$12922415
>106 DATA $425E1008,$4FCE02FC,$531E0300,$000FF03E
>107 DATA $02FC4EC2,$0080B0F2,$20060000,$0080D0F2
>108 DATA $D0F20002,$00000080,$02FE425E,$02FF9EC2
>109 DATA $4E5F2409,$531E0310,$003FF03E,$02FE4EC2
>110 DATA $10041292,$120E3FD8,$02FD425E,$02FC9EC2
>111 DATA $4ED22409,$00770300,$F03E531E,$4EC2000F
>112 DATA $3C0302FD,$0080C0F2,$413E0000,$120E1300
>113 DATA $02FF425E,$007642DE,$531E0310,$003FF03E
>114 DATA $02FF4EC2,$1300413E
>199 DATA $FFFFFFFF
```

Nejprve program spustíme. Pak už stačí přejít do monitoru a spustit aplikaci klávesou F9:

```
>RUN
>MONITOR
:APP $0200
```

Případně můžeme zkusit spuštění rovnou z BASICu, tak přidáme řádek do programu:

```
>70 A=USR $0200
```

Příkaz **USR** spustí strojový program na zadané adrese.

Na pinu P3.4 se objevují data podle stisknutých kláves, na pinu P3.5 jsou data přijímána a zobrazována na displeji. Když piny P3.4 a P3.5 propojíme, pak na displeji vidíme znaky zadávané z klávesnice.

3.3. Příklady kombinovaných programů v BASICu a assembleru

3.3.1. Logický analyzátor

Dříve uvedené příklady využívaly pro uložení programu paměť, která je společná pro monitor i BASIC. Další programy už budou využívat spolupráci programu napsaného v BASICu i assembleru. Proto musíme uložit program buď do Flash nebo do části RAM, která není BASICem využívána. K tomu slouží paměť od adresy **TOPRAM** (viz kapitola 3.4.4). Jen je potřeba dát pozor na to, aby nedošlo k přepsání programu zásobníkem systému! (Aktuální informace o využití zásobníku nám poskytne příkaz **INFO 1**; jakmile však do této oblasti začneme zapisovat, bude příkaz INFO 1 poskytovat nepravdivé údaje až do resetu počítače!)

Dobrý postup je nejprve si v monitoru dobře otestovat program v assembleru s využitím sdílené RAM na adrese $0200 nebo na adrese TOPRAM, teprve po důkladném otestování program přepsat do Flash, kde zůstane uložený i po resetu systému. Příklady zde uvedené jsou připraveny pro zápis do TOPRAM.

První složitější příklad nám nabízí funkci logického analyzátoru. Po spuštění program čeká na spouštěcí puls na vstupu procesoru a následně do paměti uloží a zobrazí tvar zachyceného signálu. Funkce je tedy podobná příkladu v kapitole 1.5.4, pouze signály nejsou analogové, ale digitální. Výhodou může být, že je takto současně zachyceno více signálů najednou.

Nejprve si tedy napíšeme program v assembleru, který čeká na vstupní puls na pinu P6.7 a pak uloží zachycená data celého portu P6 do paměti. Kvůli dodržení stálé rychlosti načítání dat je nutné zakázat na začátku přerušení; na konci je opět přerušení povoleno. Pokud bychom takovou funkci ladili, je dobré vypustit instrukci DINT, případně ji nahradit např. instrukcí **NOP**, která nic nedělá, ale „drží“ místo pro doplnění instrukce DINT později, až si budeme jisti, že nám funkce správně funguje.

```
:$0840           ;R15=adresa
:$0840           ;R14=delka cekani
:$0840 DINT      ;zakazani preruseni
:$0842 MOV #128,R12
:$0846 BIT.B #$80,&P6IN
```

```
:$084C JNZ $0846 ;cekani na klidovy stav
:$084E BIT.B #$80,&P6IN
:$0854 JZ $084E  ;cekani na spousteci puls
:$0856 MOV.B &P6IN,0(R15)
:$085C INC R15
:$085E MOV R14,R13
:$0860 DEC R13
:$0862 JNZ $0860 ;cekaci smycka
:$0864 DEC R12
:$0866 JNZ $0856 ;vsechny vzorky nactene?
:$0868 EINT      ;povoleni preruseni
:$086A RET
```

Teď si napíšeme program v BASICu, který bude s naším strojovým podprogramem spolupracovat.

```
>10 REM Logicky analyzator
>20 DIM D(32)
>30 P=VARPTR D(0)
>40 D=$10000*P
>50 N=1 'delka cekani na každý vzorek
>60 REM Testovaci data
>70 REM FOR I=0 TO 31:D(I)=RND:NEXT
>80 REM Smazani displeje
>90 BPUT 12
>100 REM Cyklus pro mereni a zobrazovani
>110 REPEAT
>120 REM Zachyceni dat
>130 I=USR $840,D+N
>140 REM Prekresleni displeje
>150 FOR B=0 TO 7
>160 REM Smazani radku
>170 IF B BPUT 10
>180 IF B=FALSE BPUT 27,"[H"
>190 BPUT "                          " '42 mezer +2 pixely
>200 REM Nastaveni promennych pro další bit
>210 L=P?0 BIT B
>220 X=0
>230 Y=B*8+6
>240 Z=Y-5
>300 REM Cyklus pro kresleni jednoho prubehu
>310 FOR I=1 TO 125
>320 C=P?I BIT B
>330 IF C=L GOTO 400
>340 J=I-1
>350 IF L LINE X,Z,J,Z
>360 IF L=FALSE LINE X,Y,J,Y
>370 IF L LINE I,Y,J,Z
>380 IF L=FALSE LINE J,Y,I,Z
>390 X=I:L=C
>400 NEXT
>410 REM Dokresleni prubehu do konce
>420 IF X>=125 GOTO 500
>430 J=I-1
>440 IF L LINE X,Z,J,Z
>450 IF L=FALSE LINE X,Y,J,Y
>500 NEXT
>510 UNTIL FALSE
```

Pokud chceme program používat trvale, můžeme si uložit strojový podprogram do paměti Flash. Při

přepisování programu je potřeba zadat také správné adresy pro instrukce skoku. Upravená verze je uvedena zde:

```
:$FC00 DINT      ;zakazani preruseni
:$FC02 MOV #128,R12
:$FC06 BIT.B #$80,&P6IN
:$FC0C JNZ $FC06 ;cekani na klidovy stav
:$FC0E BIT.B #$80,&P6IN
:$FC14 JZ $FC0E  ;cekani na spousteci puls
:$FC16 MOV.B &P6IN,0(R15)
:$FC1C INC R15
:$FC1E MOV R14,R13
:$FC20 DEC R13
:$FC22 JNZ $FC20 ;cekaci smycka
:$FC24 DEC R12
:$FC26 JNZ $FC16 ;vsechny vzorky nactene?
:$FC28 EINT      ;povoleni preruseni
:$FC2A RET
```

Také je nutné upravit také řádek 130 v BASICu a použít tam správnou adresu, kde bude program uložený.

```
>130 I=USR $FC00,D+N
```

3.3.2. Osciloskop

Protože už jsme si jeden takový program ukazovali v kapitole 1.5.4, můžeme z něho převzít velkou část programu v BASICu. Upravíme ho tak, aby se data z ADC vyčítala v assemblerovém podprogramu. Nejprve zapíšeme část programovanou v assembleru.

```
:$0840 ;R15=adresa
:$0840 ;R14=delka cekani
:$0840 ;dalsi parametry jsou v pameti
:$0840 DINT              ;zakazani preruseni
:$0842 ;-------- priprava prevodu
:$0842 MOV #128,R12
:$0846 BIS #2,&ADC12CTL0 ;povoleni prevodu
:$084A CALL #$0880       ;spusteni prvniho prevodu
:$084E ;-------- cekani na signal
:$084E CALL #$087A       ;cekani na prevod
:$0852 CMP @R15,R13
:$0854 JNC $084E
:$0856 CALL #$087A       ;cekani na prevod
:$085A CMP @R15,R13
:$085C JC $0856
:$085E ;-------- hlavni smycka
:$085E MOV R13,0(R15)    ;ulozeni vysledku do pameti
:$0862 INCD R15
:$0864 MOV R14,R13
:$0866 DEC R13
:$0868 JNZ $0866         ;cekaci smycka
:$086A CALL #$087A       ;cekani na prevod
:$086E DEC R12
:$0870 JNZ $085E
:$0872 ;-------- ukonceni prevodu
:$0872 EINT
:$0874 BIC #2,&ADC12CTL0
:$0878 RET
:$087A ;-------- podprogram pro prevod
:$087A BIT #1,&ADC12IFG
```

```
:$087E JZ $087A          ;cekani na dokonceni prevodu
:$0880 MOV &ADC12MEM0,R13
:$0884 BIS #1,&ADC12CTL0 ;spusteni dalsiho prevodu
:$0888 RET
```

Teď si připravíme zobrazovací část v BASICu:

```
>10 REM nastaveni ADC
>20 ADC12CTL0=$8810
>30 ADC12CTL1=$0238
>40 ADC12MCTL0=$87
>50 P6SEL=$80
>60 REM pamet pro vzorky
>70 DIM V(64)
>80 P=VARPTR V(0)
>90 R=$10000*P
>100 N=1 'delka cekani
>110 T=2000 'uroven spusteni
>120 D=64 'meritko zobrazeni
>200 REPEAT
>210 REM spusteni prevodu
>220 V(0)=T
>230 A=USR $0840,R+N
>300 REM zobrazeni vysledku
>310 BPUT 12
>320 X=0
>330 Y=P
>400 FOR I=1 TO 127
>410 Q=O+2
>420 LINE J,@O/D,I,@Q/D
>430 O=Q:J=I
>440 NEXT
>490 UNTIL FALSE
```

Můžeme si teď ukázat jiný způsob zápisu programu ve strojovém kódu. Do programu v BASICu si zapíšeme instrukce ve formě hexadecimálních čísel. Tato čísla program zapíše do paměti RAM. Toto nám umožní mít strojový podprogram v RAM a přitom ho tam vždy automaticky zapsat po spuštění programu. Data pro zápis získáme v monitoru z výpisu MEMW.

```
:MEMW $0840
0840 C232 403C 0080 D3A2
0848 01A0 12B0 0880 12B0
0850 087A 9F2D 2BFC 12B0
0858 087A 9F2D 2FFC 4D8F
0860 0000 532F 4E0D 831D
0868 23FE 12B0 087A 831C
0870 23F6 D232 C3A2 01A0
0878 4130 B392 01A4 27FD
0880 421D 0140 D392 01A0
0888 4130 DEAD DEAD DEAD
```

Pokud chceme trochu šetřit místem, zapisujeme v BASICu program po 32-bitových slovech. První 16-bitové slovo je v 32-bitovém slově vždy v nižší části, následující 16-bitové slovo je ve vyšší části. Proto to vypadá, jako by každá dvě slova byla prohozena. Do programu pak doplníme následující části:

```
>150 REM zapis programu do RAM
>160 READ N:A=$840
>170 FOR I=1 TO N
>180 READ B:!A=B:A=A+4
>190 NEXT
```

```
>600 DATA 19 'pocet slov
>610 DATA $403CC232,$D3A20080,$12B001A0,$12B00880
>620 DATA $9F2D087A,$12B02BFC,$9F2D087A,$4D8F2FFC
>630 DATA $532F0000,$831D4E0D,$12B023FE,$831C087A
>640 DATA $D23223F6,$01A0C3A2,$B3924130,$27FD01A4
>650 DATA $0140421D,$01A0D392,$4130
```

3.4. *Detaily programového vybavení počítače*

V následujících kapitolách si popíšeme některé důležitější části systému počítače.

3.4.1. Obsazení paměti RAM

Adresa	Velikost	Popis
0200	1024	Paměť pro DIM a zásobník BASICu
0600	26× 4	Proměnné BASICu
0668	26× 4	Informace o blocích alokovaných příkazem DIM
06D0	128	Příkazová řádka
0750	16× 2	Adresy vektorů přerušení
0770		Blok globálních proměnných; viz kapitola 3.4.5
		Lokální proměnné systému
TOPRAM	450	Zásobník procesoru

3.4.2. Tabulka vektorů přerušení

Pokud chceme použít pro některou periferii zpracování dat v přerušení, musíme napsat obslužný program v assembleru, zapsat adresu obslužného programu do tabulky vektorů a povolit dané přerušení příslušným bitem.

Adresa	Vektor	Adresa	Vektor
0750	Nepoužitý	0760	USART0 TX
0752	Rezervovaný	0762	USART0 RX
0754	USART1 TX	0764	WDT
0756	USART1 RX	0766	Comparator
0758	P1 edge	0768	Rezervovaný
075A	TimerA (CC1-2, TA)	076A	Rezervovaný
075C	TimerA CC0	076C	NMI
075E	ADC	076E	Rezervovaný

3.4.3. Rozdělení paměti Flash

Adresa		Popis
1000	10FF	Tabulka adres funkcí
1100		Basic, Monitor
8000		Uživatelský program v BASICu
		Volná Flash

Adresa		Popis
	FDFF	Uživatelské programy v assembleru
FE00	FFFF	Systémová tabulka vektorů

3.4.4. Tabulka funkcí

Systémové funkce je možné používat ve svých programech, pokud budeme dodržovat pár jednoduchých pravidel:

- registry R12 až R15 se používají na předávání parametrů a jejich hodnoty mohou být po návratu z funkce změněny
- pokud je potřeba hodnoty v R12 až R15 zachovat, musí se před voláním funkcí uložit a po návratu z funkce opět obnovit (třeba na zásobník: instrukce PUSH a POP)
- registry R4 až R11 je možné libovolně používat v programu
- chování funkcí PUTC, PUTS, GETC, KEY, GETKEY, EDITSTR a PRINTVAL je určeno hodnotou globální proměnné current_channel (viz Tabulka 1).
- 32-bitové hodnoty jsou uloženy R14 = nižší slovo, R15 = vyšší slovo

Adresa	Funkce	R15	R14	R13	R12	Popis funkce
1000	GLOBAL					Adresa bloku globálních proměnných
1002	TOPRAM					Adresa volné části RAM
1004	PUTC	znak				Tisk zadaného znaku
1006	PUTS	text				Tisk textu od zadané adresy
1008	GETC	→znak				Čekání na klávesu a vrácení ASCII kódu
100A	KEY	→0/1				Pokud je připravena klávesa ke zpracování, vrátí hodnotu 1, jinak vrátí 0
100C	GETKEY	→znak				Vrací hodnotu ASCII kódu nebo kód řídících funkcí (šipky, Fn aj.)
100E	EDITSTR	text →0/1	max			Umožní editaci textu na displeji; R15 obsahuje adresu textu, R14 maximální délku textu Po ukončení vrací hodnotu 1, je-li editace ukončena klávesou Enter, nebo 0, je-li ukončena klávesou CTRL+C
1010	PRINTVAL	32-bitová hodnota		režim		Tisk zadaného čísla podle vypraného režimu: 0=dekadicky, 1=hexadecimálně
1012	LCDMODE	xor	or			Nastavení režimu výpisu znaků nebo grafiky na displeji
1014	LCDPIXEL	X	Y			Vykreslení bodu na displeji v zadaných souřadnicích
1016	LCDLINE	X1	Y1	X2	Y2	Vykreslení úsečky na displeji z bodu X1,Y1 do bodu X2,Y2
1018	RND	→náhoda				Vrátí pseudonáhodnou 32-bitovou hodnotu
101A	DELAY	čas				Čekání podle času zadaného v milisekundách
101C	BREAK					Při spuštění programu v monitoru slouží k zastavení na vybraném místě; dále je pak možné program krokovat, měnit registry nebo znovu spustit

3.4.5. Tabulka globálních proměnných

Adresa	Velikost	Popis
0770	1	Aktuálního komunikační kanál; hodnota se mění, pokud jsou vykonány příkazy BASICu se zadaným kanálem

Adresa	Velikost	Popis
0771	1	Defaultní komunikační kanál; hodnota se nastavuje po resetu podle aktuálně připojených komunikačních prostředků (klávesnice, terminál)
0772	4	Hodnota systémového časovače (TICK)
0776	4	Vnitřní hodnota pro výpočet pseudonáhodného čísla (RANDOMIZE)

3.4.6. Tabulka kódů

Nejprve uvedeme ASCII tabulku, která obsahuje kódy znaků od 0 do 127. Tyto kódy vrací shodně obě funkce GETC i GETKEY. Kódy znaků jsou uvedeny v hexadecimálním tvaru. Každý sloupeček obsahuje kód pro vyšší nibl kódu, řádek obsahuje nižší nibl znaku. Kód vybraného znaku získáme složením obou niblů (např. znak F má kód $4_ a $_6, tedy $46).

	$0_	$1_	$2_	$3_	$4_	$5_	$6_	$7_
$_0		CTRL+P	mezera	0	@	P	`	p
$_1	CTRL+A	CTRL+Q	!	1	A	Q	a	q
$_2	CTRL+B	CTRL+R	"	2	B	R	b	r
$_3	CTRL+C	CTRL+S	#	3	C	S	c	s
$_4	CTRL+D	CTRL+T	$	4	D	T	d	t
$_5	CTRL+E	CTRL+U	%	5	E	U	e	u
$_6	CTRL+F	CTRL+V	&	6	F	V	f	v
$_7	CTRL+G	CTRL+W	'	7	G	W	g	w
$_8	Backspace	CTRL+X	(	8	H	X	h	x
$_9	Tab	CTRL+Y	)	9	I	Y	i	y
$_A	CTRL+J	CTRL+Z	*	:	J	Z	j	z
$_B	CTRL+K	Esc	+	;	K	[	k	{
$_C	CTRL+L	CTRL+\	,	<	L	\	l	|
$_D	Enter	CTRL+]	-	=	M	]	m	}
$_E	CTRL+N	CTRL+^	.	>	N	^	n	~
$_F	CTRL+O	CTRL+_	/	?	O	_	o	Delete

Další tabulka obsahuje kódy řídících kláves, jak je vrací funkce GETKEY.

Klávesa	Kód	Klávesa	Kód	Klávesa	Kód	Klávesa	Kód
Home	$101	F1	$10B	F7	$112	nahoru	$165
Insert	$102	F2	$10C	F8	$113	dolů	$166
End	$104	F3	$10D	F9	$114	vlevo	$167
Page Up	$105	F4	$10E	F10	$115	vpravo	$168
Page Down	$106	F5	$10F	F11	$117		
		F6	$111	F12	$118		

Držení klávesy CTRL spolu s řídící klávesou nastaví bit $200, držení klávesy SHIFT nastaví bit $400 a držení klávesy ALT nastaví bit $800. Např. CTRL+vlevo vrátí kód $367, SHIFT+vlevo vrátí kód $567.

4. Kapitola pro experty

Pokud chceme začít využívat více možností, které nám procesor nabízí, můžeme se seznámit s tím, jak procesor funguje a jakými periferiemi je vybaven. V následujících kapitolách je pouze stručný úvod k jednotlivým částem. Pro detailnější informace je potřeba se vypravit na web výrobce procesoru: www.ti.com (více informací v kapitole 5.3).

4.1. Popis zapojení počítače

Počítač MSP430SBC obsahuje pouze nejnutnější obvody pro to, aby bylo možné začít programovat: napájecí zdroj, mikrokontrolér s připojeným oscilátorem, grafický LCD displej a PS/2 konektor pro připojení klávesnice, 40-pinový konektor pro řízení vlastních periferií. Schéma zapojení najdete v příloze 5.4.

Funkce jednotlivých pinů konektoru je popsána v následující tabulce:

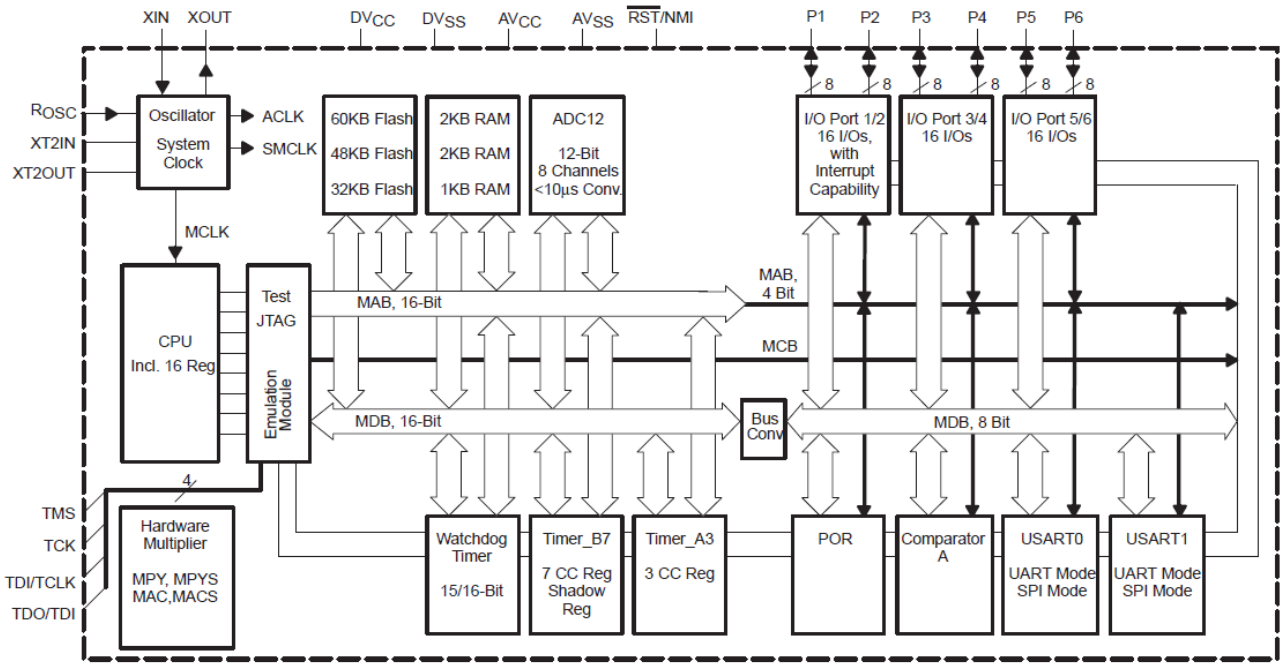
Pin	Port	Funce	Pin	Port	Funkce
1	Vcc		2	P6.0	A0
3	P6.1	A1	4	P6.2	A2
5	P6.3	A3	6	P6.4	A4
7	P6.5	A5	8	P6.6	A6
9	P6.7	A7	10	GND	
11	Vcc		12	P1.0	TACLK
13			14	P1.2	TA1
15	P1.3	TA2	16	P1.4	SMCLK
17	P1.5	TA0	18	P1.6	TA1
19	P1.7	TA2	20	GND	
21	Vcc		22	P3.0	STE0
23	P3.1	SIMO0	24	P3.2	SOMI0
25	P3.3	UCLK0	26	P3.4	UTXD0
27	P3.5	URXD0	28	P3.6	UTXD1
29	P3.7	URXD1	30	GND	
31	Vcc		32	P4.0	TB0
33	P4.1	TB1	34	P4.2	TB2
35	P4.3	TB3	36	P4.4	TB4
37	P4.5	TB5	38	P4.6	TB6
39	P4.7	TBCLK	40	GND	

Tabulka 4: Popis I/O konektoru

4.2. Popis procesoru

Mikrokontrolér MSP430F149 patří do rodiny obvodů MSP430 firmy Texas Instruments. Jedná se o 16-bitové procesory určené pro zpracování analogových a digitálních signálů (Mixed Signal Processor) s důrazem na nízkou spotřebu. Architektura procesoru pochází z roku 1992, první obvody z této řady se objevily na trhu v roce 1996. Použitý obvod MSP430F149 se začal prodávat v roce 2000.

Mikrokontrolér se skládá z procesorového jádra, bloku pro generování hodinových signálů, paměti Flash a RAM a množství různých periferií (12-bitový ADC, 2 16-bitové časovače, 2 sériové porty UART/SPI, množství vstupně/výstupních pinů).



Ilustrace 3: Blokové schéma MSP430F149

Procesor při své práci načítá instrukce z paměti Flash nebo RAM a vykonává je. Při tom může číst data z jednotlivých periferií a také je do periferií zapisovat. V následujících kapitolách si popíšeme funkce jednotlivých částí.

4.2.1. Procesorové jádro

Procesor obsahuje 16 16-bitových registrů R0 – R15. Některé registry mají speciální určení, ostatní jsou pro volné použití v programu.

Registr R0 obsahuje adresu načítaných instrukcí. Označuje se také **PC** (Program Counter). Protože jsou instrukce 16-bitové, načítá se vždy 16-bitové slovo z sudé adresy. Proto je nejnižší bit tohoto registru vždy nulový.

Čítač instrukcí	0

Registr R1 je použitý jako ukazatel zásobníku **SP** (Stack Pointer). Sem si může program ukládat data z registrů, které právě potřebuje program pro jiné výpočty. Také sem ukládá návratové adresy při volání podprogramů. Zásobník je 16-bitový, proto i zde je nejnižší bit vždy nulový.

Ukazatel zásobníku	0

Registr R2 obsahuje stavové bity procesoru **SR** (Status Register): výsledky operací, stav přerušení, uspání procesoru.

							V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C

C bit je nastaven, pokud během operace došlo k přenosu z nejvyššího bitu

Z bit je nastaven, pokud je výsledek operace nulový

N bit je nastaven, pokud je výsledek operace záporný

V bit je nastaven, pokud během operace došlo k přetečení rozsahu znaménkového čísla

GIE pokud je bit nastaven, je povoleno přerušení programu

CPUOFF pokud je bit nastaven, program neběží a procesor spí

OSCOFF pokud je bit nastaven, jsou všechny oscilátory procesoru zakázány (nejnižší spotřeba)

SCG1:SCG0 určují nastavení hodin během spánku procesoru

Registry R4 až R15 je možné používat libovolně v programu.

4.2.2. Adresovací režimy

Adresovací režimy procesoru říkají, jakým způsobem má získat data pro vykonání instrukce. Režimy jsou uvedeny v tabulce.

Adresovací režim	Způsob zápisu
Registrový režim	R4, R15
Indexový režim	4(R11)
Symbolický režim	<adresa>
Absolutní režim	&<adresa>
Nepřímý registrový režim	@R5
Nepřímý registrový režim s autoinkrementací	@R6+
Konstanty	#<hodnota>

Registrový režim je nejjednodušší a nejrychlejší režim: hodnota pro výpočet je získána z registru nebo výsledek je zapsán do registru.

Pokud je potřeba pro výpočet použít přímo hodnotu, máme pro to režim načtení konstanty.

Absolutní režim použijeme, pokud potřebujeme načíst nebo zapsat hodnotu třeba do registru nějaké periferie procesoru nebo přistupovat k nějakému místu v paměti procesoru.

Indexový režim se používá pro přístup k datovým strukturám v paměti.

Symbolický režim můžeme používat například pro přístup k datům, která jsou součástí strojového kódu.

Nepřímý registrový režim se používá pro čtení dat, jejichž adresu máme uloženou v registru (třeba proto, že jsme ji museli někde načíst nebo nějak spočítat). Tento režim je možné použít pouze pro načtení, ale ne pro ukládání dat.

Nepřímý registrový režim s autoinkrementací používáme podobně jako předchozí režim pouze pro načtení dat, například pro postupné načítání textu. Po každém načtení se totiž obsah adresovacího registru zvýší o 1 nebo 2 podle toho, jestli načítáme byte (8 bitů) nebo word (16 bitů).

4.2.3. Instrukce procesoru

Nyní si uvedeme přehlednou tabulku se všemi instrukcemi procesoru:

Instrukce		Popis	Funkce	V	N	Z	C
ADC(.B)†	dst	Přičtení bitu C k dst	dst + C → dst	*	*	*	*
ADD(.B)	src,dst	Přičtení src k dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	Přičtení src a bitu C k dst	src + dst + C → dst	*	*	*	*
AND(.B)	src,dst	Logický součin src a dst	src .and. dst → dst	0	*	*	*
BIC(.B)	src,dst	Nulování bitů v dst podle masky src	.not. src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	Nastavení bitů v dst podle masky src	src .or. dst → dst	–	–	–	–
BIT(.B)	src,dst	Testování bitů v dst podle masky src	src .and. dst	0	*	*	*
BR†	dst	Skok na adresu dst	dst → PC	–	–	–	–
CALL	dst	Skok do podprogramu na adrese dst	PC + 2 → stack dst → PC	–	–	–	–
CLR(.B)†	dst	Nulování dst	0 → dst	–	–	–	–
CLRC†		Nulování bitu C	0 → C	–	–	–	0

Instrukce		Popis	Funkce	V	N	Z	C
CLRN†		Nulování bitu N	0 → N	–	0	–	–
CLRZ†		Nulování bitu Z	0 → Z	–	–	0	–
CMP(.B)	src,dst	Porovnání src a dst	dst - src	*	*	*	*
DADC(.B)†	dst	BCD kód: přičtení bitu C k dst	dst + C → dst (BCD)	*	*	*	*
DADD(.B)	src,dst	BCD kód: přičtení src a bitu C k dst	src + dst + C → dst (BCD)	*	*	*	*
DEC(.B)†	dst	Odečtení 1 od dst	dst - 1 → dst	*	*	*	*
DECD(.B)†	dst	Odečtení 2 od dst	dst - 2 → dst	*	*	*	*
DINT		Zakázání přerušení	0 → GIE	–	–	–	–
EINT		Povolení přerušení	1 → GIE	–	–	–	–
INC(.B)†	dst	Přičtení 1 k dst	dst + 1 → dst	*	*	*	*
INCD(.B)†	dst	Přičtení 2 k dst	dst + 2 → dst	*	*	*	*
INV(.B)†	dst	Bitová inverze dst	.not. dst → dst	*	*	*	*
JC	adresa	Skok na adresu, je-li bit C nastavený	dst → dst	–	–	–	–
JZ	adresa	Skok na adresu, je-li bit Z nastavený		–	–	–	–
JGE	adresa	Skok na adresu, jsou-li bity N a V stejné		–	–	–	–
JL	adresa	Skok na adresu, jsou-li bity Na V rozdílné		–	–	–	–
JMP	adresa	Skok na adresu	PC + 2*offset → PC	–	–	–	–
JN	adresa	Skok na adresu, je-li bit N nastavený		–	–	–	–
JNC	adresa	Skok na adresu, je-li bit C nulový		–	–	–	–
JNZ	adresa	Skok na adresu, je-li bit Z nulový		–	–	–	–
MOV(.B)	src,dst	Přesun dat z src do dst	src → dst	–	–	–	–
NOP†		Žádná operace		–	–	–	–
POP(.B)†	dst	Odebrání položky ze zásobníku	@SP → dst SP + 2 → SP	–	–	–	–
PUSH(.B)	src	Uložení položky na zásobník	SP – 2 → SP src → @SP	–	–	–	–
RET†		Návrat z podprogramu	@SP → PC SP + 2 → SP	–	–	–	–
RETI		Návrat z přerušení	@SP → SR SP + 2 → SP @SP → PC SP + 2 → SP	*	*	*	*
RLA(.B)†	dst	Rotace vlevo aritmetická	C ← dst ← 0	*	*	*	*
RLC(.B)†	dst	Rotace vlevo přes bit C	C ← dst ← C	*	*	*	*
RRA(.B)	dst	Rotace vpravo aritmetická	dst.msb → dst → C	0	*	*	*
RRC(.B)	dst	Rotace vpravo přes bit C	C → dst → C	0	*	*	*
SBC(.B)†	dst	Odečtení negovaného bitu C od dst	dst + FFFF + C → dst	*	*	*	*
SETC†		Nastavení bitu C	1 → C	–	–	–	1
SETN†		Nastavení bitu N	1 → N	–	1	–	–
SETZ†		Nastavení bitu Z	1 → Z	–	–	1	–
SUB(.B)	src,dst	Odečtení src od dst	dst + .not. src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	Odečtení src a negovaného bitu C od dst	dst + .not. src + C → dst	*	*	*	*
SWPB	dst	Prohození bytů ve slově dst		–	–	–	–
SXT	dst	Znaménkové rozšíření bytu do slova		0	*	*	*
TST(.B)†	dst	Porovnání dst s nulou	dst + FFFF + 1	0	*	*	1

Instrukce		Popis	Funkce	V	N	Z	C
XOR(.B)	src,dst	Logický „xor“ src a dst	src .xor. dst → dst	*	*	*	*

Instrukce označené znakem † jsou tzv. emulované instrukce. Dají se zapsat jako skutečné instrukce procesoru, ale v této zjednodušené formě usnadňují programování — zapisuje se menší počet operandů instrukce (nebo také žádný).

Sloupečky V, N, C a Z udávají, jaké příznaky jsou instrukcemi ovlivňované:

*	instrukce daný příznak ovlivňuje
–	instrukce daný příznak neovlivňuje
0	instrukce daný příznak nuluje
1	instrukce daný příznak nastavuje
src	zdrojová data; data nejsou operací změněna
dst	cílová (pro některé instrukce zároveň zdrojová) data; data jsou operací změněna

Příklady instrukcí ovlivňujících bity SR

instrukce	src	dst	výsledek	V	N	Z	C
ADD.B	70	0F	7F	0	0	0	0
	00	00	00	0	0	1	0
	70	91	01	0	0	0	1
	70	90	00	0	0	1	1
	70	8F	FF	0	1	0	0
	FF	FF	FE	0	1	0	1
	80	81	01	1	0	0	1
	80	80	00	1	0	1	1
	70	10	80	1	1	0	0
SUB	FFFF	0001	0002	0	0	0	0
	0000	0001	0001	0	0	0	1
	0000	0000	0000	0	0	1	1
	0001	0000	FFFF	0	1	0	0
	0001	FFFF	FFFE	0	1	0	1
	0001	8000	7FFF	1	0	0	1
	8000	0000	8000	1	1	0	0
RRC.B	C=0	FE	7F	0	0	0	0
	C=0	FF	7F	0	0	0	1
	C=0	00	00	0	0	1	0
	C=0	01	00	0	0	1	1
	C=1	00	80	0	1	0	0
	C=0	01	00	0	1	0	1
RRA		0002	0001	0	0	0	0
		0000	0000	0	0	1	0
		0001	0000	0	0	1	1
		8000	C000	0	1	0	0
		8001	C000	0	1	0	1
AND.B BIT.B	01	01	01	0	0	0	1
	01	02	00	0	0	1	0
	81	82	80	0	1	0	1
XOR	0001	0000	0001	0	0	0	1

instrukce	src	dst	výsledek	V	N	Z	C
	0001	0001	0000	0	0	1	0
	8000	0000	8000	0	1	0	1
	8000	8001	0001	1	0	0	1
	8000	8000	0000	1	0	1	0
SXT		0001	0001	0	0	0	1
		0000	0000	0	0	1	0
		0080	FF80	0	1	0	1

Tabulka 5: Příklad ovlivnění stavových bitů

4.2.4. Mapa paměti procesoru MSP430F149

Následující tabulka uvádí rozložení jednotlivých oblastí v adresovém prostoru procesoru.

Adresa		Popis
0000	00FF	8-bitové registry periferií
0100	01FF	16-bitové registry periferií
0200	09FF	Paměť RAM
0C00	0FFF	Program BSL (bootloader = zavaděč programu)
1000	10FF	Paměť Flash
1100	FFFF	Paměť Flash

Přehled všech registrů periferií je uvedený v tabulkách v kapitole 5.2.

4.3. Popis periferií

Důkladný popis všech periferií můžeme najít v manuálu k procesorům MSP430x1xx, který je volně dostupný na webu firmy TI. Přesný název dokumentu najdeme v kapitole 5.3. Význam značek pro stav registrů po resetu najdeme v kapitole 5.2.

4.3.1. Vstupně–výstupní porty

Každý port P1 až P6 obsahuje 8 bitů. Řídí se proto několika 8-bitovými registry: vstupní data se čtou z registru PxIN, výstupní data se zapisují do registru PxOUT. Funkce vstupu nebo výstupu se volí zápisem do registru PxDIR, speciální HW funkce se vybírá zápisem do registru PxSEL.

Porty P1 a P2 mají ještě funkci přerušení při změně vstupní úrovně. K tomu slouží registry PxIES, které nastavují vzestupnou nebo sestupnou hranu. Registr PxIE povoluje přerušení pro jednotlivé bity portu a stav příznaků přerušení od jednotlivých bitů portu se čte z registru PxIFG.

Registry vstupně–výstupních portů

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0018	**P3IN**									
0019	**P3OUT**	uuuuuuuu								
001A	**P3DIR**	00000000								
001B	**P3SEL**	00000000								
001C	**P4IN**									
001D	**P4OUT**	uuuuuuuu								
001E	**P4DIR**	00000000								

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
001F	**P4SEL**	00000000								
0020	**P1IN**									
0021	**P1OUT**	uuuuuuuu								
0022	**P1DIR**	00000000								
0023	**P1IFG**	00000000								
0024	**P1IES**	uuuuuuuu								
0025	**P1IE**	00000000								
0026	**P1SEL**	00000000								
0028	**P2IN**									
0029	**P2OUT**	uuuuuuuu								
002A	**P2DIR**	00000000								
002B	**P2IFG**	00000000								
002C	**P2IES**	uuuuuuuu								
002D	**P2IE**	00000000								
002E	**P2SEL**	00000000								
0030	**P5IN**									
0031	**P5OUT**	uuuuuuuu								
0032	**P5DIR**	00000000								
0033	**P5SEL**	00000000								
0034	**P6IN**									
0035	**P6OUT**	uuuuuuuu								
0036	**P6DIR**	00000000								
0037	**P6SEL**	00000000								

V následující tabulce je uvedený popis funkcí bitů registrů pro řízení vstupně–výstupních portů.

Bity	Hodnota	Význam
PxDIR	0	Vstup
	1	Výstup
PxIN		Vstupní hodnota na pinu
PxOUT		Výstupní hodnota pro pin (PxDIR musí být 1 a PxSEL musí být 0)
PxSEL	0	Vstupně výstupní funkce
	1	Alternativní funkce (viz Tabulka 4).
PxIES	0	Přerušení pro změnu 0 → 1
	1	Přerušení pro změnu 1 → 0
PxIE	0	Přerušení zakázáno
	1	Přerušení povoleno
PxIFG	0	Nastavená změna podle PxIES na pinu neproběhla
	1	Nastavená změna na pinu proběhla

4.3.2. Watchdog časovač

Watchdog časovač se zpravidla používá pro časový dohled nad funkcí zařízení. Pokud nefunguje zařízení správně, tj. pokud se přestane nulovat watchdog časovač, dojde k resetu procesoru. Kromě funkce watchdog může WDT fungovat i jako obyčejný časovač (bez resetování procesoru) s pevně nastavitelným časem.

Pro nastavení se používá jediný 16-bitový registr WDTCTL. Při zápisu do tohoto registru je potřeba zapisovat horní bajt s hodnotou $5A00, jinak dojde k okamžitému resetu procesoru!

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0120	**WDTCTL**	01101001	Čtení: $69; Zápis: $5A							
		00000000	WDT HOLD	WDT NMIES	WDT NMI	WDT TMSEL	WDT TCNTCL	WDT SSEL	WDTISx	

Funkce jednotlivých bitů jsou popsány v následující tabulce.

Bity	Hodnota	Význam
WDTHOLD	0	Časovač pracuje
	1	Časovač je zastavený
WDTNMIES	0	NMI funkce pro signál 0 → 1
	1	NMI funkce pro signál 1 → 0
WDTNMI	0	RST/NMI pin má funkci Reset
	1	RST/NMI pin má funkci NMI
WDTTMSEL	0	Režim watchdog
	1	Režim časovač
WDTCNTCL	0	
	1	Nulování časovače
WDTSSEL	0	Čítač používá jako zdroj hodin signál SMCLK (8MHz)
	1	Čítač používá jako zdroj hodin signál ACLK (32768Hz)
WDTISx	00	Perioda časovače je 32768 (1s pro ACLK, 4ms pro SMCLK)
	01	Perioda časovače je 8192 (250ms pro ACLK, 1ms pro SMCLK)
	10	Perioda časovače je 512 (15.6ms pro ACLK, 64µs pro SMCLK)
	11	Perioda časovače je 64 (1.95ms pro ACLK, 8µs pro SMCLK)

4.3.3. Časovač TimerA

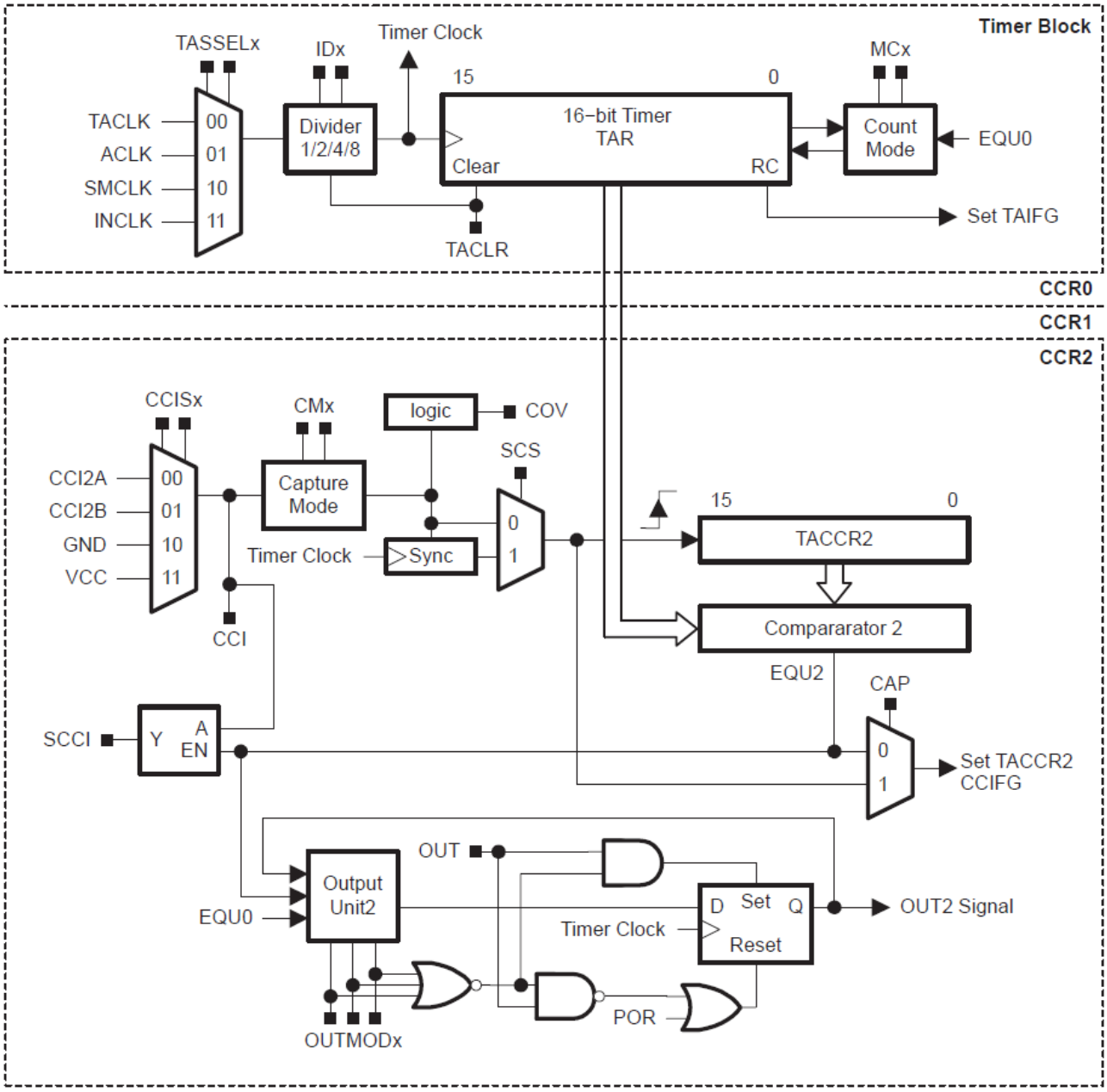
Hlavní registr časovače je 16-bitový čítač TAR. Tento registr podle nastavení v bitech MCx čítá od 0 do $FFFF nebo od 0 do hodnoty v registru TACCR0 a opět od 0 případně po dosažení hodnoty TACCR0 odčítá zpět k 0. Rychlost čítání se nastaví bity TASSELx (výběr zdroje hodin) a IDx (předdělička).

K dispozici jsou 3 registry TACCRx, které mohou fungovat v režimu porovnání (compare) nebo zachycení (capture) podle nastavení bitu CAP.

Režim compare porovnává neustále hodnotu TAR a TACCRx a pokud je hodnota shodná, provede akci podle nastavení bitů OUTMODx. Registr TACCR0 může být navíc použitý k omezení rozsahu čítání čítače TAR.

Režim capture slouží k zachycení obsahu čítače TAR do registru TACCRx při detekci vzestupné nebo sestupné hrany (případně obou hran) signálu CCIxA nebo CCIxB podle nastavení bitů CMx a CCISx.

Na následujícím obrázku je blokové schéma časovače TA s vyobrazenými vazbami mezi jednotlivými částmi.



Registry časovače TA

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
012E	**TAIV**	00000000	0	0	0	0	0	0	0	0
		00000000	0	0	0	0	TAIVx			0
0160	**TACTL**	00000000							TASSELx	
		00000000	IDx		MCx			TACLR	TAIE	TAIFG
0162	**TACCTL0**	00000000	CMx		CCISx		SCS	SCCI		CAP
		00000000	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
0164	**TACCTL1**	00000000	Viz TACCTL0							
		00000000								
0166	**TACCTL2**	00000000	Viz TACCTL0							
		00000000								

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0170	**TAR**	00000000	Čítač TimerA							
		00000000								
0172	**TACCR0**	00000000	Capture/compare registr 0							
		00000000								
0174	**TACCR1**	00000000	Capture/compare registr 1							
		00000000								
0176	**TACCR2**	00000000	Capture/compare registr 2							
		00000000								

Dále jsou popsané funkce jednotlivých bitů.

Bity	Hodnota	Význam
TASSELx	00	Čítač používá jako zdroj hodin vstup TACLK
	01	Čítač používá jako zdroj hodin vnitřní signál ACLK (32768Hz)
	10	Čítač používá jako zdroj hodin vnitřní signál SMCLK (8MHz)
	11	Čítač používá jako zdroj hodin vstup INCLK
IDx	00	Zdroj hodin je připojený přímo
	01	Zdroj hodin je dělený 2
	10	Zdroj hodin je dělený 4
	11	Zdroj hodin je dělený 8
MCx	00	Čítač je zastavený a nečítá
	01	Čítač čítá a nuluje se po dosažení hodnoty nastavené v registru TACCR0
	10	Čítač čítá a nuluje se po dosažení hodnoty $FFFF
	11	Čítač čítá, po dosažení hodnoty nastavené v registru TACCR0 čítá zpět k nule
TACLR	0	
	1	Registr TAR se vynuluje
TAIE	0	Přerušení zakázáno
	1	Přerušení při dosažení hodnoty 0 povoleno
TAIFG	0	Nedošlo k dosažení hodnoty 0 čítačem
	1	Došlo k dosažení hodnoty 0 čítačem
CMx	00	Nepoužívá se zachycení (capture)
	01	Zachycení na vzestupnou hranu
	10	Zachycení na sestupnou hranu
	11	Zachycení na vzestupnou i sestupnou hranu
CCISx	00	Pro zachycení se používá vstup CCIxA
	01	Pro zachycení se používá vstup CCIxB
	10	Vstup pro zachycení je připojený na GND
	11	Vstup pro zachycení je připojený na Vcc
SCS	0	Asynchronní zachycení
	1	Synchronní zachycení
SCCI		Synchronizovaný stav vybraného vstupu pro zachycení
CAP	0	Režim porovnání (compare); signál TAx je výstupní
	1	Režim zachycení (capture); signál CCIx je vstupní
OUTMODx	000	Výstup je přímo řízený bitem OUT

Bity	Hodnota	Význam
	001	Při dosažení hodnoty TACCRx je výstup nastavený na 1
	010	Při dosažení hodnoty TACCRx je změněný stav výstupu, při dosažení 0 je výstup nastavený na 0
	011	Při dosažení hodnoty TACCRx je výstup nastavený na 1, při dosažení 0 je výstup nastavený na 0
	100	Při dosažení hodnoty TACCRx je změněný stav výstupu
	101	Při dosažení hodnoty TACCRx je výstup nastavený na 0
	110	Při dosažení hodnoty TACCRx je změněný stav výstupu, při dosažení 0 je výstup nastavený na 1
	111	Při dosažení hodnoty TACCRx je výstup nastavený na 0, při dosažení 0 je výstup nastavený na 1
CCIE	0	Přerušení pro capture/compare je zakázáno
	1	Přerušení pro capture/compare je povoleno
CCI		Stav vybraného vstupu pro zachycení
OUT	0	Výstup v režimu 000 je nastavený na hodnotu 0
	1	Výstup v režimu 000 je nastavený na hodnotu 1
COV	0	Nedošlo k přetečení při zachycení
	1	Došlo k přetečení při zachycení; předešlá data z registru TACCRx jsou ztracena
CCIFG	0	Nedošlo k zachycení nebo porovnání TACCRx
	1	Došlo k zachycení nebo porovnání TACCRx
TAIVx	000	Žádné přerušení časovače není aktivní
	001	TACCR1 CCIFG přerušení je aktivní
	010	TACCR2 CCIFG přerušení je aktivní
	101	TAIFG přerušení je aktivní

	TA0	TA1	TA2
CCIxA	TA0	TA1	TA2
CCIxB	TA0	CAOUT	ACLK

4.3.4. Časovač TimerB

Pro některé úlohy lze částečně použít také časovač TimerB, respektive registry TBCCTL2 až TBCCTL6 a k nim příslušející registry TBCCRx. Režim a hodiny časovače TB není možné měnit, protože je využíván pro časování celého systému. Časovač používá hodiny 8MHz a čítá v režimu od 0 do $FFFF s periodou 122Hz. V knize je popsáno několik příkladů, jak je možné tento časovač použít např. pro řízení LED pomocí PWM.

Nastavení a používání časovače je shodné s časovačem TB až na některé odlišnosti, kterými se však vzhledem k omezenému rozsahu knihy nebudeme zabývat.

Registry časovače TB

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0186	**TBCCTL2**	00000000	CMX		CCISx		SCS	CLLDx		CAP
		00000000	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
0188	**TBCCTL3**	00000000	Viz TBCCTL2							

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
		00000000								
018A	**TBCCTL4**	00000000	Viz TBCCTL2							
		00000000								
018C	**TBCCTL5**	00000000	Viz TBCCTL2							
		00000000								
018E	**TBCCTL6**	00000000	Viz TBCCTL2							
		00000000								
0196	**TBCCR2**	00000000	Capture/compare registr 2							
		00000000								
0198	**TBCCR3**	00000000	Capture/compare registr 3							
		00000000								
019A	**TBCCR4**	00000000	Capture/compare registr 4							
		00000000								
019C	**TBCCR5**	00000000	Capture/compare registr 5							
		00000000								
019E	**TBCCR6**	00000000	Capture/compare registr 6							
		00000000								

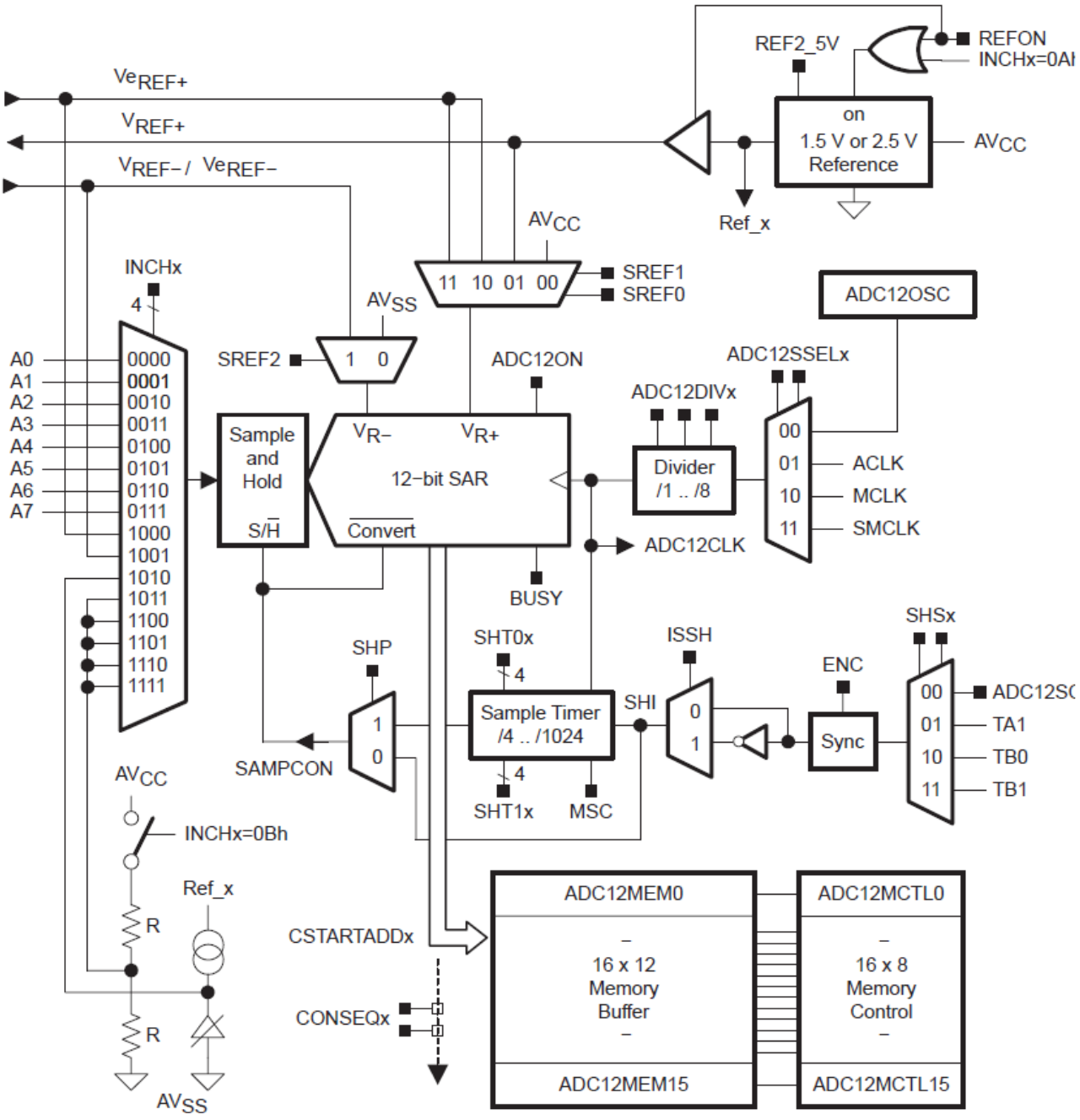
V následující tabulce je popsané pouze rozdílné nastavení vstupů CCIx u časovače TB.

	TB2	TB3	TB4	TB5	TB6
CCIxA	TB2	TB3	TB4	TB5	TB6
CCIxB	TB2	TB3	TB4	TB5	ACLK

4.3.5. Analogově–digitální převodník

Analogově–digitální převodník (ADC) měří napětí připojené na zvoleném analogovém vstupu portu P6 na digitální hodnotu. Nastavení ADC je poměrně komplikované, úplný popis je v dokumentaci výrobce procesoru.

Před spuštěním převodu je potřeba nastavit časování převodníku: frekvenci (výběr hodin ADC12SSELx a dělička ADC12DIVx), časování vzorkování signálu (SHT1x a SHT0x), musíme si vybrat referenci (tj. maximální napětí, které dokáže ADC měřit; REF2_5V, REFON, SREFx), nastavit režim převodu (bity MSC, SHSx, CSTARADDx, SHP, ISSH, CONSEQx) a vybrat převáděný kanál nebo více kanálů (INCHx, EOS). AADC musí být zapnutý (bit ADC12ON), musí být povoleno spuštění převodu (bit ENC) a nakonec musí být spuštěný převod (bit ADC12SC), pokud není nastavené automatické spouštění převodu (např. od časovače v přesně daném intervalu nebo se převod může spouštět stále dokola). Následující blokové schéma ADC zobrazuje vnitřní funkci a vazby v modulu.



Registry ADC

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0080	**ADC12MCTL0**	00000000	EOS	SREFx			INCHx			
0081	**ADC12MCTL1**	00000000								
0082	**ADC12MCTL2**	00000000								
0083	**ADC12MCTL3**	00000000								
0084	**ADC12MCTL4**	00000000								
0085	**ADC12MCTL5**	00000000								
0086	**ADC12MCTL6**	00000000								
0087	**ADC12MCTL7**	00000000								

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0088	**ADC12MCTL8**	00000000								
0089	**ADC12MCTL9**	00000000								
008A	**ADC12MCTL10**	00000000								
008B	**ADC12MCTL11**	00000000								
008C	**ADC12MCTL12**	00000000								
008D	**ADC12MCTL13**	00000000								
008E	**ADC12MCTL14**	00000000								
008F	**ADC12MCTL15**	00000000								

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0140	**ADC12MEM0**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0142	**ADC12MEM1**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0144	**ADC12MEM2**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0146	**ADC12MEM3**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0148	**ADC12MEM4**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
014A	**ADC12MEM5**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
014C	**ADC12MEM6**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
014E	**ADC12MEM7**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0150	**ADC12MEM8**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0152	**ADC12MEM9**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0154	**ADC12MEM10**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0156	**ADC12MEM11**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0158	**ADC12MEM12**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
015A	**ADC12MEM13**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
015C	**ADC12MEM14**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
015E	**ADC12MEM15**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
01A0	**ADC12CTL0**	00000000	SHT1x				SHT0x			
		00000000	MSC	REF2_5V	REFON	ADC12	ADC12	ADC12	ENC	ADC12

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
						ON	OVIE	TOVIE		SC
01A2	**ADC12CTL1**	00000000	CSTARTADDx				SHSx		SHP	ISSH
		00000000	ADC12DIVx			ADC12SSELx		CONSEQx		ADC12 BUSY
01A4	**ADC12IFG**	00000000								
		00000000								
01A6	**ADC12IE**	00000000								
		00000000								
01A8	**ADC12IV**	00000000	0	0	0	0	0	0	0	0
		00000000	0	0	ADC12IVx					0

Funkce jednotlivých bitů jsou popsány v následující tabulce:

Bity	Hodnota	Význam
SHT1x SHT0x	0000	Čas vzorkování analogového signálu 4 takty
	0001	8 taktů
	0010	16 taktů
	0011	32 taktů
	0100	64 taktů
	0101	96 taktů
	0110	128 taktů
	0111	192 taktů
	1000	256 taktů
	1001	384 taktů
	1010	512 taktů
	1011	768 taktů
	11xx	1024 taktů
MSC	0	Každý převod musí být samostatně spuštěný
	1	Po prvním spuštění převodu jsou další převody spouštěny automaticky
REF2_5V	0	Vnitřní reference 1.5V
	1	Vnitřní reference 2.5V
REFON	0	Vnitřní reference vypnutá
	1	Vnitřní reference zapnutá
ADC12ON	0	Převodník vypnutý
	1	Převodník zapnutý
ADC12OVIE	0	Přerušení při přetečení ADC12MEMx zakázané
	1	Přerušení při přetečení ADC12MEMx je povolené
ADC12TOVIE	0	Přerušení při přetečení času převodu je zakázané
	1	Přerušení při přetečení času převodu je povolené
ENC	0	Převod je zakázaný; všechny šedivé položky je možné nastavovat
	1	Převod je povolený; šedivé položky není možné nastavovat!
ADC12SC	0	
	1	Spuštění převodu (pokud je povoleno parametrem SHSx)
CSTARTADDx	xxxx	Výběr registru ADC12MCTLx/ADC12MEMx pro první převod

Bity	Hodnota	Význam
SHSx	00	Převod je odstartován manuálně bitem ADC12SC
	01	Převod je odstartován signálem z časovače TA1
	10	Převod je odstartován signálem z časovače TB0
	11	Převod je odstartován signálem z časovače TB1
SHP	0	Vzorkování je řízeno vybraným signálem podle SHSx
	1	Vzorkování je řízeno časovačem podle SHTx
ISSH	0	Vzorkovací signál je neinvertovaný
	1	Vzorkovací signál je invertovaný
ADC12DIVx	000	Zdroj hodin je připojený přímo
	001	Zdroj hodin je dělený 2
	010	Zdroj hodin je dělený 3
	011	Zdroj hodin je dělený 4
	100	Zdroj hodin je dělený 5
	101	Zdroj hodin je dělený 6
	110	Zdroj hodin je dělený 7
	111	Zdroj hodin je dělený 8
ADC12SSELx	00	Převodník používá jako zdroj hodin vnitřní oscilátor ADC12OSC (5MHz)
	01	Převodník používá jako zdroj hodin vnitřní signál ACLK (32768Hz — to je příliš nízká frekvence!)
	10	Převodník používá jako zdroj hodin vnitřní signál SMCLK (8MHz)
	11	Převodník používá jako zdroj hodin vnitřní signál MCLK (8MHz)
CONSEQx	00	Každý spouštěcí signál provede pouze jeden převod pro nastavený vstup
	01	Každý spouštěcí signál spustí převod pro všechny nastavené vstupy
	10	Spouštěcí signál spustí opakovaný převod jednoho nastaveného vstupu
	11	Spouštěcí signál spustí opakovaný převod všech nastavených vstupů
ADC12BUSY	0	Převodník není spuštěný
	1	Převodník pracuje
EOS	0	Není poslední převáděný vstup
	1	Poslední vstup v pořadí
SREFx	000	Vr+=AVcc, Vr-=AVss
	001	Vr+=Vref+, Vr-=AVss
	010	Vr+=Veref+, Vr-=AVss
	011	Vr+=Veref+, Vr-=AVss
	100	Vr+=AVcc, Vr-=Vref-/Veref-
	101	Vr+=Vref+, Vr-=Vref-/Veref-
	110	Vr+=Veref+, Vr-=Vref-/Veref-
	111	Vr+=Veref+, Vr-=Vref-/Veref-
INCHx	0000	Kanál A0
	0001	Kanál A1
	0010	Kanál A2
	0011	Kanál A3
	0100	Kanál A4
	0101	Kanál A5

Bity	Hodnota	Význam
	0110	Kanál A6
	0111	Kanál A7
	1000	Kanál Veref+
	1001	Kanál Vref-/Veref-
	1010	Kanál je teplotní snímač
	1011 11xx	Kanál je připojený na napětí (Avcc-AVss)/2
ADC12IVx	00000	Žádné přerušení není aktivní
	00001	Přetečení při zápisu do ADC12MEMx
	00010	Přetečení času převodu
	00011	Přerušení po do ADC12MEM0
	00100	Přerušení po do ADC12MEM1
	00101	Přerušení po do ADC12MEM2
	00110	Přerušení po do ADC12MEM3
	00111	Přerušení po do ADC12MEM4
	01000	Přerušení po do ADC12MEM5
	01001	Přerušení po do ADC12MEM6
	01010	Přerušení po do ADC12MEM7
	01011	Přerušení po do ADC12MEM8
	01100	Přerušení po do ADC12MEM9
	01101	Přerušení po do ADC12MEM10
	01110	Přerušení po do ADC12MEM11
	01111	Přerušení po do ADC12MEM12
	10000	Přerušení po do ADC12MEM13
	10001	Přerušení po do ADC12MEM14
	10010	Přerušení po do ADC12MEM15

4.3.6. Asynchronní sériový port UART/SPI

Sériový port je možné použít pro asynchronní komunikaci přes rozhraní UART nebo pro komunikaci synchronní přes rozhraní SPI. Režim UART/SPI se vybírá bitem SYNC. Některé bity se používají pouze v režimu UART, jiné pouze v režimu SPI. Tyto bity jsou označené číslem [1] nebo [2]. Většina bitů je společných pro oba režimy.

Pro nastavení režimu UART je potřeba nastavit rychlost komunikace (BR1:BR0 = vybrané hodiny (SSELx) / rychlost dat) a parametry přenosu: parita (PENA, PEV), délka přenášených dat (CHAR), počet STOP bitů (SPB). Pro speciální použití je možné nastavit režim multiprocesorové komunikace (MM). Odesílaná data se zapisují do registru UxTXBUF, pokud je bit UTXIFGx nastavený. Zároveň je potřeba číst data z registru UxRXBUF, pokud je nastavený bit URXIFGx. Zároveň je potřeba kontrolovat bity FE, PE, OE a BRK pro detekci chyb přenosu.

Režimu SPI je nejprve potřeba povolit bitem SYNC. Stejně jako v režimu UART je potřeba nastavit přenosovou rychlost zápisem do registrů BR1 a BR0 podle požadované rychlosti přenosu a nastavených hodin SSELx. Dále se musí nastavit bity CHAR, MM, režim hodin pomocí bitů CKPH a CKPL a režim STC. Odesílaná data se zapisují stejně jako v režimu UART do registru UxTXBUF a čtou z registru UxRXBUF, pokud jsou nastavené příslušné bity UTXIFGx nebo URXIFGx.

Registry UART/SPI

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0000	**IE1**	0000xx00	UTXIE0	URXIE0	ACCVIE	NMIIE			OFIE	WDTIE
0001	**IE2**	xx00xxxx			UTXIE1	URXIE1				
0002	**IFG1**	10x0xx10	UTXIFG0	URXIFG0		NMIIFG			OFIFG	WDTIFG
0003	**IFG2**	xx10xxxx			UTXIFG1	URXIFG1				
0004	**ME1**	00xxxxxx	UTXE0	URXE0 USPIE0						
0005	**ME2**	xx00xxxx			UTXE1	URXE1 USPIE1				
0070	**U0CTL**	00000001	PENA	PEV	SPB	CHAR	LISTEN	SYNC	MM	SWRST
0071	**U0TCTL**	00000001		CKPL	SSELx		URXSE	TXWAKE		TXEPT
0072	**U0RCTL**	00000000	FE	PE	OE	BRK	URXEIE	URXWIE	RXWAKE	RXERR
0073	**U0MCTL**	uuuuuuuu								
0074	**U0BR0**	uuuuuuuu	Nižší byte děličky							
0075	**U0BR1**	uuuuuuuu	Vyšší byte děličky							
0076	**U0RXBUF**	uuuuuuuu	Přijímaná data							
0077	**U0TXBUF**	uuuuuuuu	Vysílaná data							
0078	**U1CTL**	00000001	PENA	PEV	SPB	CHAR	LISTEN	SYNC	MM	SWRST
0079	**U1TCTL**	00000001		CKPL	SSELx		URXSE	TXWAKE		TXEPT
007A	**U1RCTL**	00000000	FE	PE	OE	BRK	URXEIE	URXWIE	RXWAKE	RXERR
007B	**U1MCTL**	uuuuuuuu								
007C	**U1BR0**	uuuuuuuu	Nižší byte děličky							
007D	**U1BR1**	uuuuuuuu	Vyšší byte děličky							
007E	**U1RXBUF**	uuuuuuuu	Přijímaná data							
007F	**U1TXBUF**	uuuuuuuu	Vysílaná data							

V následující tabulce jsou popsané funkce bitů pro řízení UART/SPI

Bity	Hodnota	Význam
PENA[1]	0	Parita zakázána
	1	Parita povolena
PEV[1]	0	Lichá parita
	1	Sudá parita
SPB[1]	0	1 STOP bit
	1	2 STOP bity
CHAR	0	7-bitová data
	1	8-bitová data
LISTEN	0	UART funguje normálně
	1	TXD je vnitřně propojený na RXD
SYNC	0	Režim UART
	1	Režim SPI
MM[1]	0	Normální režim komunikace
	1	Multiprocesorový režim komunikace
MM[2]	0	SPI je v režimu slave
	1	SPI je v režimu master
SWRST	0	UART funguje normálně

Bity	Hodnota	Význam
	1	UART je držený ve stavu reset.
CKPH[2]	0	Signál UCLK je normální
	1	Signál UCLK je posunutý o ½ bitu
CKPL	0	Polarita hodin UCLK normální
	1	Polarita hodin UCLK invertovaná
SSELx	00	Zdroj hodin je připojený na vstup UCLKI
	01	Zdroj hodin je připojený na signál ACLK (32768Hz)
	1x	Zdroj hodin je připojený na signál SMCLK (8MHz)
URXSE[1]	0	
	1	START bit probudí procesor a spustí příjem
TXWAKE[1]	0	Přenášený byte obsahuje data
	1	Přenášený byte obsahuje adresu
STC[2]	0	Signál STE je povolený (4-drátový SPI režim)
	1	Signál STE je zakázyný (3-drátový SPI režim)
TXEPT	0	UART přenáší data
	1	UART nepřenáší data
FE	0	
	1	Chyba STOP bitu
PE[1]	0	
	1	Chyba parity
OE	0	
	1	Přetečení při zápisu do UxRXBUF
BRK[1]	0	
	1	Detekován signál BREAK
URXEIE[1]	0	Chybně přijatý znak nebude zpracován
	1	Chybně přijatý znak bude zpracován
URXWIE[1]	0	Všechny přijaté byty budou zpracovány
	1	Pouze přijatá adresa bude zrpacována
RXWAKE[1]	0	Přijatý byte obsahuje data
	1	Přijatý byte obsahuje adresu
RXERR[1]	0	
	1	Detekována chyba příjmu
UTXEx[1]	0	
	1	Povolení vysílací části obvodu USART
URXEx USPIEx	0	
	1	Povolení přijímací části obvodu USART nebo SPI obvodu
UTXIEx	0	
	1	Povolení přerušení při vyprázdnění vysílacího bufferu
URXIEx	0	
	1	Povolení přerušení při příjmu znaku
UTXIFGx	0	
	1	Příznak přerušení vysílacího bufferu; do UxTXBUF je možné zapsat další data
URXIFGx	0	

Bity	Hodnota	Význam
	1	Příznak přerušení přijímacího bufferu; z UxRXBUF je možné číst přijatá data

[1] Pouze pro režim UART
[2] Pouze pro režim SPI

5. Přílohy

5.1. Technické specifikace

Napájení	12V=; 10mA (bez připojené klávesnice a s nezapojenými piny)
Displej	Monochromatický LCD 128x64 bodů s podsvícením Řadič ST7565R
Klávesnice	PS/2
Procesor	MSP430F149; 8MHz
Paměť	Flash 60KB (32KB pro volné použití) RAM 2KB (1KB pro volné použití)
Možnosti připojení	31 uživatelsky ovládaných pinů 2x UART/SPI Časovač, PWM, 8x ADC
Digitální vstupy	Úroveň 0: (0.0–0.8)V Úroveň 1: (2.0–3.6)V
Digitální výstupy	Úroveň 0: max 0.4V při zátěži 1.5mA; max 1.2V při zátěži 6mA Úroveň 1: min 2.95V při zátěži 1.0mA; min 2.1V při zátěži 6mA Celkový proud na všech pinech nesmí v součtu překročit 50mA
Analogové vstupy	Napětí (0.0–3.3)V

5.2. Tabulka všech periferních registrů procesoru MSP430F149

Registry pro řízení periferií jsou rozděleny podle adresy na 8-bitové (adresy $0000 až $00FF) a 16-bitové (adresy $0100 až $01FF). Podle toho jsou také rozděleny do dvou následujících tabulek.

Stav registrů po resetu udávají následující značky:

0 bit je vynulovaný
1 bit je nastavený
x stav bitu je neznámý
u stav bitu je nezměněný

8-bitové registry

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0000	**IE1**	0000xx00	UTXIE0	URXIE0	ACCVIE	NMIIE			OFIE	WDTIE
0001	**IE2**	xx00xxxx			UTXIE1	URXIE1				
0002	**IFG1**	10x0xx10	UTXIFG0	URXIFG0		NMIIFG			OFIFG	WDTIFG
0003	**IFG2**	xx10xxxx			UTXIFG1	URXIFG1				
0004	**ME1**	00xxxxxx	UTXE0	URXE0 USPIE0						
0005	**ME2**	xx00xxxx			UTXE1	URXE1 USPIE1				
0018	**P3IN**									
0019	**P3OUT**	uuuuuuuu								
001A	**P3DIR**	00000000								
001B	**P3SEL**	00000000								
001C	**P4IN**									
001D	**P4OUT**	uuuuuuuu								
001E	**P4DIR**	00000000								
001F	**P4SEL**	00000000								
0020	**P1IN**									

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0021	**P1OUT**	uuuuuuuu								
0022	**P1DIR**	00000000								
0023	**P1IFG**	00000000								
0024	**P1IES**	uuuuuuuu								
0025	**P1IE**	00000000								
0026	**P1SEL**	00000000								
0028	**P2IN**									
0029	**P2OUT**	uuuuuuuu								
002A	**P2DIR**	00000000								
002B	**P2IFG**	00000000								
002C	**P2IES**	uuuuuuuu								
002D	**P2IE**	00000000								
002E	**P2SEL**	00000000								
0030	**P5IN**									
0031	**P5OUT**	uuuuuuuu								
0032	**P5DIR**	00000000								
0033	**P5SEL**	00000000								
0034	**P6IN**									
0035	**P6OUT**	uuuuuuuu								
0036	**P6DIR**	00000000								
0037	**P6SEL**	00000000								
0056	**DCOCTL**	01100000	DCOx			MODx				
0057	**BCSCTL1**	10000100	XT2OFF	XTS	DIVAx		XT5V	RSELx		
0058	**BCSCTL2**	00000000	SELMx		DIVMx		SELS	DIVSx		DCOR
0059	**CACTL1**	00000000	CAEX	CARSEL	CAREFx		CAON	CAIES	CAIE	CAIFG
005A	**CACTL2**	00000000					P2CA1	P2CA0	CAF	CAOUT
005B	**CAPD**	00000000	CAPD7	CAPD6	CAPD5	CAPD4	CAPD3	CAPD2	CAPD1	CAPD0
0070	**U0CTL**	00000001	PENA	PEV	SPB	CHAR	LISTEN	SYNC	MM	SWRST
0071	**U0TCTL**	00000001		CKPL	SSELx		URXSE	TXWAKE		TXEPT
0072	**U0RCTL**	00000000	FE	PE	OE	BRK	URXEIE	URXWIE	RXWAKE	RXERR
0073	**U0MCTL**	uuuuuuuu								
0074	**U0BR0**	uuuuuuuu	Nižší byte děličky							
0075	**U0BR1**	uuuuuuuu	Vyšší byte děličky							
0076	**U0RXBUF**	uuuuuuuu	Přijímaná data							
0077	**U0TXBUF**	uuuuuuuu	Vysílaná data							
0078	**U1CTL**	00000001	PENA	PEV	SPB	CHAR	LISTEN	SYNC	MM	SWRST
0079	**U1TCTL**	00000001		CKPL	SSELx		URXSE	TXWAKE		TXEPT
007A	**U1RCTL**	00000000	FE	PE	OE	BRK	URXEIE	URXWIE	RXWAKE	RXERR
007B	**U1MCTL**	uuuuuuuu								
007C	**U1BR0**	uuuuuuuu	Nižší byte děličky							
007D	**U1BR1**	uuuuuuuu	Vyšší byte děličky							
007E	**U1RXBUF**	uuuuuuuu	Přijímaná data							
007F	**U1TXBUF**	uuuuuuuu	Vysílaná data							
0080	**ADC12MCTL0**	00000000	EOS	SREFx			INCHx			
0081	**ADC12MCTL1**	00000000								

adresa	název	stav po resetu	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0082	**ADC12MCTL2**	00000000								
0083	**ADC12MCTL3**	00000000								
0084	**ADC12MCTL4**	00000000								
0085	**ADC12MCTL5**	00000000								
0086	**ADC12MCTL6**	00000000								
0087	**ADC12MCTL7**	00000000								
0088	**ADC12MCTL8**	00000000								
0089	**ADC12MCTL9**	00000000								
008A	**ADC12MCTL10**	00000000								
008B	**ADC12MCTL11**	00000000								
008C	**ADC12MCTL12**	00000000								
008D	**ADC12MCTL13**	00000000								
008E	**ADC12MCTL14**	00000000								
008F	**ADC12MCTL15**	00000000								

16-bitové registry

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
011E	**TBIV**	00000000	0	0	0	0	0	0	0	0
		00000000	0	0	0	0	TBIVx			0
0120	**WDTCTL**	01101001	Read: 69h; Write: 5Ah							
		00000000	WDT HOLD	WDT NMIES	WDT NMI	WDT TMSEL	WDT TCNTCL	WDT SSEL	WDTIS	
0128	**FCTL1**	10010110	Read: 96h; Write: A5h							
		00000000	BLKWRT	WRT				MERASE	ERASE	
012A	**FCTL2**	10010110	Read: 96h; Write: A5h							
		01000010	FSSEL		FNx					
012C	**FCTL3**	10010110	Read: 96h; Write: A5h							
		00011000			EMEX	LOCK	WAIT	ACCVIFG	KEYV	BUSY
012E	**TAIV**	00000000	0	0	0	0	0	0	0	0
		00000000	0	0	0	0	TTAIVx			0
0130	**MPY**	uuuuuuuu	1. operand pro neznaménkové násobení							
		uuuuuuuu								
0132	**MPYS**	uuuuuuuu	1. operand pro znaménkové násobení							
		uuuuuuuu								
0134	**MAC**	uuuuuuuu	1. operand pro neznaménkové násobení a přičtení							
		uuuuuuuu								
0136	**MACS**	uuuuuuuu	1. operand pro znaménkové násobení a přičtení							
		uuuuuuuu								
0138	**OP2**	uuuuuuuu	2. operand (po zapsání se vykoná operace vybraná zápisem 1. operandu)							
		uuuuuuuu								
013A	**RESLO**	????????	Nižší slovo výsledku							
		????????								
013C	**RESHI**	????????	Vyšší slovo výsledku							
		????????								

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
013E	**SUMEXT**	????????	Příznak znaménka (MPYS, MACS) nebo přetečení (MAC)							
		????????								
0140	**ADC12MEM0**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0142	**ADC12MEM1**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0144	**ADC12MEM2**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0146	**ADC12MEM3**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0148	**ADC12MEM4**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
014A	**ADC12MEM5**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
014C	**ADC12MEM6**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
014E	**ADC12MEM7**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0150	**ADC12MEM8**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0152	**ADC12MEM9**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0154	**ADC12MEM10**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0156	**ADC12MEM11**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0158	**ADC12MEM12**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
015A	**ADC12MEM13**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
015C	**ADC12MEM14**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
015E	**ADC12MEM15**	uuuuuuuu	0	0	0	0				
		uuuuuuuu								
0160	**TACTL**	00000000							TASSELx	
		00000000	IDx		MCx			TACLR	TAIE	TAIFG
0162	**TACCTL0**	00000000	CMx		CCISx		SCS	SCCI		CAP
		00000000	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
0164	**TACCTL1**	00000000	Viz TACCTL0							
		00000000								
0166	**TACCTL2**	00000000	Viz TACCTL0							
		00000000								
0170	**TAR**	00000000	Čítač TimerA							
		00000000								
0172	**TACCR0**	00000000	Capture/compare registr 0							

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
		00000000								
0174	**TACCR1**	00000000	Capture/compare registr 1							
		00000000								
0176	**TACCR2**	00000000	Capture/compare registr 2							
		00000000								
0180	**TBCTL**	00000000		TBCLGRPx		CNTLx			TBSSELx	
		00000000	IDx		MCx			TBCLR	TBIE	TBIFG
0182	**TBCCTL0**	00000000	CMX		CCISx		SCS	CLLDx		CAP
		00000000	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
0184	**TBCCTL1**	00000000	Viz TBCCTL0							
		00000000								
0186	**TBCCTL2**	00000000	Viz TBCCTL0							
		00000000								
0188	**TBCCTL3**	00000000	Viz TBCCTL0							
		00000000								
018A	**TBCCTL4**	00000000	Viz TBCCTL0							
		00000000								
018C	**TBCCTL5**	00000000	Viz TBCCTL0							
		00000000								
018E	**TBCCTL6**	00000000	Viz TBCCTL0							
		00000000								
0190	**TBR**	00000000	Čítač TimerB							
		00000000								
0192	**TBCCR0**	00000000	Capture/compare registr 0							
		00000000								
0194	**TBCCR1**	00000000	Capture/compare registr 1							
		00000000								
0196	**TBCCR2**	00000000	Capture/compare registr 2							
		00000000								
0198	**TBCCR3**	00000000	Capture/compare registr 3							
		00000000								
019A	**TBCCR4**	00000000	Capture/compare registr 4							
		00000000								
019C	**TBCCR5**	00000000	Capture/compare registr 5							
		00000000								
019E	**TBCCR6**	00000000	Capture/compare registr 6							
		00000000								
01A0	**ADC12CTL0**	00000000	SHT1x				SHT0x			
		00000000	MSC	REF2_5V	REFON	ADC12 ON	ADC12 OVIE	ADC12 TOVIE	ENC	ADC12 SC
01A2	**ADC12CTL1**	00000000	CSTARTADDx				SHSx		SHP	ISSH
		00000000	ADC12DIVx			ADC12SSELx		CONSEQx		ADC12 BUSY
01A4	**ADC12IFG**	00000000								
		00000000								
01A6	**ADC12IE**	00000000								

adresa	název	stav po resetu	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
			bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
		00000000								
01A8	**ADC12IV**	00000000	0	0	0	0	0	0	0	0
		00000000	0	0	ADC12IVx					0

5.3. Seznam použité literatury

1. MSP430x1xx Family User's Guide (SLAU049E)
2. MSP430F149 Datasheet (SLAS272F)
3. Butterfly Basic (http://www.rowley.co.uk/msp430/basic.htm)
4. Atomic Theory and Practice (http://members.casema.nl/hhaydn/howel/Acorn/Atom/atap/atap.htm)

5.4. Schéma zapojení