Príručka programovania v ASM a C s RTX51 od Diversant Software s príkladmi

Guide to programming in C and ASM with RTX51 from Diversant Software with examples

https://diversant.sk

Predslov

Mnoho programátorov sa skôr či neskôr dostane pred problém, ktorý už nemožno elegantným spôsobom riešiť v programovacom jazyku za pomoci cyklického „pooling-u“. Je potrebné vykonávať množstvo úloh súčasne a zároveň je potrebné definovať presné reakčné časy. Na tento problém už dnes existuje riešenie. Je to operačný systém RTX51 Tiny, alebo RTX51 Full. Je vhodný pre embedded systémy a všade tam, kde je potrebné minimalizovať dĺžku vývoja riadiaceho softwaru.

Ing. Eduard Jadroň

Autor

Autor

Obsah

1 Tvorba programov v jazyku assembler A51

1.1. Príkazy assembleru A51, direktívy, riadiace príkazy, inštrukcie

Program napísaný v assembleri A51 je tvorený príkazmi, direktívami, riadiacimi príkazmi a inštrukciami. Každý riadok programu môže obsahovať len jeden príkaz, direktívu, inštrukciu. Príkazy napísané v A51 nerozlišuje veľké, alebo malé písmená a môžu začínať na ľubovoľnom riadku a stĺpci. Každý program musí na svojom konci obsahovať direktívu ***END***.

Direktívy sú interné príkazy assembleru ktoré nevykonávajú žiadnu činnosť, ale ovplyvňujú len samotný preklad programu. Používajú sa na definíciu konštánt, rezerváciu pamäťového priestoru a pre premenné v programe.

Riadiace príkazy ovplyvňujú preklad programu na základe operátorov, čo umožňuje jednoduché vytvorenie programu ktorý je konfigurovateľný pre rôzne varianty aplikácií.

Inštrukcie sú elementárne príkazy ktoré sú vykonávané CPU. Inštrukcia pozostáva z mnemotechnickej skratky inštrukcie a operandu. Inštrukcia môže mať aj viac operandov. Inštrukcie sú po preklade programu uložené do tzv. „***object***“ súboru, ktorý sa neskôr používa na vytvorenie finálneho programu vo formáte INTEL HEX[1] pre naprogramovanie do pamäti programu CPU.

1.2. Použitie komentárov a symbolov

Komentár je textový reťazec obsahujúci informácie o programe, autorovi a iné informácie. Komentár je nepovinný. Za komentár je považované všetko, čo sa nachádza za znakom „**;**“.

Symbol je názov pozostávajúci z kombinácie znakov a číslic ktorý môže reprezentovať číselnú hodnotu, textový blok, alebo meno registra. Symbol nesmie začínať číslom! Vždy musí byť pre definíciu názvu symbolu ako prvé použité písmeno „**A**“-„**Z**“, alebo „**a**“-„**z**“, prípadne znak „**_**“, alebo „**?**“.

[1] Výsledný súbor s príponou HEX obsahuje program ktorý je možné naprogramovať do mikroprocesora. Jeho štruktúra je odlišná od formátu BIN, pretože obsahuje navyše aj CRC zabezpečovacie kódy pre odstránenie možných chýb v súbore.

1.3. Návestia, operandy, výrazy a operátory

Návestie je symbolická adresa skoku. V programe sa používa ako náhrada za číselné vyjadrenie adresy. Návestie musí mať unikátny názov ukončený znakom **„:“** a v programe sa už nesmie nachádzať premenná, alebo návestie s týmto menom. Návestie poskytuje prekladaču a debuggeru dôležité informácie o umiestnení jednotlivých častí programu a dát v pamäti.

Výrazy sú vzájomné kombinácie čísel, znakov, symbolov a operátorov poskytujúcich vypočítané hodnoty ktoré sú používané pri preklade programu. Výsledok výrazu môže byť v prípade A51.EXE maximálne 16 bitová hodnota.

Operandy sú argumenty, alebo výrazy ktoré sú špecifikované assemblerovskými direktívami, alebo inštrukciami. Assemblerovská direktíva vyžaduje operand ktorý je tvorený konštantou, alebo symbolom.

Operátory sú použité na kombinovanie a porovnávanie operandov vo vnútri assemblerovského programu. Nie sú viazané inštrukciami assembleru a teda ich použitím sa nevytvára žiaden programový kód. Operátory poskytujú len základné výpočty a ich hodnoty sú následne odovzdávané prekladanému programu.

1.4. Deklarácia premenných a konštánt v A51

Adresovanie pamäti dát internej a externej pamäti RAM mikroprocesora je vykonávané dvomi spôsobmi:

- Priamym adresovaním, (DATA, BDATA)
- Nepriamym adresovaním, (IDATA, PDATA, XDATA)

Priame adresovanie používajúce pre prístup do pamäti dát pamäťovú triedu ***DATA***, a ***BIT***. V prvom prípade deklarujeme pamäťové premenné s minimálnou dĺžkou 1 Byte. Druhý prípad umožňuje deklarovať premenné s veľkosťou jedného bitu. Priamym adresovaním je možné v prípade 8051 adresovať pamäťový priestor od adresy 00H do 80H. Bitovo adresovateľný priestor sa nachádza v pamäťovom priestore od adresy 20H do 30H.

mov a,10h *;obsah pamäťovej bunky na adrese 10h skopíruje do registra A*

Nepriame adresovanie používajúce pre prístup do pamäti niektorý indexový register ktorý umožňuje prístup do celého adresného priestoru mikroprocesora 8051. Nepriame adresovanie používa register R0, R1 a DPTR. Pamäťová trieda je v tomto prípade ***IDATA***, ***XDATA*** a ***CODE*** pre pamäť programu. Pre adresovanie externého pamäťového priestoru dát ***XDATA*** a programu ***CODE*** je použitá 16 bitová adresa uložená v registri DPTR. Niektoré deriváty napr. AT89C51RD2 využívajú pre prístup väčšieho počtu registrov DPTR, čo v konečnom dôsledku urýchľuje vykonávanie programu.

mov r0,#10h *;register R0 bude ukazovať nepriamo na adresu 10h*

mov a,@r0 *;obsah pamäťovej bunky na ktorú ukazuje svojim obsahom R0 zapíše do A*

Obrázok 1, Naplnenie súvislého bloku pamäti pomocou nepriameho adresovania

```
29  ;------------------------------------------------------------------
30  Prg1      segment code
31            rseg Prg1
32  ;------------------------------------------------------------------
33  Start:    mov r0,#7fh     ;Pocet byte mazanej pamati v RAM ...
34            clr a           ;Tymto prepisem pamat ...
35  Mazem:    mov @r0,a       ;Teraz vymazem konkretny byte ....
36            djnz r0,Mazem   ;Opakujem skok na Mazem pokial R0<>0
37            nop
38            nop
39            nop
40  ;------------------------------------------------------------------
```

V tomto prípade obsahuje register R0 hodnotu, t.j. počet bajtov ktoré sa použijú v cykle na naplnenie pamäti RAM. Inštrukcia **mov @r0,a** v cykle inicializuje konkrétne oblasti pamäte. Pre prístup do pamäti programu (CODE) sa používa inštrukcia **movc a,@a+dptr** kde sa z adresy pamäťovej bunky vzniknutej súčtom obsahov registrov (A+DPTR) načíta byte a uloží do A. Register DPTR[2] sa používa ako univerzálny ukazovateľ dát v pamäti programu a dát.

Obrázok 2, Príklad správneho zápisu programu v assembleri

```
1   NAME  CHAR_IO
2
3   PUBLIC  PUTCHAR
4
5   CHAR_ROUTINES SEGMENT  CODE       ;definicia programoveho segmentu
6                 RSEG  CHAR_ROUTINES ;segment oznacim ako relokovatelny v pamati
7
8   PUTCHAR:      JNB  TI,$           ;kontrolujem bit TI t.j. odoslanie znaku
9                 CLR  TI             ;znak je uz odoslany, nastavim TI=0
10                MOV  SBUF,A         ;vysielany znak ulozim do vysielacieho registra SBUF
11                RET
12
13  VAR2          SEGMENT  DATA       ;definicia datoveho segmentu
14                RSEG  VAR2          ;segment oznacim ako relokovatelny v pamati
15  DUMMY:        DS  40H             ;vytvorim 64 byte
16
17                END
18
```

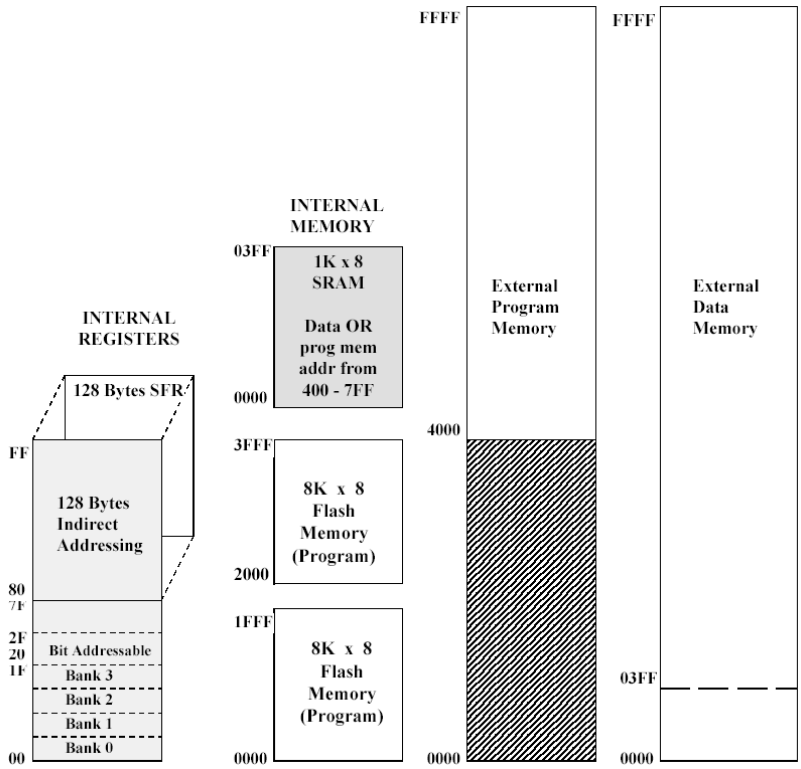
[2] Moderné procesory už majú viacero samostatných registrov DPTR, čo umožňuje rýchlejší prístup k zapísaným údajom v pamätiach procesora.

Vyššie uvedený program obsahuje niekoľko definovaných segmentov. Jeden je programový segment s názvom CHAR_ROUTINES a druhý je dátový segment s názvom VAR2 o veľkosti 40h t.j. 64 Byte. Definované bloky sa pri preklade programu rozmiestnia a optimalizujú v pamäti procesora.

1.5. Pamäťový podsystém mikroprocesora 8051

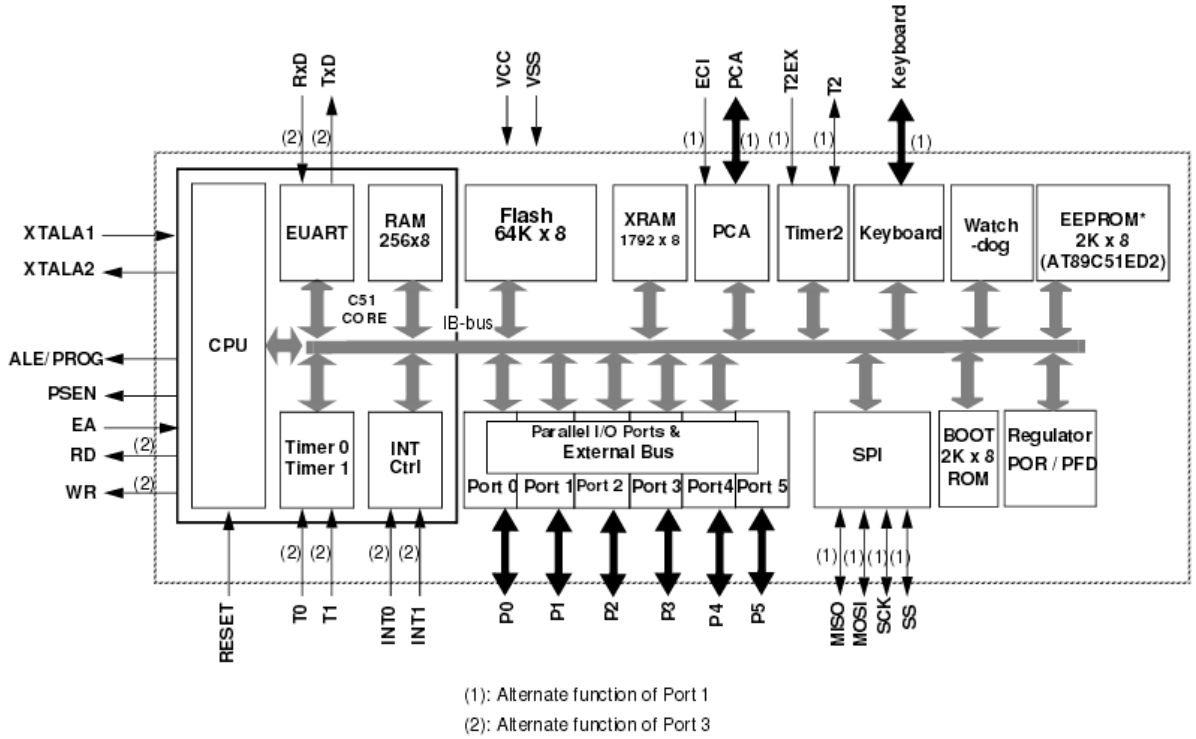
Mikroprocesor rady 8051 obsahuje niekoľko pamäťových priestorov[3], ktoré je možné využívať programátorom. Obrázok 3 zobrazuje fyzické umiestnenie pamäti v 8051.

Obrázok 3, Pamäťový model a mikroprocesora AT89C51RD2 kompatibilného s 8051



[3] V súčasnosti sa výrobcovia snažia vyhovieť požiadavkám trhu a do svojich mikroprocesorov integrujú veľké množstvo predtým externých súčiastok do spoločného kremíkového čipu. Napr. A/D, alebo D/A prevodníky, I^2C, RS485, PWM, PLA ....

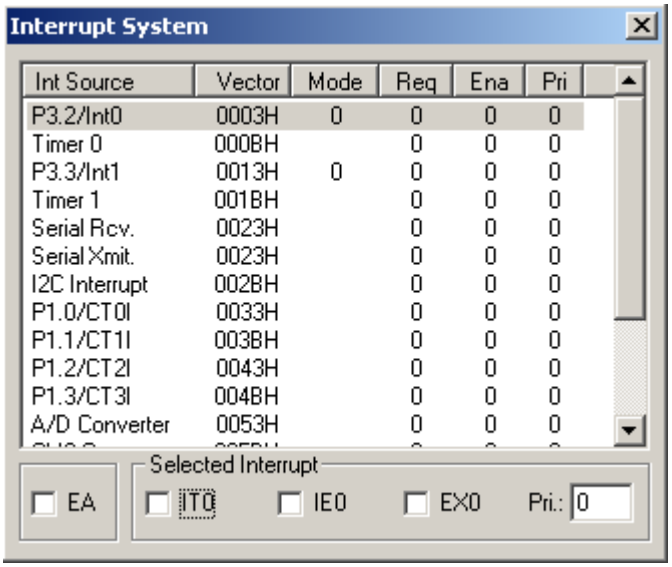
Obrázok 4, Bloková schéma mikroprocesora AT89C51ED2



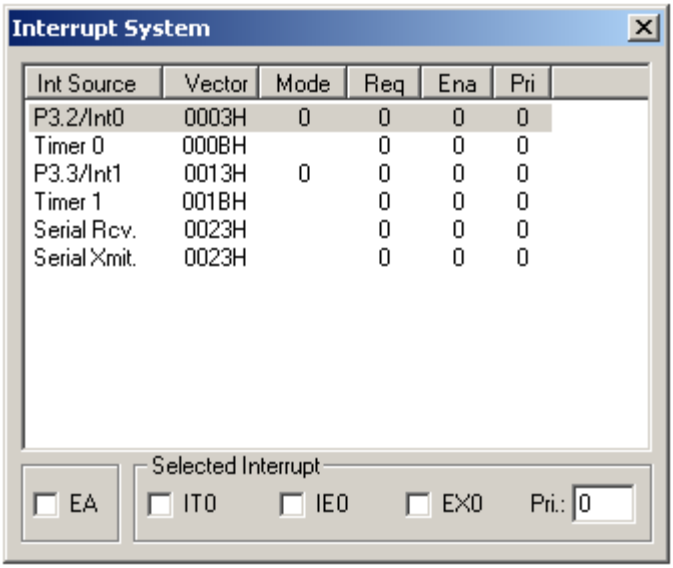
1.6. Prerušovací podsystém mikroprocesora 8051

Architektúra 8051 umiestňuje svoje vektory prerušenia na absolútne adresy v pamäti programu od adresy 0003H. Vektor prerušenia je v podstate adresa podprogramu, na ktorú mikroprocesor vykoná skok ak je prijaté platné prerušenie. Vykonávaná časť programu sa prijatím prerušenia dočasne pozastaví, na vrchol zásobníka sa uloží návratová adresa nasledujúcej inštrukcie a vykoná sa skok na adresu obsluhy prerušenia t.j. vektor prerušenia. Návrat z prerušenia je vykonaný inštrukciou ***RETI***. Táto inštrukcia z vrcholu zásobníka vyzdvihne 2 byte, ktoré použije ako cieľovú adresu skoku, ktorou sa vykoná návrat do programu odkiaľ bol „násilne“ prerušený a pokračuje vo vykonávaní programu.

Obrázok 5, Prerušovacia jednotka mikroprocesora 80C552 v µVision4



Obrázok 6, Prerušovacia jednotka mikroprocesora AT89C2051 v µVision4



Jednotlivé ladiace okná a ich obsahy sa značne odlišujú od typu použitých procesorov. Veľká variabilita umožňuje programátorovi vyvíjať aplikáciu doslova podľa vopred stanovených požiadaviek.

Obrázok 7, Vzorový príklad programu v macro assembleri[4]

C:\Omega\Data\Dropbox\Záloha\C51\i2c_dsw\ADC_Converter.asm

```
1   ;------------------------------------------------------------------------------------
2   Repeat              macro   Num
3                           anl a,#Num
4                           rl a
5                           rl a
6                       endm
7   ;------------------------------------------------------------------------------------
8   public              _ADC_CONVERSION         ;ADC_Conversion(unsigned char Channel)
9   public              CONVERSION              ;Conversion(unsigned char Channel)
10  ;------------------------------------------------------------------------------------
11  ADCON               data 0C5h
12  ADCH                data 0C6h
13  ;------------------------------------------------------------------------------------
14  ADCS                bit acc.3               ;Spustim prevod AD prevodnika
15  ADCI                bit acc.4               ;Kontrolujem ci je prevod ukonceny ADCON.4...
16  MASKA               equ 11000000B
17  ;------------------------------------------------------------------------------------
18  Prog_ADC_Converter  segment code
19                      rseg Prog_ADC_Converter
20  ;------------------------------------------------------------------------------------
21  _ADC_CONVERSION:    mov a,r7                ;Vstupny parameter je cislo kanala pre meranie v R7
22                      setb ADCS               ;Nastavim "ADC prevod"
23                      xch a,ADCON             ;Obsah A potrebujem do ADCON aby sa AD prevod spustil
24  ?PR?ADC_END:        mov a,ADCON             ;ADCI signalizuje ukonceny prevod a pritomnost
    vysledku
25                      jnb ADCI,?PR?ADC_END    ;Je prevod ukonceny ???
26                      clr ADCI                ;bit ADCI nastavujem na 0
27                      mov ADCON,a
28                      jmp NO_INTERRUPT
29  CONVERSION:         mov a,ADCON
30  NO_INTERRUPT:       Repeat MASKA            ;LSB vysledku je v D7,D6 v ADCON[6..7]
31                      mov r7,a                ;LSB AD prevodu 2 bitov R7=(ADCON[6..7]<<2) ...
32                      mov a,ADCH
33                      Repeat ~MASKA           ;Ziskavam ADCH[0..5]<<2 & (~MASKA) a scitam ich s R7
34                      orl a,r7                ;Scitavam ADCH[2..7] | R7
35                      xch a,r7                ;vysledok ADCH[2..7] | (ADCON[6..7]<<2)
36                      mov a,ADCH
37                      Repeat MASKA            ;Ziskavam ADCH[6..7]&MASKA
38                      mov r6,a                ;Vysledok je MSB AD prevodu
39                      ret
40  ;------------------------------------------------------------------------------------
41  ;                   cseg at 0053h           ;os_attach_interrupt(10)
42  ;                   push acc
43  ;                   pop acc
44  ;                   reti
45  ;------------------------------------------------------------------------------------
46                      end
```

[4] **Macro** je direktíva prekladača umožňujúca opakujúce sa časti programového kódu opätovne použiť k zjednodušeniu a sprehľadneniu programu napísaného v assembleri. Samozrejme „macro“ a jeho použitie má význam len vtedy ak veľkosť napísaného programu obsahuje množstvo riadkov programového kódu.

Tabuľka 1, Zoznam niektorých vektorov prerušení[5]

Číslo prerušenia	Adresa	Symbol	Zdroj prerušenia
0	0003h	$\overline{INT0}$	Externé prerušenie 0
1	000Bh	**T0**	Prerušenie od časovača T0
2	0013h	$\overline{INT1}$	Externé prerušenie 1
3	001Bh	**T1**	Prerušenie od časovača T1
4	0023h	**ES**	Prerušenie od sériovej linky RS232
5	002Bh	**T2**	Prerušenie od časovača T2
6	0033h	**PCA**	Prerušenie od PCA
10	0053h	**ADC**	Prerušenie od 10-bit ADC prevodníka

Tabuľka 2, Prehľad SFR registrov mikroprocesora AT89C51RD2 kompatibilného s 8051

	Bit Addressable	Non Bit addressable							
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
F8h		CH 0000 0000	CCAP0H XXXX XXXX	CCAP1H XXXX XXXX	CCAP2H XXXX XXXX	CCAP3H XXXX XXXX	CCAP4H XXXX XXXX		FFh
F0h	B 0000 0000								F7h
E8h	P5 1111 1111	CL 0000 0000	CCAP0L XXXX XXXX	CCAP1L XXXX XXXX	CCAP2L XXXX XXXX	CCAP3L XXXX XXXX	CCAP4L XXXX XXXX		EFh
E0h	ACC 0000 0000								E7h
D8h	CCON 00X0 0000	CMOD 00XX X000	CCAPM0 X000 0000	CCAPM1 X000 0000	CCAPM2 X000 0000	CCAPM3 X000 0000	CCAPM4 X000 0000		DFh
D0h	PSW 0000 0000	FCON XXXX 0000	EECON XXXX XX00	EETIM 0000 0000					D7h
C8h	T2CON 0000 0000	T2MOD XXXX XX00	RCAP2L 0000 0000	RCAP2H 0000 0000	TL2 0000 0000	TH2 0000 0000			CFh
C0h	P4 1111 1111							P5 1111 1111	C7h
B8h	IP X000 000	SADEN 0000 0000							BFh
B0h	P3 1111 1111							IPH X000 0000	B7h
A8h	IE 0000 0000	SADDR 0000 0000							AFh
A0h	P2 1111 1111		AUXR1 XXXX 00X0				WDTRST XXXX XXXX	WDTPRG XXXX X000	A7h
98h	SCON 0000 0000	SBUF XXXX XXXX							9Fh
90h	P1 1111 1111								97h
88h	TCON 0000 0000	TMOD 0000 0000	TL0 0000 0000	TL1 0000 0000	TH0 0000 0000	TH1 0000 0000	AUXR XX0X 1000	CKCON X000 0000	8Fh
80h	P0 1111 1111	SP 0000 0111	DPL 0000 0000	DPH 0000 0000				PCON 00X1 0000	87h
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

reserved

[5] Počet vektorov prerušenia a ich umiestnenie v pamäti programu závisí od typu mikroprocesora. Vektory prerušenia (adresy vektorov prerušenia) niektorých mikroprocesorov a napríklad mikroprocesora Dallas Semiconductor DS87C550 nie sú podľa katalógových listov rovnaké, a v prípade použitia náhrady procesora 80C552 priamo DS87C550 je potrebné „presmerovať“ niektoré vektory prerušení oproti používanému štandardu u mikroprocesora 8051.

1.7. Deklarácia relokovateľných segmentov v pamäti dát a programu

Program napísaný v ľubovoľnom jazyku je zložený z veľkého množstva samostatných procedúr a funkcií ktoré sú rozmiestnené na rôznych miestach v pamäti programu. Pri preklade programu ktorý sa skladá z väčšieho počtu relokovateľných menších častí je výsledný súbor lepšie optimalizovateľný v rámci pamäte programu a údajov mikroprocesora čo v konečnom dôsledku má priaznivý vplyv na rýchlosť vykonávania programu a jeho stabilitu.

Obrázok 8, Príklad zápisu programu v assembleri A51

```
1   LED          BIT  P1.7          ;Deklarácia premennej LED s veľkosťou 1 bit
2
3                CSEG AT   0000H    ;Preklad programu od adresy 0000H
4                LJMP START         ;Skok na návestie START
5
6                CSEG AT   001BH    ;Preklad programu od adresy 001BH
7                JMP  INT_T1        ;Skok na INT_T1
8
9                CSEG AT   0100H    ;Preklad programu od adresy 0100H
10  START :      MOV  R0,#07FH      ;Nepriama adresa do R0
11               CLR  A             ;Register A naplním číslom 00h
12  CYKL :       MOV  @R0,A         ;Obsah A zapíšem na adresu uloženú v R0
13               DJNZ R0,CYKL       ;Dekrementujem R0 a pokiaľ R0<>0 skočím na CYKL
14               MOV  SP,#50H       ;Nastavím zásobník na adresu 50H v RAM
15               LCALL INIT
16  LOOP :       JMP  LOOP          ;Vytvorím nekonečný cyklus
17
18  PRG0         SEGMENT CODE       ;Vytváram relokovateľný segment
19               RSEG PRG0          ;Relokovateľný - premiestniteľný
20  INT_T1 :     INC  COUNTER       ;COUNTER=COUNTER+1
21               CPL  FLAG          ;FLAG=/FLAG
22               CPL  LED
23               RETI
24
25  PRG1         SEGMENT CODE       ;Vytváram relokovateľný segment
26               RSEG PRG1          ;Relokovateľný - premiestniteľný
27  INIT :       MOV  TMOD,#20H
28               MOV  A,#9CH
29               MOV  TH1,A
30               SETB EA
31               SETB ET1
32               SETB PT1
33               SETB TR1
34               RET
35
36  VAR          SEGMENT DATA       ;Deklarácia segmentu dát v pamäti RAM
37               RSEG VAR
38  COUNTER :    DS   1             ;Deklarácia pamäťovej premennej s veľkosťou 1 Byte
39
40  VARBIT       SEGMENT DATA       ;Deklarácia segmentu dát v pamäti RAM
41               RSEG VARBIT
42  FLAG :       DBIT 1             ; Deklarácia pamäťovej premennej s veľkosťou 1 bit
43               END
44
```

Časti programu deklarované ako „SEGMENT CODE“ sú programové segmenty[6] a „SEGMENT DATA“ sú deklarácie dátového segmentu t.j. deklarácia pamäťovej premennej *COUNTER* údajového typu *byte* a *bitovej* premennej *FLAG*. Použitá direktíva *RSEG* slúži na deklaráciu relokovateľného[7] segmentu programu, alebo dát.

[6] Časti programového kódu, alebo pamäti údajov

[7] „Relokovateľnosť“ (Relocate segment) je možné voľne preložiť ako schopnosť premiestniť program, alebo dát v rámci pamäte bez použitia absolútnej direktívy prekladu od adresy ***ORG***, alebo ***CSEG AT***.

1.8. Spájanie programov napísaných v A51 a ich preklad

Programy napísané v A51 sa skladajú zo segmentov. Segment je časť programu ktorá môže byť preložená absolútnej adrese, alebo relatívnej. Prvý spôsob sa používa ak je potrebné preložiť program od absolútnej adresy, t.j. obsluha vektorov prerušenia. Druhý je zaujímavejší tým, že neobsahuje adresu na ktorú má byť umiestnený. Hovoríme, že je relokovateľný t.j. premiestniteľný v rozsahu pamäťového programu, alebo dát v priestore mikroprocesora. Linker L51.EXE je zodpovedný za spojenie všetkých segmentov do jedného výsledného, ktorý sa neskôr použije pre naprogramovanie do mikroprocesora. Použite relokovateľných segmentov prináša nesporné výhody, napriek zdanlivej komplikovanosti. Keďže program pozostáva z niekoľkých menších celkov tzv. segmentov, linker je schopný omnoho lepšie využiť pamäťový priestor pamäti programu, alebo dát a jednotlivé časti segmentu rozmiestni tak, aby zaberali menej pamäti a aby adresy podprogramov a údajových štruktúr sa nachádzali na adrese bez zvyšku deliteľnej dvomi čo zabezpečí rýchlejší prístup mikroprocesora k nim.

Ak potrebujeme spojiť[8] niekoľko samostatných programov do jedného kompaktného tvaru, tak musíme každý program v assembleri preložiť pomocou programu A51.EXE. Po preklade je vytvorený tzv. „object“ súbor s príponou OBJ. Tento súbor je možné spojiť pomocou špeciálneho programu BL51.EXE nazývaného linker, ktorý “zlinkuje” všetky súbory do jedného celku. Použitím OH51.EXE je možné preložiť program do spustiteľnej podoby vo formáte INTEL HEX, ktorý sa naprogramuje do mikroprocesora 8051.

Preklad programu:

```
REM This Batch-File generates a Sample-Program for the A51-Assembler
a51.exe  asample1.a51 debug xref
a51.exe  asample2.a51 debug xref
a51.exe  asample3.a51 debug xref
bl51.exe asample1.obj,asample2.obj,asample3.obj to asample precede (var1) ixref
oh51.exe asample
```

[8] Rovnako spájame aj programy napísané v C, prípadne ich vzájomnú kombináciu napísanú v A51 a C. Vtedy je nevyhnutné použiť direktívu ***extern*** umožňujúcu sprístupniť jednotlivé procedúry, funkcie a premenné napísané v ľubovoľných jazykoch. Napr. v jazyku C sprístupnenie premennej deklarovanej v A51 bude: ***extern unsigned char premenna;***. V A51 je nutné vyššie uvedenú premennú uviesť v hlavičke programu ako ***public premenna*** aby bola viditeľná z jazyka A51 v C.

Obrázok 9, Zápis programu v assembleri a jeho pripojenie k jazyku C

```
;------------------------------------------------
public        _Delay            ;Volanie funkcie s parametrom
;------------------------------------------------
Prog2         segment  code     ;Vytvaram cakaciu slucku cca. 300us pre zapis do radica HD pri 0x80
              rseg  Prog2
;------------------------------------------------
_Delay:       djnz  r7,_Delay   ;Cas do registra R7
              ret               ;Vratim sa z rutiny podprogramu
;------------------------------------------------
              end

```

Obrázok 10, Pripojenie programu v assembleri do jazyka C

```
#include  <reg51.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <dsw.h>

#include  <adresy.inc>

extern  Delay(unsigned  char);              //Rutinka pre oneskorenie ...

void  Init_Serial(void)
{
 TMOD|=0x22;        //Nastavim rezim casovaca
 SCON|=0x50;        //Nastavim rezim serioveho rozhrania
 PCON|=0x80;        //Nastavim rezim dvojnasobnej rychlosti
 TH1=TL1=0xFB;      //Nastavim rezim s danou prenosovou rychlostou 19200bps
 TR1=1;             //Spustim casovac ...
 Delay(50);         //Chvilku pockam pred inicializaciou ...
}
```

Obrázok 10 názorne ukazuje pripojenie procedúry[9] Delay() ktorá je napísaná v assembleri a je prilinkovaná ako externá procedúra pomocou *extern Delay(unsigned char)*. V zátvorkách je očakávaný vstupný parameter ktorý následne vstupuje do funkcie ako číslo. Niekedy sa stretávame s tzv. parametrom *(void)*, čo v skutočnosti znamená, že do procedúry, alebo funkcie nevstupuje žiadny parameter. Vo vyšších programovacích jazykoch ako je napr. C je vhodné upozorniť prekladač na túto skutočnosť preto, aby sme lepšie využili jeho optimalizačné schopnosti a zbavili sa zbytočných chybových hlásení.

[9] Jazyk C nepozná procedúry, len funkcie. Ak nepotrebujeme návratovú hodnotu funkcie mali by sme použiť **void**, ktorý prekladaču povie, že návratová hodnota funkcie sa nepoužije.

1.9. Zápis inštrukcií assembleru 8051 v jazyku C

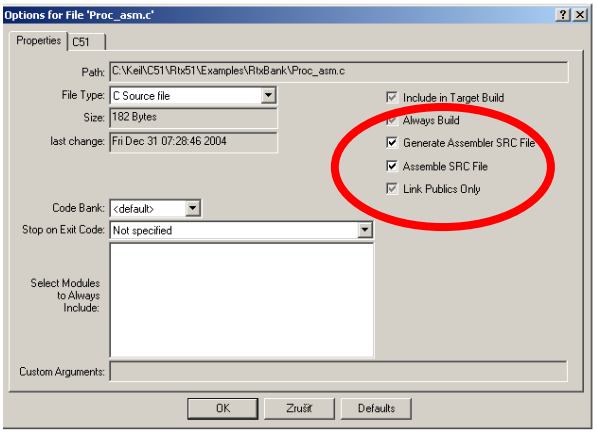
Použitie vyšších programovacích jazykov pri tvorbe aplikácii má svoje nesporné výhody. Aj napriek týmto výhodám potrebujeme niekedy vytvoriť vysoko efektívny, krátky a rýchly program pomocou inštrukcií assembleru mikroprocesora 8051.

Na tieto účely sa používa direktíva prekladača C tzv. **#pragma asm**[10] a **#pragma endasm**, ktorá umožňuje zápis assembleru do jazyka C. Jediné obmedzenie tohto spôsobu programovania je, že takýto program musí byť zapísaný v samostatnej časti projektu ako samostatný súbor s nastavenými voľbami v okne Options.

Obrázok 11, Príklad zápisu programu v assembleri

```
1   void Delay (unsigned char Dlzka )    //Zapis procedury v jazyku C
2   {
3    while ((-- Dlzka ) != 0x00 );
4   }
5
6   void Proc_asm (void )                //Zapis procedury v assembleri 8051
7   {
8   #pragma asm
9     PUSH DPH
10    PUSH DPL
11    INC DPTR
12    MOVC A,@A+DPTR
13    MOV R7,#50h
14    DJNZ R7,$
15    POP DPL
16    POP DPH
17  #pragma endasm
18  }
19
```

Pred samotným prekladom programu je potrebné v okne Options µVision3 zvoliť tieto voľby: Generate Assembler SRC File a Assemble SRC File.



[10] **#pragma asm** nie je použiteľná v demoverziách.

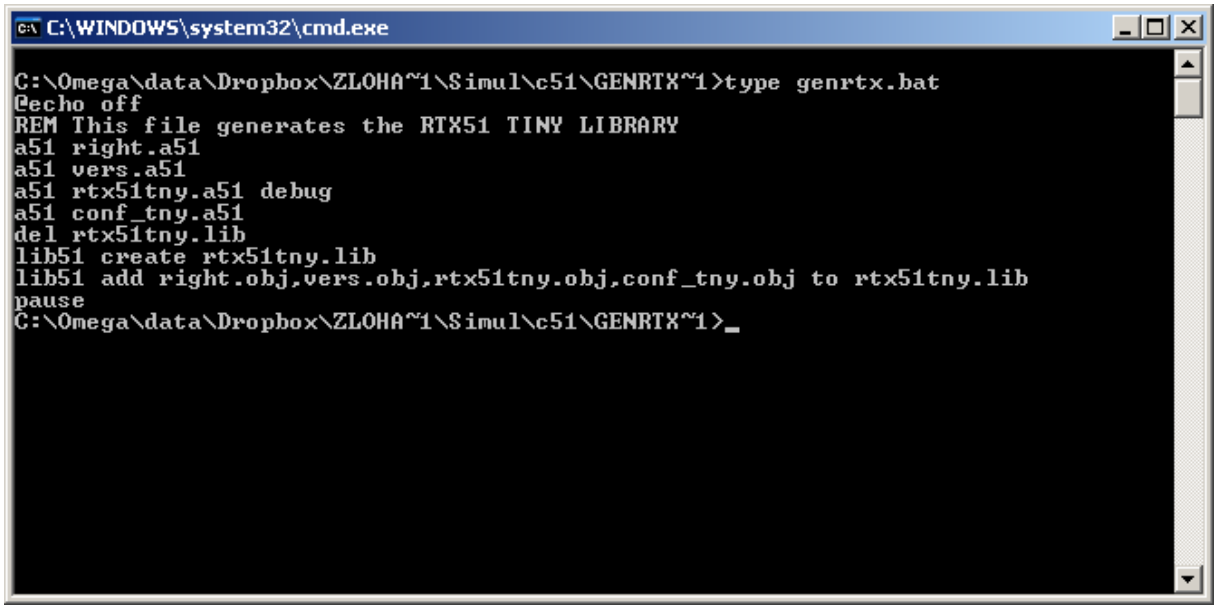
Výpis súboru PROC_ASM.C po preklade kompilátorom C

```
; .\Objects\Proc_asm.SRC generated from: Proc_asm.c
; COMPILER INVOKED BY:
;        C:\Keil\C51\BIN\C51.EXE Proc_asm.c LARGE OMF2 OPTIMIZE(9,SPEED) BROWSE DEBUG
PRINT(.\Listings\Proc_asm.lst) SRC(.\Objects\Proc_asm.SRC)


NAME    PROC_ASM

?PR?Proc_asm?PROC_ASM                SEGMENT CODE
            PUBLIC  Proc_asm
; void Proc_asm(void)                                   //Zapis procedury v assembleri 8051

            RSEG  ?PR?Proc_asm?PROC_ASM
Proc_asm:
            USING   0
                                    ; SOURCE LINE # 1
; {
                                    ; SOURCE LINE # 2
; #pragma asm
; PUSH DPH
              PUSH DPH
; PUSH DPL
              PUSH DPL
; INC DPTR
              INC DPTR
; MOVC A,@A+DPTR
              MOVC A,@A+DPTR
; MOV R7,#50h
              MOV R7,#50h
; DJNZ R7,$
              DJNZ R7,$
; POP DPL
              POP DPL
; POP DPH
              POP DPH
; #pragma endasm
; }
                                    ; SOURCE LINE # 13
            RET
; END OF Proc_asm

            END
```

1.10. Preklad programovej knižnice rtx51tny.lib z príkazového riadku

Prekladač C51 do verzie 3.40[11] umožňoval len preklad programu z prostredia príkazového riadku (Command Line) operačného systému MSDOS verzie 6.22, alebo Windows XP. Výsledkom prekladu je súbor **rtx51tny.lib** ktorý je potrebný pre začlenenie do existujúceho projektu C51 aby bolo možné používať operačný systém RTX51 Tiny[12].

Obrázok 12, Preklad programovej knižnice RTX51 Tiny verzie 1.01



Obrázok 13, Preklad programovej knižnice RTX51 Tiny verzie 2.02

```
@echo off
a51.exe version.a51
a51.exe os_clear_signal.a51
a51.exe os_create.a51
a51.exe os_delete.a51
a51.exe os_reset_interval.a51
a51.exe os_running_task.a51
a51.exe os_send_signal.a51
a51.exe os_set_ready.a51
a51.exe copyright.a51
a51.exe conf_tny.a51
lib51.exe create rtx51tny.lib
lib51.exe add os_clear_signal.obj, os_create.obj, os_delete.obj, os_reset_interval.obj,
os_running_task.obj, os_send_signal.obj, os_set_ready.obj, copyright.obj, conf_tny.obj to rtx51tny.lib
```

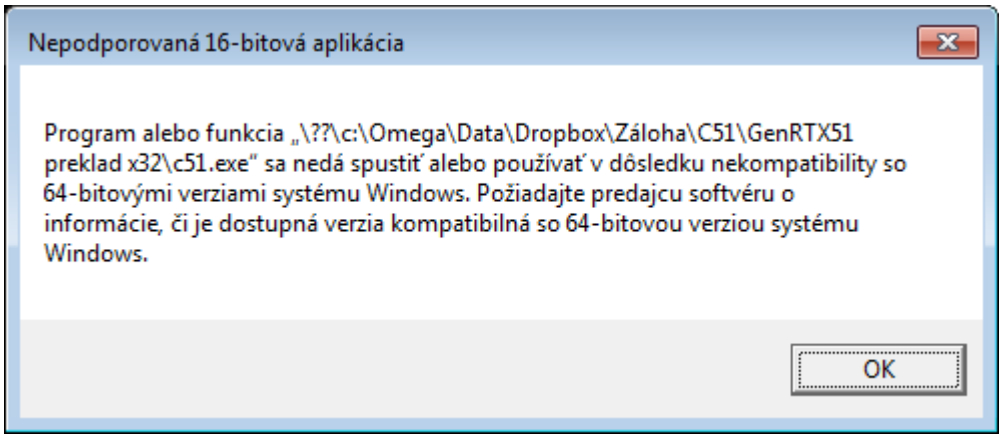
[11] Prekladač jazyka C51 verzie 3.40 vyžadoval pre správnu funkciu prítomnosť tzv. dongle (hardwareového kľúča) ako ochranu proti nelegálnemu kopírovaniu. Ak bol tento **„dongle“** na porte LPT1, LPT2 pri zapojenej tlačiarni, dochádzalo pravidelne vždy pri kompilácii programu napísaného ho v C k „RESET-u“ tlačiarne, alebo nesprávnej funkcii zariadení ktoré boli pripojené na paralelnom LPT porte. Program C51.EXE cez LPT1, alebo LPT2 port zisťoval prítomnosť hardwareového kľúča na porte.

[12] RTX51 Tiny 2.02 je oproti verzii 1.01 podstatne vylepšený čím sa dosiahlo vysokej stability operačného systému. Od verzie 9.xx sa RTX51 Tiny dodáva v rámci PK51 zadarmo ako súčasť programového vybavenia.

1.11. Preklad programov v 64-bitových systémoch z príkazového riadku

V niektorých prípadoch je potrebné preložiť program napísaný v jazyku C bez vývojového prostredia µVision priamo v konzole operačného systému Windows.[13] Problémy spôsobuje hlavne prostredie 64-bitových[14] operačných systémov, ktoré neumožňuje spustenie prekladača jazyka C pomocou spustiteľných súborov **c51.exe**, **l51.exe**, alebo **oh51.exe** verzie 3.40 určenej ešte pre 16-bitový operačný systém MS-DOS. Preto je potrebné z adresára **c:\keil\c51\bin** legálnej distribúcie KEIL µVision[15] skopírovať do zvoleného adresára nasledujúce súbory: a51.exe, ax51.exe, cx51.exe, lx51.exe, ohx51.exe, rtx51tny.lib, c51fps.lib, c51fpc.lib, c51fpl.lib, reg51.h, stdio.h, stdlib.h, rtx51tny.h, l51.dll, tools.ini.[16]

Obrázok 14, Výpis chybového hlásenia o nepodporovanej 16-bitovej aplikácii



Prvá možnosť ako je možné preložiť program v 64-bitovom systéme je nainštalovanie virtuálneho počítača v prostredí 32, alebo 64 bitového operačného systému ako je napr. VirtualBox, alebo VMware[17], v ktorých je už priamo možné nainštalovať 16-bitové prostredie MS-DOS a preložiť program. Druhá možnosť už bola načrtnutá a spočíva v skopírovaní niektorých súborov ktoré sú obsiahnuté v legálnej inštalácii µVision. Dávkovým súborom potom môžeme preložiť program a pomocou ohx51.exe exportovať do „hex“ súboru ktorá je pomocou špeciálneho programátora, alebo pomocou ISP naprogramovaná do mikroprocesora.

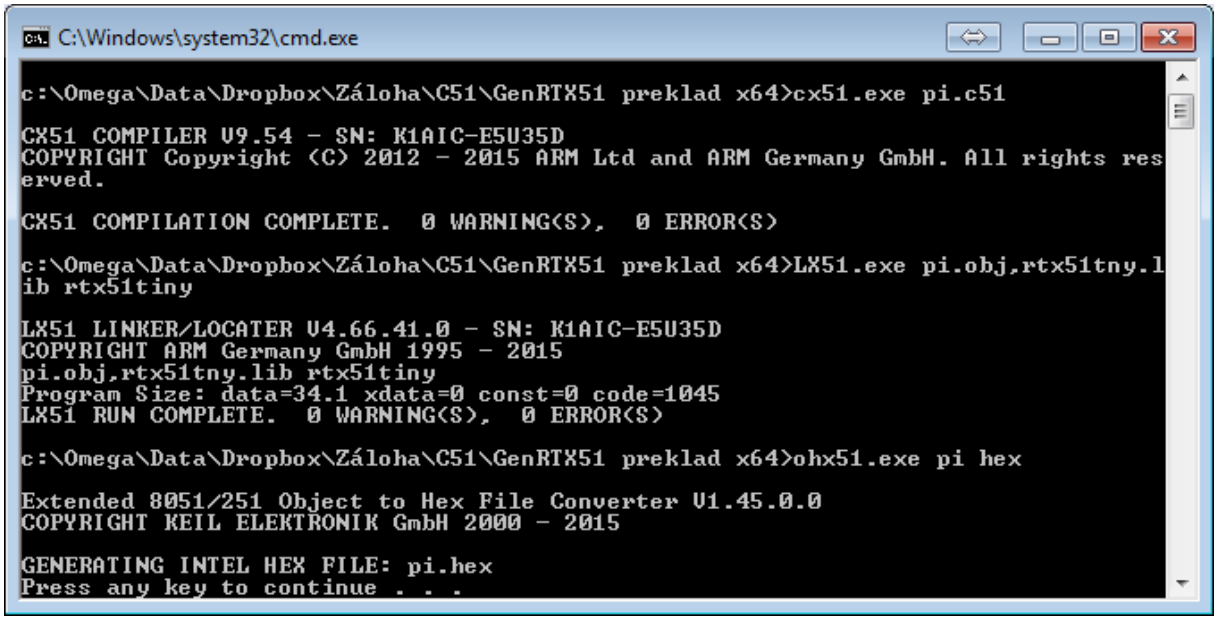
[13] Windows XP 64-bit, Windows 7 64-bit, Windows 8.1 64-bit, Windows 10 64-bit
[14] 32-bitové operačné systémy môžu adresovať maximálne 4GB fyzickej pamäti RAM, u 64-bitových systémoch už reálne obmedzenie v podobe približne **16,8 miliónov terabyte** nie je.
[15] Vyskúšané a funkčné vo verzii Keil µVision 9.54a
[16] Súbor **tools.ini** obsahuje registračné údaje o Keil µVision ako je napr. S/N, alebo LIC ktoré sú priamo naviazané na konkrétny hardware PC.
[17] http://www.vmware.com

Obrázok 15, Preklad programu pomocou dávkového súboru

```
@echo on
cx51.exe pi.c51
lx51.exe pi.obj,rtx51tny.lib rtx51tiny
ohx51.exe pi hex
@echo off
del *.lst *.m51 *.
pause
```

Dávkový súbor môžeme pomenovať napr. **pi.bat**[18] a v tomto tvare je spustiteľný aj v prostredí 64-bitového operačného systému.

Obrázok 16, Výpis prekladu programu v 64-bitovom systéme



Z vyššie uvedeného obrázku je vidieť, že preklad programu bol úspešný a bol vygenerovaný „hex“ súbor, ktorý už môžeme naprogramovať do mikroprocesora.

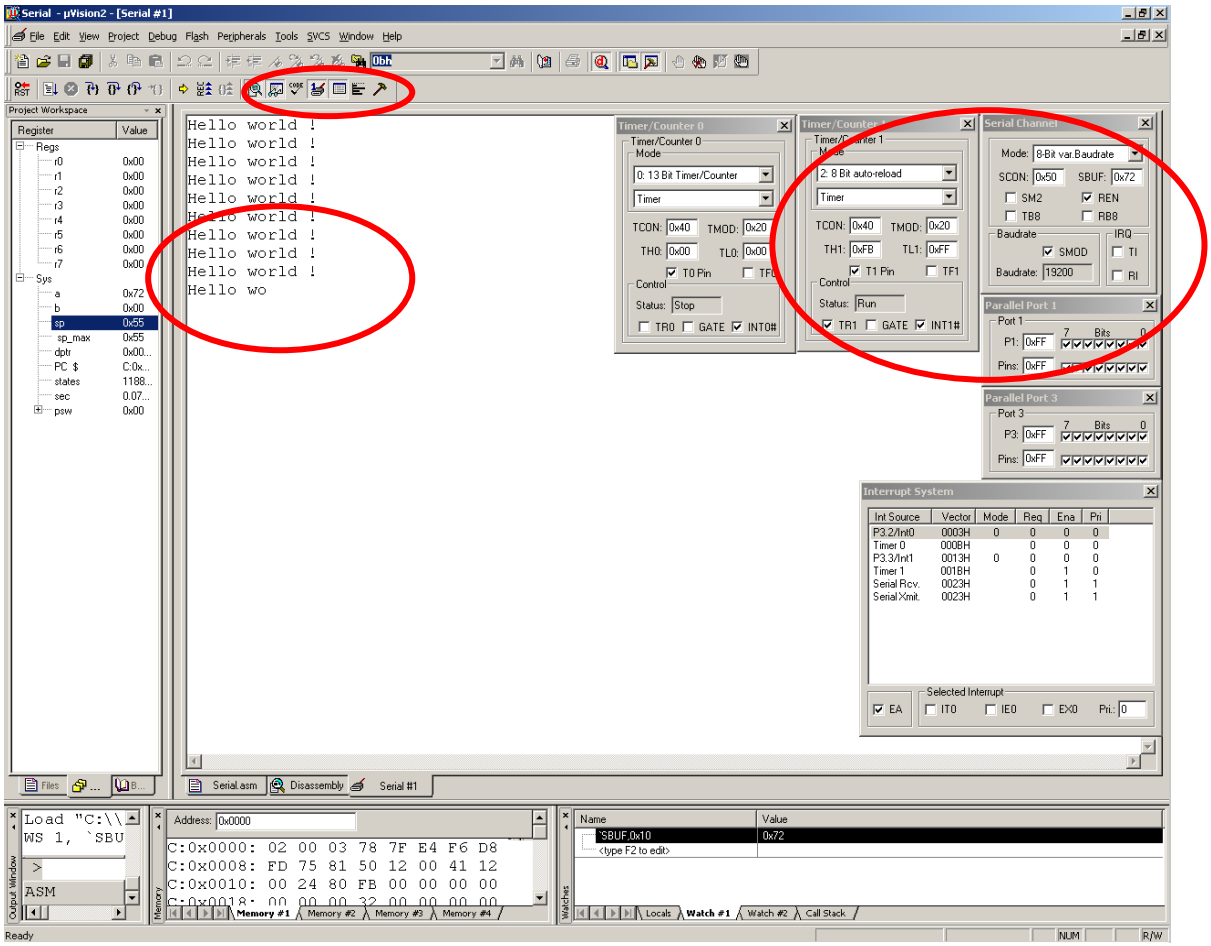
[18] Niektoré antivírusové programy upozorňujú pri spustení na možnú nebezpečnosť pri použití dávkového súboru. Odporúčame preto radšej tieto hlásenia antivírusového vypnúť, aby neustále nevypisoval možnosť ohrozenia PC, čo je nesprávny oznam pre užívateľa.

1.12. Ladenie aplikácií napísaných v A51 pomocou µVision

Každý program sa musí po napísaní v počiatočnej fáze odladiť – odskúšať. Na túto činnosť existuje množstvo nástrojov líšiacich sa možnosťami a samozrejme aj cenou. Medzi najjednoduchšie môžeme zaradiť tzv. softwarové (programové) simulátory, ktoré síce umožnia daný program odladiť, ale s väčšími, alebo menšími nepresnosťami. Lepšie výsledky nám dávajú tzv. emulátory, ktoré sú v osadené cieľovým mikroprocesorom v ktorom je vykonávaný odladovaný program.

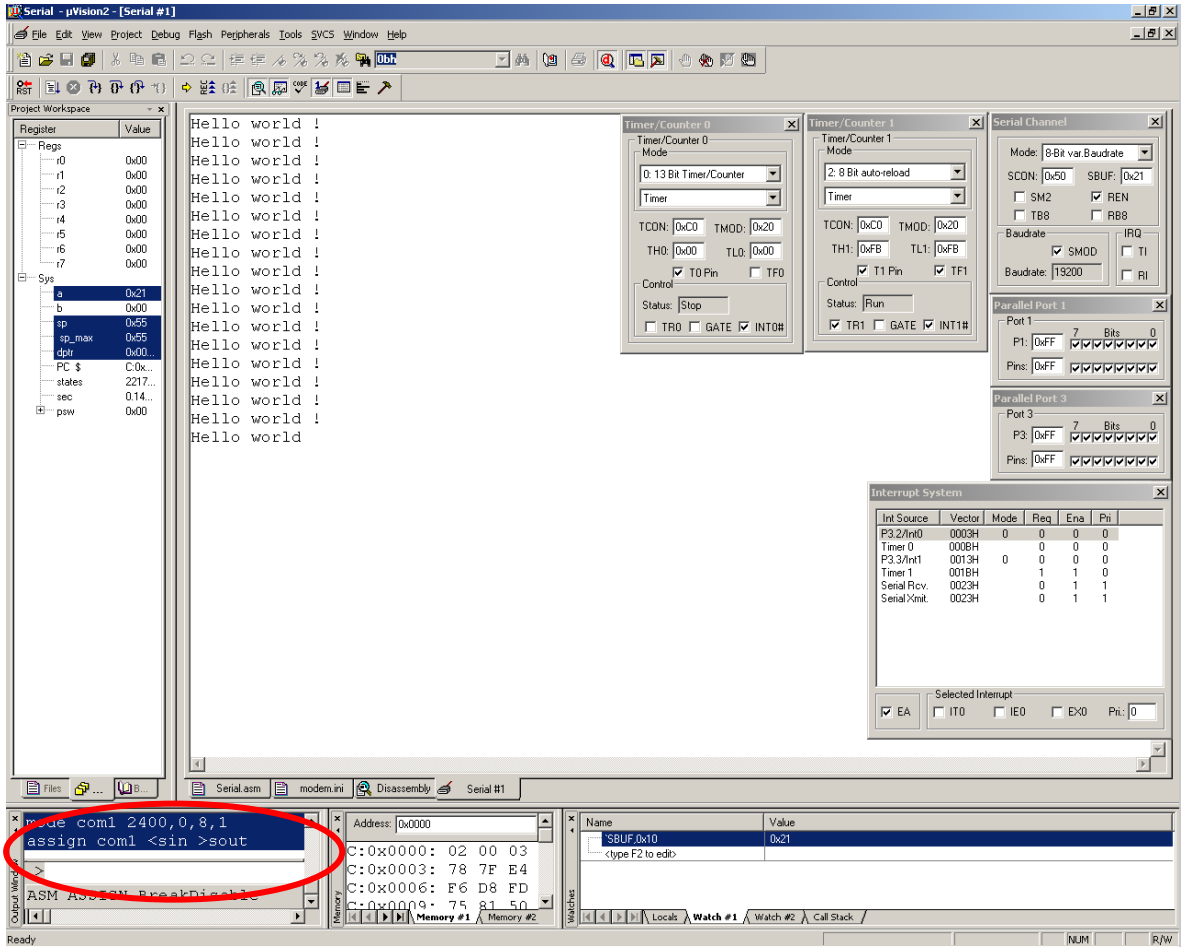
Pre naše potreby bude postačovať ladiaci program – debugger ktorý je súčasťou prostredia µVision. Pre úspešné odladenie programu je potrebné v µVision3 zvoliť v menu programu položku „Serial Window #1" a ak je program správne napísaný tak po jeho spustení začne vypisovať celú komunikáciu mikroprocesora na virtuálny[19] sériový kanál prostredia µVision.

Obrázok 17, Odladovaný program sériového rozhrania v µVision



[19] Výpis musí byť presmerovaný len na port COM1 až COM4 t.j. fyzický port v PC, ináč nepracuje správne! Porty COM1 až COM4 (3F8h, 3E8h, 2F8h, 2E8h) je možné použiť ako obojsmerné (bidirectional EPP. ECP) porty pri ladení aplikácií s rôznym hardware pomocou funkcií *scanf()* a *printf()*, prípadne s registrom SBUF.

Obrázok 18, Nastavenie µVision pre presmerovanie sériového kanála na port COM.



Ak sa pri odlaďovaní[20] použije možnosť presmerovanie sériového kanála prostredia µVision na porty PC, celá komunikácia mikroprocesora 8051 bude zachytávaná, zobrazovaná na obrazovke a zároveň odosielaná na porty COM1 až COM4. Je nutné upozorniť, že každá operácia čítania, alebo zápisu s registrom SBUF spôsobí zmenu na príslušných portoch. Určitou nevýhodou tohto programu je obmedzený rozsah komunikačných portov obmedzujúcich[21] sa len na fyzické porty COM1 až COM4. Aj napriek tejto nevýhode je možno pripojiť napríklad modem, GSM mobil, sériovú tlačiareň, čítačku čiarových kódov a iných zariadení ktoré komunikujú cez sériové rozhranie RS232.

Pre nastavenie parametrov sériového kanála v prostredí µVision sa používa príkaz MODE ktorý poznáme z prostredia MS DOS-u s parametrami.

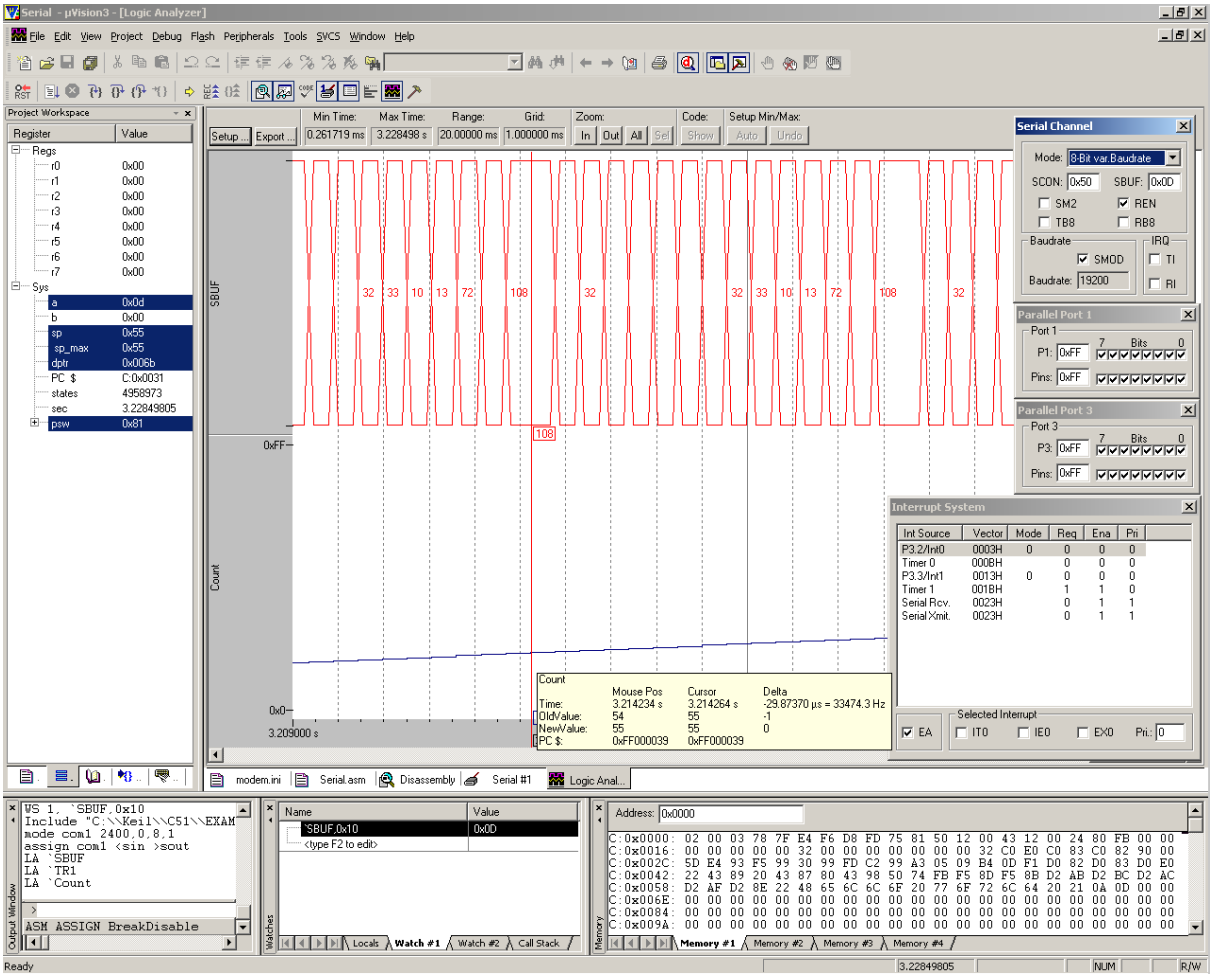
[20] Pre správnu činnosť funkcie ***printf()*** je nutné ihneď po inicializácii sériového kanála nastaviť **TI=1**. Príznakový bit **TI=0** je potrebné nastaviť **len vtedy** ak používame pri prenose údajov prerušenie od sériového kanála pre sériový prenos s adresou obsluhy prerušenia na adrese 0023H.

[21] Nové PC majú už obvykle len jeden fyzický port COM1, alebo žiaden , ktorý tam je len kvôli spätnej kompatibilite s existujúcim hardware a software. S výhodou je možné použiť USB Serial napr. s čipom CH340C, ktorý umožní vytvorenie virtuálneho sériového portu COM cez rozhranie USB.

MODE COM1 2400,0,8,1 (2400 – prenosová rýchlosť, bez parity, 8 bitov, 1 stop bit)

ASSIGN COM1 <sin >sout (Priradenie vstupu/výstupu portu COM1)

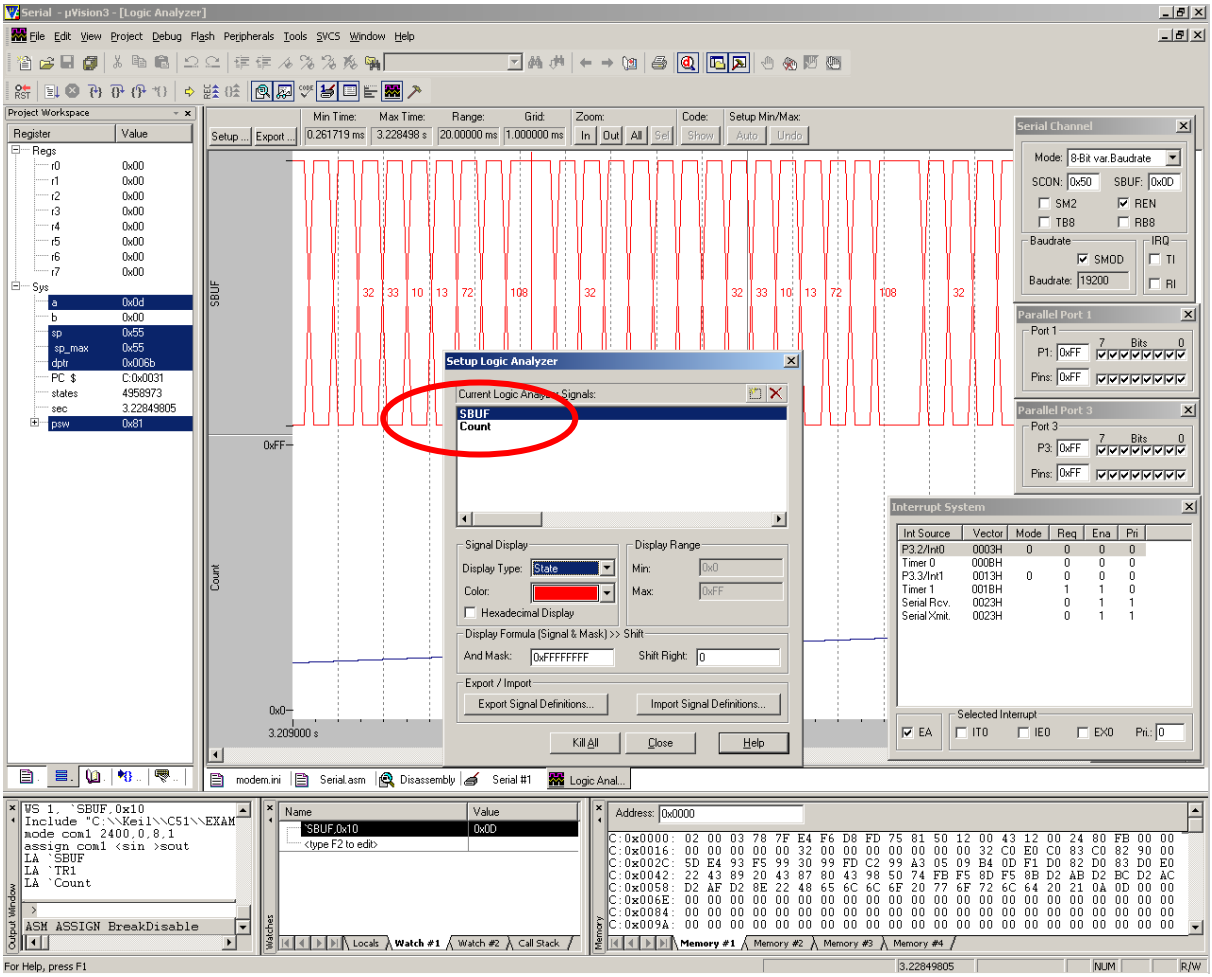
Obrázok 19, µVision3 a využitie funkcie Logic Analyzer - osciloskop



Zobrazenie časového priebehu logických úrovní na jednotlivých portoch mikroprocesora 8051 je bez použitia drahých logických analyzátorov veľmi komplikované a náročné. Túto náročnú operáciu je možné práve v softwarových simulátoroch veľmi dobre integrovať. Pri odlaďovaní programu môže programátor sledovať priebehy signálov na portoch mikroprocesora. Túto funkciu je možné použiť len od µVision3[22] od firmy Keil.

[22] Užívateľ môže od tejto verzie použiť v sekcii „**Option for Target**“ tzv. „**Initialization File**“ ktorý je v podstate program C, ktorý môže generovať rôzne analógové, alebo číslicové priebehy t.j. signály pre hlavný program. Veľkou výhodou je, že v tomto programe je možné pracovať so všetkými premennými ktoré sú deklarované v hlavnom programe ktorý odlaďujeme. Premenné ktoré vstupujú do simulácie musia byť deklarované ako globálne. Problém môžu spôsobovať mikroprocesory s viacerými sériovými rozhraniami RS232, CAN, I²C, kde je potom potrebné zmeniť názov simulovaného kanála z **SIN** na **SIN0**.

Obrázok 20, Nastavenie funkcie Logic Analyzer - osciloskop



Nastavenie funkcie osciloskopu je možné vykonať v okne „Setup Logic Analyzer“. V tomto okne je možné nastaviť typ premennej[23], portu a spôsob ich reprezentovaného zobrazenia na obrazovke. Zároveň je možné všetky nastavenia importovať, exportovať, alebo maskovať a zapísať do súboru pre neskoršie použitie. Zobrazované názvy signálov a parametre sériového rozhrania je možné vidieť aj v ľavom dolnom rohu prostredia.

[23] Aby bol obsah premennej viditeľný na osciloskope musí byť premenná deklarovaná ako globálna. Lokálne deklarované premenné a premenné ktoré sú zapuzdrené vo funkciách, alebo task-och RTX51 nie je možné zobrazovať.

1.13. Programovanie jednoduchých aplikácií v A51 - príklady

Príklad: Napíšte program v A51, ktorý bude vykonávať reguláciu pohonu jednosmerného motora s cudzím budením pomocou PSD regulátora *(prírastkového)*[24], kde skutočná hodnota nameranej veličiny sa bude nachádzať na porte P1, žiadaná hodnota na P3 a akčná veličina na porte P2. Všetky premenné, konštanty a vstupné hodnoty budú kvôli jednoduchosti programu 8 bitové celočíselné zo znamienkom *(signed char)*[25]:

$$y_N = y_{N-1} + A \cdot e_N - B \cdot e_{N-1} + C \cdot e_{N-2} \qquad (1.)$$

Kde ***A, B, C*** sú konštanty a $\boldsymbol{e_N}$ je odchýlka žiadanej a skutočnej regulovanej veličiny.

$$A = K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T}\right) \qquad (2.)$$

$$B = K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T}\right) \qquad (3.)$$

$$C = K \cdot \left(\frac{T_d}{\Delta T}\right) \qquad (4.)$$

Teoretický opis jednotlivých variantov regulátora je uvedený v kapitole 17.1, kde je jednoduchým spôsobom vysvetlená diskrétna teória regulácie s použitím mikroprocesorov. V kapitole 17.10 a 17.11 je vyššie uvedený diskrétny PSD regulátor naprogramovaný v prostredí operačného systému RTX51 s 32 bitovou aritmetikou, kde jednotlivé premenné majú reálny údajový typ. V praxi sa snažíme reálnu[26] aritmetiku nahradiť celočíselnou, ktorá je v mikroprocesoroch vypočítaná omnoho rýchlejšie.

[24] Prírastkový regulátor sa od absolútneho odlišuje výrazne v rýchlosti výpočtu, pretože počíta v algoritme len konkrétne prírastky z regulačnej odchýlky a nie súčty jednotlivých „súm“, ktoré sa vo výpočte pripočítavajú k zásahu na výstupe regulátora. Pre urýchlenie výpočtu je vhodné pri reálnej aritmetike použiť tzv. tabuľkový spôsob, ktorý spočíva v uložení výsledkov jednotlivých krokov výpočtov do tabuľky, ktoré neskôr používame vo svojich procedúrach a funkciách.

[25] Použitý údajový typ bol len pre verziu napísanú v jazyku assembler, pre jazyk C je použitý celočíselný údajový typ ***int***, alebo reálny údajový typ ***float***.

[26] Nie je to možné vždy splniť, pretože vstupný rozsah hodnôt a regulačný vzťah to vždy neumožňuje.

Príklad zápisu programu v jazyku assembler:

c:\Omega\data\Dropbox\Záloha\C51\PSD8bit\PSD_8bit_asm.asm

```
1   ;------------------------------------------------
2   ?PR?main?PSD8  SEGMENT  CODE
3   ?DT?PSD8  SEGMENT  DATA
4   ;------------------------------------------------
5       RSEG   ?DT?PSD8
6   ;------------------------------------------------
7   KA:       DS    1
8   KB:       DS    1
9   KC:       DS    1
10  En:       DS    1
11  Is:       DS    1
12  Iz:       DS    1
13  Yn:       DS    1
14  En1:      DS    1
15  En2:      DS    1
16  Yn1:      DS    1
17  ;------------------------------------------------
18      RSEG   ?PR?main?PSD8
19      USING    0
20  ;------------------------------------------------
21      MOV       KA,#01H
22      MOV       KB,#02H
23      MOV       KC,#03H
24  Cykl:
25      MOV       Iz,P1
26      MOV       Is,P3
27  ;------------------------------------------------
28  ;    En=Iz-Is;
29  ;------------------------------------------------
30      CLR       C
31      MOV       A,Iz
32      SUBB      A,Is
33      MOV       En,A
34  ;------------------------------------------------
35  ;   Yn=Yn1+(KA*En)-(KB*En1)+(KC*En2);
36  ;------------------------------------------------
37      MOV       A,KA
38      MOV       B,En
39      MUL       AB
40      ADD       A,Yn1
41      MOV       R7,A
42      MOV       A,KB
43      MOV       B,En1
44      MUL       AB
45      MOV       R6,A
46      CLR       C
47      MOV       A,R7
48      SUBB      A,R6
49      MOV       R7,A
50      MOV       A,KC
51      MOV       B,En2
52      MUL       AB
53      ADD       A,R7
54      MOV       Yn,A
55  ;------------------------------------------------
56      MOV       Yn1,A
57      MOV       En2,En1
58      MOV       En1,En
59      SJMP      Cykl
60  ;------------------------------------------------
61      END
62
```

Poznámka.

V programe nie je použitá funkcia ***Delay()***, ktorá zabezpečuje periodické vykonávanie regulačného algoritmu v presne definovaných časových intervaloch.

Príklad zápisu programu v jazyku C:

C:\Omega\Data\Dropbox\Záloha\C51\PSD 8bit\PSD8.c

```
 1  #include <reg51.h>
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4  #include <string.h>
 5
 6  signed char KA,KB,KC,En,En1,En2,Yn,Yn1,Iz,Is;
 7
 8  void main(void)
 9  {
10   KA=1;
11   KB=2;
12   KC=3;
13   while(1)
14    {
15     Iz=P1;
16     Is=P3;
17     En=Iz-Is;
18     Yn=Yn1+(KA*En)-(KB*En1)+(KC*En2);
19     Yn1=Yn;
20     En2=En1;
21     En1=En;
22    }
23  }
24
```

Príklad v jazyku C je uvedený len pre ilustráciu špecifikácií pri tvorbe aplikácií medzi vyšším programovacím jazykom a jazykom symbolických adries. Vo vyššom programovacom jazyku je možné jednoduchým spôsobom zmeniť údajový typ premenných na ***float***, alebo ***double*** a výpočty budú vykonávané s reálnou aritmetikou. Teoretický rozbor regulátora je venovaný v kapitole 17.1, kde čitateľ je oboznámený so základnými poznatkami z teórie oblasti diskrétnej regulácie a regulátorov.

Poznámka.

V programe nie je použitá funkcia ***Delay()***, ktorá zabezpečuje periodické vykonávanie regulačného algoritmu v presne definovaných časových intervaloch.

Príklad: Napíšte program typu „reklama“ ktorý bude v jazyku assembler periodicky vypisovať na porte P1 údaje nachádzajúce sa v tabuľke. Tabuľka obsahuje tzv. spínací plán LED diód pripojených na porte P1. Koniec tabuľky bude označený znakom ‘*’.

```
;------------------------------------------------------------------
BANK        macro   N
            mov psw,#(&N shl 3)
            endm
;------------------------------------------------------------------
            cseg at 0000h
            jmp Start
;------------------------------------------------------------------
Prg0        segment code
            rseg Prg0
;------------------------------------------------------------------
Tabulka:    db  10000000b,01000000b,00100000b,00010000b
            db  00001000b,00000100b,00000010b,00000001b
            db  00000001b,00000010b,00000100b,00001000b
            db  00010000b,00100000b,01000000b,10000000b
            db  11001100b,00110011b,11001100b,00110011b
            db  11001100b,00110011b,11001100b,00110011b
            db  11110000b,00001111b,11110000b,00001111b
            db  11110000b,00001111b,11110000b,00001111b
            db  11111111b,00000000b,00000000b,00000000b
            db  '*'
;------------------------------------------------------------------
Prg1        segment code
            rseg Prg1
;------------------------------------------------------------------
Start:      mov r0,#7fh         ;Pocet byte mazanej pamati v RAM ...
            clr a               ;Tymto prepisem pamat ...
Mazem:      mov @r0,a           ;Teraz vymazem konkretny byte ....
            djnz r0,Mazem       ;Opakujem skok na Mazem pokial R0<>0
Zobrazuj:   mov dptr,#Tabulka   ;Kde je Tabulka ???
Cykluj:     clr a
            movc a,@a+dptr      ;Nacitam prvy byte z Tabulky
            cjne a,#'*',Pokracuj
            jmp Zobrazuj
;------------------------------------------------------------------
Prg2        segment code
            rseg Prg2
;------------------------------------------------------------------
Pokracuj:   mov P1,a            ;Zobrazim na LED pripravku
            call Delay          ;Oneskorim to ....
            inc dptr            ;Posuniem sa na dalsi byte v Tabulke
            cjne a,#'*',Cykluj  ;Ak A<>'*' tak zobrazujem !!!
            jmp Zobrazuj
;------------------------------------------------------------------
Prg3        segment code
            rseg Prg3
;------------------------------------------------------------------
Delay:      BANK 3              ;Prepnem sa na inu banku registrov
            mov r0,#0FFh        ;Oneskorim to o asi 255*255 us
            mov r1,#0FFh
CCC:        djnz r1,$
            djnz r0,CCC
            BANK 0              ;Vratim sa na povodnu banku registrov
            ret
;------------------------------------------------------------------

            end
```

Poznámka.

Funkcia ***Delay()*** v programe má len demonštračnú funkciu, negeneruje časovo presne definované časové intervaly.

Príklad v jazyku C:

```
1   #include <reg51.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4
5
6   unsigned char Tabulka[]={1,2,4,8,16,32,64,128,64,32,16,8,4,2,1,0};
7   unsigned char n;
8
9   sbit Tlacidlo = P3^0;
10
11  void Delay(long int Time)
12  {
13   while(--Time!=0);
14  }
15
16  void main(void)
17  {
18   while(1)
19   {
20    n=0;
21    while(Tabulka[n]!=0)
22     {
23      if(Tlacidlo==0) P1=Tabulka[n++];
24      Delay(100000);
25     }
26   }
27  }
```

Poznámka.

Funkcia ***Delay()*** v programe má len demonštračnú funkciu, negeneruje časovo presne definované časové intervaly.

Príklad: Napíšte program, ktorý bude cez sériové rozhranie RS232 vysielať rýchlosťou 19,2kb.s^{-1} textový reťazec "Hello world !!!". Časti opakujúceho sa kódu napíšte pomocou makra. V programe definujte premenné umiestnené v programových, dátových segmentoch.

```
1               cseg at 0000h   ;org 0000h
2               jmp Init
3   ;-----------------------------------------
4   Text        segment code
5               rseg Text
6   ;-----------------------------------------
7   Info:       db 'Hello world !!!',0dh,0ah,00h
8   ;-----------------------------------------
9   Prog0       segment code
10              rseg Prog0
11  ;-----------------------------------------
12  Init:       clr a
13              mov r0,#7fh     ;Vymazem pamat od 00h do 7fh
14  Cyklus:     mov @r0,a
15              djnz r0,Cyklus
16              orl tmod,#20h   ;T1 ako generator prenosovej rychlosti
17              mov scon,#50h   ;MODE 1 seriovej linky
18              orl pcon,#80h   ;Nastavim 2x rychlost
19              mov a,#0fbh     ;Nastavim rychlost 19200bps pri 18.432MHz
20              mov th1,a
21              mov tl1,a
22              setb tr1        ;Spustim casovac T1
23              jmp Send
24  ;-----------------------------------------
25  Prog1       segment code
26              rseg Prog1
27  ;-----------------------------------------
28  Send:       mov dptr,#Info  ;Ukazujem na data pre vysielanie ...
29  Pokracuj:   clr a
30              movc a,@a+dptr  ;Nacitam jeden znak z pamati ....
31              mov sbuf,a      ;Posielam na seriovu linku znak !!!
32              inc dptr        ;Posuniem sa na dalsi znakm ....
33              jnb ti,$        ;Cakam pokial nie je TI=1
34              clr ti          ;TI=0
35              call Delay
36              cjne a,#00h,Pokracuj
37              jmp Send
38  ;-----------------------------------------
39  Prog2       segment code
40              rseg Prog2
41  ;-----------------------------------------
42  Delay:      mov r0,#0ffh
43  CCC:        mov r1,#0ffh
44              djnz r1,$
45              djnz r0,CCC
46              ret
47  ;-----------------------------------------
48              end
```

Poznámka.

Funkcia ***Delay()*** v programe má len demonštračnú funkciu, negeneruje časovo presne definované časové intervaly.

Príklad v jazyku C:

```
1   #include <reg51.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   char Text[] = {"Hello world !!!\n\r"};
6
7   void Delay(unsigned int Time)
8   {
9    while(--Time!=0x00);
10  }
11
12  void main(void)
13  {
14   TH1=TL1=0xFB;
15   TMOD|=0x20;
16   SCON=0x50;
17   PCON|=0x80;
18   TR1=1;
19   TI=1;
20   while(1)
21    {
22     printf("%s",Text);
23     Delay(0xFFFF);
24    }
25  }
```

Poznámka.

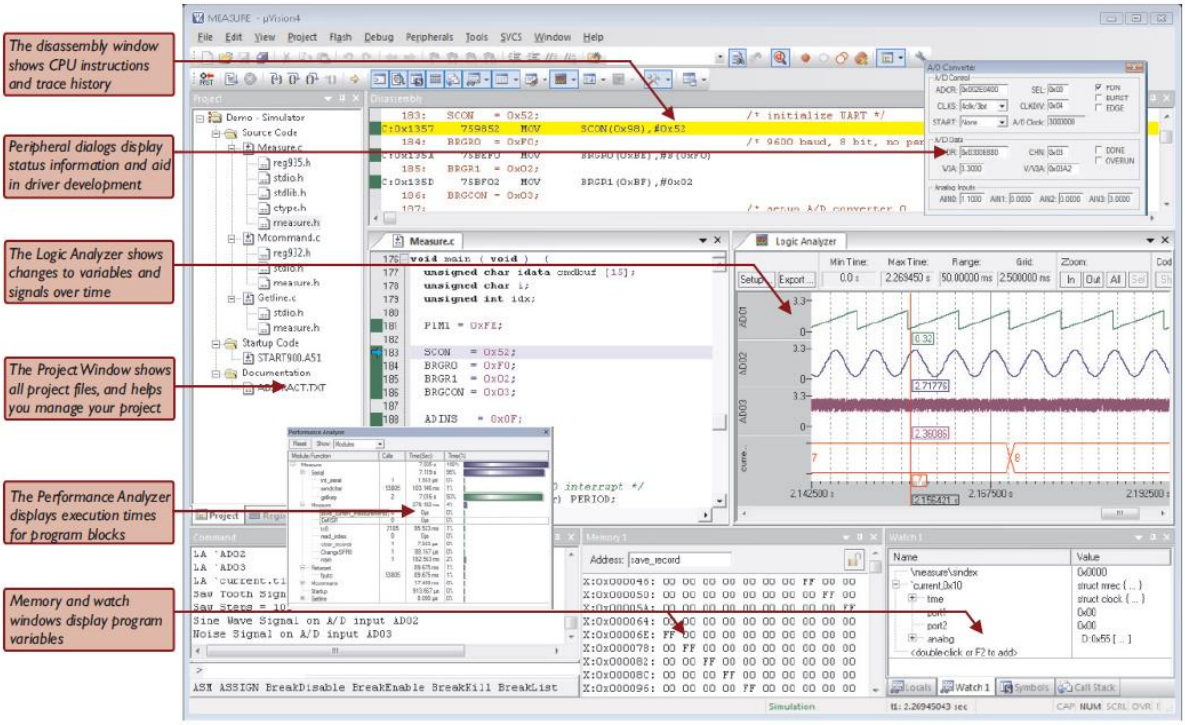
Funkcia ***Delay()*** v programe má len demonštračnú funkciu, negeneruje časovo presne definované časové intervaly.

2 Práca v prostredí µVision

Cieľom tejto kapitoly bude čitateľa oboznámiť s filozofiou tvorby projektu v prostredí µVision4 a ich vyšších verzií. Spôsob tvorby projektu sa diametrálne líši od spôsobu pri práci v prostredí prekladača AS552, ale minimálne v iných IDE[27] prostrediach.

Projekt je súhrn všetkých súborov, knižníc ktoré sú potrebné pre vytvorenie výsledného spustiteľného tvaru HEX, BIN v mikroprocesore 8051, alebo knižnice s príponou LIB.

Obrázok 21, Vzorový príklad projektu µVision4



The µVision development platform is easy to use and it helps you to quickly create embedded programs that work. The µVision editor and debugger are integrated in a single application that provides a seamless embedded project development environment for editing, simulating, Flash programming and testing in target hardware.

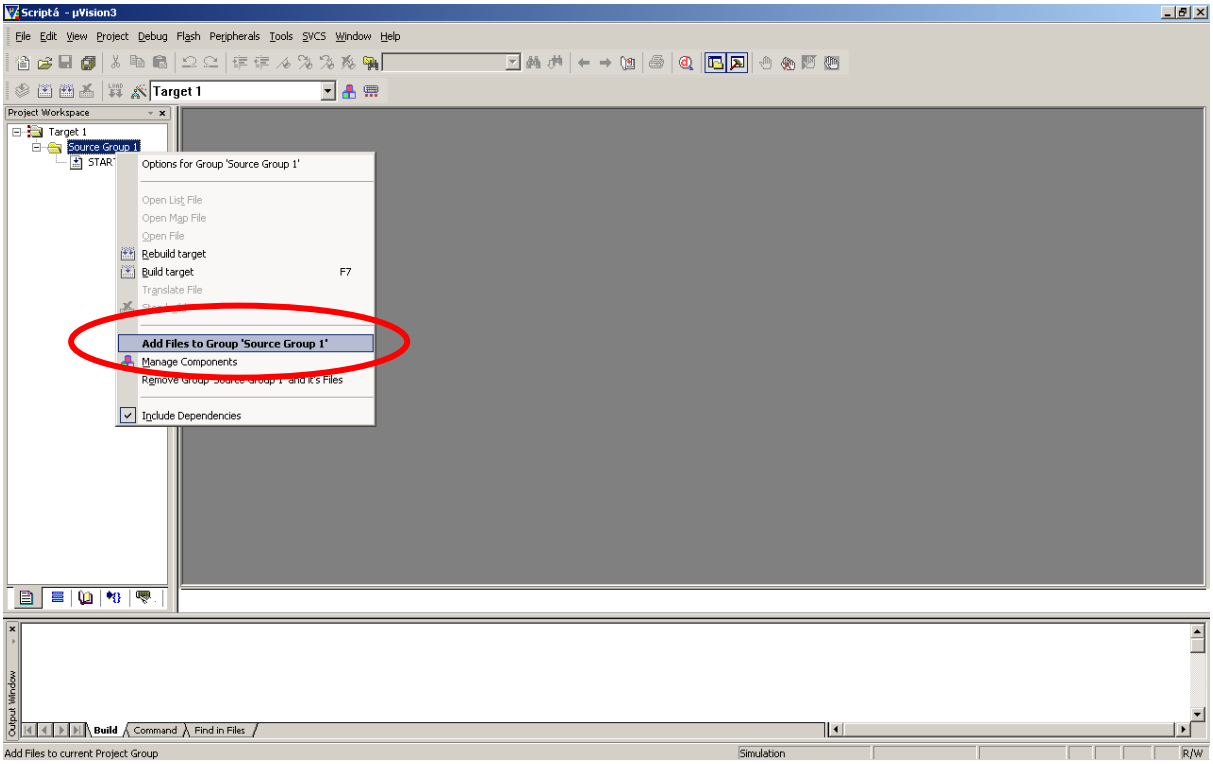
2.1. Postup pri vytváraní projektu

a. vytvorenie adresára pre nový projekt v menu „Create New Project“
b. voľba mikroprocesora v menu „Select Device for Target“ , viď Obrázok 23
c. v okne „Project Workspace“ načítať súbory (asm, c, h, inc, lib ....), viď Obrázok 22
d. v okne „Options for Target“ zvoliť vlastnosti projektu, viď Obrázok 24
e. nastaviť výstupný formát preloženého súboru, Obrázok 25
f. nastaviť veľkosti optimalizačného stupňa (použiteľné len pre C) , Obrázok 26

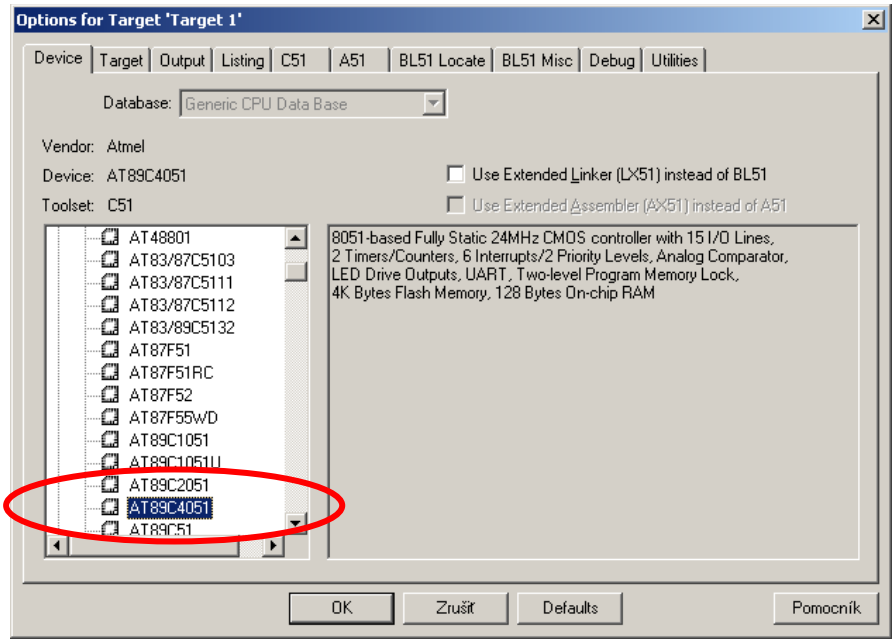
[27] IDE - Integrated Development Environment

2.2. Tvorba projektu krok za krokom

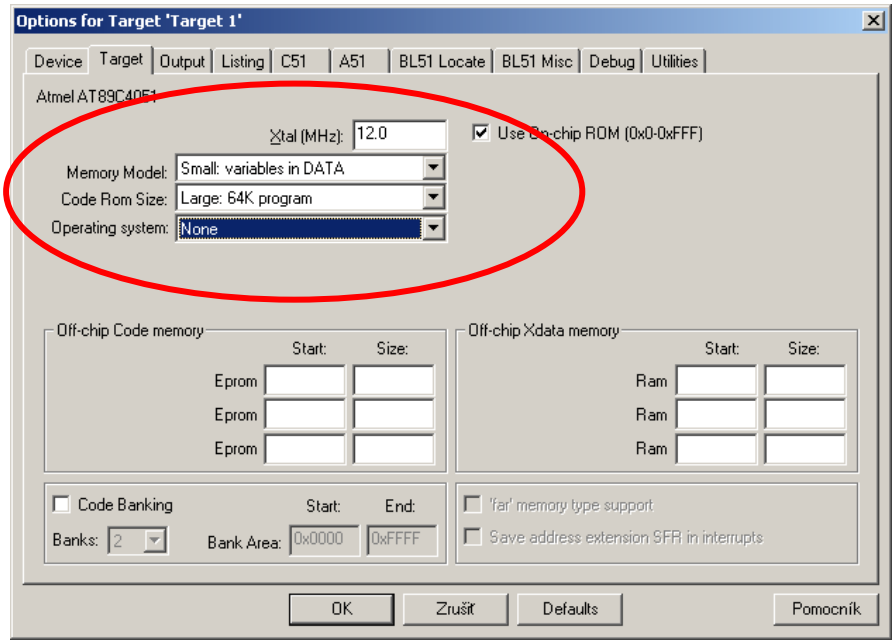
Obrázok 22, Voľba úborov pre preklad projektu v menu „Project Workspace“



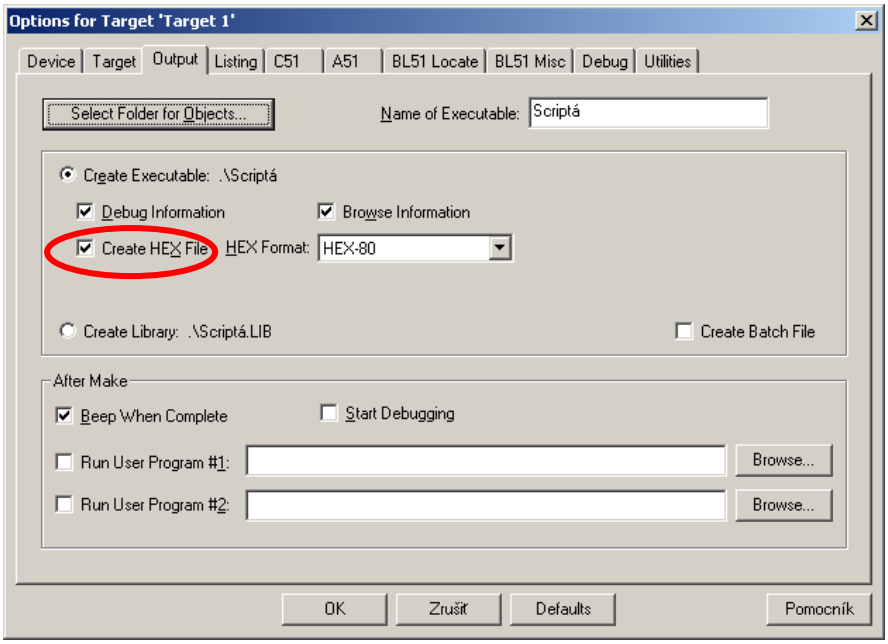
Obrázok 23, Voľba mikroprocesora v okne „Options for Target“



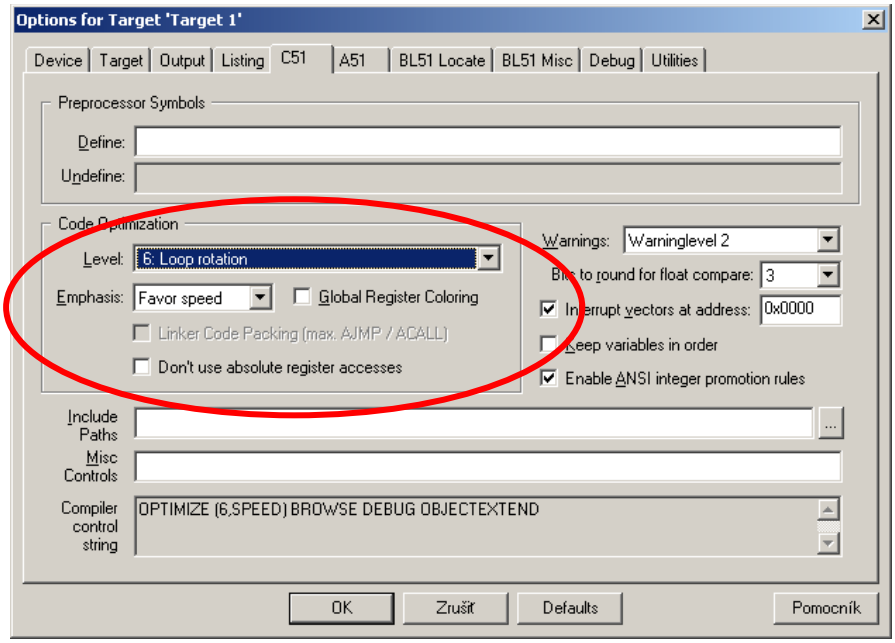
Obrázok 24, Voľba vlastností projektu v okne „Options for Target“



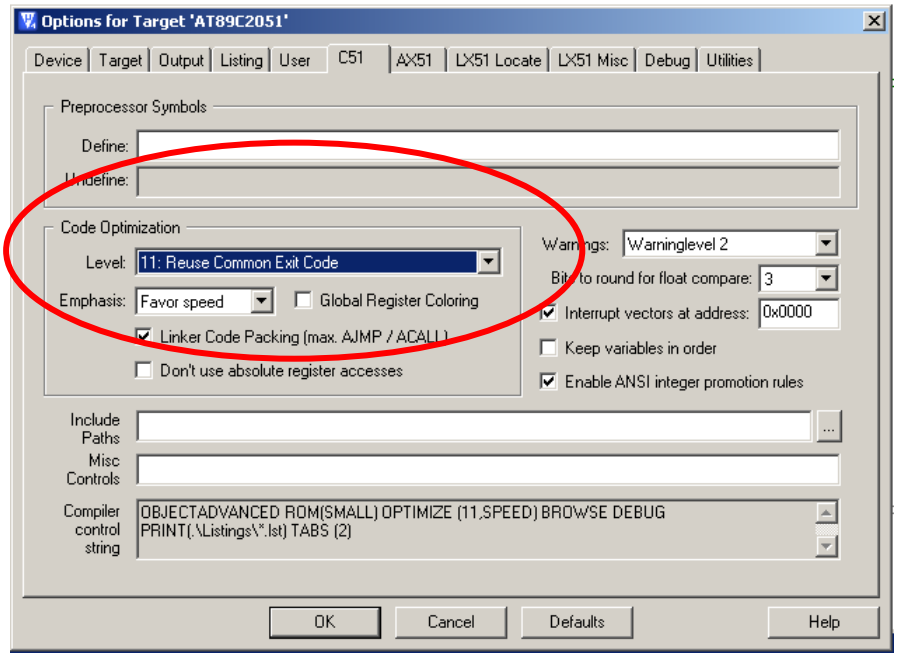
Obrázok 25, Nastavenie formátu výstupu prekladaného súboru do HEX v µVision3



Obrázok 26, Nastavenie optimalizácie a optimalizačného stupňa v µVision3[28]



Obrázok 27, Použitie najvyšších optimalizačných stupňov v Ax51 a Cx51

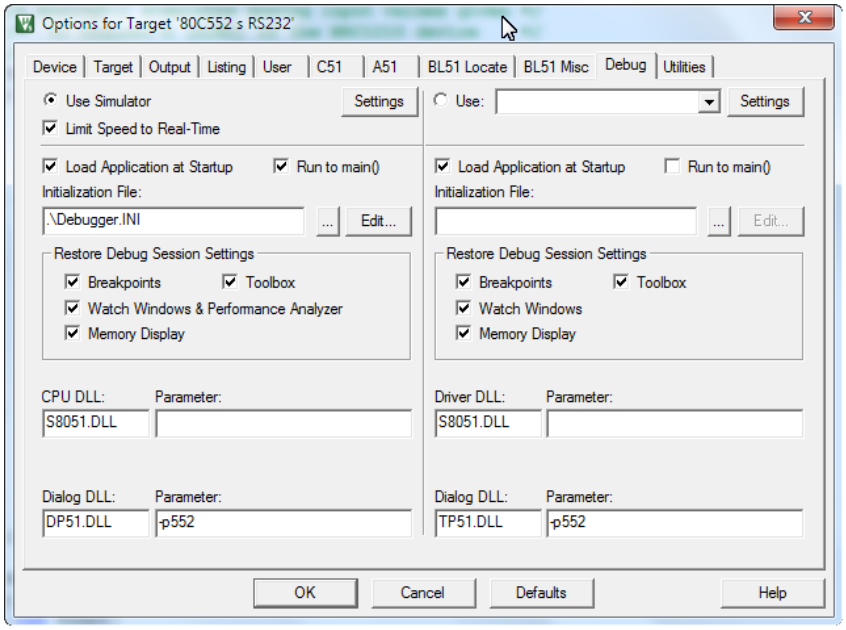


[28] Optimalizačný stupeň je jedným z dôležitých parametrov ovplyvňujúcim preklad výstupného programu do spustiteľného tvaru pre mikroprocesor 8051 v Intel hex formáte. Pri použití kompilátora Ax51 a Cx51 je možné zvoliť až optimalizačný stupeň 11. Pri použití kompilátora Ax51, alebo Cx51 nie je možné ladiť aplikácie napísané s RTX51. Je to preto, lebo RTX51 používa možnosti adresovať pamäť dát a programu v rozsahu 64kB, prípadne s použitím xBanking-u n*64kB, nepodporuje totiž 24 bitové adresovanie používané v modernejších a architektúrne modifikovaných procesoroch.

2.3. Ladenie programov v µVision pomocou integrovaného Debugger-a

Vývojové nástroje často obsahujú rôzne doplňujúce funkcie ladenia t.j. experimentálne overovanie správnej činnosti vytváraného programu. Jedným z najmocnejších nástrojov v prostredí µVision je integrovaný[29] Debugger. V súbore s príponou **.ini** napr. **debugger.ini** sú použité časti programov, procedúr, funkcií napísané v jazyku C ktoré sa súčasne vykonávajú paralelne s odlaďovaným programom. Všetky zmeny obsahov premenných[30], registrov sa okamžite zapisujú v prostredí a vytvárajú tak variabilne interaktívne prostredie ktoré reaguje na vstupné parametre prichádzajúce z externého programu t.j. **debugger.ini**. V tomto súbore je možné definovať pomocou množstva interných premenných prostredia Keil µVision ďalšie správanie sa odlaďovaného programu v závislosti na vstupných hodnotách a parametroch[31].

Obrázok 28, Nastavenie integrovaného Debugger-a



[29] Existuje možnosť použitia aj externého debugger-a, čo je v podstate špecializovaný hardware obsahujúci cieľový mikroprocesor ktorý komunikuje cez špeciálne rozhranie JTAG, pomocou ktorého je schopný čítať obsahy registrov, premenných, štruktúr pričom je možné vykonávanie programu pozastaviť, spustiť a krokovať po jednotlivých inštrukciách. Cena uvedeného hardware je pre začínajúceho programátora často veľmi vysoká a preto je lepšie keď si odlaďovanie programu skúsi ovládať cez integrovaný debugger v Keil µVision.

[30] Všetky premenné v hlavnom programe ktoré chceme použiť pri ladení v súbore **.ini** musia byť deklarované ako globálne, deklarácie funkcií majú mierne obmedzenia, preto je potrebné sledovať pri spustení **Debugger-a** okno „**Command**“, kde je vypisovaný stav úspešnosti prekladu súboru **.ini**, ktorý umožní úspešne odhaliť prípadnú chybu. Je vhodné oboznámiť sa pred použitím debugger-a s internými premennými pre debugger **VTREGS** v Keil µVision.

[31] Položka **Limit Speed Real-Time** umožňuje prispôsobiť rýchlosť ladeného programu na reálne hodnoty ktoré budeme dosahovať v praxi. Efekt „reálnosti“ vykonávaného programu vidíme najlepšie na výkonných počítačoch, kde by sme vykonávanie kódu bez zaškrtnutej položky nemohli kvôli rýchlosti vykonávania programu ani len sledovať.

Obrázok 29, Príklad zápisu programu pre Debugger

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Debugger.INI

```
/*-----------------------------------------------*/
/* Analog0() simulates analog input values given */
/*  to channel-0 (AIN0) of the MSC1210 device    */
/*-----------------------------------------------*/
Signal void analog0 (float limit)
{
  float volts,rnd;

  printf ("Analog0 (%f) entered.\n", limit);
  while (1)
 {          /* forever */
    volts = 0;
    while (volts <= limit)
    {
     ain0 = volts;     /* analog input-0 */
     swatch (0.01);     /* 0.1 Sec */
     volts += 0.1;     /* increase voltage */
    }
    volts = limit;
    while (volts >= 0.0)
    {
     ain0 = volts;
     swatch (0.01);
     volts -= 0.1;
    }
  }
}

Signal void su()
{
 int Count;
 while(1)
  {
//    if(++Count>=1000) P1.3=0;
   swatch(0.00333);
  }
}

Signal void Fazy_Right()
{
 int Citac;
 float TX,x;
 while(1)
  {
   x+=0.00000001;
   TX=(0.0100/3.0)+x;
   swatch(TX);
   P1.0=0; P1.1=1; P1.2=0;    //0x02
   swatch(TX);
   P1.0=0; P1.1=1; P1.2=1;    //0x06
   swatch(TX);
   P1.0=0; P1.1=0; P1.2=1;    //0x04
   swatch(TX);
   P1.0=1; P1.1=0; P1.2=1;    //0x05
   swatch(TX);
   P1.0=1; P1.1=0; P1.2=0;    //0x01
   swatch(TX);
   P1.0=1; P1.1=1; P1.2=0;    //0x03
  }
}

Signal void Fazy_Left()
{
 int Citac;
 float TX,x;
 while(1)
  {
   x+=0.00000001;
   TX=(0.0100/3.0)+x;
   swatch(TX);
   P1.0=0; P1.2=1; P1.1=0;    //0x04
   swatch(TX);
   P1.0=0; P1.2=1; P1.1=1;    //0x06
   swatch(TX);
   P1.0=0; P1.2=0; P1.1=1;    //0x02
```

Page 1

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Debugger.INI

```
   76       swatch(TX);
   77       P1.0=1; P1.2=0; P1.1=1;    //0x03
   78       swatch(TX);
   79       P1.0=1; P1.2=0; P1.1=0;    //0x01
   80       swatch(TX);
   81       P1.0=1; P1.2=1; P1.1=0;    //0x05
   82      }
   83    }
   84
   85    // Create signal functions for buttons
   86    signal void pb1 (void)
   87    {
   88      P3^=0x10;
   89      swatch (2);
   90    }
   91
   92    signal void pb2 (void)
   93    {
   94      P1^=0x10;
   95      swatch (2);
   96      P1^=0x10;
   97    }
   98
   99    signal void pb3 (void)
  100    {
  101      P4^=0x80;
  102      swatch (2);
  103      P4^=0x80;
  104    }
  105
  106    // Create signal functions for car detectors
  107    signal void cd1 (void)
  108    {
  109      P3^=0x04;
  110      swatch (2);
  111      P3^=0x04;
  112    }
  113
  114    signal void cd2 (void)
  115    {
  116      P3^=0x08;
  117      swatch (2);
  118      P3^=0x08;
  119    }
  120
  121    // Assign toolbox buttons to signal functions
  122    define button "ZSF", "pb1()"
  123    define button "OP", "pb2()"
  124    define button "Zavesenie frekvencie", "pb3()"
  125    define button "Nadprud slaby", "cd1()"
  126    define button "Nadprud silny", "cd2()"
  127
  128    // start volatage ramp on AIN0
  129    analog0 (4.9)
  130    //su()
  131    Fazy_Right()
  132    //Fazy_Left()
  133    //g,main
  134
  135    // fixed voltage on AIN0 .. AIN7
  136    AIN0=0.000000
  137    AIN1=0.000000
  138    AIN2=2.500000
  139    AIN3=0.000000
  140    AIN4=0.000000
  141    AIN5=0.000000
  142    AIN6=0.000000
  143    AIN7=0.000000
```

Page 2

3 Kompilátor C pre architektúru 8051

3.1. Minimálne požiadavky na prekladač

- Inteligentné rozmiestňovanie kódu programu a dát,
- Možnosť použitia dátových typov (bit, byte, word, dword, longint, real ...),
- Špeciálne metódy adresovania (xBanking, 24 bit adresovanie),
- Rôzne pamäťové modely (SMALL, COMPACT, LARGE, HUGE),
- Riadenie aplikácií na úrovni prerušení.

Množstvo aplikácií je napísaných vo vyšších programovacích jazykoch C, EC++, Pascal, Basic, Fortran a iných. Po podrobnejšom preskúmaní systémových zdrojov jednotlivých mikroprocesorov architektúry 8051 je možné povedať, že ak samotná systémová výbava architektúry procesora je obmedzenejšia v konečnom dôsledku sa znižuje efektívnosť samotného generovaného programovaného kódu prekladačom.

U niektorých architektúr mikroprocesorov, napríklad AVR, alebo ARM je zaistená optimálna veľkosť výsledného programového kódu nielen pomocou hardwaru, ale aj vďaka použitej architektúry ktorá obsahuje 32 univerzálnych registrov (GPR – General Purpose Register). Vďaka tejto 16 bitovej architektúre je už možné použiť ľubovoľný register pre aritmetickú, logickú alebo vo funkcii ukazovateľa dát a v spojení s jednostupňovým paralelným spracovaním inštrukcií (pipeling) je možné v jedinom hodinovom cykle spracovať viacero inštrukcií.

3.2. Moderné požiadavky na prekladač

Ideálny kompilátor – prekladač by mal teda vytvoriť cieľový program tak dobre ako keby bol napísaný v assembler-i. Vďaka neustálemu vývoju množstva existujúcich optimalizačných[32] metód prekladačov sa k tomuto cieľu blížime. Pri účinnom využití architektúry mikroprocesora je možné nadbytočnosť (overhead) kódu znížiť o cca. 30% v porovnaní s programom, ktorý je napísaný v assembler-i. V prípade že je algoritmus napísaný programátorom neefektívne, nepomôže ani špičkový kompilátor s optimalizačnými modulmi.

[32] Od verzie prostredia MDK 5.28 je podľa praktických skúseností prínosom zapnutie optimalizačného stupňa –O3, ktorý podstatným vylepší fungovanie aplikácie s RTX5. Bez optimalizácie –O3 aplikácia v náhodných intervaloch „zamŕzala“ a bola značne nestabilná. Nevýhodou tohto optimalizačného stupňa môže byť len to, že štandardný debugging v prostredí Keil µVision5 nemusí správne fungovať.

Až pri nových architektúrach AVR, alebo ARM je braný do úvahy ohľad na **HLL** (High Level Language), kde dochádza k zníženiu prebytočnému programovému kódu vygenerovanému v HLL takmer na nulu. Tento proces je ale závislý hlavne na programátorovi ktorý musí dodržovať pomerne veľké množstvo pravidiel pri písaní programu, aby vytvoril výsledný vysoko efektívny programový kód.

4 Tvorba programov v jazyku C51

4.1. Prehľad používaných architektúr mikroprocesorov

- Harvard,
- von Neuman,
- Hybridná *(Harvard + Von Neuman)*,
- CISC *(Complex Instruction Set Computer)*,
- RISC *(Reduced Instruction Set Computer)*,
- CISP *(Configurable Instruction Set Computer)*.

4.2. Prekladače jazyka C pre mikroprocesory

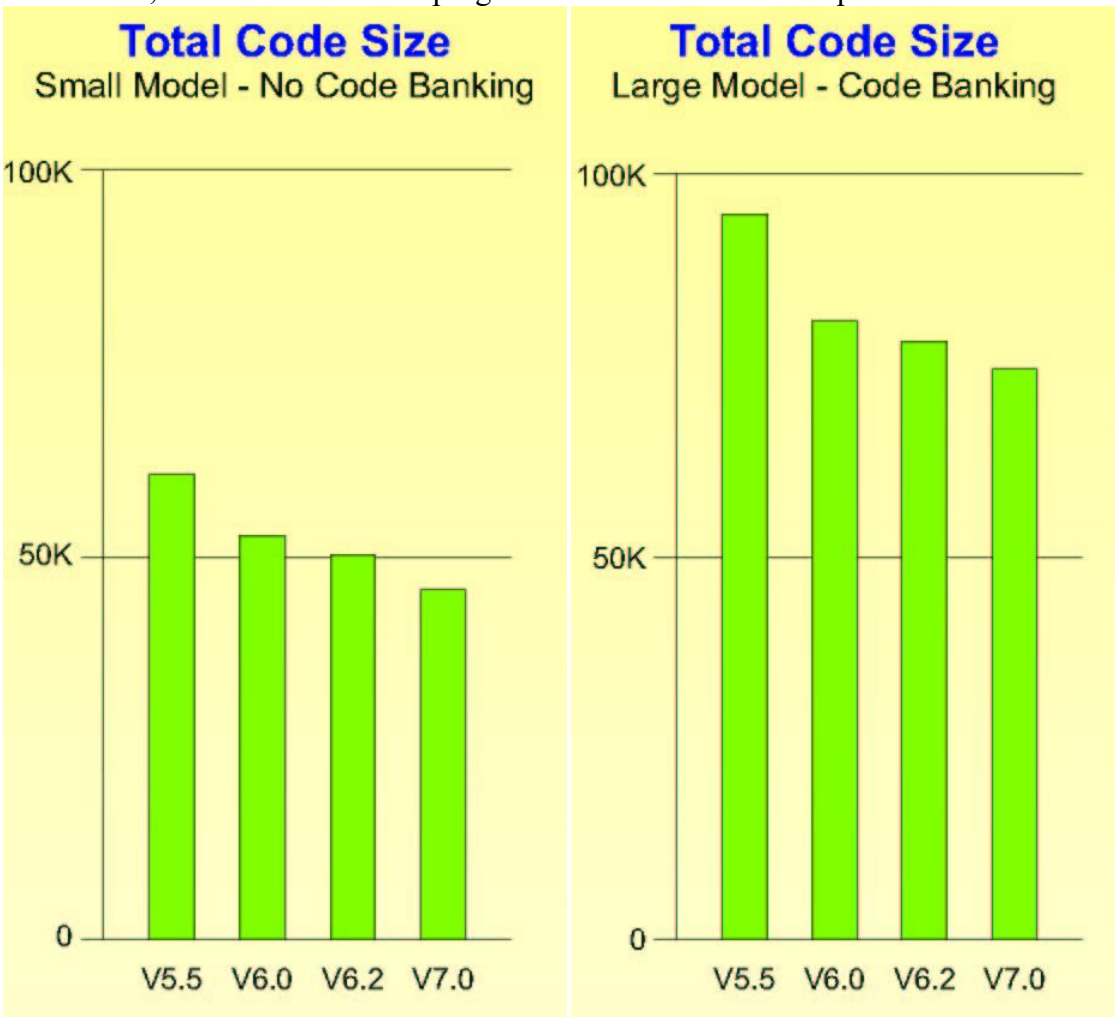
Hlavnou úlohou prekladača jazyka C je:

- Automaticky rozmiestňovať programový kód a dát,
- Mať otvorenú architektúru umožňujúcu využitie častí programového kódu, alebo knižníc napísaných v inom jazyku *(BASIC, PASCAL, C, FORTRAN, ASM ...)*,
- Umožňovať rôzne metódy adresovania pre iné CPU,
- Používať rôzne pamäťové modely[33] *(SMALL, COMPACT, LARGE ...)*, a štandardné údajové typy *(char, int, float, double, string, pointer ...)* a štruktúry,
- Riadiť a ladiť program na úrovni prerušenia.

Obrázok 30, Programovací model vývojového nástroja µVision4 a vyšší

[33] Vyššie programovacie jazyky generujú síce univerzálny programový kód, ale pri porovnaní s assembler-om je značne väčší „nafúknutý“ a samozrejme menej efektívny, ale dĺžka písania algoritmu programu je výrazne menšia.

Obrázok 31, Porovnanie veľkostí programu v závislosti na verzii prekladača C51



Veľkosť pamäťového priestoru použitá prekladačom C v jednotlivých pamäťových modeloch nie je pre programátora príliš podstatná, pretože je to pamäťový priestor nevyhnutne potrebný pre činnosť programu v závislosti na množstve použitých premenných. V prípade nedostatku pamäťového priestoru prekladač ohlási tento nedostatok pamäti programátorovi. Len v jednom prípade je potrebné dávať pozor na maximálne množstvo pridelenej pamäti a to vtedy ak používame reentrant[34] funkcie, pretože funkcie ktoré sú reentrantné používajú pamäť IDATA[35] mikroprocesora pre svoju činnosť. Takto sa môže veľmi ľahko stať, že niekoľkonásobné vyvolanie funkcie môže mať za následok vyčerpania prideleného

[34] Reentrantné funkcie používajú vlastný „Environment“ t.j. prostredie alokujúce oddelený pamäťový priestor pre každú súčasne viacnásobne volanú funkciu z hlavného programu, alebo z prerušenia.
[35] Oblasť pamäti RAM mikroprocesora 8051 v oblasti adries od 080h do 0FFh, ktorá je nepriamo adresovateľná pomocou registrov **R0** a **R1**, napr. ***mov a,@R0***.

pamäťového priestoru a tým k pádu[36] programu. Konkrétne sa jedná o pamäťový model COMPACT, ktorý sa tak často nepoužíva ako model, prípadne LARGE[37].

Operačný systém RTX51 Full vyžaduje nevyhnutne k svojej činnosti navyše externú pamäť dát XRAM o veľkosti minimálne 1kB plus pamäť pre použité premenné a úlohy. Pre RTX51 Tiny nie je vyžadovaná externá pamäť, stačí pamäť[38] RAM o veľkosti 256 Byte.

Podrobnejšie informácie o špecifikách a spôsobe tvorby programov v programovacom jazyku C v prostredí µVision nájdete (Hrubý, 2009).

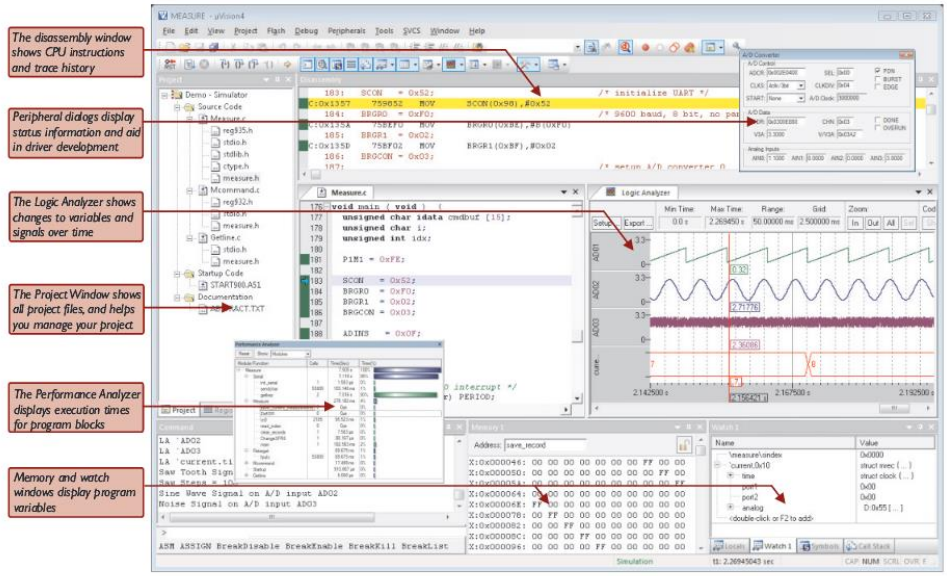
Obrázok 32, Vývoj aplikácie s Cx51 a RTX51

[36] Myslí sa tým zacyklenie programu

[37] Využíva oblasť pamäti RAM v rozsahu od 00h do 0FFh t.j. DATA+IDATA, 0000h do 0FFFFh t.j. XRAM, všetko v rozsahu 0 až 64kB.

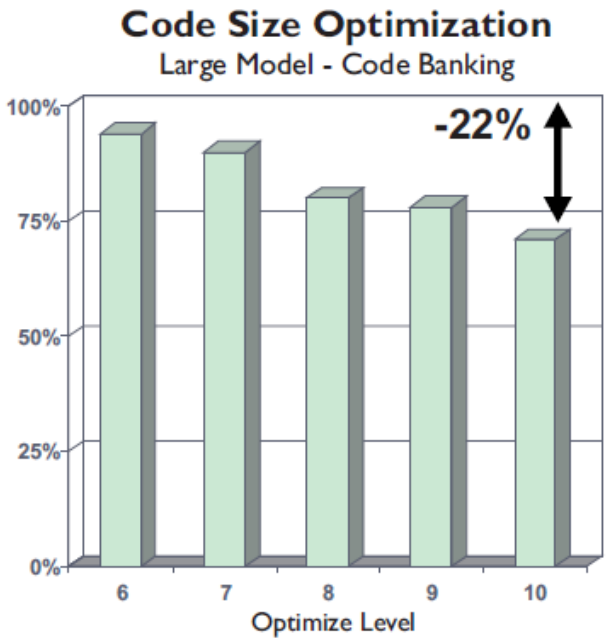
[38] Nastavenie je závislé na hodnote **RAMTOP** v súbore **conf_tny.a51**, kde hodnota **07Fh** je použiteľná pre mikroprocesory osadených maximálne 128 Byte internej pamäti DATA v rozsahu 00h až 07Fh. RTX51 Tiny správne funguje pokiaľ nedosiahne 90% obsadenosti pamäti DATA, IDATA t.j. RAM. V opačnom prípade to vedie k nekorektnej funkcii niektorých taskov, čo je pomerne veľmi ťažko v praxi identifikovať. V prípade potreby väčšieho množstva pamäti pre program je možné hodnotu **STACK** zmeniť na hodnotu 00h, čím zrušíme potrebnú pamäť pre zásobník a aplikácie dostanú k dispozícii väčšiu uvoľnenú pamäť na úkor zásobníka.

Obrázok 33, Vývoj aplikácie v prostredí µVision4



The µVision development platform is easy to use and it helps you to quickly create embedded programs that work.The µVision editor and debugger are integrated in a single application that provides a seamless embedded project development environment for editing, simulating, Flash programming and testing in target hardware.

Obrázok 34, Optimalizácia programového kódu kompilátorom Cx51



Obrázok 35, Optimalizačné stupne µVision

Level	Description
0	**Constant Folding**: The compiler performs calculations that reduce expressions to numeric constants, where possible. This includes calculations of run-time addresses. **Simple Access Optimizing**: The compiler optimizes access of internal data and bit addresses in the 8051 system. **Jump Optimizing**: The compiler always extends jumps to the final target. Jumps to jumps are eliminated.
1	**Dead Code Elimination**: Unused code fragments and artifacts are eliminated. **Jump Negation**: Conditional jumps are closely examined to see if they can be streamlined or eliminated by the inversion of the test logic.
2	**Data Overlaying**: Data and bit segments suitable for static overlay are identified and internally marked. The BL51 Linker/Locator has the capability, through global data flow analysis, of selecting segments which can then be overlaid.
3	**Peephole Optimizing**: Redundant MOV instructions are removed. This includes unnecessary loading of objects from the memory as well as load operations with constants. Complex operations are replaced by simple operations when memory space or execution time can be saved.
4	**Register Variables**: Automatic variables and function arguments are located in registers when possible. Reservation of data memory for these variables is omitted. **Extended Access Optimizing**: Variables from the IDATA, XDATA, PDATA and CODE areas are directly included in operations. Intermediate registers are frequently unnecessary. **Local Common Subexpression Elimination**: The compiler detects multiple uses of the same expression or subexpression. The result of the first expression is saved and reused when possible. Superfluous expression calculations are eliminated from the code. **Case/Switch Optimizing**: Code involving switch and case statements is optimized using jump tables or jump strings. **Simple Loop Optimizing**: Program loops that fill a memory range with a constant are converted and optimized.
5	**Global Common Subexpression Elimination**: Identical subexpressions within a function are calculated only once when possible. The intermediate result is stored in a register and reused.
6	**Loop Rotation**: Program loops are rotated if the resulting program code is faster and more efficient. The loop expression of **for** and **while** loops is evaluated once at the top of the loop and then at the bottom of the loop. This optimization generates more code but speeds up execution.
7	**Extended Index Access Optimizing**: DPTR is used for register variables where appropriate. Pointer and array access are optimized for both execution speed and code size.
8	**Common Tail Merging**: When there are multiple calls to a single function, some of the setup code can be reused, thereby reducing program size.
9	**Common Block Subroutines**: Recurring instruction sequences are detected and converted into subroutines. The Cx51 Compiler rearranges code to obtain larger recurring sequences.
10	**Rearrange Code (Linker Optimization)**: When detecting common block subroutines, code is rearranged to obtain larger recurring sequences.
11	**Reuse of Common Exit Code (Linker Optimization)**: Identical exit sequences are reused. This may reduce the size of common block subroutines even further. This optimization level generates the most compact program code possible.

4.3. Príklad zápisu programu v C51

Príklad: Napíšte program, ktorý vyšle na sériové rozhranie textový reťazec “Hello world !" s prenosovou rýchlosťou 1200b.s^{-1}, 1 štart a stop bit, 8 bit slovo, bez parity s 16MHz kryštálom.

Príklad zápisu programu v jazyku C51:

```
1   /*---------------------------------------------------------------------------
2   HELLO.C
3
4   Copyright 1995-2005 Keil Software, Inc.
5   ---------------------------------------------------------------------------*/
6
7   #include <REG52.H>                 /* special function register declarations    */
8                                      /* for the intended 8051 derivative          */
9
10  #include <stdio.h>                 /* prototype declarations for I/O functions */
11
12
13  #ifdef MONITOR51                          /* Debugging with Monitor-51 needs    */
14  char code reserve [3] _at_ 0x23;          /* space for serial interrupt if      */
15  #endif                                    /* Stop Exection with Serial Intr.    */
16                                            /* is enabled                         */
17
18
19  /*-----------------------------------------------
20  The main C function.   Program execution starts
21  here after stack initialization.
22  -----------------------------------------------*/
23  void main (void)
24  {
25
26  /*-----------------------------------------------
27  Setup the serial port for 1200 baud at 16MHz.
28  -----------------------------------------------*/
29  #ifndef MONITOR51
30      SCON  = 0x50;                 /* SCON: mode 1, 8-bit UART, enable rcvr       */
31      TMOD |= 0x20;                 /* TMOD: timer 1, mode 2, 8-bit reload         */
32      TH1   = 221;                  /* TH1:   reload value for 1200 baud @ 16MHz   */
33      TR1   = 1;                    /* TR1:  timer 1 run                           */
34      TI    = 1;                    /* TI:    set TI to send first char of UART    */
35  #endif
36
37  /*-----------------------------------------------
38  Note that an embedded program never exits (because
39  there is no operating system to return to).   It
40  must loop and execute forever.
41  -----------------------------------------------*/
42    while (1)
43    {
44      P1 ^= 0x01;                   /* Toggle P1.0 each time we print */
45      printf ("Hello World\n");     /* Print "Hello World" */
46    }
47  }
```

Výpočet hodnoty TH1 pre časovač v režime 1 pre vysoké a nízke rýchlosti:

$$b^{-s} = \frac{2^{SMOD}}{32} \cdot \frac{f_{osc}}{12 \cdot (256 - TH1)}$$

$$b^{-s} = \frac{2^{SMOD}}{32} \cdot \frac{f_{osc}}{12 \cdot (65536 - [TH1, TL1])}$$

$$TH1 = 256 - \frac{2^{SMOD} \cdot f_{osc}}{384 \cdot b^{-s}} \qquad (5.)$$

4.4. Deklarácia jednoduchých premenných v jazyku C

Programátor pri tvorbe algoritmu programu využíva rôzne premenné[39], alebo štruktúry na ukladanie údajov. Jazyk C obsahuje niekoľko jednoduchých základných celočíselných a reálnych premenných. Veľkosť pamäti ktorú zaberajú závisí od ich typu. Pre podrobnejšie informácie o jednotlivých typoch premenných, ich použití a spôsobe zápisu je možné nájsť v (Keringham, a iní, 1998), prípadne pre definovanú cieľovú platformu mikroprocesorov v publikácii (Burkhard, 2003).

4.4.1. Celočíselné premenné

```
char znak;
signed char sch;
unsigned char cislo;
int cele;
unsigned int cele;
long int dlhe;
```

4.4.2. Reálne premenné

```
float x;
double y;
```

4.4.3. Bitové premenné

```
bit Flag;
sbit LED = P1^0;
```

Premenná typu „bitové pole“ nie je v prípade kompilátora jazyka C51 implementovaná, lebo mikroprocesor 8051 obsahuje priamo vo svojom vnútri bitové oblasti ktoré môže programátor použiť pri tvorbe programu.

[39] Vyššie uvedené deklarácie typu premenných jazyka C sú použité z prekladača jazyka C51 nachádzajúceho sa v Keil µVision, takže sa môžu líšiť od premenných ktoré sú použité v iných prekladačoch.

Obrázok 36 ukazuje prípade použitia kompilátora pre procesory ARM napr. STM32F407 je možné s výhodou použiť pre prístup do bitovo orientovanej premennej ***union*** ako bližšie upresňuje.

Obrázok 36, Deklarácia premennej pre LCD s prístupom po bitoch

```
typedef union
{
  struct
    {
      uint8_t rs:1;     //RS[0] je to 7 bit !!!
      uint8_t rw:1;     //RW[1]
      uint8_t en:1;     //EN[2]
      uint8_t led:1;    //LED[3]
      uint8_t data:4;   //Data[4..7]
    }
  bit;
  uint8_t word;
}
LCD_4bit_type;
```

Obrázok 37, Použitie bitovej premennej LCD v programe

```
void lcd_send_statement(uint8_t data,LCD_4bit_type wire)
{
  static lcd_status_t lcd_ctrl;
  static LCD_4bit_type value;

  value.word=bitmask_one(data);
  value.word|=wire.word;

  value.bit.en=1;
  lcd_ctrl.data[0]=value.word;
  value.bit.en=0;
  lcd_ctrl.data[1]=value.word;

  value.word=bitmask_two(data);
  value.word|=wire.word;

  value.bit.en=1;
  lcd_ctrl.data[2]=value.word;
  value.bit.en=0;
  lcd_ctrl.data[3]=value.word;
```

Príklad deklarácie 16 bitovej premennej typu integer[40],

int x;

a deklarácie premennej typu ukazovateľ[41] na integer.

*int *x;*

4.5. Jednorozmerné a viacrozmerné polia

Spôsob zápisu deklarácie jednorozmerného poľa:

int x[1024];

code char x[1024];

xdata char x[1024];

xdata char x[]={0,1,2,3,4,5,6,7,8,9,10};

Deklarácia jednorozmerného poľa sa v jazyku C môže spojiť s identifikátorom prefixu segmentu ktorý určuje kde sa má deklarované pole uložiť. Prefix ako pojem môže byť uvažovaný ako modifikátor (Hrubý, 2009). Jazyk C pozná dva druhy prefixov: ***const*** *a* ***volatile***. Prvý prefix ***const*** zabráni možnej modifikácii objektu ktorému bol priradený v priebehu vykonávania programu, druhý prefix ***volatile*** hovorí kompilátoru, že obsah premennej môže byť zmenený napr. prerušením, alebo čítaním hodnôt z externých zariadení napr. RTC[42] a pri vykonávaní príkazov[43] v programe je potrebné s tým počítať. Prefix ***volatile*** je relatívne málo používaný. V prípade, že potrebujeme do poľa uložiť hodnoty ktoré sa nebudú počas vykonávania programu meniť, môžeme použiť prefix „code“, ktorý uloží pole do pamäti programu. Segment „xdata“ slúži na vytvorenie poľa v externej pamäti RAM[44] do ktorého je možné zapisovať a čítať podľa potreby.

Deklarácia viacrozmerného poľa:

int x[128][256];

code int x[128][256];

xdata int x[128][256];

[40] Do premennej **x** je uložená 16 bitová hodnota čísla

[41] Do premennej **x** je uložená len **adresa** premennej **x** uloženej v pamäti, tzv. ukazovateľ, alebo vo väčšine odbornej literatúry je uvádzaný len pojem **smerník**.

[42] Real Time Clock, napr. DS1307.

[43] Napr. príkazy cyklu for(), while(), alebo do {} while().

[44] Myslí sa XRAM v adresnom priestore od 0 do 64kB, kde sa prístup vykonáva pomocou adresných vodičov na porte P0 a P2 spoločne so signálmi ALE, /RD a /WR.

V prípade deklarácie viacrozmerného poľa si musíme uvedomiť, že nároky na pamäťový priestor ktoré zaberá pole vzrastie úmerne s počtom veľkosti[45] jednotlivých rozmerov.

4.6. Deklarácia zložitejších pamäťových štruktúr v C (statická a dynamická alokácia)

Pod pojmom zložitejšie pamäťové štruktúry rozumieme deklarovanie viacprvkovej heterogénnej množiny rôznych typov premenných, pričom sú zvonku viditeľné ako jednoliaty samostatný celok. Zložitejšie deklarácie uvádza v kapitole 1.17.2 (Hrubý, 2009).

Obrázok 38, Deklarácia štruktúry

```
typedef xdata struct
{
 unsigned char Preamble,Dst,Src;
 unsigned int Size;
 unsigned char Data[SizeOfPacketData];
 unsigned char Crc;
}
Packet51;
```

Obrázok 39, Deklarácia a použitie union-u

```
union
{
 char addr[2];
 int Addr;
}
Register;

idata Register REG;

REG.addr[0]=0x12;          //Prístup do pamäťovej bunky union-u ako 2x byte
REG.addr[1]=0x34
.....
REG.Addr=0x1234;           //Prístup do pamäťovej bunky union-u ako 1x int
```

Pri niektorých prekladačoch[46] sa môžeme stretnúť so štruktúrami a atribútom určujúcim veľkosť premennej v bitoch.

Obrázok 40, Deklarácia pamäťovej štruktúry s určením veľkosti

[45] V prípade umiestnenia poľa, alebo štruktúry do dynamickej pamäti nemôže veľkosť prekročiť 7,5 kB. Je to obmedzenie kompilátora jazyka C v Keil µVision5. Celkovo nesmie prekročiť 64 kB pamäti RAM, alebo CODE. U iných kompilátorov sa môže situácia odlišovať, jazyk C v princípe obmedzenia veľkosti pamäti nepozná.

[46] Vývojové prostredie MDK-ARM umožňuje navyše deklarovať pamäťovú štruktúru s atribútom ***packed***, umožňujúcim do jednej premennej (štruktúry) uložiť väčší počet menších premenných ktorých spoločná veľkosť je rovná súčtu údajového typu použitej premennej, ale vo všeobecnosti platí, že veľkosť union-u je rovná veľkosti typu najväčšej deklarovanej premennej vo vnútri union-u.

```
typedef xdata struct
{
 unsigned int  z:10;          //Len 10 bitov
 unsigned int  x:8;           //Len 8 bitov
 unsigned char y:2;           //Len 2 bity
}
ADC_Result;
```

Prípadne zápis premennej ADCON a ADCH uchovávajúcu výsledok AD prevodu v C:

```
typedef union
{
        struct
        {
                unsigned char _ADCH:8;    //8..15 bit     Value = 0x0FF..0x3F0
                unsigned char _AADR:3;    //0..2 bit      Value = 0..7 a ich kombinacie
                unsigned char _ADCS:1;    //3..3 bit      Value = 0..1
                unsigned char _ADCI:1;    //4..4 bit      Value = 0..1
                unsigned char _ADEX:1;    //5..5 bit      Value = 0..1
                unsigned char _ADCX:2;    //6..7 bit      Value = 0..3
        }
        b;
        unsigned int word;
}
ADC_t;
```

4.6.1. Statická alokácia premenných v jazyku C

Priradenie štruktúry „Packet51“ premennej „packet“ zabezpečíme v jazyku C pomocou nasledujúceho zápisu:

```
xdata Packet51 packet;
```

4.6.2. Dynamická alokácia premenných v jazyku C

Dynamické alokovanie pamäti sa v jazyku C používa v prípadoch, keď je potrebné dynamicky prideľovať podľa skutočných potrieb množstvo pamäti, ktorá nie je vopred známa.

Na zabezpečenie dostatočného množstva pamäte (alokovanie) sa v programovacom jazyku C používa funkcia ***malloc_mempool[]*** a funkcia ***init_mempool()***[47] s funkciou ***malloc()***[48], ktorá inicializuje množstvo pamäti, alebo alokovanú pamäť pre potreby programátora. Obrázok 41 názorne zobrazuje rozmiestnenie jednotlivých segmentov v BL51 v pamäťovom priestore aplikácie.

V prípade dynamickej alokácie premennej „Recv_Message“ štruktúry „Packet51“ je potrebné použiť nižšie uvedený postup:

a. Vytvoriť v pamäti **xdata**[49] dostatočný priestor pre dynamicky prideľovanú pamäť:
   unsigned char xdata malloc_mempool[2(sizeof(Packet51)+5)];*
b. inicializácia celého pamäťového bloku pamäti **xdata** pomocou funkcie jazyka C:
   init_mempool(&malloc_mempool,sizeof(malloc_mempool));
c. priradenie statickej štruktúry „Packet51“ na štruktúru s názvom „*Recv_Message“:
   *xdata Packet51 *Recv_Message;*
d. inicializácia premennej „Recv_Message“, ktorá obsahuje adresu prideleného pamäťového priestoru:
   Recv_Message=(Packet51)malloc(sizeof(Packet51)));*
e. naplnenie jednotlivých položiek štruktúry údajmi:

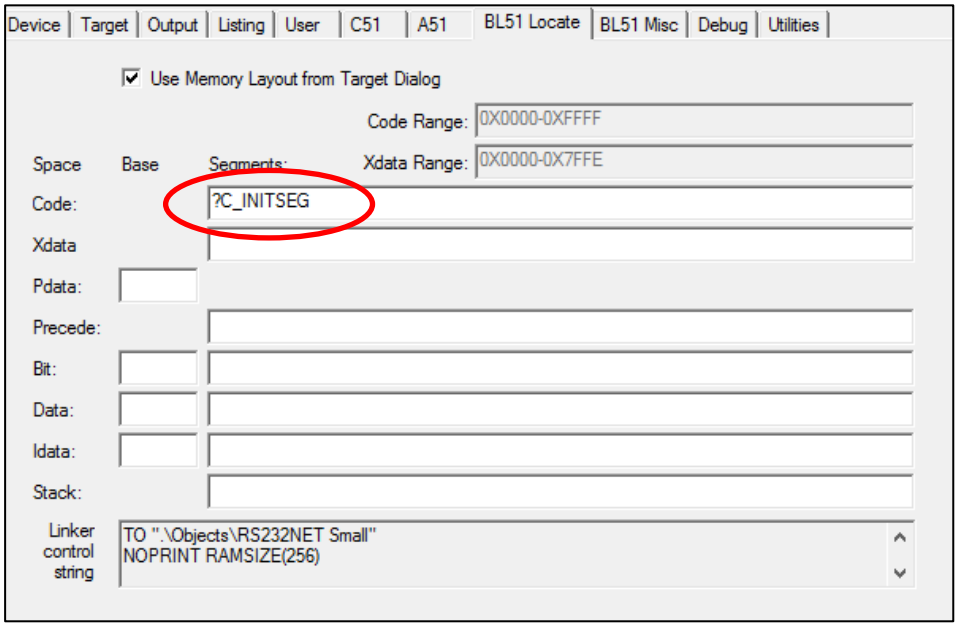
```
Recv_Message->Preamble='^';
Recv_Message->Dst=0xAB;
Recv_Message->Dst=0xCD;
...
...
...
```

[47] Pri použití tejto funkcie hlási prekladač chybu WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS, SEGMENT: ?C_INITSEG. Na odstránenie vyššie uvedeného chybového hlásenia je potrebné v menu **Options for Target** -> **BL51 Locate** -> v položke **Code** nastaviť **?C_INITSEG**. Po tejto úprave už nebude prekladač hlásiť chybu a preklad programu bude dokončený bez chybového hlásenia. Chybové hlásenie bolo zaznamenané pri Keil µVision 9.56 a je možné, že v predchádzajúcich verziách sa rovnako tento stav vyskytuje. Direktívu **?C_INITSEG** je potrebné nastaviť len vtedy ak je potrebné v prostredí operačného systému RTX51 Full ešte použiť navyše alokáciu pamäti pre dáta pomocou ***init_mempool()***!!!

[48] V prípade nedostatku pamäte pre alokovanie premennej je funkciou ***malloc()*** vrátená hodnota ***NULL***.

[49] Programátor musí vytvoriť potrebný priestor v dynamickej pamäti pre údaje a zároveň pre tzv. popisovač bloku, ktorého veľkosť môže závisieť od pamäťového modelu mikroprocesora a použitého prekladača. Celá požadovaná dostupná pamäť musí byť vytvorená v externej pamäti programu. Programátor túto dostupnú pamäť musí inicializovať v súbore **init.a51** projektu ktorý potom obsahuje kompletný pamäťový management dynamicky alokovanej pamäte pre aplikáciu. Pri prideľovaní (alokovaní) pamäte pomocou funkcie ***init_mempool()*** je potrebné blok pamäti zvýšiť navyše ešte o tzv. popisovač, ktorý je v prípade použitia prekladača C51 o 5 Byte. Pri alokácii je teda potrebné ku každému alokovanému bloku pamäti pripočítať **5 Byte** a navýšiť tak požadovanú pamäť pre dynamické premenné.

Obrázok 41, Nastavenie BL51 Locate s ?C_INIT SEG



V prípade, že potrebujeme v programovacom jazyku C alokovať externý pamäťový priestor na absolútnej adrese 0000h pomocou funkcie ***init_mempool()*** o veľkosti 4096 Byte, stačí len správne doplniť inicializáciu poľa ***malloc_mempool()*** pomocou direktívy *_at_* s adresou umiestnenia[50] v pamäti ako upresňuje.

Použitý ukazovateľ bol deklarovaný ako netypizovaný pomocou konštrukcie *xdata void *p*, pričom môže byť ekvivalentne použitý aj typizovaný ukazovateľ *xdata char *p*. Prístup do takéhoto poľa po alokácii pomocou funkcie ***malloc()*** je možné jednoducho pomocou *((char xdata*)p)[i]=i*. Funkcia ***malloc()*** vracia adresu umiestnenia v pamäti. Uvoľnenie pamäti je možné pomocou funkcie ***free()***.
Obrázok 42 chronologicky zobrazuje celý proces alokovania externej pamäti v programovacom jazyku C.

Obrázok 42, Alokovanie externej pamäti v C a s prístupom netypizovaného ukazovateľa

[50] Umiestnenie poľa na absolútnej adrese má význam len v prípade ladenia programu, kde je potrebné sledovať obsah poľa na pevnej adrese. Po ukončení ladenia programu sa môže direktíva *_at_* zrušiť a prekladač si umiestni pole podľa potreby.

```
unsigned char i;
xdata void *p;
unsigned char xdata malloc_mempool [4096] _at_ 0x0000;

void main(void)
{
 init_mempool (&malloc_mempool, sizeof(malloc_mempool));
 while(1)
  {
   p = malloc (100);
   for (i = 0; i < 100; i++)
    {
     ((char *) p)[i] = i;
     ((char xdata*)p)[i]=i;
    }
   free (p);
  }
}
```

Alokácia[51] externej pamäti v jazyku C a samotný princíp prístupu do pamäti je prevzatý s malými úpravami z existujúceho príkladu ktorý je súčasťou distribúcie programu Keil µVision.

Pamäťové oblasti sú bloky s pevnou veľkosťou pamäte, ktoré sú bezpečné pre vlákna. Fungujú oveľa rýchlejšie ako dynamicky pridelená „hromada“ HEAP a hlavne netrpia fragmentáciou. Keďže ich použitie je bezpečnejšie pre „threads” vlákna, môžu sa použiť vlákna aj v spojení s ISR[52].

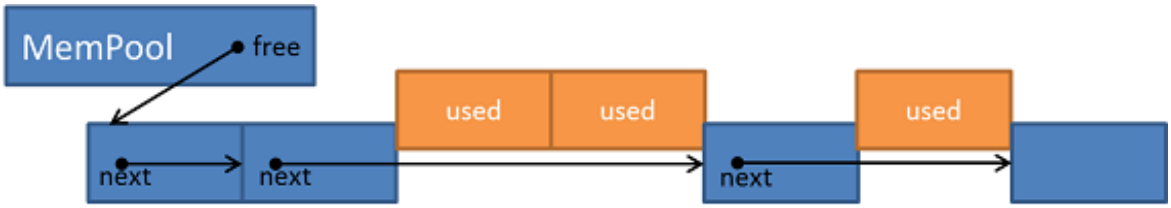
Na pamäťovú oblasť “pool” sa dá pozerať ako na vzájomne prepojený zoznam dostupných (nepoužitých) pamäťových blokov s pevnou a rovnakou veľkosťou. Pridelenie pamäte z „pool-u“ (pomocou funkcie *osMemoryPoolAlloc*) jednoducho zruší blok zo zoznamu a odovzdá používateľovi. Uvoľnenie pamäte[53] pre „pool“ (pomocou funkcie *osMemoryPoolFree*) jednoducho znovu naformátuje blok do zoznamu.

[51] Pozor na použitie danej funkcie **init_mempool()** spolu s operačným systémom RTX51 Full. Odporúčame použiť pre vytvorenie dynamickej oblasti pamäte pre premenné radšej funkciu **os_create_pool()**. Pre pridelenie a uvoľnenie pamäti musíme použiť zodpovedajúce funkcie **os_get_block()** a **os_free_block()**. V opačnom prípade môže dôjsť k prepísaniu už predtým pridelenej pamäti funkciou **init_mempool()**.

[52] Interrupt Service Routine

[53] Do uvoľneného bloku sa už nesmie zapisovať bez opätovnej alokácie pomocou funkcie *osMemoryPoolAlloc()*.V opačnom prípade hrozí poškodenie riadiacej štruktúry prepojených ukazovateľov zoznamu pamäťových blokov.

Obrázok 43, Model štruktúry MemoryPools v CMSIS



4.7. Pridelenie pamäte

Objekty RTX5 (thread, mutex, semaphore, timer, message queue, thread, event flags, memory pool) vyžadujú vyhradenú pamäť RAM. Objekty môžu byť vytvorené pomocou volania funkcie ***osObjectNew()*** a odstránené pomocou volania ***osObjectDelete()***. Pamäť súvisiacich objektov musí byť k dispozícii počas životnosti objektu.

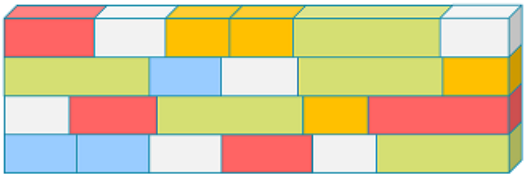
RTX5 ponúka tri rôzne spôsoby alokácie pamäte pre objekty:

- **Globálna pamäťová oblasť** používa jednu globálnu pamäťovú oblasť pre všetky objekty. Je ľahké ju nakonfigurovať, ale môže mať nevýhodu fragmentácie pamäte, keď sa vytvárajú a ničia objekty s rôznymi veľkosťami.
- **Špecifické pamäťové oblasti** pre objekt používajú oblasť pamäte pevnej veľkosti pre každý typ objektu. Metóda je časovo deterministická a zabraňuje fragmentácii pamäte.
- **Staticky prideľovaná pamäť** vyhradzuje pamäť počas kompilácie a úplne sa vyhýba tomu, že systém môže mať nedostatok pamäte. To sa zvyčajne vyžaduje pre niektoré systémy dôležité z hľadiska bezpečnosti.

V tej istej aplikácii je možné použiť všetky metódy prideľovania pamäte.

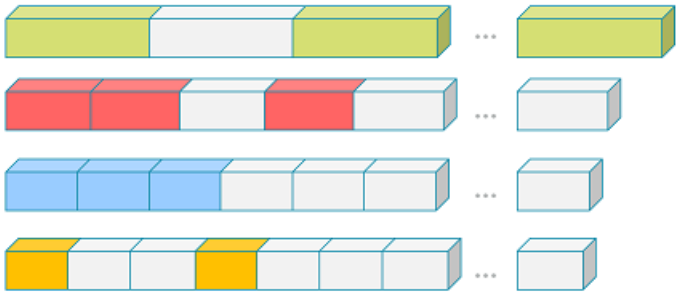
Globálna pamäťová oblasť prideľuje všetky objekty z oblasti pamäte. Táto metóda alokácie pamäte je predvolená vRTX5.

Obrázok 44, Znázornenie globálnej pamäťovej oblasti v RTOS



Ak zvolená pamäťová oblasť neposkytuje dostatok pamäte, vytvorenie objektu zlyhá, funkcia ***osObjectNew()*** vráti výsledok funkcie s hodnotou NULL.

Obrázok 45, Špecifická pamäťová oblasť v RTOS



Existuje tu len jedna pamäťová oblasť pre každý typ objektu.

Špecifická pamäťová oblasť nevytvára fragmentáciu pamäte s vyhradenou správou pamäte pre každý typ objektu. Tento typ pamäťových oblastí je úplne deterministický z časového hľadiska, čo znamená, že vytvorenie a zrušenie objektu vyžaduje vždy rovnaký čas. Pretože oblasť pamäte s pevnou veľkosťou je špecifická pre typ objektu, spracovanie situácií s nedostatkom pamäte je menej komplikované.

Spoločne využívané pamäťové oblasti sú selektívne povolené pre každý typ objektu, napr.: mutex, alebo vlákno môžeme nastaviť pomocou konfiguračného súboru RTX_Config.h. Keď pamäťová oblasť neposkytuje dostatok pamäte, vytvorenie objektu zlyhá a súvisiaca funkcia ***osObjectNew()*** vráti výsledok funkcie NULL.

- Povoliť v **Thread Configuration** pre thread objekty.
- Povoliť v **Timer Configuration** pre timer objekty.
- Povoliť v **Event Flags Configuration** pre event objekty.
- Povoliť v **Mutex Configuration** pre mutex objekty.
- Povoliť v **Semaphore Configuration** pre semafóry.
- Povoliť v **Memory Pool Configuration** pre memory pools oblasti.
- Povoliť v **Message Queue Configuration** pre messages objekty.

Staticky prideľovaná pamäť na rozdiel od dynamického priradenia pamäte vyžaduje pridelenie statickej pamäte priradenie objektovej pamäte už počas kompilácie programu.

Obrázok 46, Schéma staticky prideľovanej pamäte v RTOS

Existuje tu len staticky pridelená pamäť pre všetky objekty. Statické pridelenie pamäte sa dá dosiahnuť len pomocou manuálneho nastavenia vlastností užívateľsky definovanej pamäte pomocou atribútov v čase vytvárania objektu.

Obrázok 47, Alokoácia pamäti pomocou funkcie osMemoryPoolAlloc() s CMSIS

```
__NO_RETURN void dht_dht11_app(void *argument)                                  //Pri
optimalizacnom stupni O3 tato rutina nefunguje !!!
{
  (void)argument;
  uint8_t Count;
  osStatus_t status;
  struct dht_device dht11;
  TM_DHT11_Data_t dht11_data;
  TM_DHT11_t rslt;
  DHT_msg_t *msg;
  TM_DHT11_Init();
  //while((TM_DHT11_Init()!=TM_DHT11_OK))  osDelay(TwoSecond);                  //Initialize
DHT11 -> PD1, nekontroluje pritomnost DHT11 na porte !!!
  do
  {
   switch(Count)
    {
      default             : Count++;
                            rslt=TM_DHT11_Read(&dht11_data);
                            break;
      case Number_Retries : Count=Number_Retries;
                            #ifdef TERMINATE
                            while(osThreadTerminate(osThreadGetId())!=osOK) osDelay(OneSecond);
                            #endif
                            break;
    }
   osDelay(TwoSecond);
  }
  while(rslt!=TM_DHT11_OK);
  DHT11_msgQueue=osMessageQueueNew(DHT_POOL_OBJECTS,sizeof(DHT_msg_t*),NULL); // Message queue
used to pass pointers to msg_t
  DHT11_memPool=osMemoryPoolNew(DHT_POOL_OBJECTS,sizeof(DHT_msg_t),NULL);     // Memory pool
for actual message objects
  memcpy(&dht11.dht,&DHT_Init,sizeof(DHT_msg_t));                             //Inicializujem
DHT stukturu z tabulky ....

  for(;;)
  {
   if(TM_DHT11_Read(&dht11_data)==TM_DHT11_OK)
    {
     dht11.dht.temperature=(float32_t)dht11_data.Temp/10;
     dht11.dht.humidity=(float32_t)dht11_data.Hum/10;
     dht11.read=&DHT_Compute_Propiertys;
     dht11_read(&dht11);                                                      //Tymto naplnim len
strukturu dht11 a o sebe nevedia ...
     msg=(DHT_msg_t*)osMemoryPoolAlloc(DHT11_memPool,osWaitForever);          //Alokujem pamat pre
odosielanu spravu ...
     if(msg != NULL) memcpy(msg,&dht11.dht,sizeof(DHT_msg_t));                //Pridelilo mi pamat ?,
ak ano tak kopirujem data na odoslanie ...
     status=osMessageQueuePut(DHT11_msgQueue,&msg,0U,osWaitForever);          //Odosielam spravu na
msgQueue ...
     while(status != osOK) Error_Signal();                                    //Bolo to OK, alebo
chyba ...
     /* Print something on USART1 */
     #ifdef SERIAL
     if(TM_USART_BufferEmpty(USART1))
      {
       sprintf(str,"t=%+2.1f°C\n\rh=%2.1f%%\n\r",dht11.dht.temperature,dht11.dht.humidity);
       osMutexAcquire(mutex_usart1_id,osWaitForever);
       TM_USART_Puts(USART1,str);
       osMutexRelease(mutex_usart1_id);
      }
     #endif
    }
   osDelay(TwoSecond);                                                        //Meriam kazde 2
sekundy, castejsie je uplne zbytocne ....
  }
}
```

4.8. Práca so zložitejšími pamäťovými štruktúrami v jazyku C a funkcií RTX51

Pri programovaní aplikácií pre mikroprocesory sa niekedy nevyhneme použitiu zložitejších pamäťových premenných, alebo pamäťových štruktúr. Ich použitie umožní programátorovi lepšie a efektívnejšie vytvárať časti programového kódu a jednoduchšie pristupovať k obsahom premenných. Jednoduché premenné jazyka C a ich deklarácia sú uvedené v každej učebnici jazyka C napr. (Keringham, a iní, 1998).

Obrázok 48, Deklarácia jednoduchej štruktúry v C

```
typedef xdata struct  //Musi byt deklarovana a pouzivana ako je to uvedene v tomto tvare, inac spravne nefunguje !!!
{
 unsigned char Preamble,Dst,Src;
 unsigned int Size;
 unsigned char Data[SizeOfPacketData];
 unsigned char Crc;
}
Packet51;

xdata Packet51 *ReceivedMessage;
xdata Packet51 Packet = {Token,1,2,sizeof(Packet.Data),"!!! Diversant Software !!!",NULL};
xdata unsigned int *MessageToSend, *ReceivedToBuffer, *MTS;

xdata Packet51 AAA;

void System(void) _task_ SYSTEM _priority_ 0
{
 signed char RtxCompletion;
 unsigned char fcs;
 unsigned int *ptr;
 RtxCompletion=os_set_slice(15360);
 RtxCompletion=os_create_pool(SizeMemBlock,MemoryPool,SizeMemPool); //Vytvorim MemoryPool v pamati XRAM pre datove struktu
 RtxCompletion=os_create_task(SEND);
 while(1)
  {
   AAA=Packet;
   Vytvor_Packet((char)rand(),(char)rand(),"!!! Diversant Software !!!",&AAA);
```

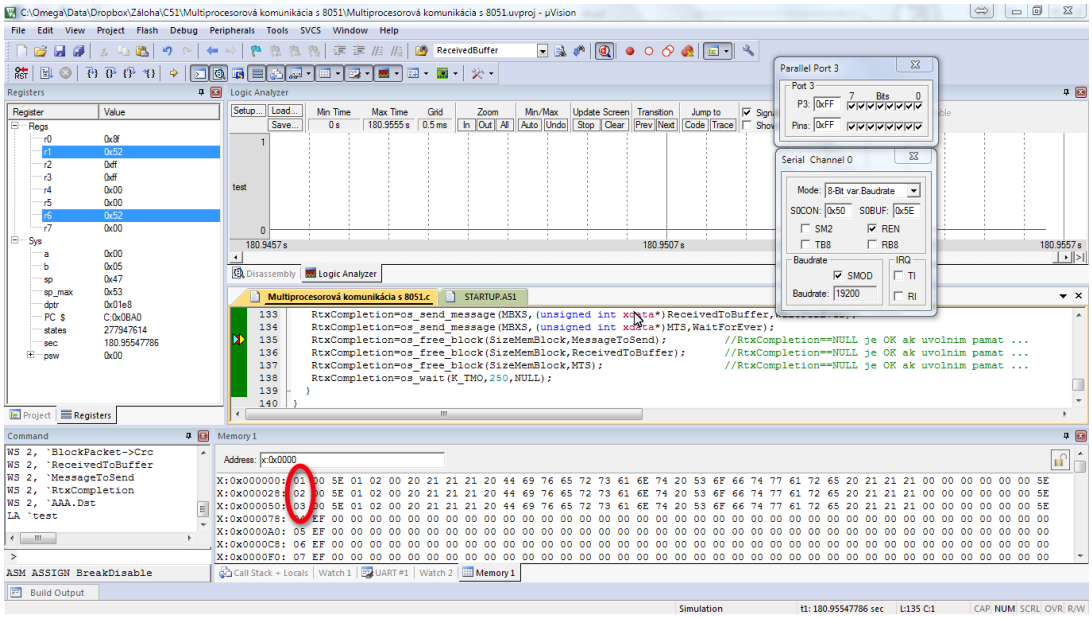
Obrázok 48 názorne ukazuje vytvorenie štruktúry **Packet51** ako pole rôznych údajových typov a jej následné priradenie pre premennú[54] **Packet** a **AAA**. Tieto dve premenné sú absolútne rovnocenné. S výhodou je ich použiť na prvotnú inicializáciu napr. **AAA=Packet**, kde sa štruktúra **AAA** naplní obsahom štruktúry **Packet** bez použitia funkcie **memcpy()**. Prístup do štruktúry je možné zápisu **Packet.Dst**, prípadne pre prístup do poľa **Packet.Data[i]** použije tento typ zápisu. V prípade ak poznáme len adresu pamäťovej premennej odoslanej mailbox-om ako údajový typ **int (unsigned int xdata*)&ReceivedMessage)** môžeme použiť zápis pre prístup do premennej **ReceivedMessage->Dst**, alebo **ReceivedMessage->Data[i]**. Premenná **ReceivedMessage** musí ukazovať svojím obsahom na adresu pamäťového bloku[55] ktorý je

[54] Takto definovaná premenná je umiestnená prekladačom staticky do externej pamäti **xdata** a nie je vôbec potrebné programátorom rezervovať miesto pre danú pamäťovú štruktúru. Pri nedostatku pamäti pre premennú prekladač hlási štandardnú chybu.

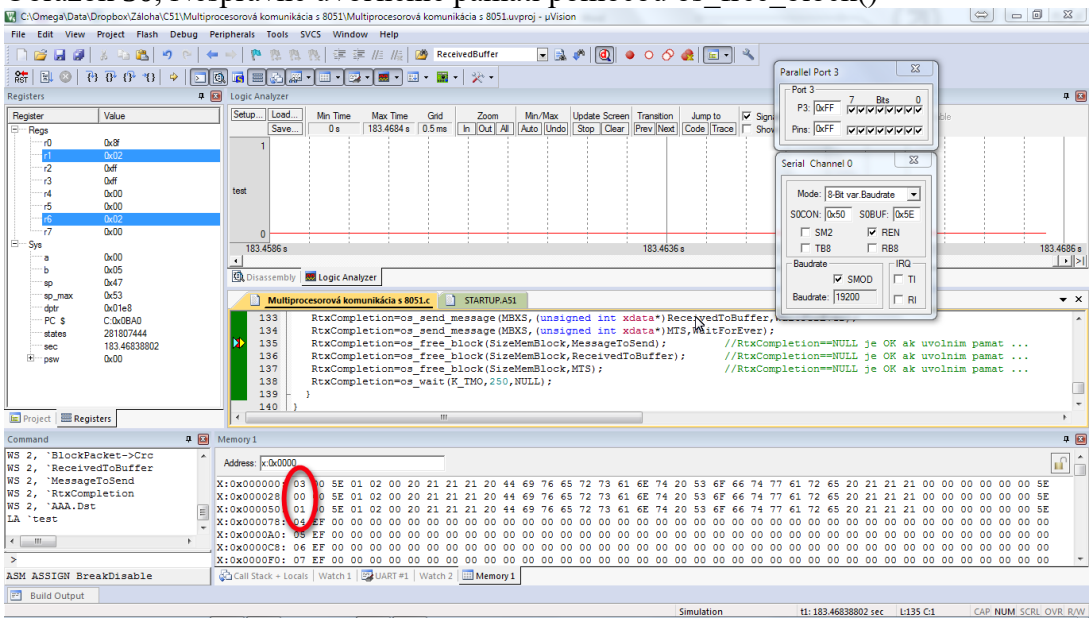
[55] Použitie dynamickej pamäti pri použití operačného systému RTX51 Full je možné len po použití funkcie os_create_pool(SizeMemBlock,MemoryPool,SizeMemPool). Pri správnom alokovaní pamäťových blokov pomocou funkcie MessageToSend=os_get_block(SizeMemBlock) je potrebné postupovať rovnako ako pri plnení

rezervovaný pomocou funkcie **MessageToSend=os_get_block(SizeMemBlock)** a po ukončení práce so štruktúrou je potrebné rezervovanú pamäť uvoľniť pomocou funkcie **os_free_block(SizeMemBlock,MessageToSend)**. Premenná **Packet** môže po vytvorení obsahovať náhodné hodnoty, preto musíme jej obsah inicializovať požadovanou hodnotou pomocou funkcie **memset(&Packet,0x00,sizeof(Packet))**.

Obrázok 49, Správne uvoľnenie pamäti pomocou funkcie os_free_block()



Obrázok 50, Nesprávne uvoľnenie pamäti pomocou os_free_block()



zásobníka procesoru pomocou inštrukcií 8051 PUSH a POP. Pri nedodržaní tohto postupu pri odlaďovaní programu je vidieť v samotnom header-i t.j. hlavičky alokovaného bloku pamäti po použití RtxCompletion=os_free_block(SizeMemBlock,MessageToSend) vo výpise pamäti „nesprávne“ priraďovanie poradových čísel alokovaných blokov pamätí, ktoré bolo zistené pri verzii Keil µVision 9.54.

4.9. Práca so zložitejšími štruktúrami v jazyku C pre Arduino

Vývojové prostredie použiteľné pre Arduino[56] umožňuje písať programy bez dostatočnej možnosti kontroly programového kódu debugger-om. Ak si uvedomíme, že je to voľne šíriteľný t.j. free kompilátor, tak jeho možnosti síce postačujú pre jeho širokú užívateľskú skupinu, ale výsledný programový kód nemá užívateľ ako odladiť.

```
#define SizeOfPacketData  100
#define Position_Crc (sizeof(Pkt)-1)
#define Time  25
#define LED 13

typedef struct
{
 long int Dst,Src;
 char Typ;
 char Data[SizeOfPacketData];
 long int Crc;
}
Pkt;

Pkt *Addr;

void setup(void)
{
 pinMode(LED, OUTPUT);
 Serial.begin(9600);
}

long int Checksum(char *Buffer)
{
  unsigned int i;
  Buffer[Position_Crc]=0x00;
  for(i=0x01; i<Position_Crc; i++) Buffer[Position_Crc]+=Buffer[i];
  return(~Buffer[Position_Crc]+1);
}

void loop(void)
{
 digitalWrite(LED, HIGH);
 delay(Time);
 digitalWrite(LED, LOW);
 delay(Time);
 if((Addr=(Pkt*)malloc(sizeof(Pkt)))==NULL) while(1); //Alokujem vysielaciu pamat
 Addr->Dst=0x01234567;
 Addr->Src=0x89ABCDEF;
 Addr->Typ=0x06;
 Addr->Crc=Checksum((char*)&Addr);
 memset((char*)&Addr->Dst,'x',sizeof(Addr->Dst));
memset((char*)&Addr->Src,'y',sizeof(Addr->Src));
 memset((char*)&Addr->Typ,'t',sizeof(Addr->Typ));
memset((char*)&Addr->Data,'d',sizeof(Addr->Data));
 memset((char*)&Addr->Crc,'c',sizeof(Addr->Crc));
 delay(Time*10);
 Serial.write((char*)Addr,sizeof(Pkt));
 Serial.println();
 free(Addr);                                                    //Uvolnim
vysielaciu pamat
}
```

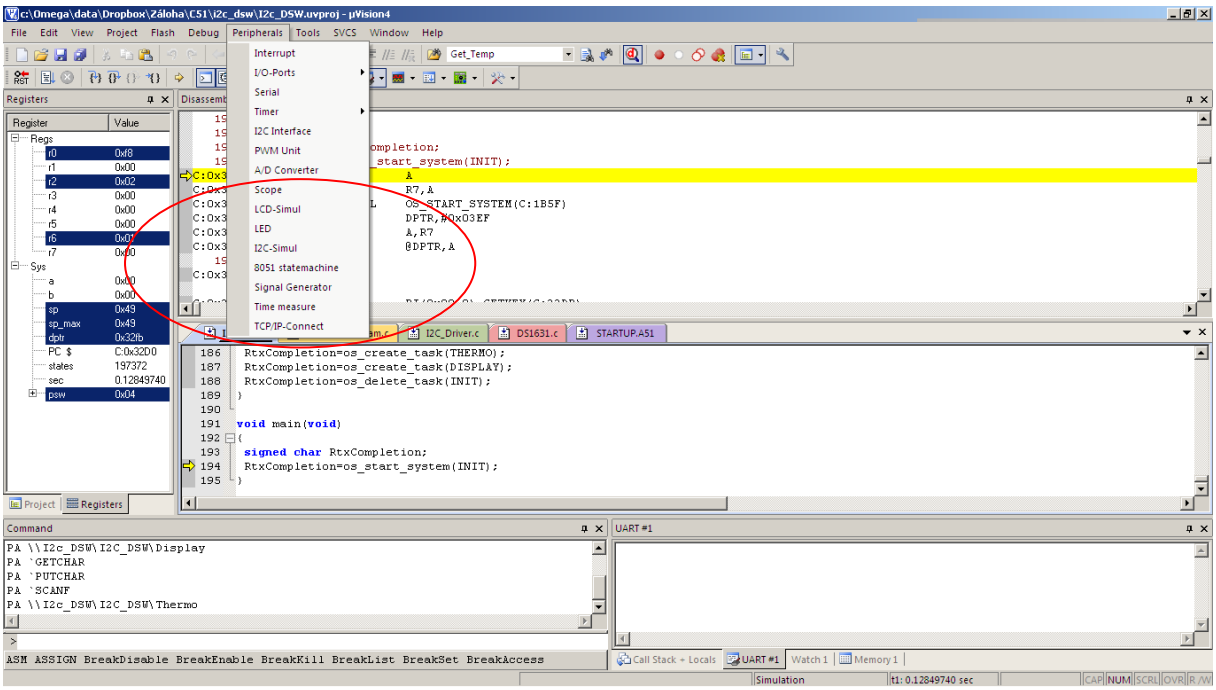
Kompilátor jazyka C pre platformu Arduino je špecifický v prípade ak sa používajú typizované údajové štruktúry a premenné. Pre porovnanie uvádzame rovnakú deklaráciu štruktúry **Pkt** ako v predchádzajúcich príkladoch a zároveň deklaráciu pre Arduino. Čitateľ si na prvý pohľad všimne rozdiely[57] v samotnej deklarácii a pri použití premennej.

[56] Bližšie informácie na www.arduino.cc
[57] Striktná definícia premennej, typecast ...

4.10. Ladenie – simulácia aplikácií pomocou prostredia µVision

Moderné mikroprocesory na svojom jadre obsahujú množstvo periférnych obvodov, ktoré programátor len s veľkými ťažkosťami dokáže odladiť – odsimulovať na PC a v konkrétnej aplikácií. Na zjednodušenie tejto náročnej činnosti bolo programátormi tretích strán naprogramovaných množstvo aplikácií[58] ktoré umožňovali jednoduchým spôsobom simulovať funkciu niektorých obvodov mikroprocesora priamo v prostredí µVision.

Obrázok 51, Modifikované prostredie µVison4 pre ladenie aplikácií s AGSI[59] ovládačmi



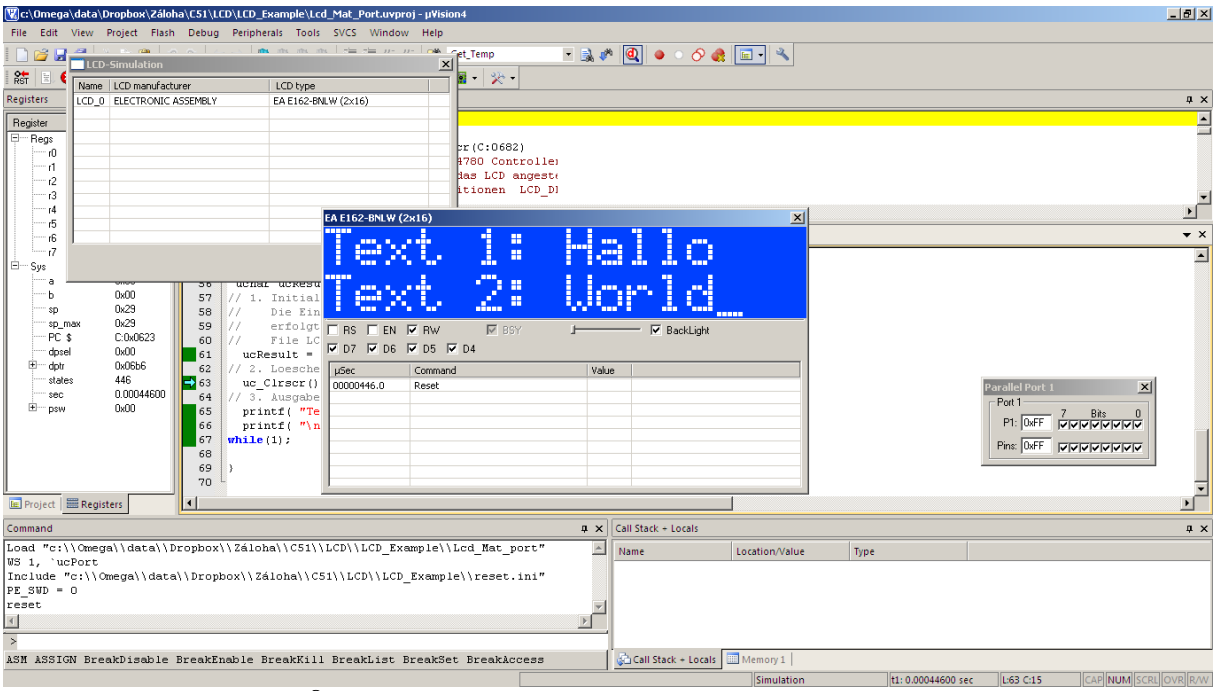
Pre správnu činnosť je potrebné modifikovať priečinok **C:\KEIL** kde je uložený súbor ***tools.ini*** s inicializačnými rutinami potrebných pre začlenenie AGSI ovládačov do systému.

Inštalácia AGSI ovládačov v TOOLS.INI

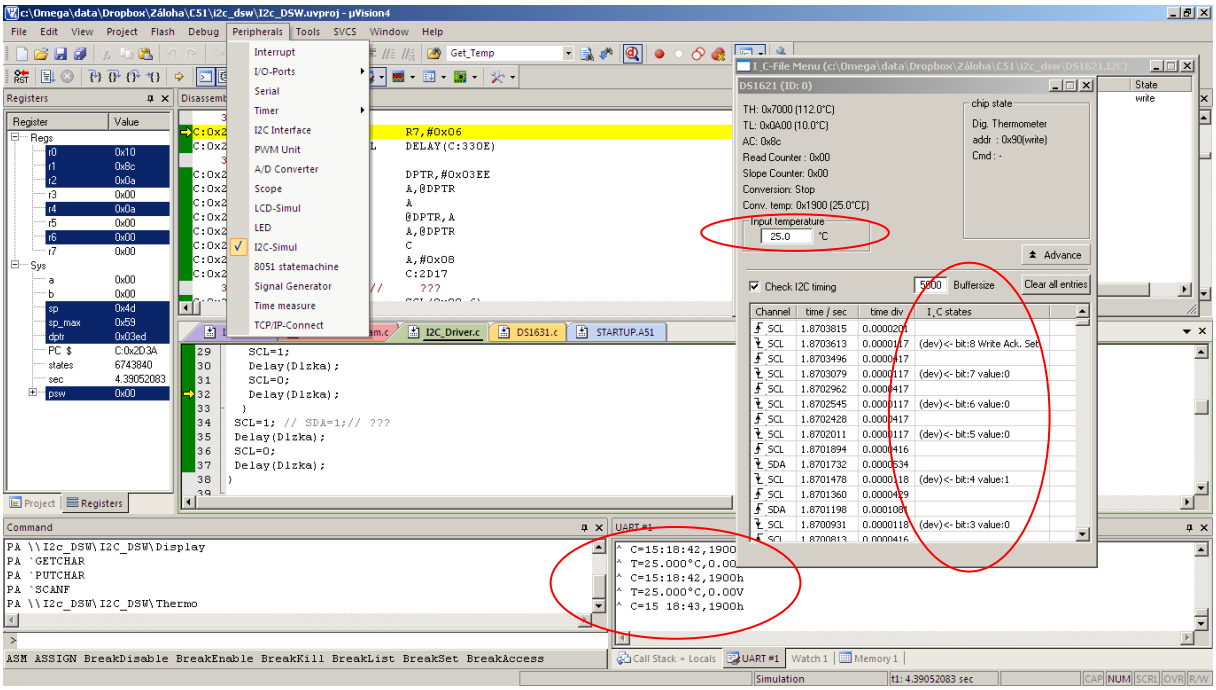
```
[C51]

PATH="C:\Keil\C51\"
VERSION=V9.54a
AGSI1=SCOPE.DLL ("Scope simulation")
AGSI2=LCD.DLL ("LCD simulation")
AGSI2=LED.DLL ("LED simulation")
AGSI3=I2C.DLL ("I2C simulation")
AGSI4=Statemachine.dll ("8051 statemachine")
AGSI5=Signalgenerator.dll ("Signal generator")
AGSI6=Timemeasure.dll ("Timemeasure")
AGSI7=TCPIP.dll ("TCPIP connection")
```

[58] AGSI ovládače sú dostupné zdarma na http://www.c51.de/c51.de/Dateien/uVision2DLLs.php?Spr=EN
[59] Bližšie informácie na http://www.keil.com/uvision/db_sim_agsi.asp

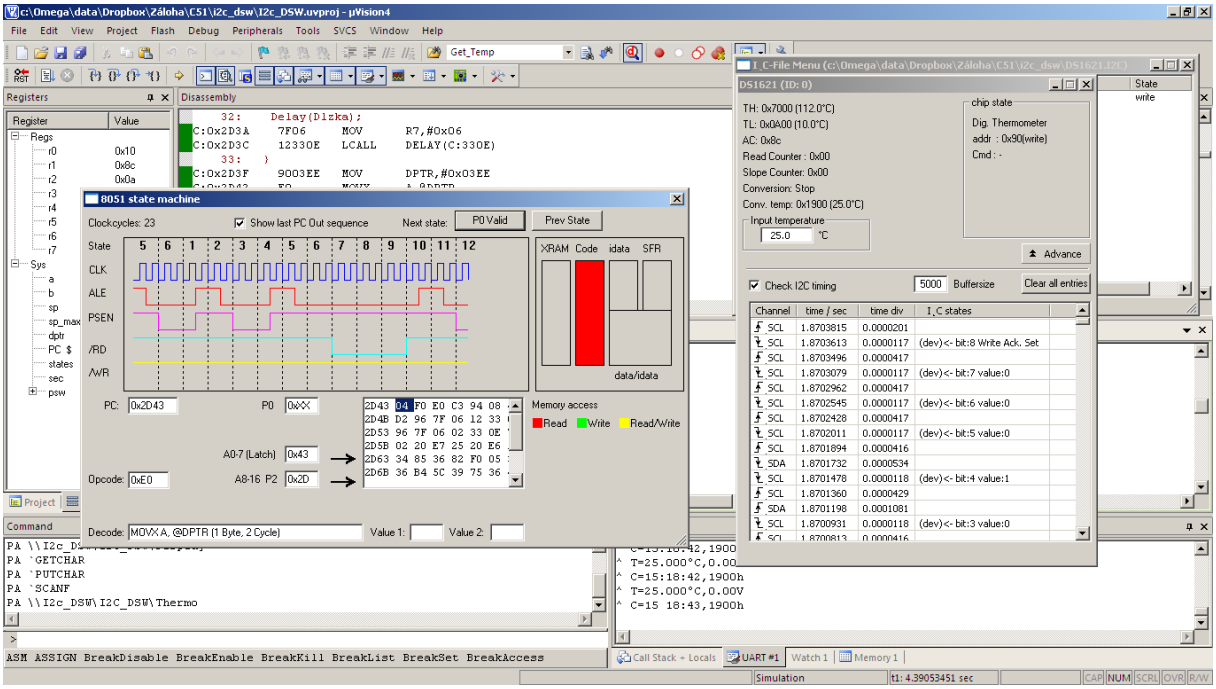
Obrázok 52, Simulácia LCD display-a AGSI ovládačmi



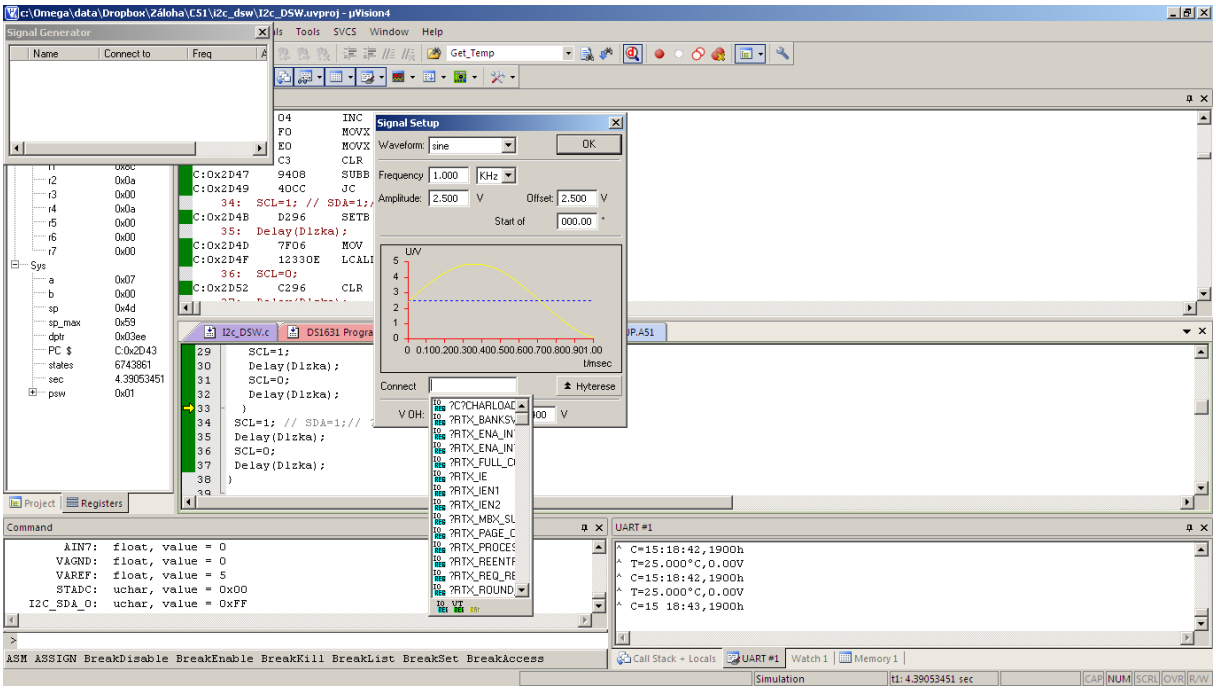
Obrázok 53, Simulácia I²C s AGSI ovládačmi



Obrázok 54, Simulácia stavu procesora s AGSI ovládačmi



Obrázok 55, Simulácia generátora priebehov s AGSI ovládačmi



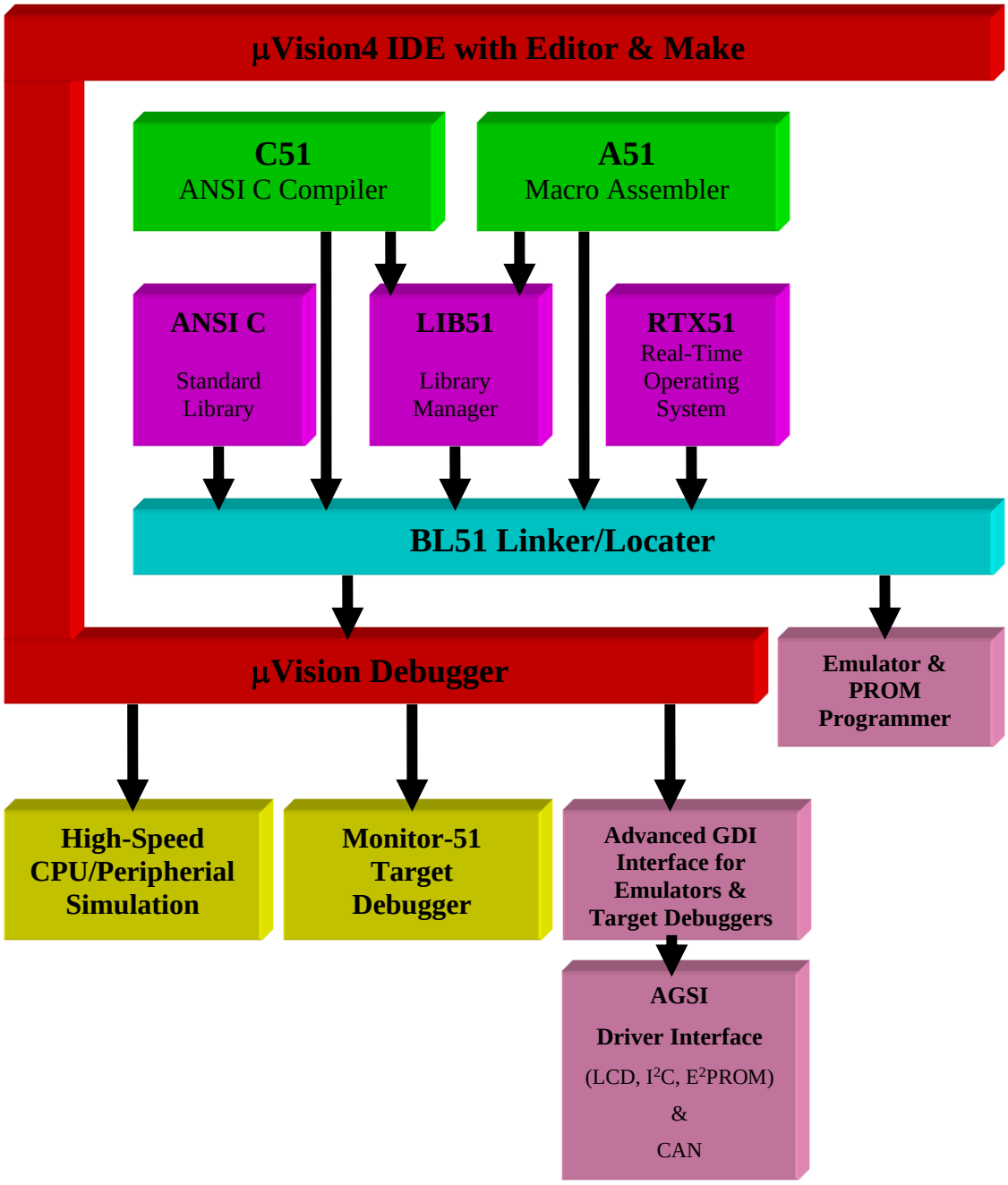
Pre správnu funkciu AGSI ovládačov je potrebné príslušné knižnice DLL nakopírovať[60] do adresára C:\KEIL\BIN\ a doplniť v TOOLS.INI používané simulačné knižnice.

[60] V niektorých prípadoch môžu mať súbory DLL odlišné názvy požadovaných súborov a je to potom potrebné správne premenovať podľa potreby Keil µVision4. Je to spôsobené staršími a neaktuálnymi AGSI na stránke, ktoré neboli dlho inovované oproti prostrediu Keil µVision4. Pôvodne boli AGSI ovládače napísané pre prostredie Keil µVision2. Iná situácia je pre ARM procesory napr. STM32F407VET6, kde existuje ULINK2, ktorý umožňuje ladenie aplikácií na úrovni zdrojového kódu pomocou integrovaného komunikačného rozhrania SW, alebo JTAG.

5 Vývojové prostriedky pre tvorbu aplikácií

5.1. Spôsoby tvorby aplikácie

Obrázok 56, Vývoj aplikácie pomocou µVision 4



5.2. Prehľad vývojových nástrojov

- KEIL PK-C51, PK-C251, PK-C166, PK-ARM, RL-ARM,
- Embedded Workenbench,
- Embedded Workenbench C++,
- MakeApp,
- Code Vision,
- Easy Code,
- Visual State,
- Version Control System,
- Rational Rose C++,
- Easy Case C, C++,
- SDL Desing Tool,
- embOS,
- RTX One[61],
- AS552[62],
- ASM251.[63]

5.3. Požiadavky na vývojové prostriedky

- Vysoká univerzálnosť použitia vývojového prostriedku.
- Zápis programu v C a assembleri, prípadne pomocou napr. stavového diagramu, alebo pravdivostnej tabuľky.
- Možnosť ladenia programu na úrovni C a assembleru.
- Softwarový simulátor s možnosťou vzájomného prepojenia s prototypovou doskou cez RS232, alebo rozhranie CAN.
- Hardwarový simulátor s možnosťou použitia breakpoint-ov v ladenom programe s možnosťou sledovania obsahov registrov, portov, pamäťových miest

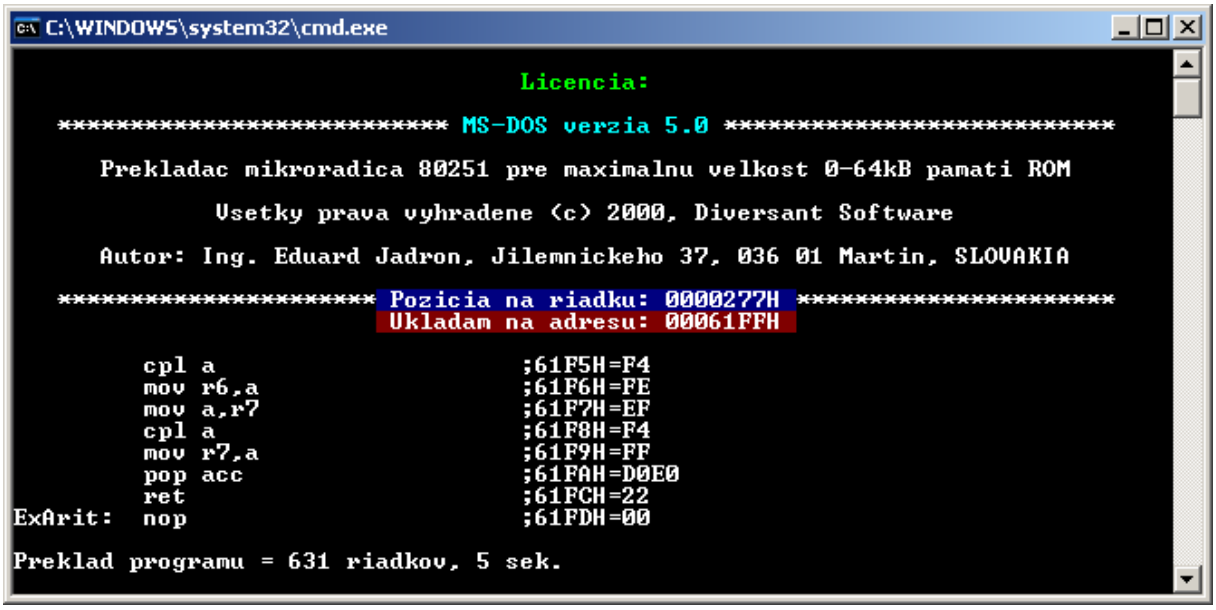
[61] Program je napísaný z dôvodu zabezpečenia vysokej efektívnosti v assembler-i a v pamäti programu zaberá cca. 400 Byte. Autorom RTX One je Ing. Eduard Jadroň.
[62] Program AS552, prípadne predchodca AS51 je už bez podpory výrobcu.
[63] ASM251 je jednoduchý, jednopriechodový assembler bez optimalizácie pre 16 bitový procesor rady 80251, ktorého autorom je Ing. Eduard Jadroň z Diversant Software. Program je napísaný v programovacom jazyku Turbo Pascal 7.0. Program **ASM251.EXE** obsahuje niekoľko doplňujúcich modulov, ktoré dokážu preložený program „zašifrovať“ v pamäti a výrazne tak sťažiť prípadný spätný preklad programu.

mikroprocesora a externej pamäti, stavu sériového komunikačného rozhrania RS232, zbernice I^2C, Ethernet a CAN.

5.4. Programovací jazyk assembler A51

Na trhu existuje obrovské množstvo platených, ale aj voľne šíriteľných[64] vývojových nástrojov pre mikroprocesory 8051 prevažne pracujúcich pod operačným systémom MSDOS[65]. Príkladom môžeme uviesť MASM51, A51, AS51[66], ASM251, AS552, 8051 Workenbench, 8051 IDE a iné. Zaujímavou alternatívou je použitie špeciálnych prekladačov[67] z ASM do C, prípadne do iného jazyka. Bližšie aplikačné možnosti sú uvedené v (RTX Real Time Operating System, 2001).

5.5. Programovací jazyk assembler ASM251 pre procesory rady 80251



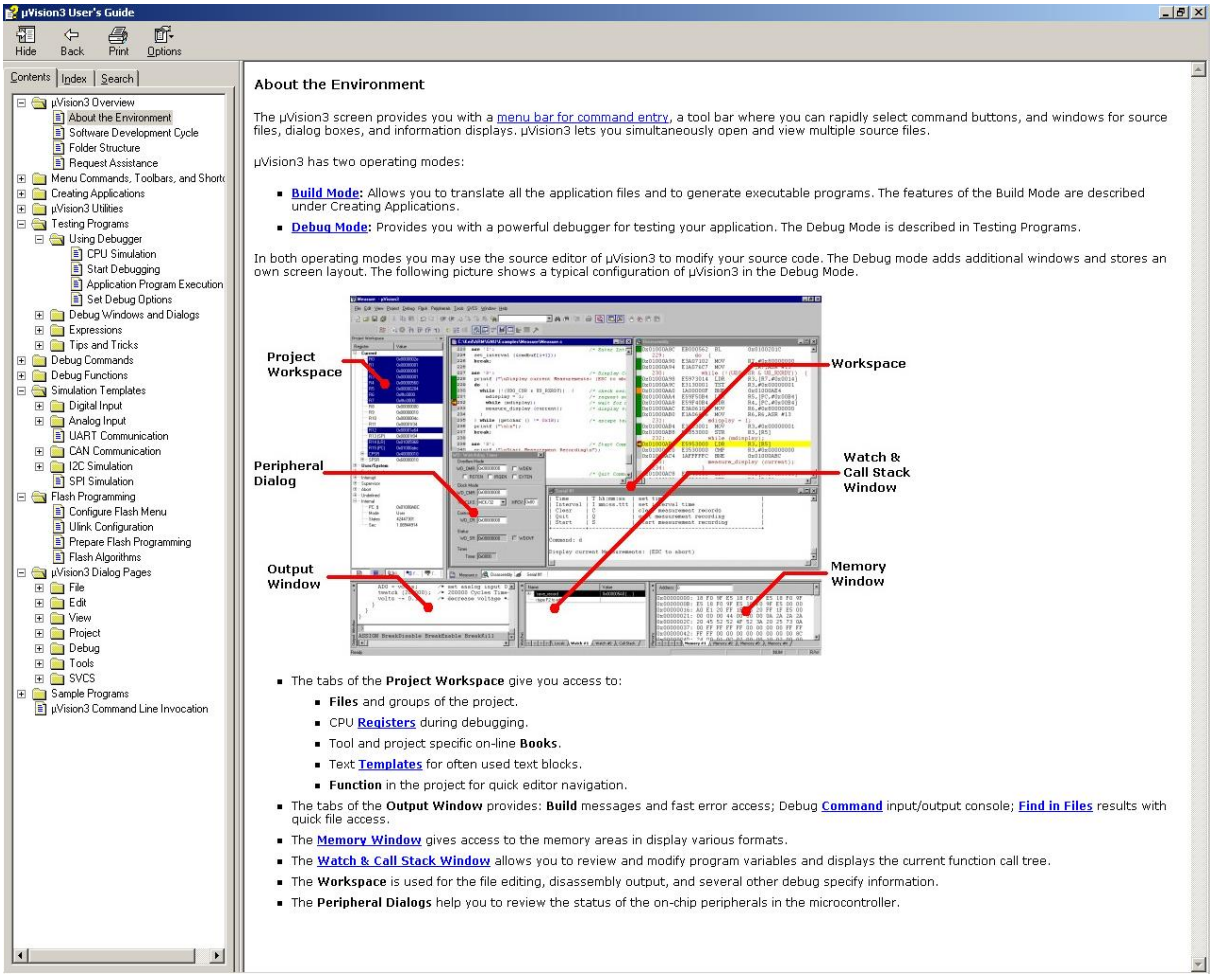
[64] Shareware, alebo freeware TPtoC.

[65] Prekladače a kompilátory ktoré boli spustiteľné pod 16-bitovým operačným systémom MSDOS nie sú podporované v niektorých 32-bitových a takmer vo všetkých 64-bitových operačných systémoch. Tento problém je možné obísť pomocou programov DosBox, VirtualBox, VMware ktoré umožňujú spustiť vo virtuálnom prostredí operačného systému ľubovoľný operačný systém.

[66] Program AS51.EXE obsahuje niekoľko závažných chýb, ktoré nesprávne prekladajú niektoré presunové inštrukcie ale v novšej verzii programu AS552.EXE od rovnakého výrobcu už uvedené chyby boli odstránené.

[67] http://www.mpsinc.com/index.html, upozorňujeme, že ide o platenú službu, kde je spoplatnený jeden riadok sumou približne 0,10 USD.

Obrázok 57, Optimálne rozdelenie pracovnej plochy vývojového nástroja

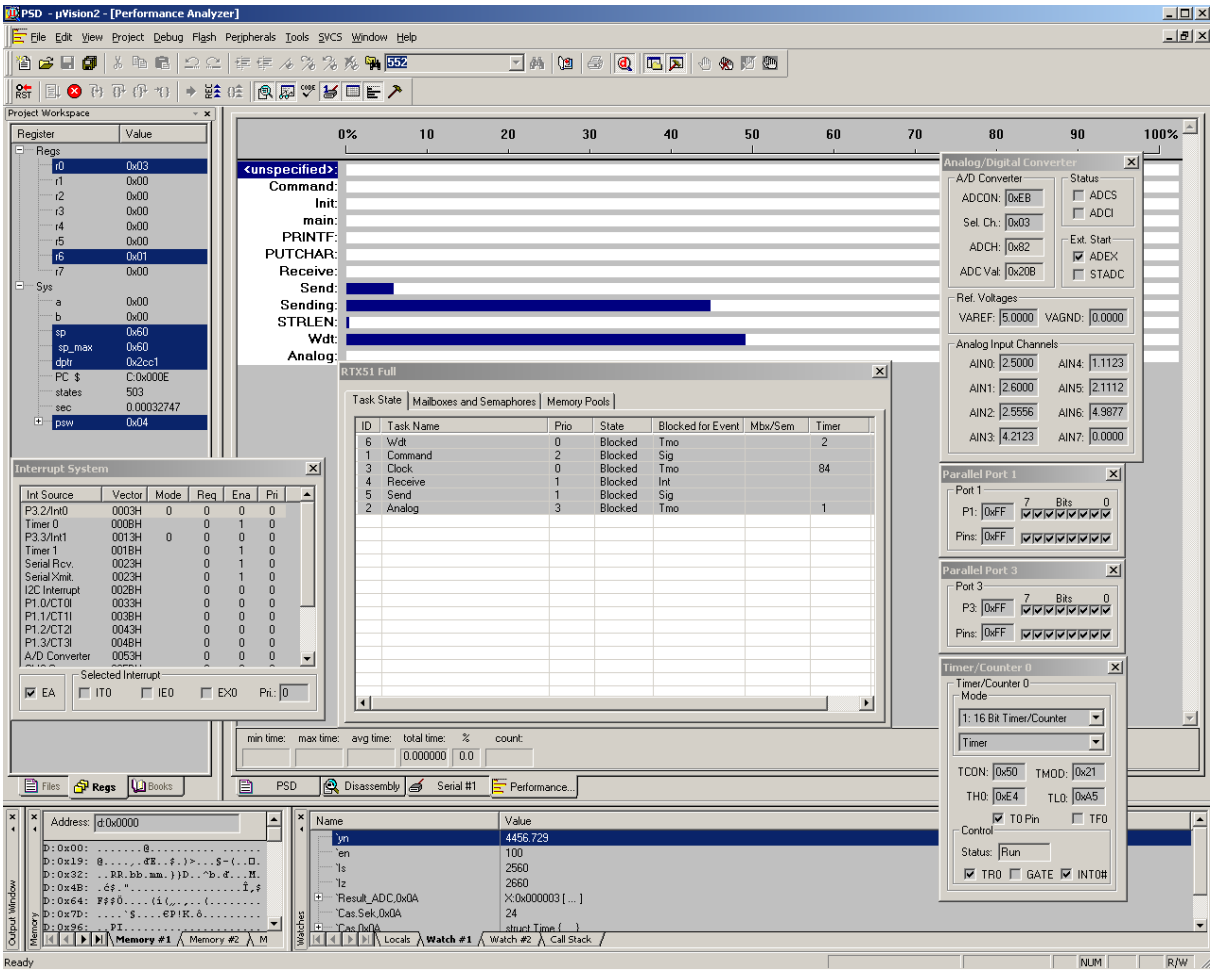


6 Prehľad vývojových nástrojov

6.1. Grafické vývojové prostredie µVision od firmy KEIL Software

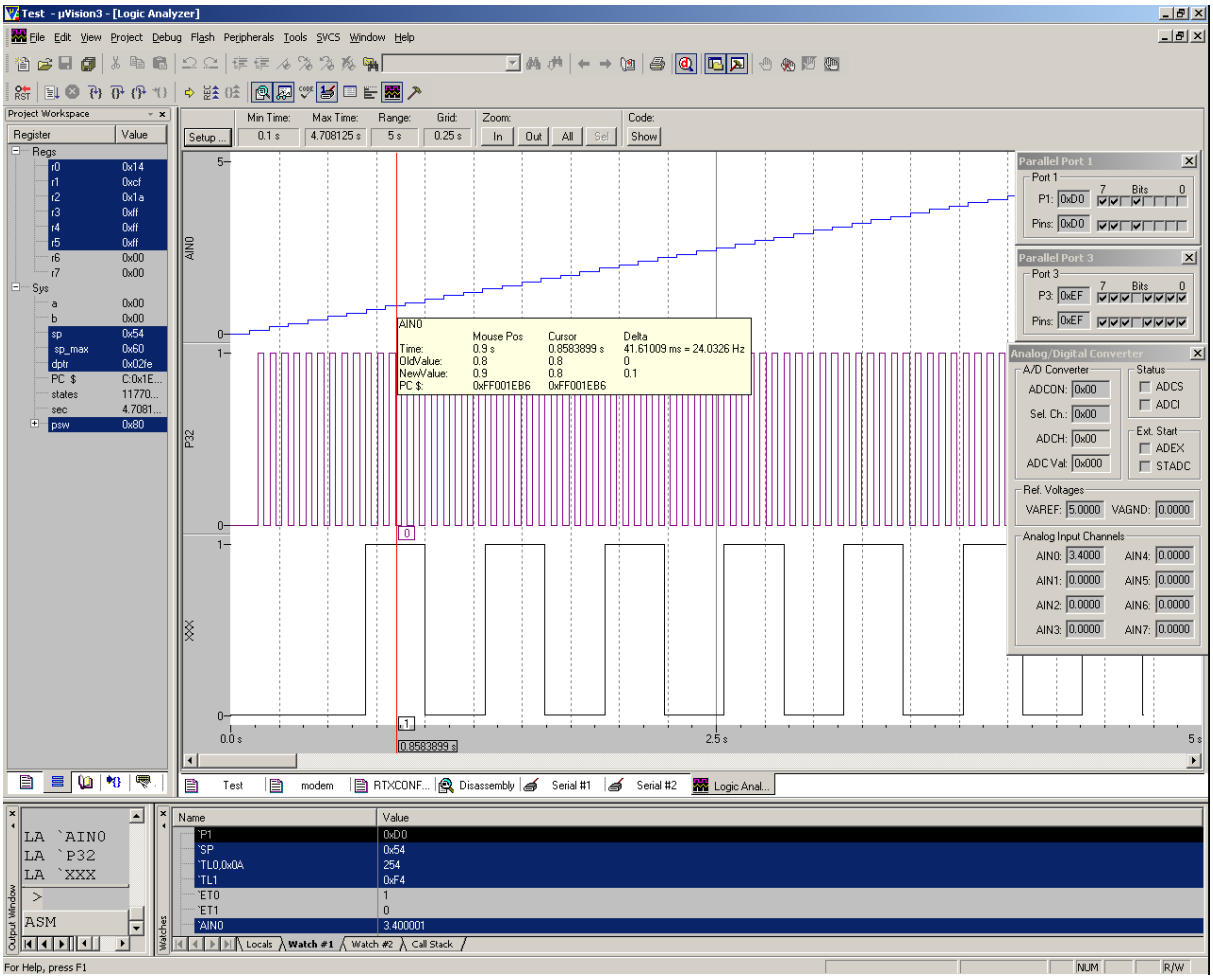
Stručná charakteristika: Prostredie µVision od verzie 2.4 je výkonným a mocným vývojovým nástrojom pre programátora. Obsahuje integrovaný programovací jazyk C a assembler, ktorý je možné ladiť až na najnižšej úrovni assembler-u, alebo na úrovni jazyka C. Výstup sériového kanála je možné presmerovať na obrazovku, alebo na štandardný port COM1 až COM4. Programátor môže sledovať množstvo obsahov registrov, ktoré sú prehľadne blokovo usporiadané. Vývojové prostredie µVision je možné používať pre tvorbu aplikácií používajúce mikroprocesory 8051, 80251, 80166, ARM. Firma KEIL Software je autorom RTOS pod obchodným označením RTX51 Tiny a RTX51 Full, ktoré sa používajú na vývoj aplikácii ktoré majú pracovať v reálnom čase. Hlavnou nevýhodou tohto programu je vysoká cena. Bližšie informácie v www.keil.com.

Obrázok 58, Vývojové prostredie µVision 2.14 od firmy KEIL pre architektúry 80C51



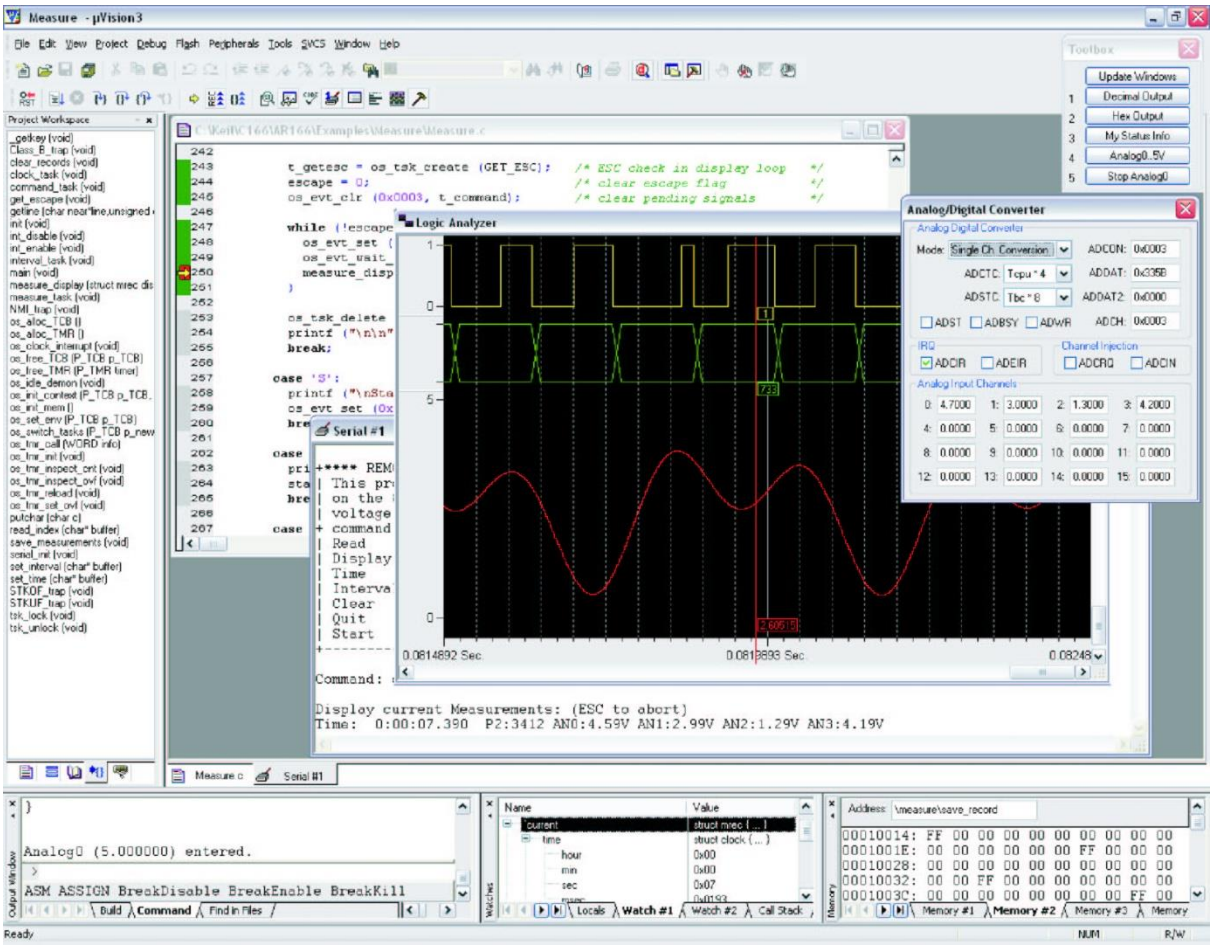
Stručná charakteristika: Vývojové prostredie µVision[68] 3 obsahuje okrem štandardných nástrojov aj logický analyzátor umožňujúci sledovanie analógových a číslicových signálov, čo posúva praktické využitie tohto nástroja smerom bližšie k HW emulátorom. Toto vývojové prostredie slúži na uľahčenie práce v náročných aplikáciách ako je napríklad TCP/IP Stack, rýchla Fourier-ová harmonická analýza v reálnom čase, skalárny alebo vektorový regulátor pohonu asynchrónneho motora a iných náročných aplikácií. K tomuto prekladaču je možné zakúpiť aj AR166, ktorý pomáha pri vytváraní tzv. Embedded aplikácií. Demonštračnú verziu tohto vývojového nástroja je možné získať na www.keil.com.

Obrázok 59, Vývojové prostredie µVision3 od firmy KEIL pre architektúry 80C51



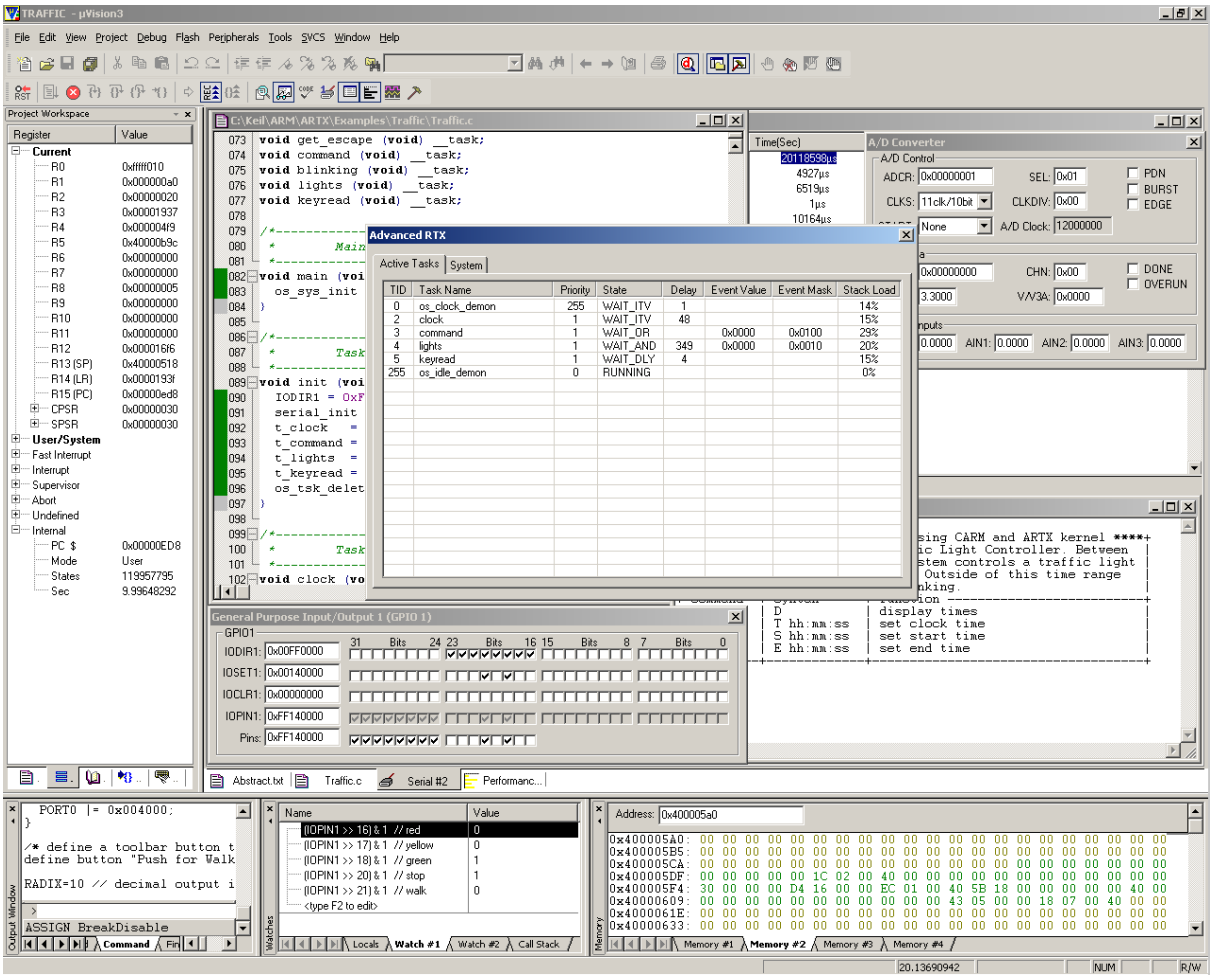
[68] Od mája 2020 je k dispozícii nová verzia PK 9.60 µVision 5, ktorá sa dočkala podstatných zmien ktoré zásadným spôsobom uľahčia vývoj aplikácií programátorom.

Obrázok 60, Vývojové prostredie µVision3 od firmy KEIL pre architektúry 80C166

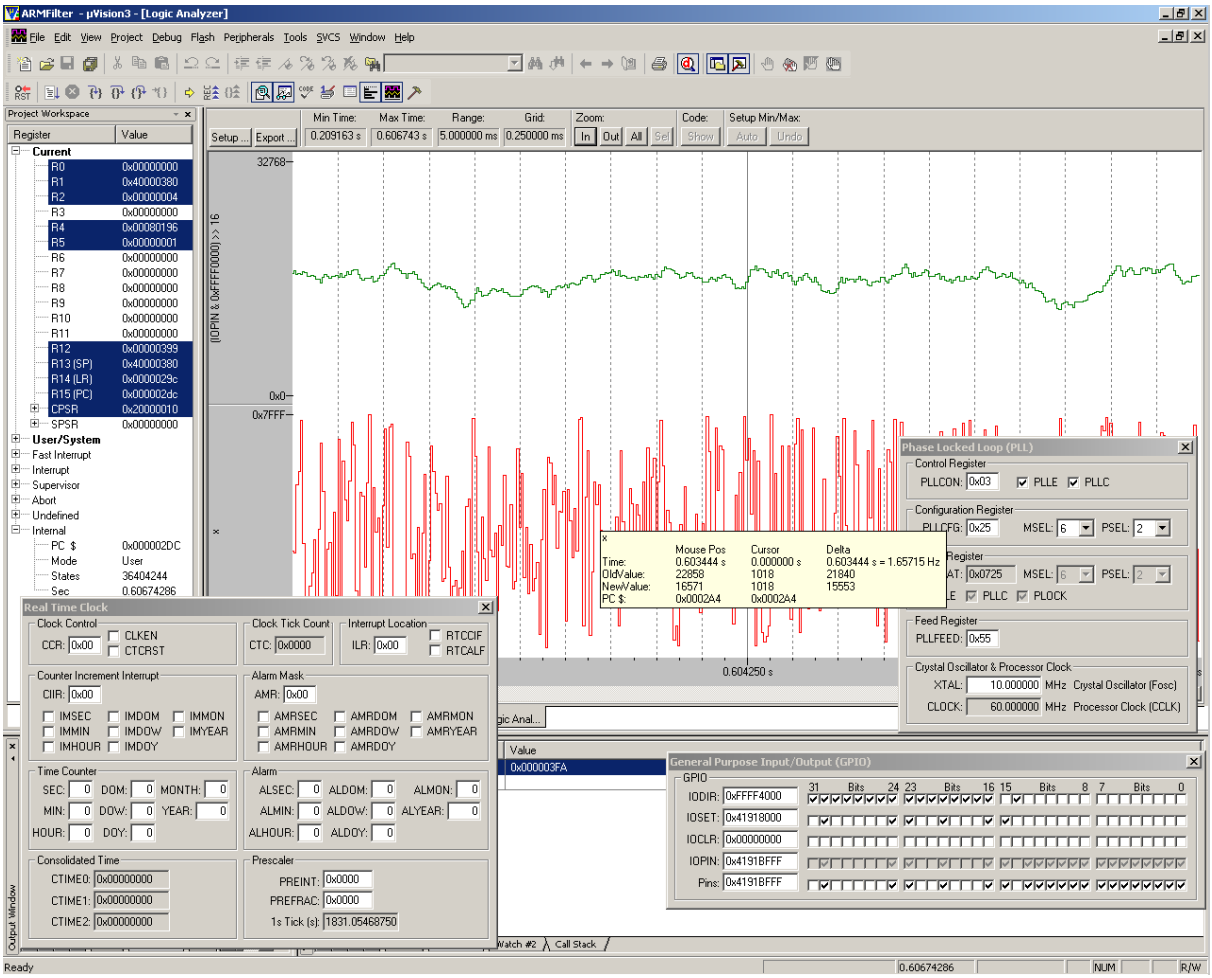


Stručná charakteristika: Toto vývojové prostredie µVision 3 slúži na prácu s rodinou najvýkonnejších 32 bitových mikroprocesorov ARM. Obsahuje podobné nástroje pre vývojového pracovníka a rovnaké prostredie na aké bol užívateľ zvyknutý doteraz. Výhodou tohto nástroja je fakt, že je distribuovaný ako ***Public Domain*** s jediným obmedzením, ktoré umožňuje ladenie programu s maximálnou veľkosťou do 16kB. Kompilátor a linker jazyka C++ je v tomto prípade bez obmedzenia a tak je možné vytvárať výsledný kód, ktorý je možné naprogramovať do mikroprocesora. K tomuto programu je dodávaný ARTX pre mikroprocesory ARM, ktorý má obdobné vlastnosti ako RTX51, alebo AR166. Demo verziu tohto vývojového nástroja je možné získať na www.keil.com.

Obrázok 61, Vývojové prostredie µVision3 od firmy KEIL pre architektúry ARM7TDMI

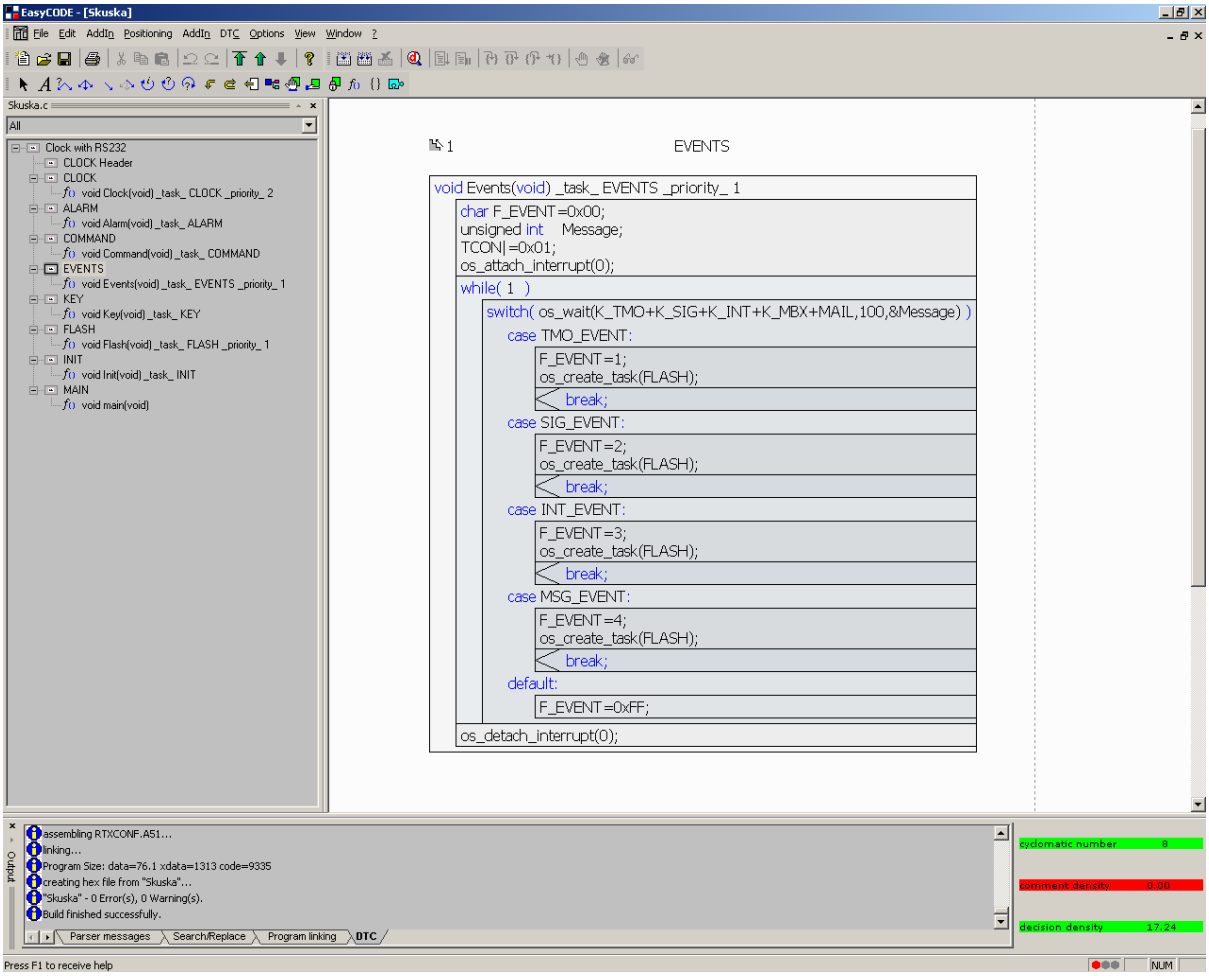


Obrázok 62, Vývojové prostredie µVision3 pre architektúry ARM



Stručná charakteristika: Vývojové prostredie EasyCODE slúži ako výkonná nadstavba pre programátora, pretože umožňuje zápis programu vo forme štruktúrovaných vývojových diagramov. Vyššie uvedený software umožňuje generovať cieľový kód pre rôzne programovacie jazyky ako je C/C++, Pascal, Cobol, Java, Basic. Takto je umožnená nezávislosť zapísaného algoritmu programu na použitom programovacom jazyku. EasyCODE využíva užívateľom používaný kompilátor a linker jazyka demo-verziu tohto vývojového nástroja je možné získať na www[69] stránke a na rozdiel od ostrej verzie programu neumožňuje zápis programu na pevný disk a použitie funkcie kopírovania a iných drobných obmedzení.

Obrázok 63, Vývojové prostredie EasyCode 7.1.0.3 od firmy EasyCode Inc. pre 8051



Nespornou výhodou EasyCODE[70] je aj možnosť prepojenia s inými vývojovými prostriedkami napr. od firmy Keil, Tasking a iných pomocou unifikovaného rozhrania[71] DTC,

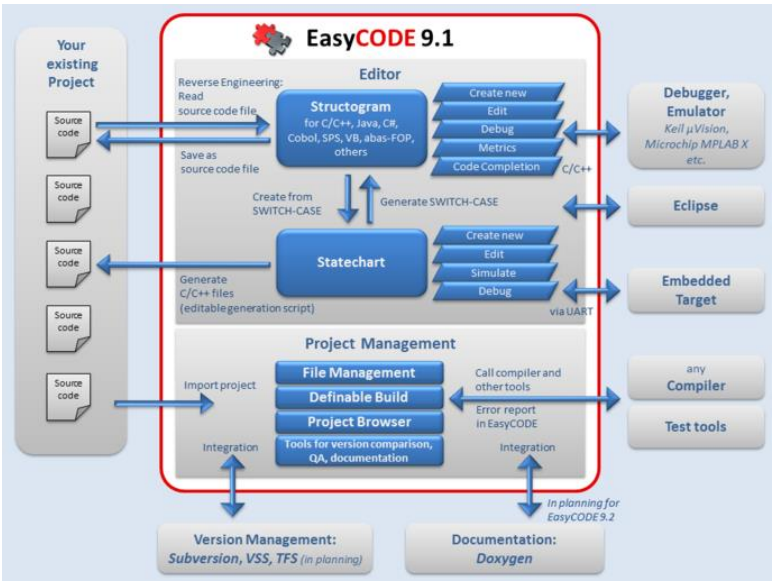
[69] www.easycode.de, www.easycode-software.com

[70] V prípade použitia je možné celý algoritmus programu vytvoriť pomocou štruktúrogramu, alebo v C jazyku napísaný program previesť do štruktúrogramu.

[71] V prípade problémov s DTC je možné použiť aj rozhranie UVSOCK (The µVision Socket Interface) (TCP/IP) na porte 4823.

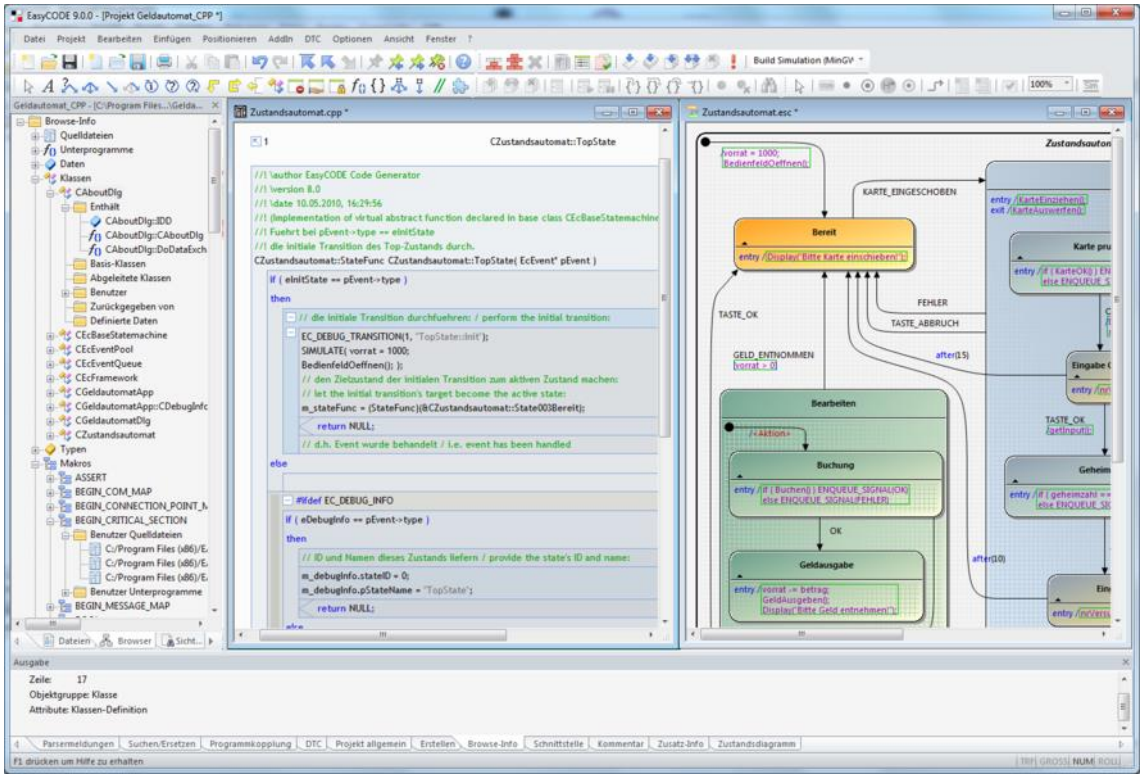
ktoré umožňuje priame ovládanie procesu kompilácie a ladenia pripojených vývojových prostredí.

Obrázok 64, Moderné možnosti programovania s EasyCODE 9.1



Zdroj: www.easycode.de

Obrázok 65, Pracovné prostredie EasyCODE 9.0.0



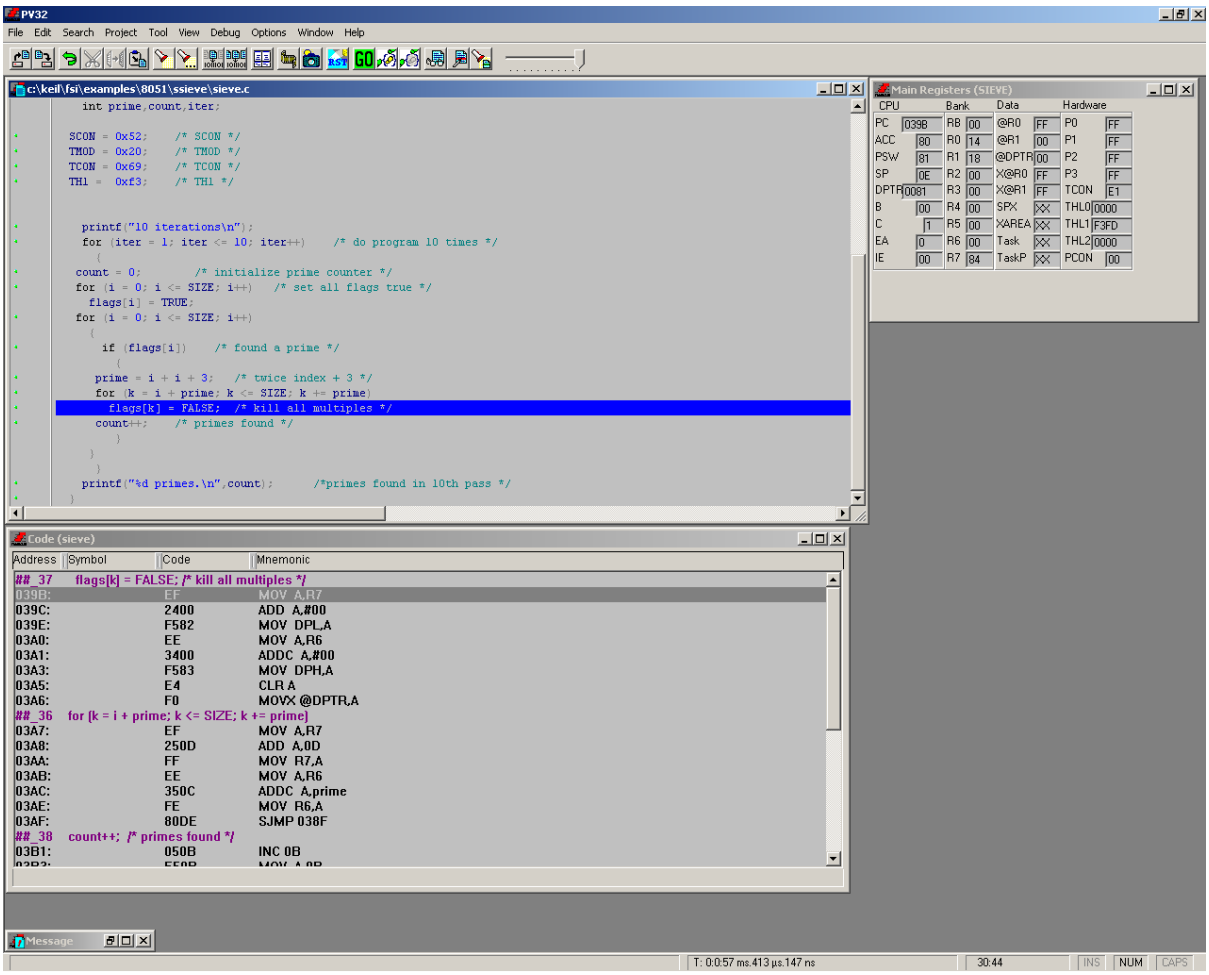
Zdroj: www.easycode.de

Prínos programu EasyCODE pre programátora je nesporný, pretože mu umožňuje vyvíjať aplikáciu pomocou vizualizačných nástrojov ktoré sú lepšie človekom prijímané. Po vizuálnom návrhu programátorom vygeneruje vysoko optimalizovaný programový kód.

6.2. Grafické vývojové prostredie ProView od Franklin Software

Stručná charakteristika: Prostredie ProView, alebo ProView32 je často používaným nástrojom profesionálnych programátorov. Umožňuje programátorovi programovať v jazyku C a assembleri. V podstate je to softwarový simulátor, ktorý nemá na rozdiel od µVision možnosť prepojenia programu ProView s vývojovou doskou pomocou sériového kanála. Cena tohto programu je v tomto prípade výrazne nižšia ako v prípade µVision. Existuje možnosť použitia RTOS v ProView od rôznych výrobcov. Na internete je možnosť získania tzv. **Evalution version** programu, ktorú môže užívateľ po dobu 30 dní bezplatne pred zakúpením vyskúšať a potom sa rozhodnúť pre kúpu vývojového nástroja. Táto Evalution verzia obsahuje niektoré obmedzenia, ale firma sa takýmto spôsobom účinne chráni pred nelegálnemu kopírovaniu svojich produktov.

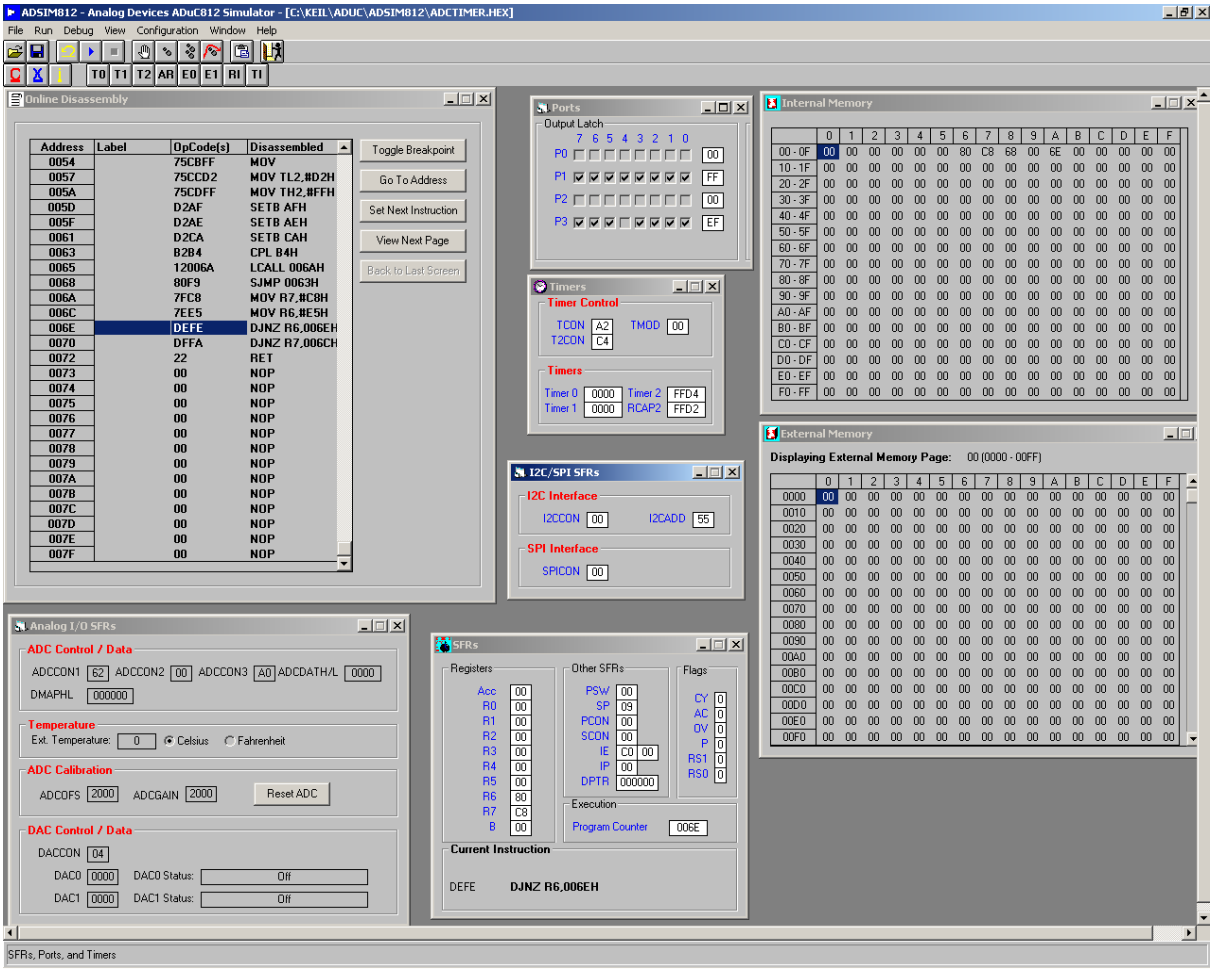
Obrázok 66, Vývojové prostredie ProView32



6.3. Grafické vývojové prostredie ADSIM812 od Analog Devices

Stručná charakteristika: Opisované vývojové prostredie umožňuje podobné funkcie ako jej konkurenčné produkty. Nespornou výhodou tohto nástroja je cena. Firma Analog Devices tento program totiž distribuuje ako freeware. Plná verzia umožňuje plnohodnotnú[72] prácu s vývojovými mikroprocesorovými doskami.

Obrázok 67, Vývojové prostredie ADSIM812

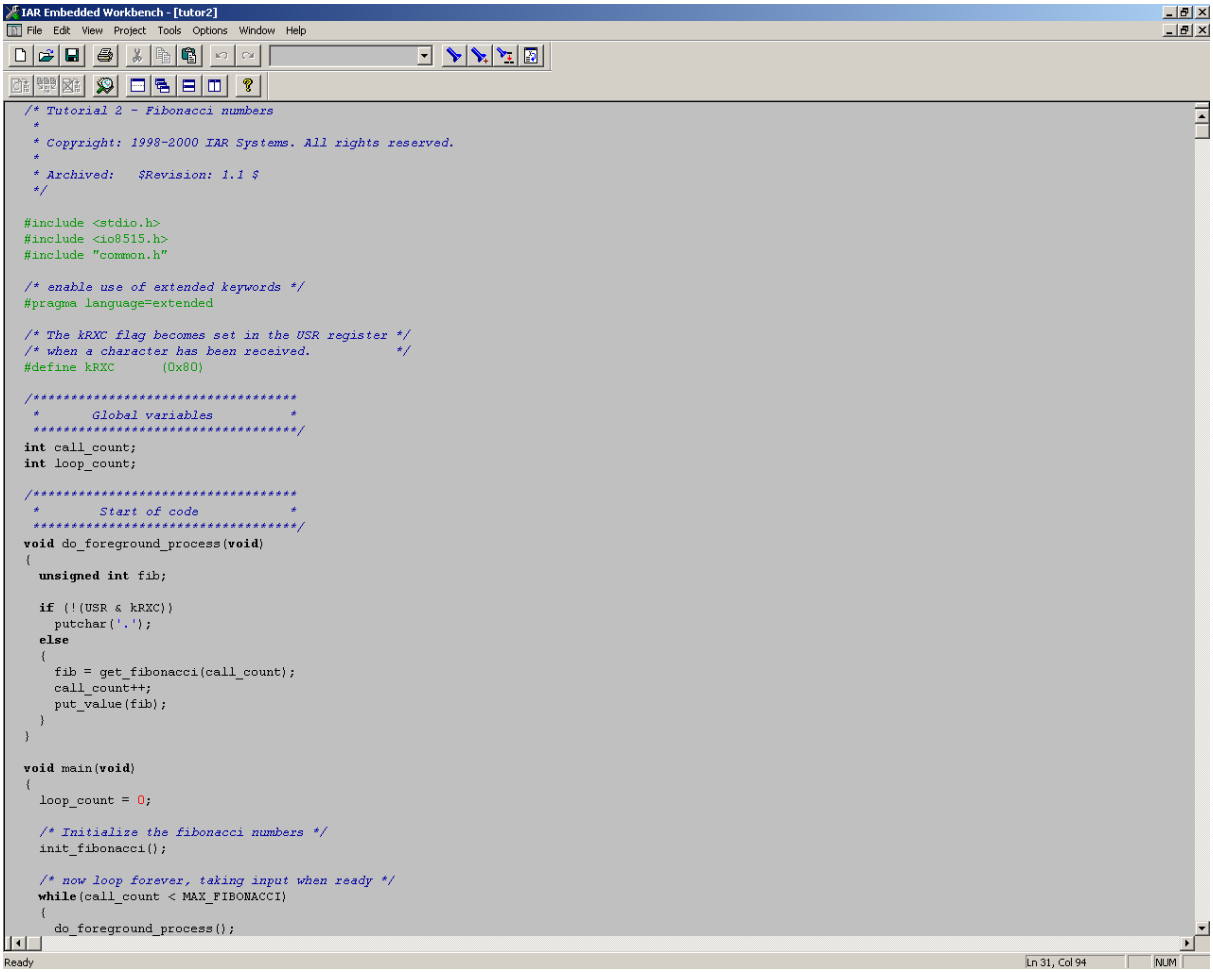


[72] Pod pojmom „plnohodnotná“ si predstavujeme len základné a jednoduché činnosti pri programovaní aplikácií pre platformu mikroprocesorov rady 8051 bez optimalizačných a simulačných nástrojov pracujúcich na úrovni HW (hardware 8051).

6.4. Grafické vývojové prostredie Embedded Worekbench od IAR Systems

Stručná charakteristika: Tento opisovaný nástroj od švédskej firmy IAR obsahuje výkonný kompilátor C a assembleru, ktorý je vybavený množstvom optimalizačných stupňov. Vybavením a funkciami je možné tento nástroj porovnať s µVision od firmy KEIL. Firma IAR Systems ponúka množstvo alternatívnych produktov ktoré výrazným spôsobom uľahčujú prácu programátora napr. MakeApp, VisualState, CodeVision, EasyCODE a iné. Programátor má možnosť použitia RTOS priamo vo svojich aplikáciách od rôznych výrobcov. Pre začínajúcich programátorov firma IAR uvoľnila programovací jazyk assembler bez obmedzení. Bližšie informácie na www.iar.com.

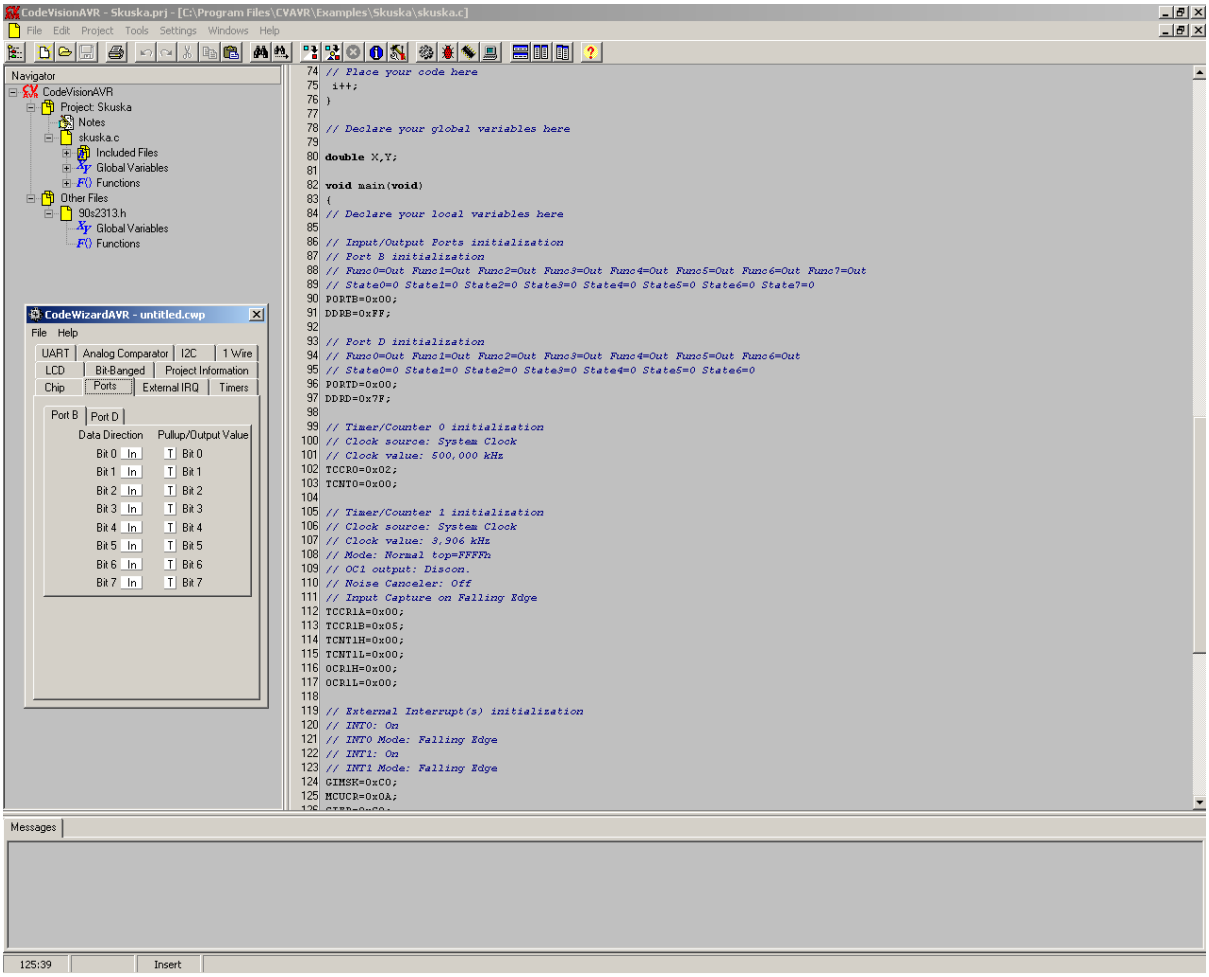
Obrázok 68, Vývojové prostredie Embedded Workbench



6.5. CodeVision AVR

Stručná charakteristika: Uvedený nástroj CodeVision sa zaraďuje do kategórie tzv. generátorov aplikácií, pretože umožňuje interaktívnym spôsobom pomocou dialógových okien nakonfigurovať jednotlivé periférne obvody mikroprocesora, čím odpadá zdĺhavé štúdium katalógových listov mikroprocesora. Výsledný kód z CodeVisionAVR môžeme priamo naprogramovať do mikroprocesora. Bližšie informácie na www.iar.com.

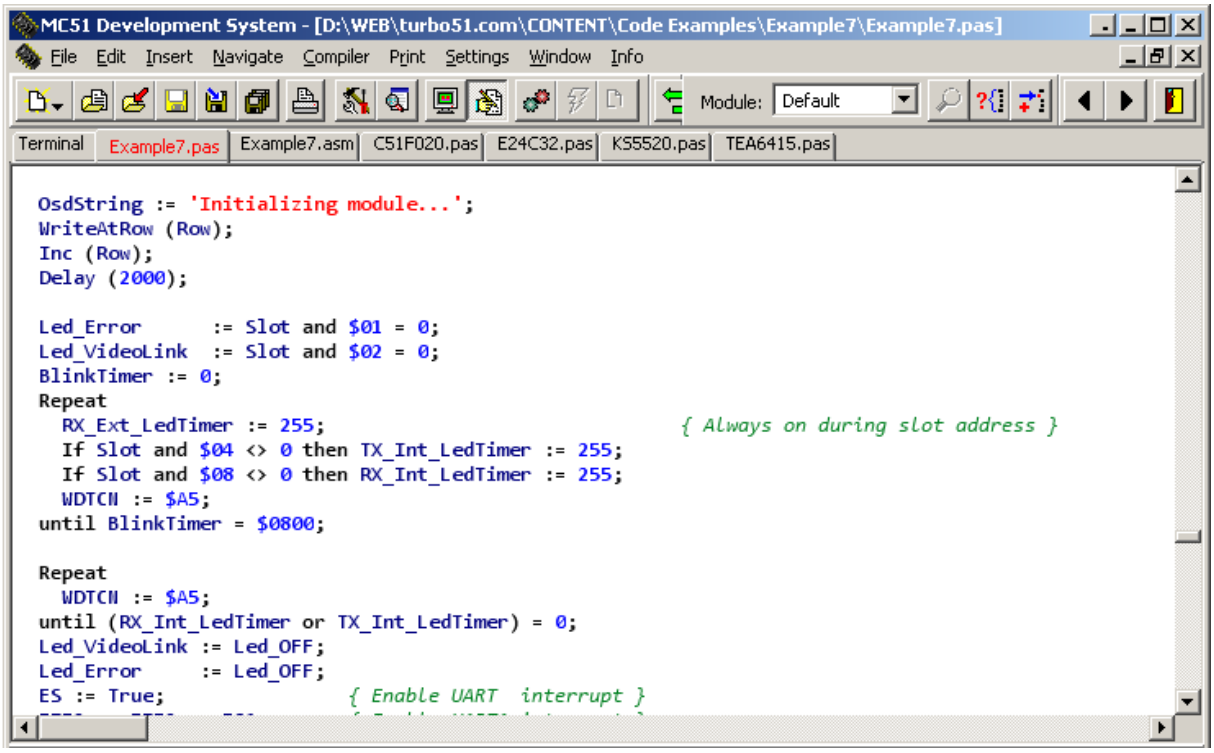
Obrázok 69, Vývojové prostredie CodeVisionAVR pre architektúry AVR



6.6. Vývojové prostredie Turbo51 pre mikroprocesory 8051

Stručná charakteristika: Tento voľne šíriteľný nástroj distribuovaný ako freeware slúži pre tvorbu aplikácií platformy 8051 v jazyku Pascal[73]. Neobsahuje grafické prostredie, ale všetky príkazy sú zadávané pomocou príkazového riadku. Autor poskytuje aj naďalej podporu pre tento program. Kvality tohto programu sú porovnateľné s inými vývojovými prostriedkami v tejto kategórii. Aktuálna verzia Turbo51 k 30.1.2011 je 0.1.3.9. Na internete je k dispozícii veľké množstvo prekladačov jazyka Pascal[74] líšiacich sa množstvom použitých technológií, preto je na užívateľovi ktorý si zvolí pre svoju profesionálnu činnosť.

Obrázok 70, Doporučený textový IDE editor k Turbo51



[73] Download programu Turbo51 je možné na stránke http://turbo51.com/download-free-pascal-compiler-8051
[74] http://www.thefreecountry.com/compilers/pascal.shtml

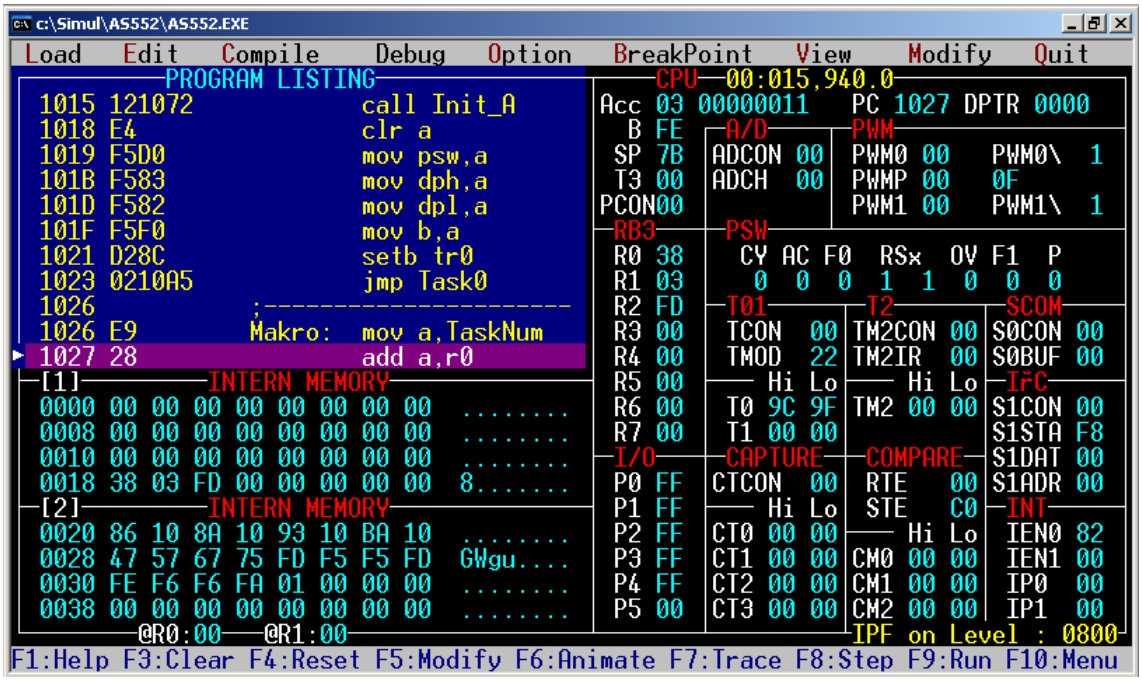
Príklad zápisu programu v prostredí Turbo51:

```
Program Calculator;

Const
 Osc = 22118400;
 BaudRateTimerValue = Byte(- Osc div 12 div 32 div 19200);

Var SerialPort: Text;
    Num1, Num2: LongInt;

Procedure WriteToSerialPort; Assembler;
Asm
  CLR   TI
  MOV   SBUF, A
@WaitLoop:
  JNB   TI, @WaitLoop
end;

Function ReadFromSerialPort: Char;
Var ByteResult: Byte absolute Result;
begin
  While not RI do;
  RI := False;
  ByteResult := SBUF;

{ Echo character }

  Asm
    MOV    A, Result
    LCALL  WriteToSerialPort
  end;
end;


Procedure Init;
begin
  TL1   := BaudRateTimerValue;
  TH1   := BaudRateTimerValue;
  TMOD  := %00100001;     { Timer1: no GATE, 8 bit timer, autoreload }
  SCON  := %01010000;     { Serial Mode 1, Enable Reception }
  TI    := True;          { Indicate TX ready }
  TR1   := True;          { Enable timer 1 }
end;

{$DefaultFile On }

begin
  Init;
  Assign(CurrentIO, ReadFromSerialPort, WriteToSerialPort);

  Writeln('Turbo51 IO file demo - integer calculator');
  Repeat
    Write   ('Enter first number: ');
    Readln  (Num1);
    Write   ('Enter second number: ');
    Readln  (Num2);
    Writeln (Num1, ' + ', Num2, ' = ', Num1 + Num2);
    Writeln (Num1, ' - ', Num2, ' = ', Num1 - Num2);
    Writeln (Num1, ' * ', Num2, ' = ', Num1 * Num2);
  until False;
end.
```

6.7. Vývojové prostredie AS552 od firmy Easy Soft.

Stručná charakteristika: Tento nástroj slúži na tvorbu aplikácií v jazyku assembler. Prioritne je určený pre mikroprocesory 80c552, ktoré obsahujú integrovaný 8 kanálový 10 bitový A/D prevodník. Obsahuje integrovaný prekladač[75] a simulátor[76], ktorý umožňuje sledovať činnosť vykonávaného programu a obsahu jednotlivých registrov mikroprocesora. Ovládanie programu AS552 je veľmi jednoduché a odpovedá možnostiam operačného systému MSDOS.

Obrázok 71, Vývojové prostredie AS552

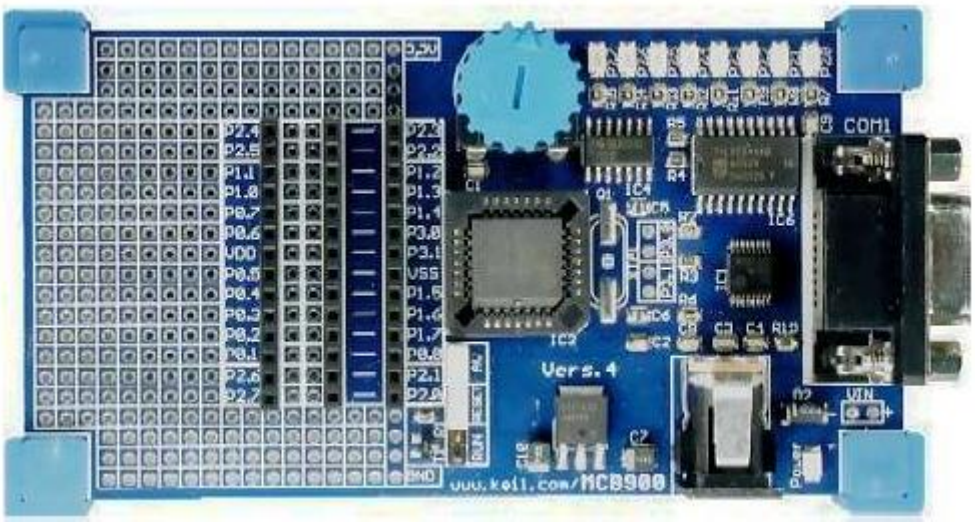


[75] Výrobca uviedol na trh aj prekladač len pre mikroprocesor 8051 pod názvom AS51, ktorý žiaľ kvôli množstvu chýb ktoré spôsobuje kompilátor v preloženom kóde nie je možné použiť pre profesionálnu aplikačnú oblasť.
[76] Modul View51.exe umožňuje simulovať vo funkcii osciloskopu prítomnosť logických úrovní na jednotlivých portoch procesora 80c552 a zobraziť ich na časovej osi.

7 Hardware pre vývojové aplikácie

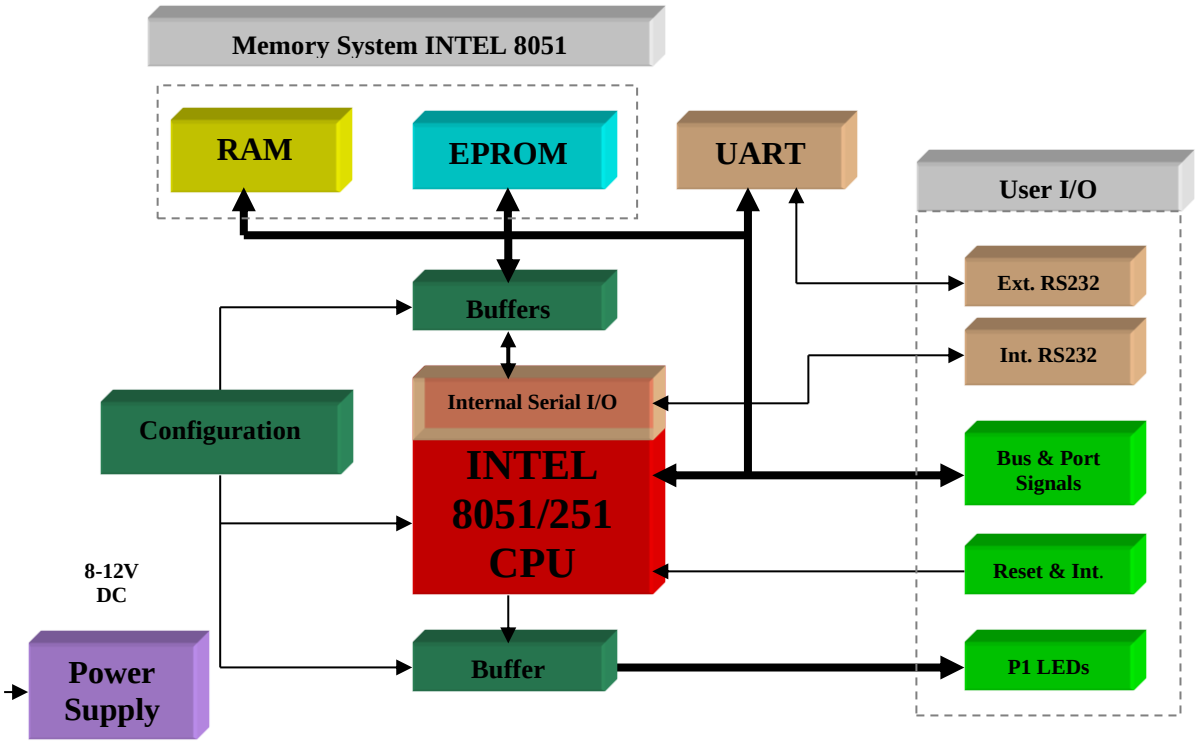
Rôznorodosť aplikácií používaných v spojitosti s mikroprocesorovou technikou má za následok množstvo rôznych vývojových dosiek na trhu, ktoré existujú v rôznych konfiguráciách. Každá aplikácia si vyžaduje špecifický hardware, ktorý umožňuje plniť svoju primárnu činnosť. Táto kapitola jednoduchým spôsobom čitateľa oboznámi s jednoduchými vývojovými doskami ktoré sú dostupné na trhu a sú schopné spolupracovať s vývojovým prostredím.

Obrázok 72, Hardware pre mikroprocesory 8051 a ARM

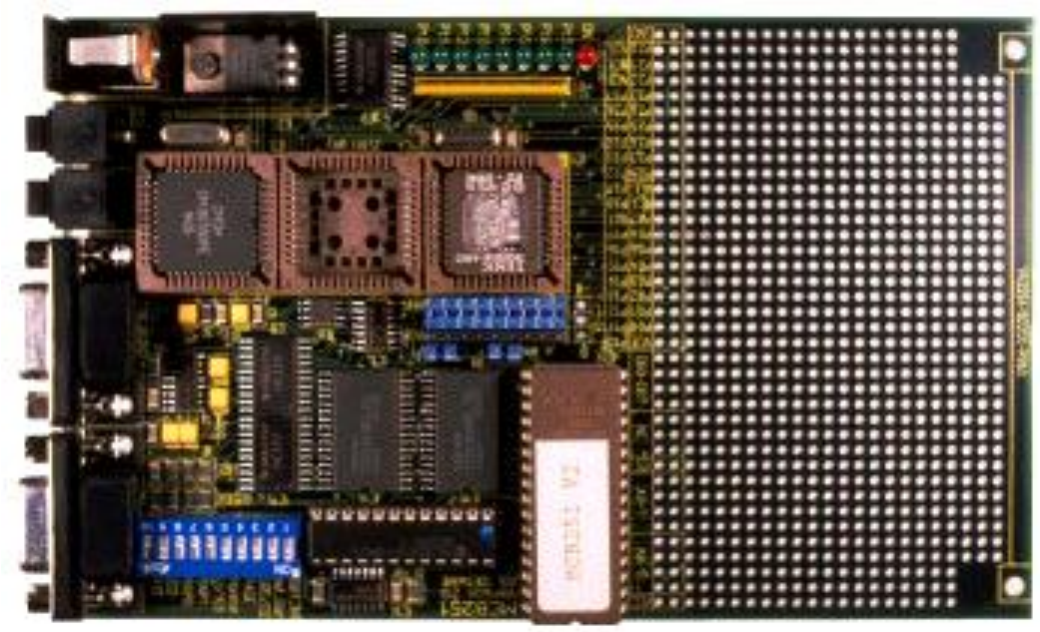


7.1. Hardware 80C51

Obrázok 73, Aplikačný model mikroprocesorového systému s 80Cx51

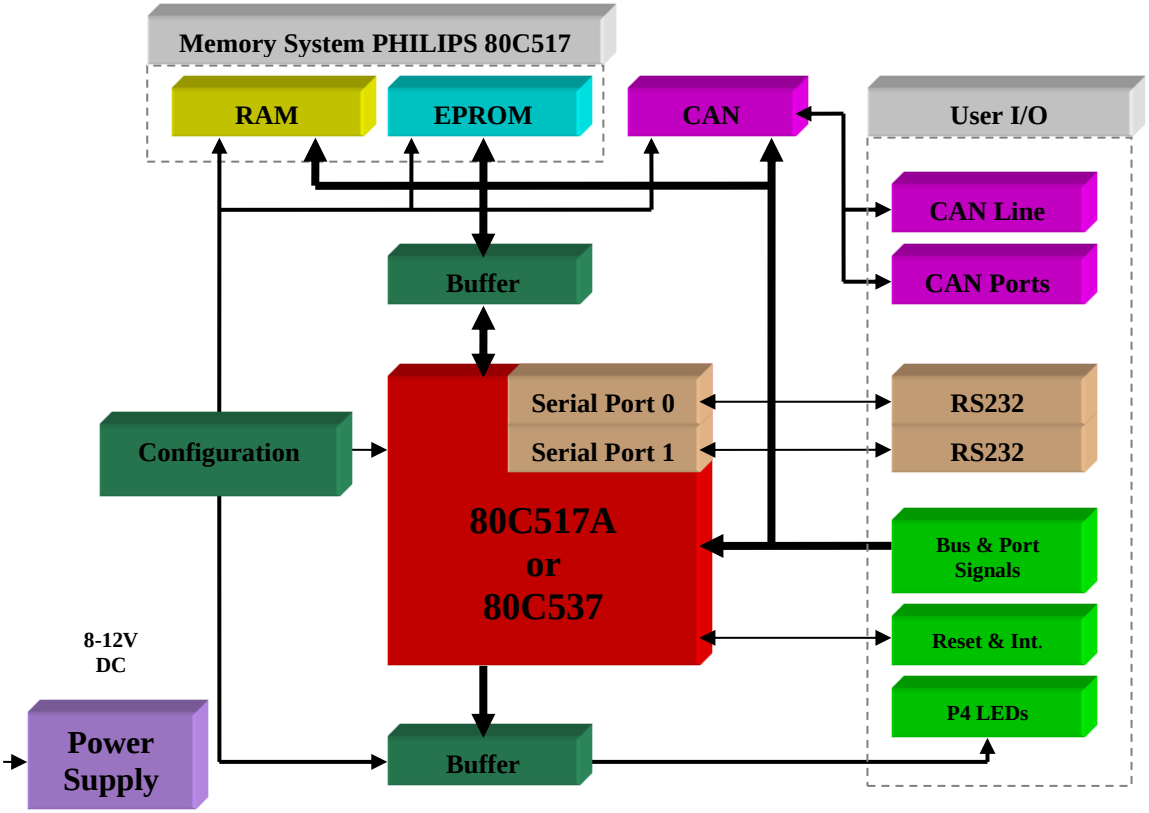


Obrázok 74, Vývojová doska s mikroprocesorom Intel 80Cx51

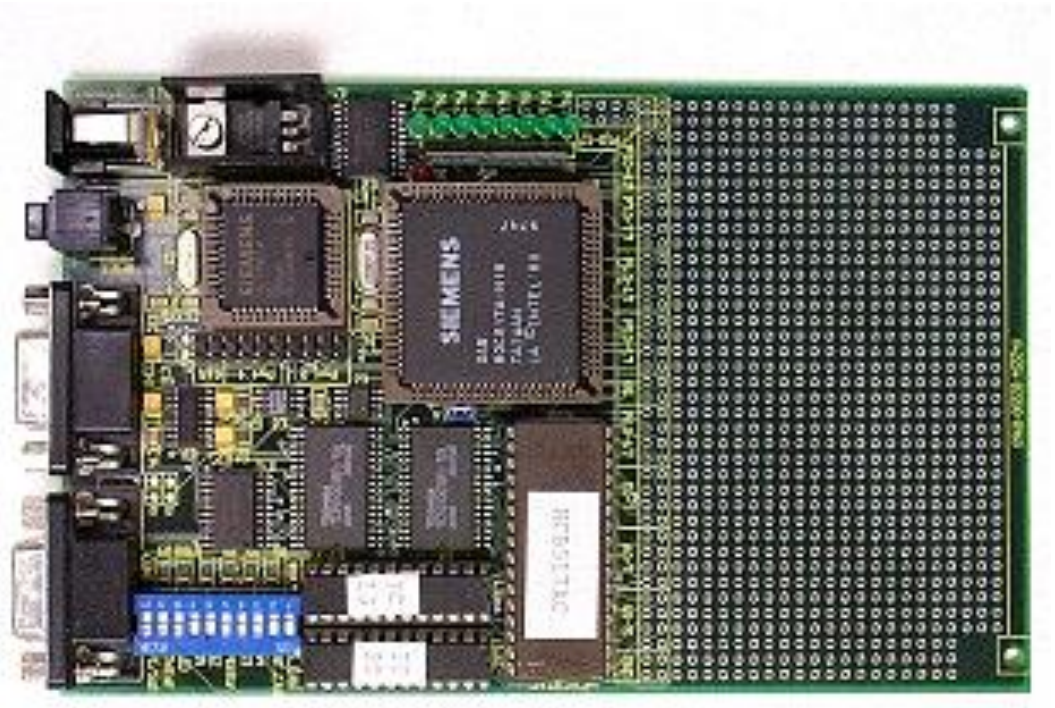


7.2. Hardware 80C517

Obrázok 75, Aplikačný model mikroprocesorového systému s 80C517

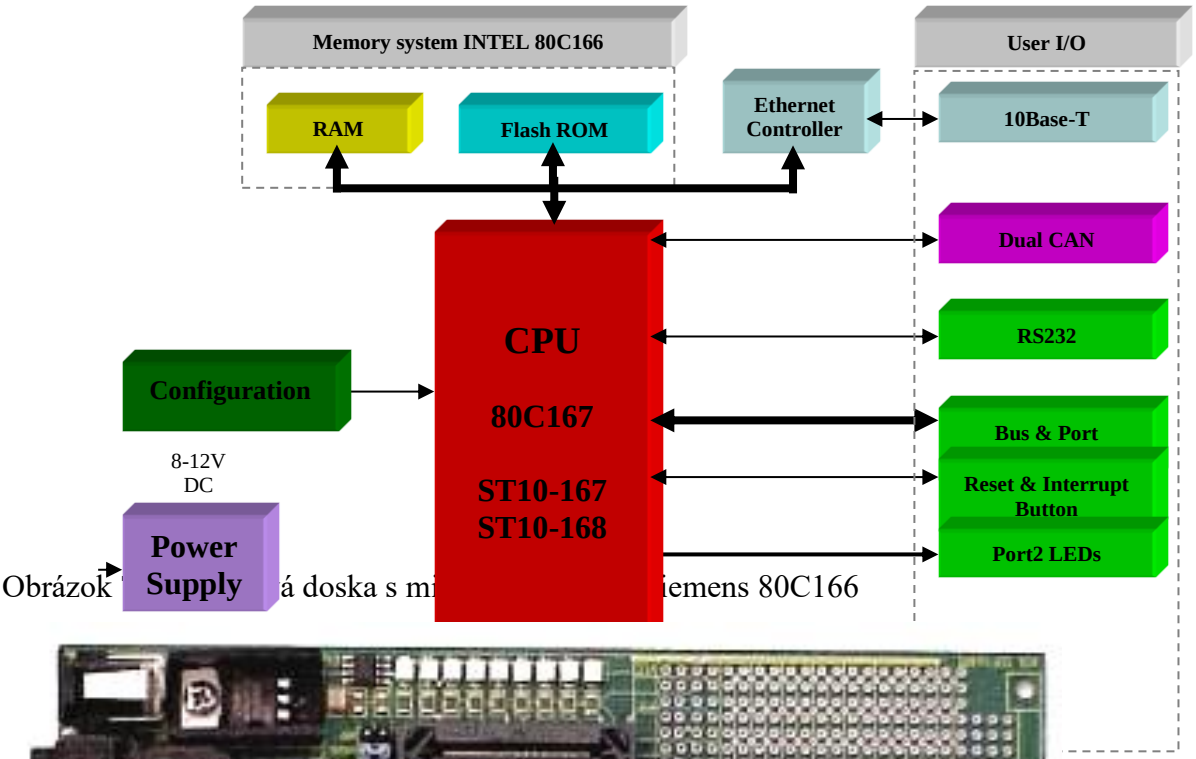


Obrázok 76, Vývojová doska s mikroprocesorom Siemens 80C517

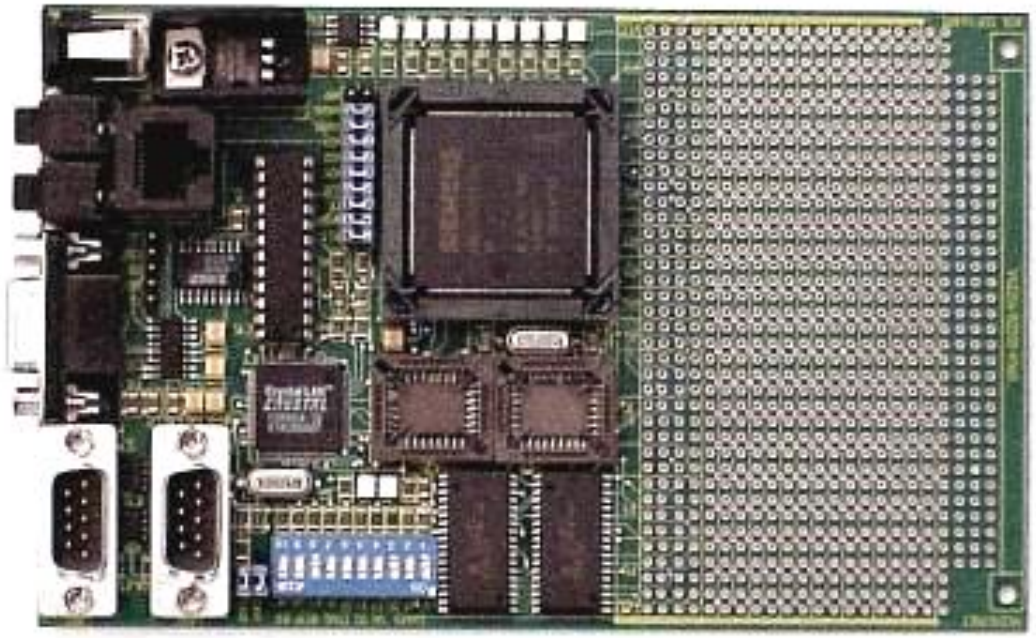


7.3. Hardware 80C166

Obrázok 77, Aplikačný model mikroprocesorového systému s 80C166

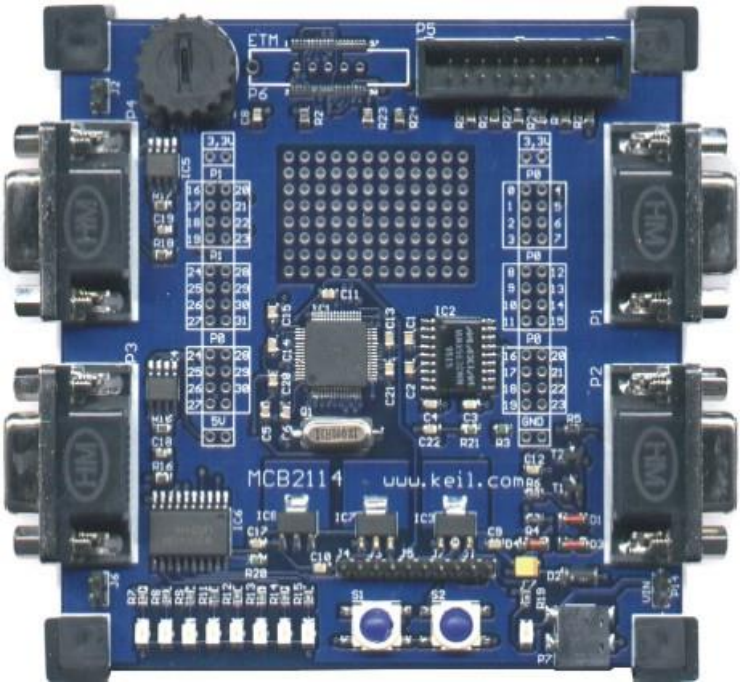


Obrázok á doska s mi iemens 80C166



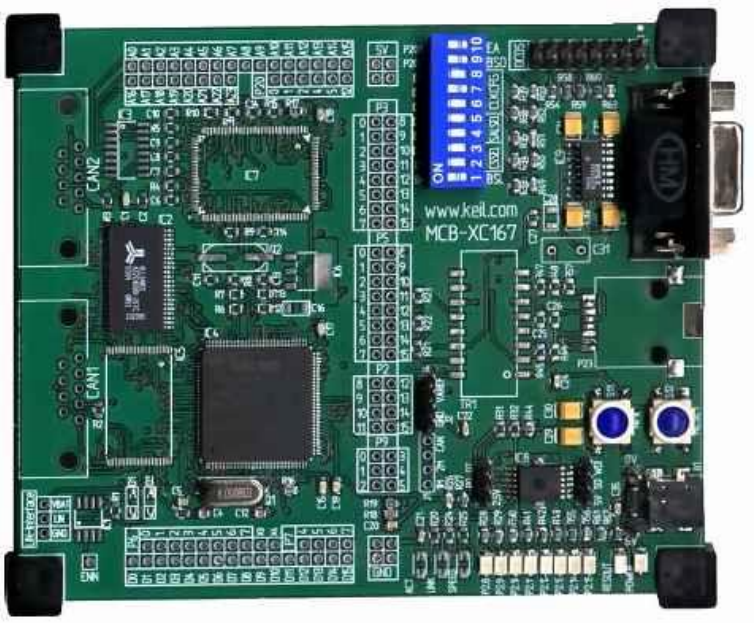
7.4. Hardware MCB2100

Obrázok 79, Vývojová doska s mikroprocesorom Philips LPC2129



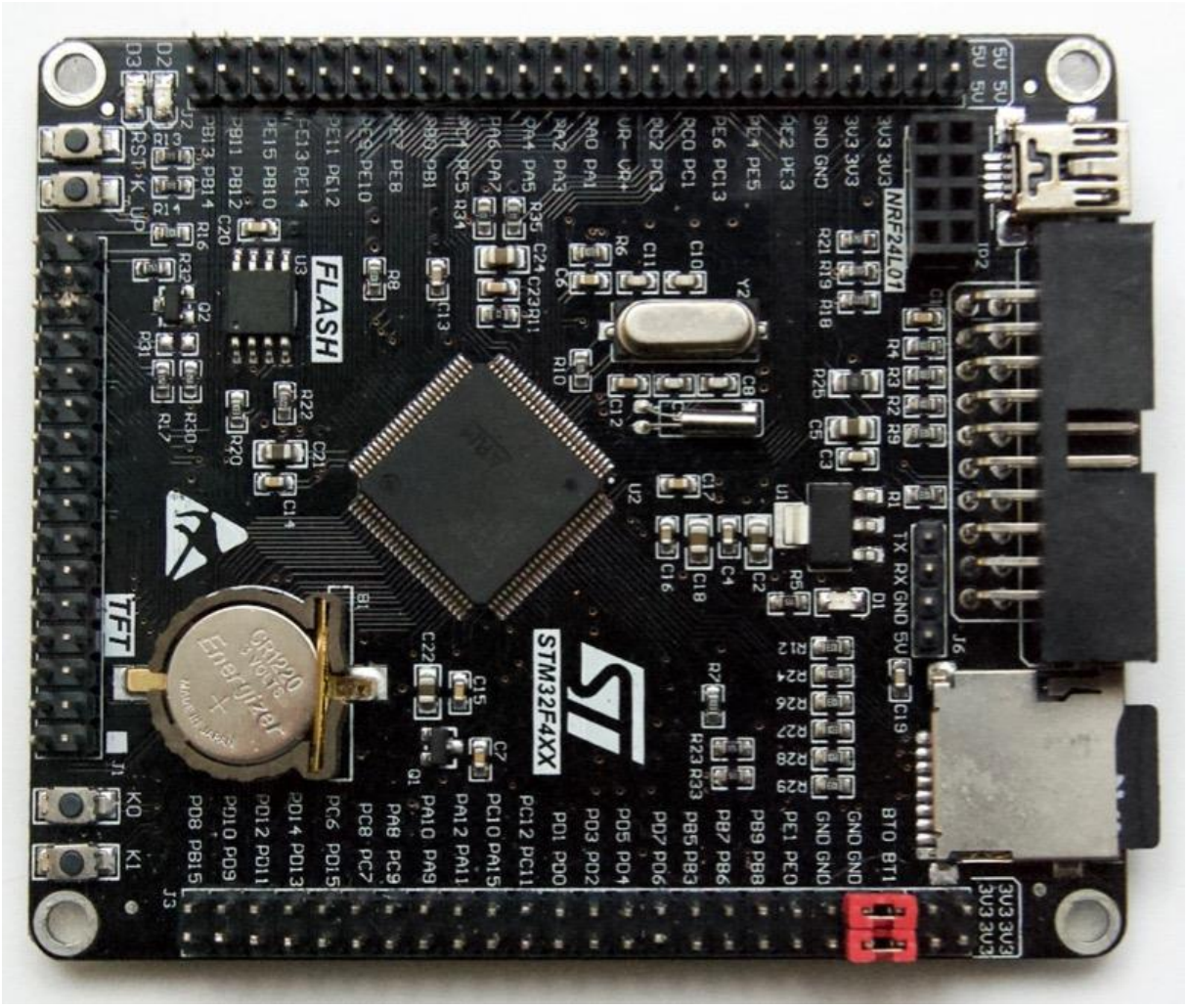
7.5. Hardware MCBXC167

Obrázok 80, Vývojová doska s mikroprocesorom Infineon XC167CI

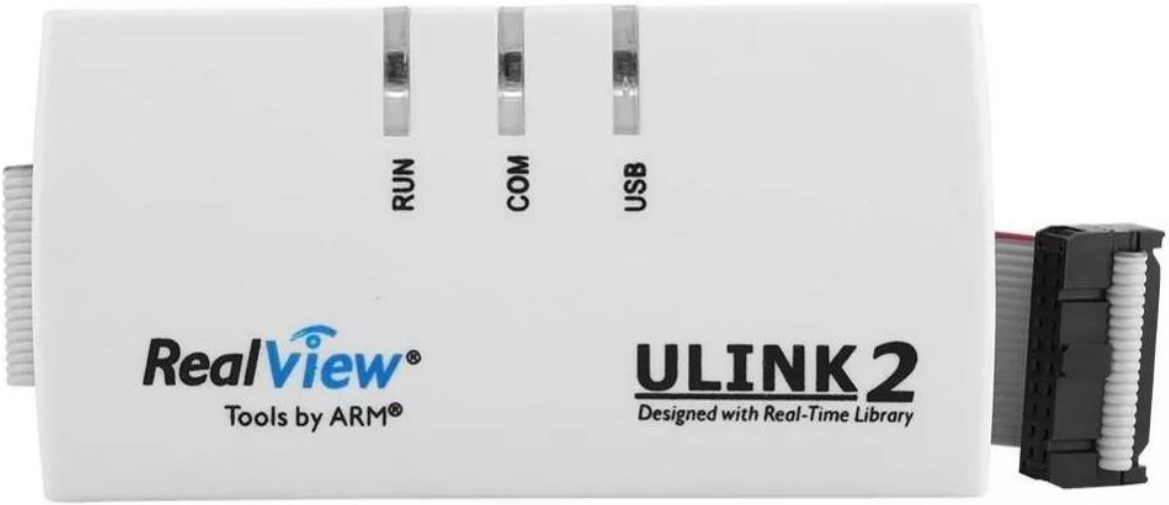


7.6. Hardware STM32F407VGTx

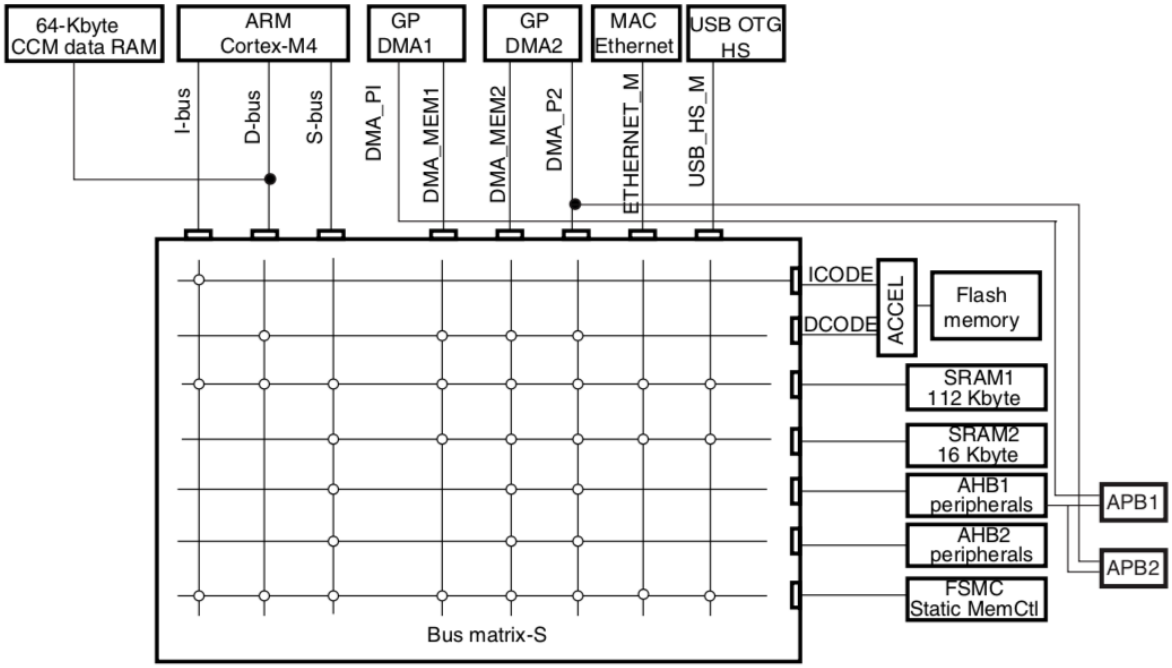
Obrázok 81, Vývojová doska s 32-bit DSP STM32F407VET6



Obrázok 82, Interface ULINK 2 pre platformu ARM



Obrázok 83, Architektúra pamäti Cortex-M4[77] STM32F407VET

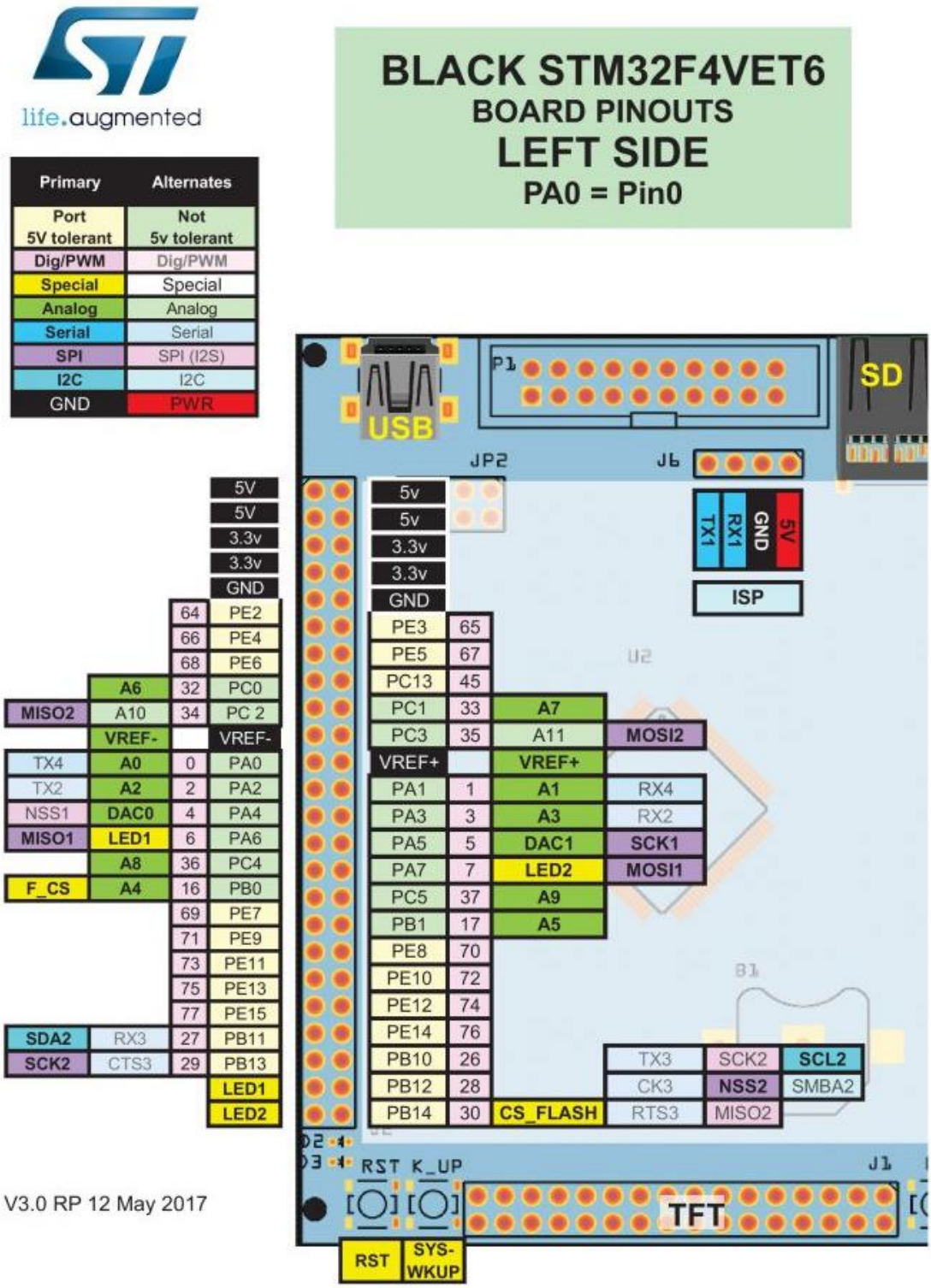


Zdroj obrázku: https://jeelabs.org/2018/z80-zexall-f407/

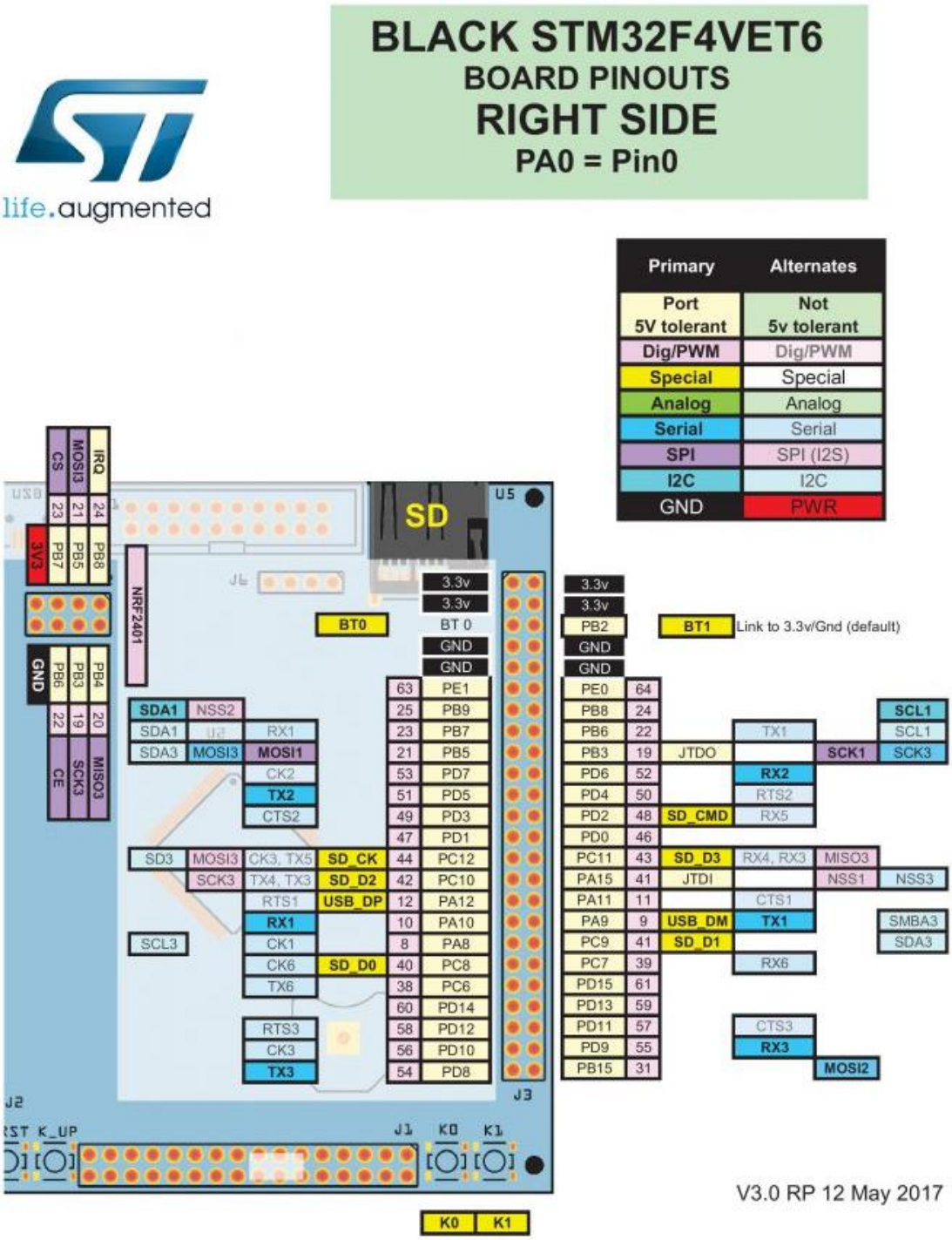
[77] STM má k dispozícii 196 KB of SRAM, ale je rozdelená do 4 oddelených častí

- 112 KB of "ordinary" on-chip SRAM
- 16 KB of "auxiliary" SRAM, contiguous with the 112 KB
- 64 KB of "core-coupled" SRAM, not contiguous with either of the above
- 4 KB "battery backup" SRAM, again not contiguous

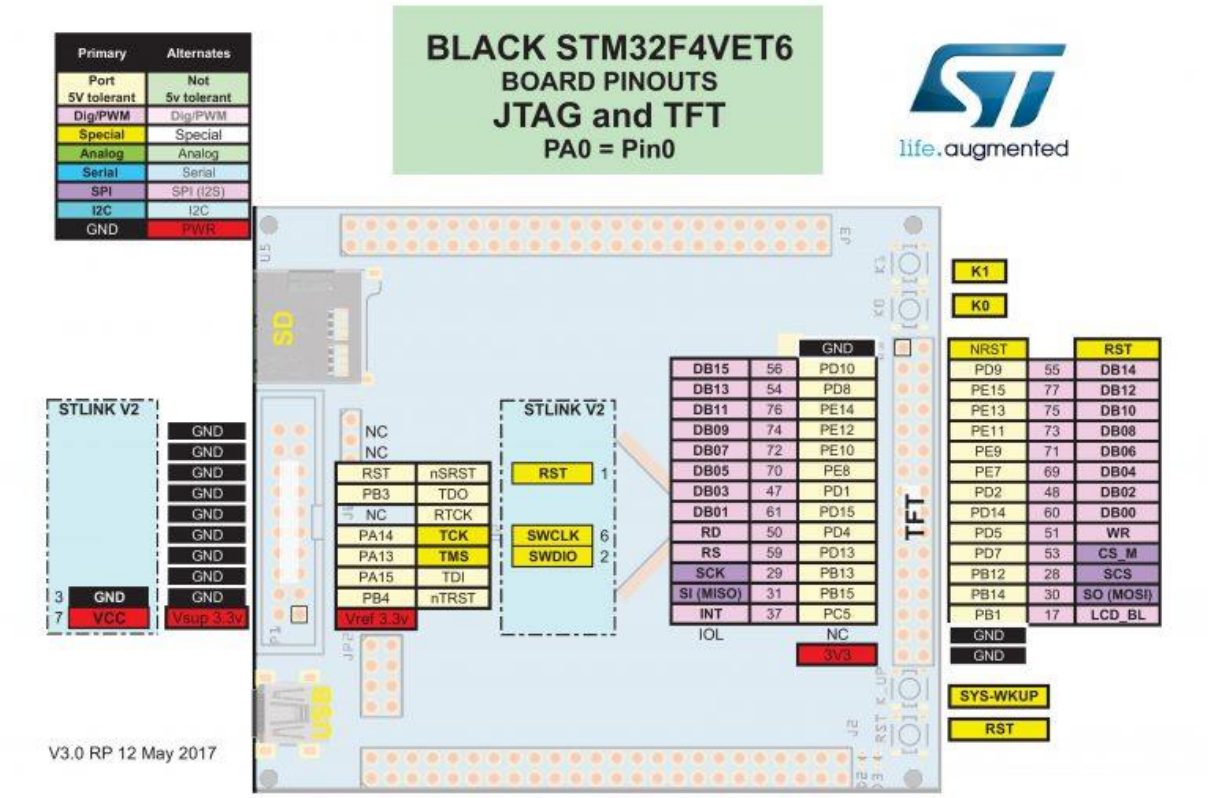
Obrázok 84, Rozloženie pinov na STM32F407VET



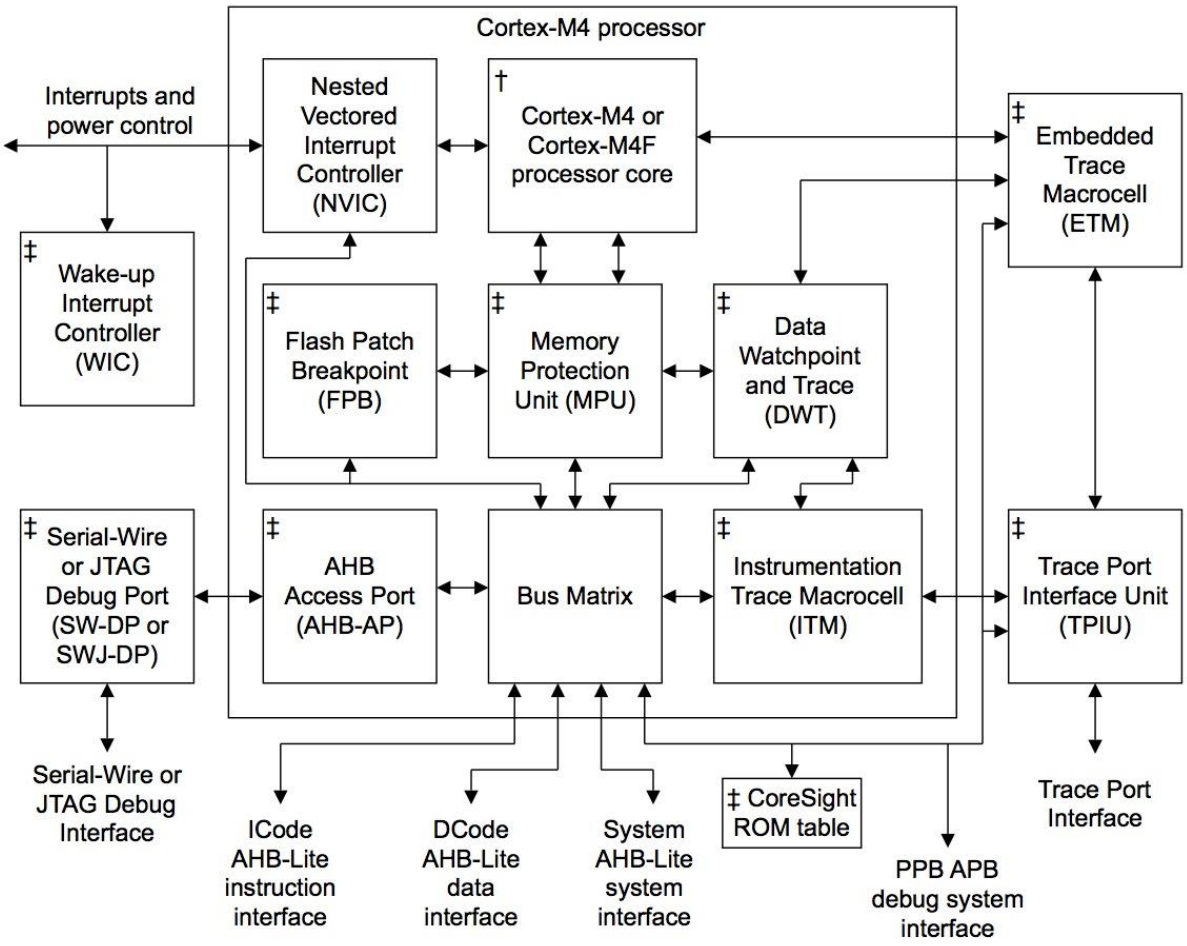
Obrázok 85, Rozloženie vývodov na STM32F407VET



Obrázok 86, Rozloženie vývodov na STM32F407VET

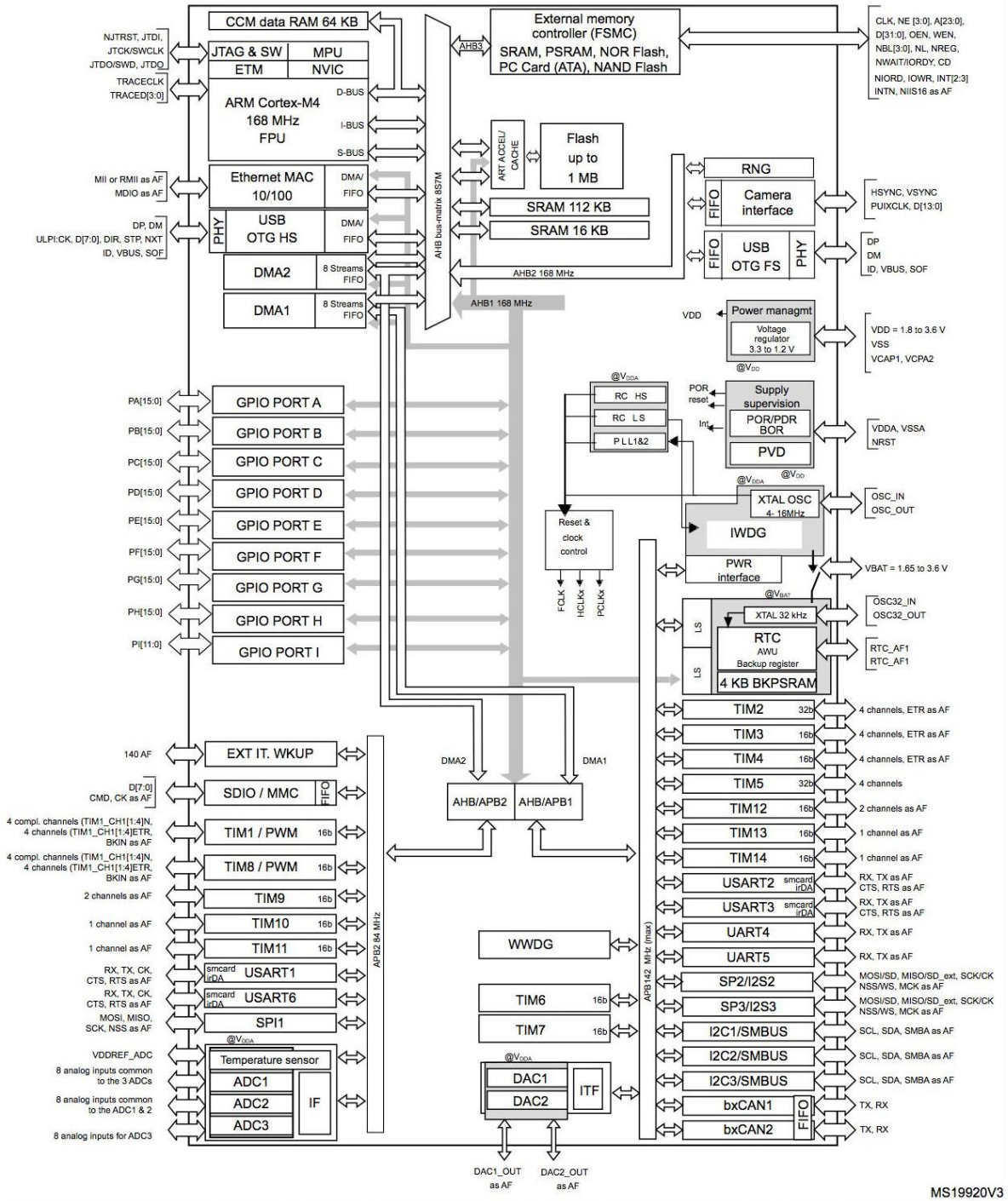


Obrázok 87, Bloková schéma CORTEX-M4



Zdroj obrázku: http://sciencezero.4hv.org/files/5/56/CortexM4BlockDiagram.jpg

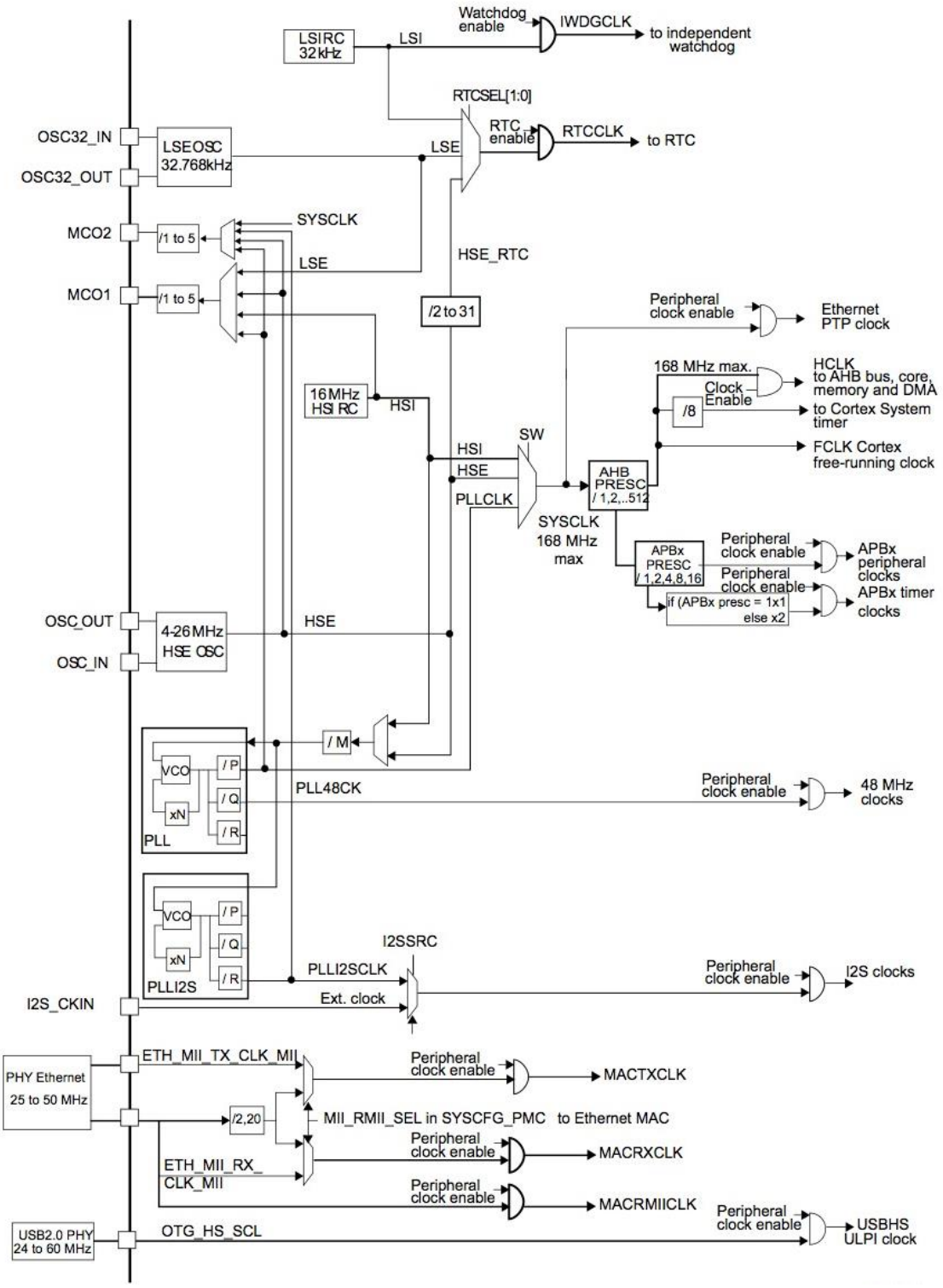
Obrázok 88, Schéma zapojenia dosky STM32F407VET BLACK



1. The timers connected to APB2 are clocked from TIMxCLK up to 168 MHz, while the timers connected to APB1 are clocked from TIMxCLK either up to 84 MHz or 168 MHz, depending on TIMPRE bit configuration in the RCC_DCKCFGR register.
2. The camera interface and ethernet are available only on STM32F407xx devices.

Zdroj obrázku: http://sciencezero.4hv.org/files/e/ee/STM32F4DeviceOverview.jpg

Obrázok 89, Schéma hodinového obvodu STM32F407VET



Zdroj obrázku: http://sciencezero.4hv.org/files/d/de/STM32F4_clock_tree.jpg[78]

Podrobnejšie informácie o architektúre CORTEX-M4 STM32F407 sú uvedené na adrese https://www.st.com/resource/en/programming_manual/dm00046982.pdf

[78] Bližšie informácie je možné nájsť na http://sciencezero.4hv.org/index.php?title=STM32F407_Microcontroller

8 Systémy pracujúce v reálnom čase s RTOS

Niektoré mikroprocesory majú pracovať v prostredí kde sa vyžaduje súčasné vykonávanie viacerých úloh. Dobrým príkladom je pravidelný zber dát z rôznych snímačov. Dáta môžu prichádzať do mikroprocesora v poradí ktoré môžeme nazvať chaotické. Snímače môžu poskytovať hodnotu nameranej veličiny mikroprocesoru ako binárne číslo, napätie, frekvencia, alebo šírka impulzu. Mikroprocesor v aplikácii má spracovať prijaté dáta zo snímačov, vykonať zásah v regulovanej sústave a prípadne zobraziť stav sústavy na zobrazovacej jednotke. Z vyššie uvedeného vidíme, že mikroprocesor musí vykonať viacero úloh za jednotku času.

Doteraz musel programátor venovať množstvo pozornosti cyklickej[79] kontrole výskytu očakávaných udalostí v uzavretej programovej slučke čo kládlo zvýšené nároky na schopnosti programátora vytvoriť efektívny kód.

S pomocou operačného systému RTX51 je možné veľmi jednoducho vytvoriť program ktorý sa bude skladať z požadovaného počtu elementárnych úloh pričom každej úlohe bude vopred predpísaná udalosť ktorá ju zaktivuje. Úlohy sú zároveň schopné vzájomnej výmeny dát, čo zaručuje ich kompaktnosť a konzistenciu, takže programátor môže venovať maximálnu pozornosť tvorbe obslužného programového kódu programu.

Reálne operačné systémy RTOS *(Real Time Operating System)* sú špeciálne operačné systémy vytvorené tak aby umožňovali rýchlu reakčnú dobu, spoľahlivosť a úsporné zaobchádzanie zo systémovými zdrojmi CPU.

RTOS rapídnym spôsobom urýchľuje vývoj aplikácií pretože obsahuje funkcie ktoré umožňujú mnoho vecí vykonávať jednoduchšie, rýchlejšie a efektívnejšie.

[79] Niekedy je tento spôsob obsluhy zariadení nazývaný ako „pooling“.

Tabuľka 3, Charakteristické vlastnosti RTOS od firmy KEIL

Description Part Number Parametric Specifications	**RTX51 FULL** FR51	**RTX51 TINY** Included with RX51
Round-Robin Multitasking	✔	✔
Preemptive Multitasking	✔	
Cooperative Multitasking	✔	✔
Timeout Events	✔	✔
Interval Events	✔	✔
Signal Events	✔	✔
Message Events	✔	
Semaphore Events	✔	
Memory Pools	✔	
Code Banking Support	✔	
CAN Libraries (for the Intel 82526 and 82527; Philips 82C200 and 8xC592; and Infineon 81C90, 81C91, C505C, and C515C)	✔	
Maximum Number of Defined Tasks	256	16
Maximum Number of Active Tasks	19	16
Required **CODE** Space	6K-8K bytes	900 bytes
Required **DATA** Space	40-46 bytes	7 bytes
Required Stack (**IDATA**) Space	20-200 bytes	3 bytes for each task
Required **XDATA** Space	650 bytes (minimum)	0 bytes
Timer Used	0, 1, or 2	0
System Clock Divisor	1,000-40,000 cycles	1,000-65,535 cycles
Interrupt Latency	Less than 50 cycles	Less than 20 cycles
Context Switch Time - Fast Task (Depends on Stack Load)	70-100 cycles	
Context Switch Time - Standard Task (Depends on Stack Load)	180-700 cycles	100-700 cycles
Task Priority Levels	4	1
Maximum Number of Semaphores (Binary)	8	
Maximum Number of Mailboxes	8	
Mailbox Size	8 entries	
Maximum Number of Memory Pools	16	

Tabuľka 4, Časovanie a vlastnosti RTOS AR166 Kernel V1.00 od firmy KEIL

Timing Specifications for AR166 Kernel V1.00

Function	Small	Compact	Medium	Large	HCompact	HLarge	XLarge
Initialize system (os_sys_init), start task	23.6	28.3	23.9	28.5	28.3	28.5	30.1
Create defined task, no task switch	8.0	10.3	8.0	10.3	10.3	10.3	12.2
Create defined task, switch task	11.6	14.3	11.8	14.5	14.3	14.5	16.0
Delete task (os_tsk_delete)	3.0	5.8	3.0	5.8	5.8	5.8	6.8
Task switch (by os_tsk_delete_self)	4.6	6.5	4.8	6.7	6.6	6.8	6.5
Task switch (by os_tsk_pass)	3.9	5.2	4.0	5.3	5.2	5.3	5.8
Task switch (upon set event)	4.7	6.3	5.1	6.7	6.3	6.7	7.1
Task switch (upon sent semaphore)	4.3	6.3	4.7	6.7	6.3	6.8	7.6
Task switch (upon sent message)	4.7	7.1	5.1	7.5	7.2	7.6	8.7
Set event (no task switch)	0.9	1.3	0.9	1.3	1.3	1.3	1.2
Send semaphore (no task switch)	0.6	1.0	0.6	1.0	1.0	1.0	1.0
Send message (no task switch)	1.1	1.8	1.1	1.8	1.8	1.8	2.5
Get own task identifier (os_tsk_self)	0.4	0.6	0.4	0.6	0.6	0.6	0.4
Interrupt response for ISR (with using-atr.)	0.3	0.3	0.3	0.3	0.3	0.3	0.2
Interrupt lockout for ISR's	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Interrupt latency for ISR's (response + lockout)	0.4	0.4	0.4	0.4	0.4	0.4	0.3

1. All times are in [µs], valid for XC16x CPU with the system clock 40 MHz, the code is executed from internal flash
2. The AR166 Kernel is configured for: 10 tasks, 10 user timers
3. These timings are measured with uVision3 debugger, simulation S166.DLL ver.: 2.42
4. Interrupt response for ISR's includes the ISR prolog as generated by C166. This prolog does the register bank switching (C166 interrupt procedure with "_using_" attribute). Note that C166 interrupt procedures have the same interrupt response time without AR166 Kernel.
5. Task switching includes swapping of the currently used stack contents. These measurements are done with 6-8 stack bytes.

Tabuľka 5, Charakteristické vlastnosti AR166 a RTX166Tiny

AR166 Kernel Technical Data

Description	AR166 Kernel	RTX166 Tiny
Number of tasks	256 max.	32 max.
Number of mailboxes	not limited	Not supported
Number of semaphores	not limited	Not supported
Number of signals	16 event flags per task	32 max.
Number of user timers	not limited	Not supported
RAM Requirements	Min. 500 Bytes	8 + 4 * number of tasks Bytes
Code requirements	Less than 4 KBytes	Less than 1.5 KBytes
Hardware Requirements	One on-chip timer (any timer may be used)	One on-chip timer (any timer may be used)
User task priorities	1 - 127	Not supported
Context switch time	Less than 25µsec @20MHz, 0 ws.	40 - 100 µsec
Interrupt lockout time	0.2 µsec	Less than 4 µsec, 0 waitstates

1. Items, which are flagged **not limited**, are not limited by the AR166 Kernel OS, however they are limited by the available system memory resources.
2. Signals are named **events** for AR166 Kernel. Each task has available 16 event flags to wait for.
3. Ram requirements depend on the numer of concurrent running tasks.

Tabuľka 6, Charakteristické vlastnosti[80] ARTX pre mikroprocesory ARM

ARTX Kernel Technical Data

Description	ARTX Kernel
Number of tasks	256 max.
Number of mailboxes	not limited
Number of semaphores	not limited
Number of signals	16 event flags per task
Number of user timers	not limited
RAM Requirements	Min. 500 Bytes
Code requirements	Less than 5 KBytes
Hardware Requirements	One on-chip timer (any timer may be used)
User task priorities	1 - 255
Context switch time	Less than 5µsec @60MHz, 0 ws.
Interrupt lockout time	1.8 µsec @60MHz, 0 ws.

1. Items, which are flagged **not limited**, are not limited by the ARTX Kernel OS, however they are limited by the available system memory resources.
2. Signals are named **events** for ARTX Kernel. Each task has available 16 event flags to wait for.
3. Ram requirements depend on the numer of concurrent running tasks.

[80] V prípade nových verzií môžu byť určité rozdiely.

Tabuľka 7, Prehľad vývojových nástrojov od firmy KEIL

Development Tools	Part Number PK51	DK51	CA51	A51
far Code/Data Support Dallas 390, Philips MX, Analog Devices ADuC812	✔			
µVision2 IDE	✔	✔	✔	✔
AX51 Ext. Macro Assembler	✔			
CX51 Ext. C Compiler	✔			
LX51 Ext. Linker	✔			
OHX51 Ext. OBJ-HEX Converter	✔			
A51 Macro Assembler	✔	✔	✔	✔
BL51 Code Banking Linker	✔	✔	✔	✔
OH51 OBJ-HEX Converter	✔	✔	✔	✔
OC51 Banked OBJ Converter	✔	✔	✔	
C51 ANSI C Compiler	✔	✔	✔	
µVision2 Debugger	✔	✔		
MON51 Target Monitor	✔	✔		
MON390 Target Monitor	✔			
ISD51 In-System Debugger	✔			
RTX51 Tiny Real-time Kernel	✔			

8.1. Opis funkcie RTOS

Aby sme pochopili význam RTOS[81], musíme si predstaviť aplikáciu ktorá má vykonávať súčasne niekoľko čiastkových úloh:

- ✓ regulovať pohon,
- ✓ snímať skutočnú hodnotu prúdu pohonu,
- ✓ snímať a vyhodnocovať teplotu pohonu,
- ✓ vykonávať funkciu regulátora pre pohon,
- ✓ obsluhovať klávesnicu a LCD display,
- ✓ komunikovať s okolím pomocou rozhrania RS232.

Z vyššie uvedeného je viditeľné, že je možné oddeliť funkciu regulátora pohonu do niekoľkých čiastkových úloh, alebo samostatných modulov. Efektívnejším spôsobom by bolo napísať program pre každú úlohu a tak vytvoriť požadované funkcie. Každý modul potom obhospodaruje samostatné premenné a funkcie, ktoré sú zapuzdrené a tak neviditeľné z okolia. Problém nastáva vtedy, ak potrebujeme vzájomne odovzdávať namerané dáta medzi jednotlivými modulmi ktoré sú spustené na rovnakej CPU. Práve v týchto činnostiach nám pomáha RTOS, kde je možné vopred definovať počet požadovaných úloh t.j. taskov[82].

RTOS potom spravuje všetky úlohy a ich systémové zdroje ako je pamäť, priority a dáta. Všetky úlohy spustené v RTOS dostanú od plánovača *(scheduler)* časový úsek v priebehu ktorého sú aktívne. Plánovač môže za niektorých okolností prerušiť vykonávanie úlohy ak je potrebné vykonať obsluhu prerušenia, alebo vykonať úlohu s vyššou prioritou, ale po vykonaní musí vrátiť CPU prerušenému procesu. S doplňujúcimi prostriedkami je možné vzájomne medzi úlohami komunikovať, synchronizovať a odovzdávať údaje.

[81] http://www.youtube.com/watch?feature=player_embedded&v=CGkl97Yghmo
[82] Task – úloha, elementárny a samostatný algoritmus procesu

8.2. Výhody použitia RTOS

- Encapsulácia - zapuzdrenie funkcií a dát v jednotlivých úlohách,
- Modulárna stavba, možnosť testovania a doplnenia systému o nové funkcie,
- Lepšia prehľadnosť a zrozumiteľnosť programu,
- Použitie funkcií v prostredí RTOS zabezpečí definované reakčné doby úloh,
- Spracovanie úloh je vykonávané vo vnútri definovaného časového úseku *(deterministické správanie sa)*,
- Všetky požiadavky na RTOS sú spracované v reálnom čase,
- Simultánne vykonávanie viacerých udalostí v reálnom čase,
- Celková absolútna spoľahlivosť systému.

Operačný systém[83] RTX51 od firmy KEIL môže v prípade potreby reagovať na vzniknutú udalosť v jednotkách ***µs***, alebo ***ms*** v súlade s požiadavkami programátora. Nemusí teda reagovať na vzniknutú udalosť čo nekonečne rýchlo[84], ale s presne definovaným časom reakcie.

8.3. Terminológia RTOS

Task je samostatná programová jednotka vykonávajúca vopred definovanú činnosť vo virtuálnom prostredí operačného systému. Každá úloha – task je v podstate nekonečná slučka tvorená v programovacom jazyku C príkazom ***while(1)***, alebo ***for(;;)***. Každý task môže RTOS individuálne ovládať. Task pre svoju činnosť v *multitaskovom* prostredí vyžaduje isté systémové zdroje *(pamäť, časovač, port, sériové rozhranie UART a iné)*. Task môže byť samostatný, alebo závislý na iných task-och *(synchronizácia s inými úlohami)*. Všetky task-y sa môžu nachádzať len v jednom z nasledujúcich stavoch: ***READY, BLOCKED, RUNNING, SLEEPING***.

Multitasking je moderná technológia zabezpečujúca simultánne vykonávanie viacerých úloh na jedinej CPU.

Round-Robin[85] s prideľovaním konštantných časových kvánt je v podstate kombinovaná metóda ktorá, umožňuje k deterministickému správaniu sa plánovača priradiť ešte plánovanie s riadením priorít. Stručne povedané ***timeslicing scheduling = Round-Robin***.

[83] http://www.keil.com/pr/article/1253.htm k dispozícii zdarma.
[84] Prerušenie sa nesmie stratiť, RTOS ktoré stratí prerušenie nie je použiteľné v aplikačnej praxi.
[85] Pri nastavení **?RTX_TIMESHARING EQU 1** t.j. Round-Robin Flag v konfiguračnom súbore **RTXSETUP.INI** nie je potrebné používať príkazy **os_wait(K_TMO+K_SIG,100,NULL)** operačného systému

Ručný **Round-Robin** odovzdáva vykonávanú úlohu samostatne CPU pre iné task-y v stave ***READY*** ktoré majú rovnaké priority. Úloha týmto samotná určuje veľkosť časového kvanta nazývaný niekedy ako ***kooperatívny multitasking***[86].

Preempting je termín ktorý vyjadruje násilné ukončenie task-u s nižšou prioritou task-om ktorý má vyššiu prioritu a vykonávaný task je uvedený do stavu ***RUNNING*** a prerušený do stavu ***READY***. Po ukončení task-u s vyššou prioritou je task s nižšou prioritou zo stavu ***READY*** prepnutý do stavu ***RUNNING***.

Inverzia priorít je jednou z hlavných úloh plánovača, ktorý má prideľovať časové kvantum pre jednotlivé task-y v závislosti na pridelenej priorite. Ak by však task **A** s vysokou prioritou čakal na uvoľnenie systémového zdroja ktorý môže byť uvoľnený len task-om **B** ktorý má nižšiu prioritu, môžeme vidieť, že task **B** by nemusel byť vzhľadom na vyššiu prioritu task-u **A** prepnutý do stavu ***RUNNING***. Preto niektoré RTOS sú vybavené mechanizmom, ktorý umožňuje vzájomnú *inverziu priorít*, ktorá uvedie task **B** z nižšou prioritou na krátky čas do vyššej priority task-u **A,** a zostane v tomto stave pokiaľ nebude task **A** blokovaný task-om **B**. Potom je uvedený task **B** do svojej pôvodnej úrovne priority pred inverziou.

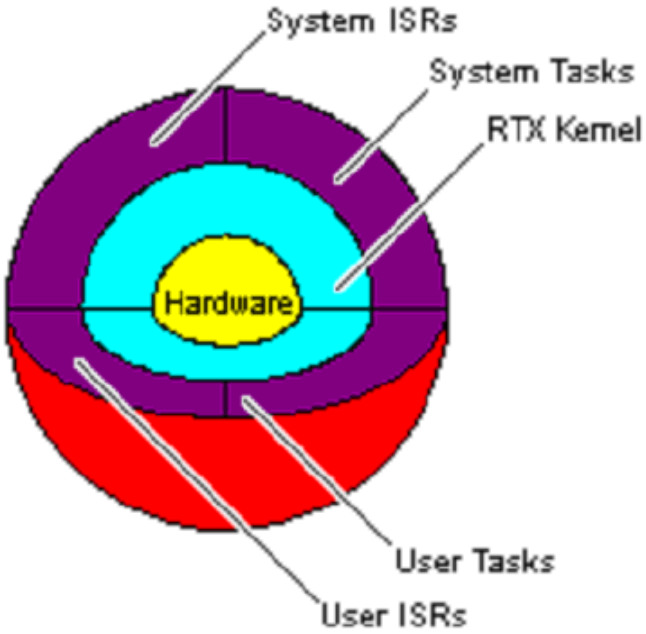
8.4. Funkcia plánovača – scheduler

Scheduler, alebo povedané plánovač je najdôležitejším prvkom reálneho operačného systému RTOS. Svojou činnosťou zabezpečuje riadenie úloh tzv. *task management* a zaisťuje, aby CPU bola pridelená úlohe s najvyššou prioritou. Všetky tieto informácie sú uložené v riadiacom bloku *(Task Block Control)*. Na vlastnostiach plánovača závisia reakčné doby a výkonnosť RTOS.

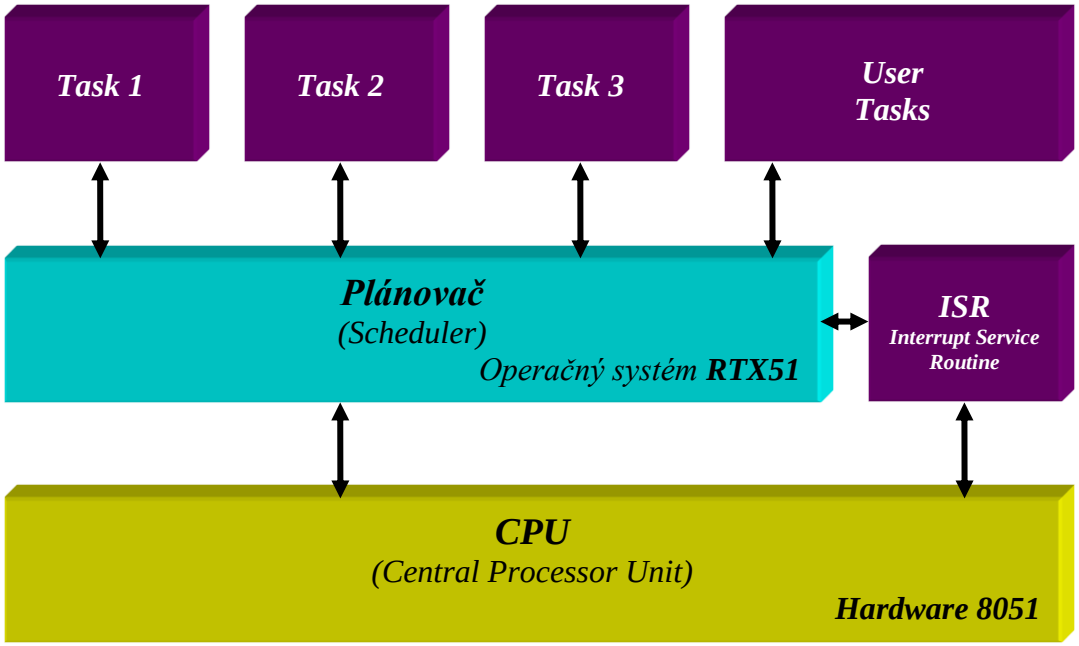
RTX51 Tiny, alebo RTX51 Full. Operačný systém dokáže prepnúť úlohy aj bez použitia príkazu **os_wait(K_TMO+K_SIG,100,NULL)**.

[86] Kooperatívny multitasking nutne **vyžaduje** použitie príkazov **os_wait()**, alebo **os_task_switch()**, pričom len **RTXCompletion=os_wait(K_TMO+K_SIG+K_INT+K_MBX+0x00,200,&Message)** je schopný v premennej **RTXCompletion** vrátiť typ udalosti ktorá spôsobila ukončenie čakania na udalosť. Vzniknutá udalosť zruší čakanie na ostatné udalosti, takže ak systém RTX čakal na vznik udalosti K_TMO a zaregistruje sa výskyt udalosti príchodu prerušenia K_INT tak sa opäť nastaví aktívne čakanie na výskyt naprogramovaných udalostí. Round-Robin je možné využiť a napísať tak jednotlivé task-y bez použitia **os_wait()**, alebo **os_task_switch()**. Plánovač v režime Round-Robin je schopný tieto task-y prepínať v pomerne krátkom čase, ktorý sa dá v súbore RTXSETUP.INI nastaviť, takže výsledný program sa správa ako „multitaskový“.

Obrázok 90, Všeobecný model RTOS



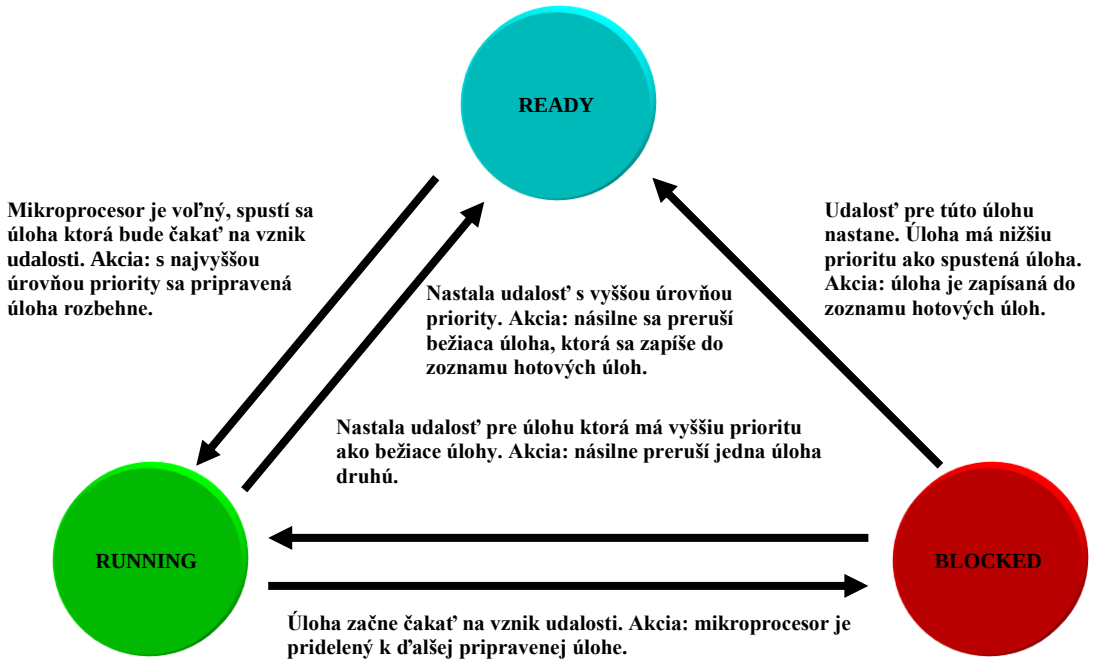
Obrázok 91, Bloková schéma systému reálneho času



Plánovač riadi:

- Správu systémových zdrojov, pamäti, časových kvánt prideľovaných jednotlivým task-om.
- Riadenie prenosu dát medzi jednotlivými úlohami pomocou semaforov, frontami správ *(Messages Queues)*.
- Aktívne riadenie taskov *(pozastavenie, spustenie, prepnutie, nútené prerušenie)* .
- Obsluhu prerušenia, na základe ktorého je možné vykonať prepnutie taskov, prerušenie musí byť obslúžené a nesmie sa stratiť. RTOS, ktoré stratia prerušenie nie sú schopné prevádzky v aplikáciách v reálnom čase.

Obrázok 92, Mechanizmus prepínania úloh v RTX51



8.5. Štandard CMSIS-RTOS pre ARM architektúry

CMSIS-RTOS verzie 2, ďalej už len (CMSIS-RTOS2) poskytuje generické rozhranie RTOS pre zariadenia založené na procesore ARM® Cortex®. Poskytuje štandardizované rozhranie API pre softvérové súčasti, ktoré vyžadujú funkčnosť RTOS, a preto prináša používateľom a softvérovému priemyslu podstatné výhody.

CMSIS-RTOS2[87] poskytuje základné funkcie, ktoré sú potrebné v mnohých aplikáciách. Zjednotený súbor funkcií CMSIS-RTOS2 znižuje potrebu štúdia a zjednodušuje zdieľanie softvérových komponentov. Komponenty tzv. middleware, ktoré používajú CMSIS-RTOS2, sú agnostické RTOS a teda sú aj ľahšie prispôsobiteľné. Štandardné šablóny projektov CMSIS-RTOS2 môžu byť dodávané s voľne dostupnými implementáciami CMSIS-RTOS2.

Aplikácie často vyžadujú vykonávať niekoľko súbežných činností naraz. CMSIS-RTOS2 dokáže spravovať viac súbežných aktivít v čase, keď sú potrebné. Každá aktivita dostane samostatné vlákno[88], ktorá vykoná špecifickú úlohu čím zjednodušuje celkovú štruktúru programu. Systém CMSIS-RTOS2 je plne škálovateľný a potrebné vlákna sa dajú ľahko pridať kedykoľvek neskôr. Vlákna majú programátorom nastavenú prioritu umožňujúcu rýchlejšiu realizáciu časovo kritických častí používateľskej aplikácie.

CMSIS-RTOS2 ponúka služby potrebné pre aplikácie ktoré sa majú vykonávať v reálnom čase, napríklad periodická aktivácia funkcií systémového časovača, správa pamäte a výmena správ medzi vláknami s časovými obmedzeniami.

[87] CMSIS-RTOS API verzia 2 definuje minimálny súbor funkcií použiteľných programátorom. Implementácie RTOS s rozšírenými funkciami môžu výrobcovia RTOS individuálne podľa svojich potrieb dopĺňať. CMSIS-RTOS2 spravuje systémové zdroje mikroprocesorov a umožňuje jednoduchú implementáciu koncepcie paralelných vlákien, ktoré sa v mikroprocesore vykonávajú súčasne. Od verzie MDK 5.28 je potrebné zapnúť v kompilátore režim C99, ináč nie je možné program preložiť. Bližšie informácie sú uvedené na http://www2.keil.com/mdk5/528.

[88] Myslíme tým niť, vlákno, thread.

CMSIS-RTOS2 komplexne rieši tieto požiadavky programátora:

- Vytváranie dynamických objektov už nevyžaduje statickú pamäť, statické pamäťové vyrovnávacie pamäte sú teraz programátorom voliteľné.
- Podpora architektúry ARMv8-M, ktorá poskytuje bezpečný a nezabezpečený stav vykonávania programového kódu.
- Ustanovenia pre odovzdávanie správ vo multijadrových systémoch.
- Plná podpora prostredia C run-time.
- C rozhrania, ktoré je kompatibilné so kompilátormi kompatibilnými s ABI.

Zmeny CMSIS-RTOS2 oproti predchádzajúcim verziám:

- Funkcie **osXxxxNew** nahrádzajú funkcie osXxxxCreate; osXxxxNew a osXxxxDelete vytvoriť a uvoľniť objekty.
- Hlavná funkcia **main()** už nie je spustená ako vlákno (to bola voliteľná funkcia v CMSIS-RTOS v1). Funkcie, vracajúce prípadný **osEvent**, boli nahradené inými.

CMSIS-RTOS2[89] poskytuje prekladovú vrstvu do CMSIS-RTOS v1. Pomocou CMSIS-RTOS C API v2 a CMSIS-RTOS C API v1 je možné spájať rovnakú aplikáciu. Postupom času môžete migrovať na nové rozhranie API, ako je vysvetlené v časti „Migrácia“ z API v1 na API v2.

Nasledujúce časti poskytujú ďalšie podrobnosti o implementácii CMSIS-RTOS2 a referenčnej implementácii RTX.

- História revízií dokumentuje zmeny vykonané v každej verzii pre CMSIS-RTOS v2 a RTX v5.
- Generické rozhranie RTOS poskytuje prehľad o rozhraniach rozhrania API dostupných v systéme CMSIS-RTOS v2.
- Prehľad funkcií obsahuje zoznam funkcií CMSIS-RTOS2 API a súboru hlavičky cmsis_os2.h.
- Validácia RTOS opisuje súbor validácie, ktorý je verejne dostupný.
- Migrácia z API v1 na API v2 ukazuje, ako používať CMSIS-RTOS2 v existujúcich projektoch a zoznam funkčných rozdielov na CMSIS-RTOS v1.

[89] CMSIS-RTOS2 nie je kompatibilný s POSIX-om, ale obsahuje ustanovenia umožňujúce podľa potreby využiť rozhranie C++11/C++14.

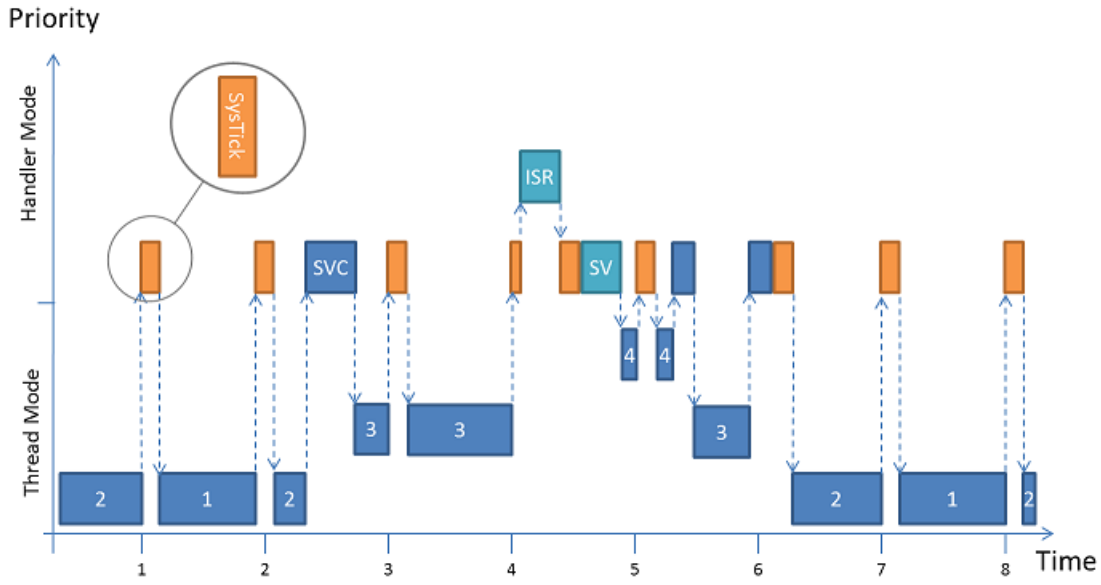
- RTX v5 implementácia poskytuje všeobecné informácie o prevádzke a používaní RTX v5.

8.6. Model plánovača v RTX5 implementovaný v CMSIS

RTX5 implementuje „preemptívny“ plánovač s nízkou časovou odozvou na vzik udalosti. Hlavné časti RTX5 sa vykonávajú v režime obsluhy, ako je napr.:

- **SysTick_Handler** sa používa na časové plánovanie.
- **SVC_Handler** sa používa na plánovanie založené na uzamknutí udalosti.
- **PendSV_Handler** sa používa na plánovanie založené na prerušeniach.

Obrázok 93, Plánovanie a prerušenie vykonávania vlákna v RTX5



Plánovač kombinuje prepínanie kontextu založené na prioritách a na plánovaní typu round-robin. Príklad zobrazený na obrázku vyššie obsahuje štyri vlákna (1, 2, 3 a 4). Vlákna 1 a 2 majú rovnakú prioritu, vlákno 3 má vyššiu prioritu a vlákno 4 najvyššiu (*osThreadAttr_t :: priority*). Pokiaľ sú vlákna 3 a 4 zablokované, plánovač prepína medzi vláknom 1 a 2 na základe prideleného časového kvanta (round-robin). Pridelené časové kvantum pre plánovač s môže byť nastavené[90] v konfigurácii systému.

[90] RTX_Config.h v Round-Robin Timeout implicitne nastavené na hodnotu 5.

Vlákno 2 odblokuje vlákno 3 ľubovoľným volaním RTOS (uskutočneným v režime obsluhy SVC) v časovom indexe 2. Plánovač sa prepne na vlákno 3 okamžite, pretože vlákno 3 má najvyššiu prioritu. Vlákno 4 je stále zablokované.

V čase 4 index nastane prerušenie (ISR) preemptívne do **SysTick_Handler**. RTX okamžite vykoná prerušenie. Rutina ISR používa volanie RTOS, ktoré odblokuje vlákno 4. Namiesto okamžitého prepnutia na vlákno 4 je nastavený príznak **PendSV** tak, aby odložil prepínanie kontextu. **PendSV_Handler** sa vykoná ihneď po návrate **SysTick_Handler** a vykoná sa odložený kontext na vlákno 4. Akonáhle vlákno 4 s najvyššou prioritou blokuje, vykoná sa volanie RTOS okamžite späť počas vlákna 5.

V čase 5 používa vlákno 3 tiež blokujúce volanie RTOS. Plánovač teda prepne späť na vlákno 2 pre časový index 6. V časovom indexe 7 plánovač používa mechanizmus round-robin pre prepnutie na vlákno 1 a tak ďalej.

8.7. Objektová koncepcia návrhu CMSIS-RTOS aplikácie

Všetky objekty[91] RTOS majú spoločnú koncepciu návrhu. Celkový životno-pracovný cyklus[92] objektu môže byť zhrnutý ako: **vytvorený** -> **spustený** -> **uvoľnený**.

8.7.1. Vytvorenie objektu

Objekt je vytvorený volaním funkcie **osXxxNew**. Nová funkcia vracia identifikátor, ktorý sa dá použiť s novým objektom. Skutočný stav objektu je typicky uložený v riadiacom bloku špecifickom pre daný objekt. Veľkosť a rozmiestnenie potrebnej pamäti pre riadiaci blok je špecifický pre každú implementáciu.

Za účelom zaistenia kontroly nad objektovo špecifickými možnosťami všetky funkcie **osXxxNew** poskytujú voliteľný argument **attr**, ktorý môže byť predvolene ako NULL. Volaním predchádzajúcej funkcie **osXxxNew** získame ukazovateľ na štruktúru atribútov daného objektu, alebo tasku:

- **name** umožňuje programátorovi nastaviť čitateľné meno pre objekt,
- **attr_bits** ovládanie objektovo špecifických vlastností,
- **cb_mem** manuálne nastaviť pamäť pre riadiaci blok,
- **cb_size** nastaví veľkosť pamäte poskytnutú pre riadiaci blok,
- **stack_mem** ukazovateľ na polohu zásobníka objektu,
- **stack_size** veľkosť pamäte pre zásobník, nie menej ako 200 Byte,
- **priority** určuje prioritu objekt, tasku v systéme RTOS
- **tz_module** používa sa len v prípade ARMv8
- **reserved** musí byť nastavená 0

Atribút **name** sa používa iba na identifikáciu objektu, napr. pomocou ladenia RTOS-aware. Na iné účely sa nepoužíva.

Atribúty **cb_mem** a **cb_size** sa môžu použiť na pridelenie pamäte riadiaceho bloku manuálne ak je potrebné prideliť iné množstvo pamäti ako implicitne pridelí systém. Treba sa ale uistiť, že množstvo pamäte, na ktoré odkazuje **cb_mem**, je dostatočné. Ak veľkosť pamäte uvedená v

[91] Nemusia byť len tasky, použité v RTX v5.4.0
[92] life-cycle

cb_size nie je dostatočná, funkcia **osXxxNew** sa vráti chybu, t.j. hodnotu **NULL**. Manuálne prideľovanie potrebnej pamäte je menej efektívne, preto vo väčšine prípadov vystačíme s pamäťou ktorú pridelí systém RTOS.

8.7.2. Použitie objektu

Potom, čo bol objekt úspešne vytvorený, môže byť použitý až po jeho uvoľnenie. Akcie definované pre objekt závisia od jeho typu. Obvykle každá použitá funkcia **osXxxDoSomething** vyžaduje odkaz na objekt odovzdaného v prvom parametri **xxx_id**.

Použitá funkcia predpokladá použitie určitej kontroly zdravia[93] na parametri **id**. Okrem toho sa tým overí konkrétny typ objektu, t.j. nie je možné zavolať prístupové funkcie jedného typu objektu s odkazom na iný typ objektu.

Všetky ďalšie kontroly parametrov, ktoré sa aplikujú, sú objektovo špecifické, konkrétne alebo úzko špecifické pre následnú implementáciu. Preto je nevyhnutné aby programátor skontroloval návratové hodnoty vykonaných funkcií cez **osErrorParameter**, aby sme zistili chybový stav.

Niektoré funkcie nie je možné použiť z obsluhy prerušenia **ISR**. To zahŕňa napr. funkcie **osXxxWait** (a podobné) obmedzené na volanie s parametrom **timeout** nastaveným na 0.

8.7.3. Uvoľnenie objektu

Objekty, ktoré už nie sú potrebné, môžu byť na požiadanie uvoľnené, aby sa uvoľnila pamäť riadiaceho bloku. Takto môžeme predpokladať, že ID objektu bude platné, kým **osXxxDelete** nebude explicitne nazvaný. Funkcia uvoľnenia objektu uvoľní použitú pamäť riadiaceho bloku. V prípade pamäte riadiaceho bloku manuálne nastaveného programátorom, pamäť musí byť tiež manuálne uvoľnená.

Jediná výnimka, o ktorú sa musí postarať, sú vlákna, ktoré nemajú explicitnú funkciu **osThreadDelete**. Vlákna môžu byť buď odpojené[94], alebo pripojiteľné[95]. Oddelené vlákna sa pri ukončení automaticky zničia, t.j. volanie **osThreadTerminate** alebo **osThreadExit** alebo návrat z vlákna. Na druhej strane spájateľné vlákna sú udržiavané spustené, kým nie je explicitne volaný **osThreadJoin**.

[93] sanity, health ...
[94] použiteľné s atribútom osThreadDetached
[95] použiteľné s atribútom osThreadJoinable

Obrázok 94, Štruktúra atribútov objektu osThreadNew

```
141  const osThreadAttr_t admin_app_attr=
142  {
143    .name="admin_app_attr",
144    .attr_bits=osThreadDetached,
145    .cb_mem=&admin_app_thread_tcb,
146    .cb_size=sizeof(admin_app_thread_tcb),
147    .stack_mem=&admin_app_thread_stk[0],
148    .stack_size=sizeof(admin_app_thread_stk),
149    .priority=osPriorityNormal,
150    .tz_module=0U,
151    .reserved=0U
152  };
```

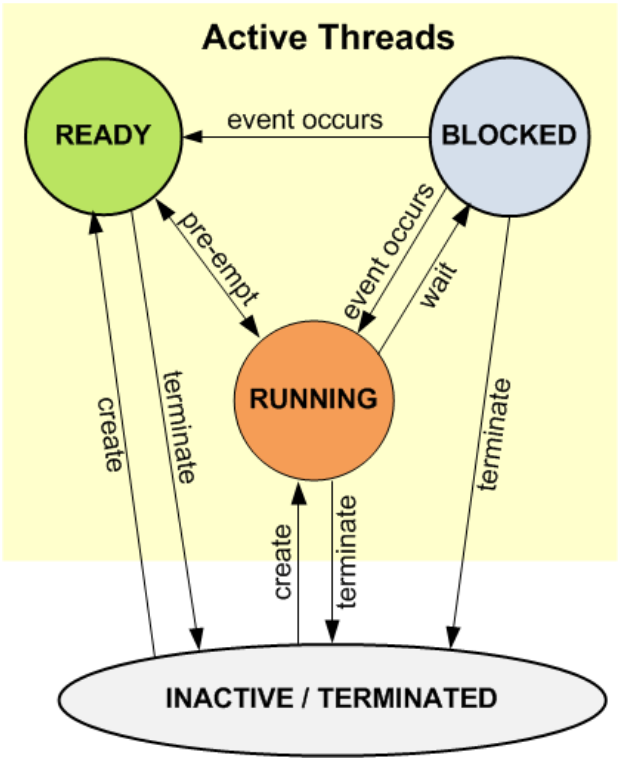
Volanie funkcie **osThreadNew**[96] vykonáme pomocou volania nasledujúceho riadku:

th0=osThreadNew(app_main,&param0,&admin_app_attr);

alebo volaním **osThreadNew** s parametrami[97] NULL

th0=osThreadNew(app_main,NULL,NULL);

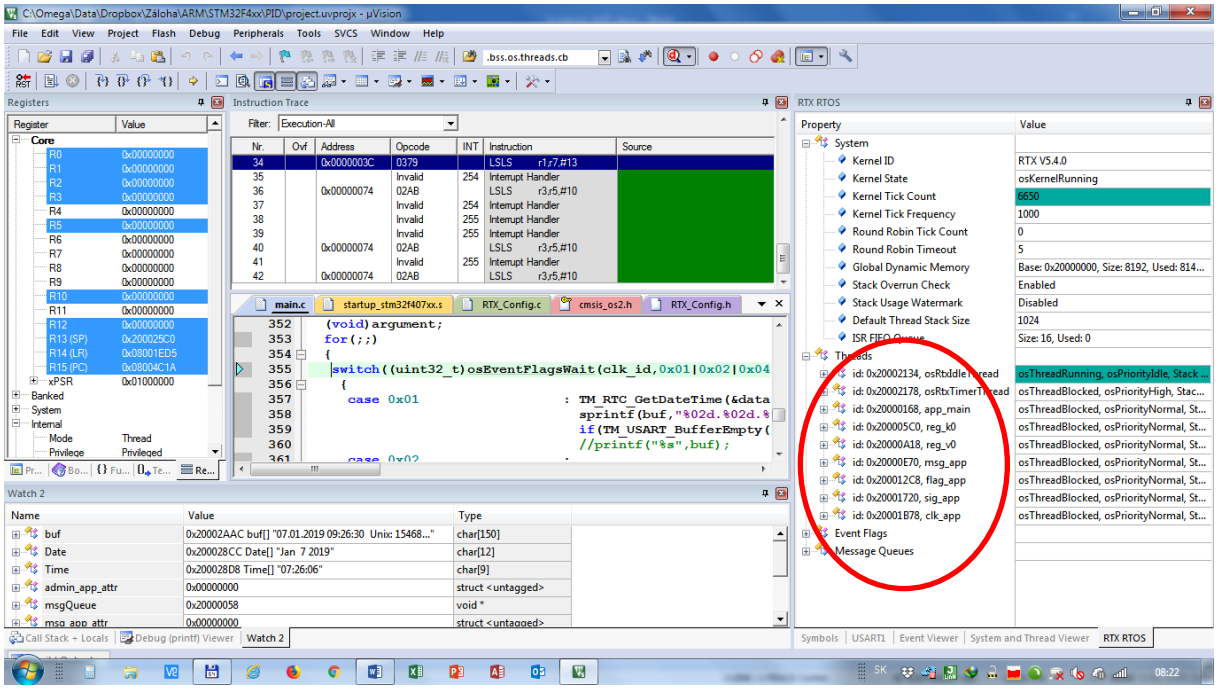
Obrázok 95, Mechanizmus prepínania úloh v CMSIS-RTOS2 API



[96] Pri tomto volaní, kde je programátorom nastavená štruktúra s atribútmi objektu z doteraz nezistených príčin sa nezobrazujú spustené objekty v RTOS simulátore v µVision5. Program funguje bez obmedzení, ale nie sú zobrazované súčasne spustené task-y v RTOS okne.

[97] Oproti predchádzajúcemu volaniu funkcie **osThreadNew** v simulátore RTOS v µVision5 správne zobrazuje spustené objekty (tasks, messages, semaphores), pravdepodobne je to tým, že jednotlivým objektom sa implicitne pridelia systémové prostriedky t.j. pamäť rovnako.

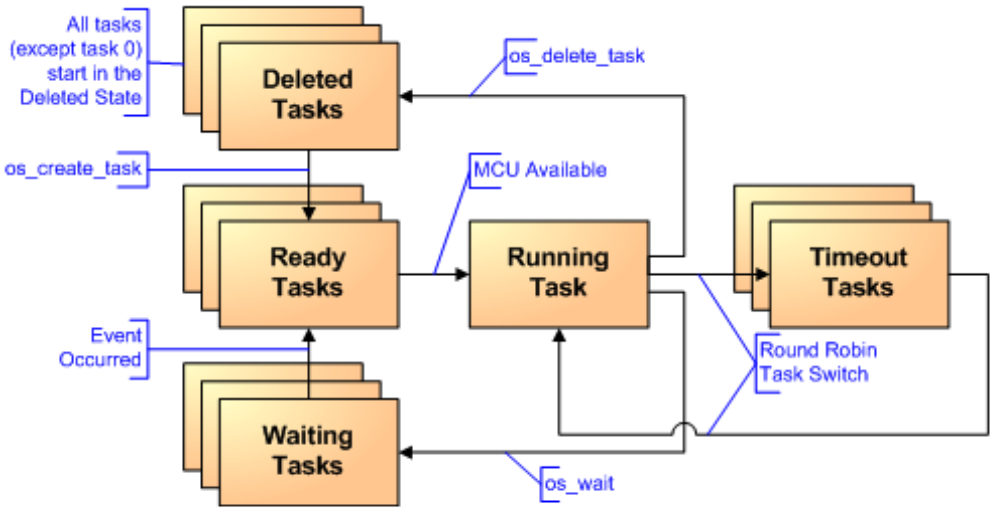
Obrázok 96, Ladenie programu CMSIS-RTOS2 s API funkciami



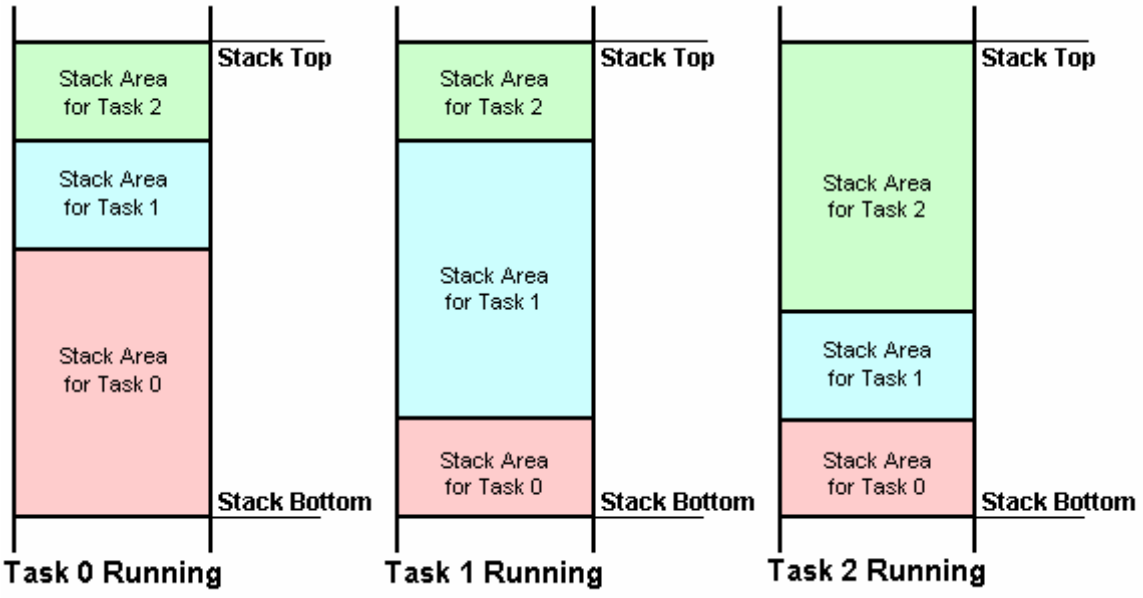
8.8. Spôsob práce plánovača v RTX51

- K prepnutiu úloh dochádza ak je ukončené vykonávanie úlohy s najvyššou prioritou a pokiaľ sa nenachádza v stave ***READY***.
- Ak sa v systéme nachádza viac čakajúcich úloh s rovnakou úrovňou priority v stave ***READY***, je spustená najdlhšie zaradená čakajúca úloha.
- K prepnutiu úlohy dochádza vtedy, ak nie je splnená prvá podmienka.
- K časovaniu pomocou ***round-robin (time-sliced)*** dochádza vtedy, ak sú splnené nasledujúce podmienky:
  a. Musí byť povolené plánovanie ***round-robin***.
  b. Vykonávaná úloha má najnižšiu prioritu 0 a nevykonáva aritmetickú operáciu s reálnym údajovým typom *(float)*.
  c. Minimálne jedna úloha s prioritou 0 musí byť v stave ***READY***.
  d. Musia byť nastavené časové kvantum funkciou ***os_set_slice(Number)***.

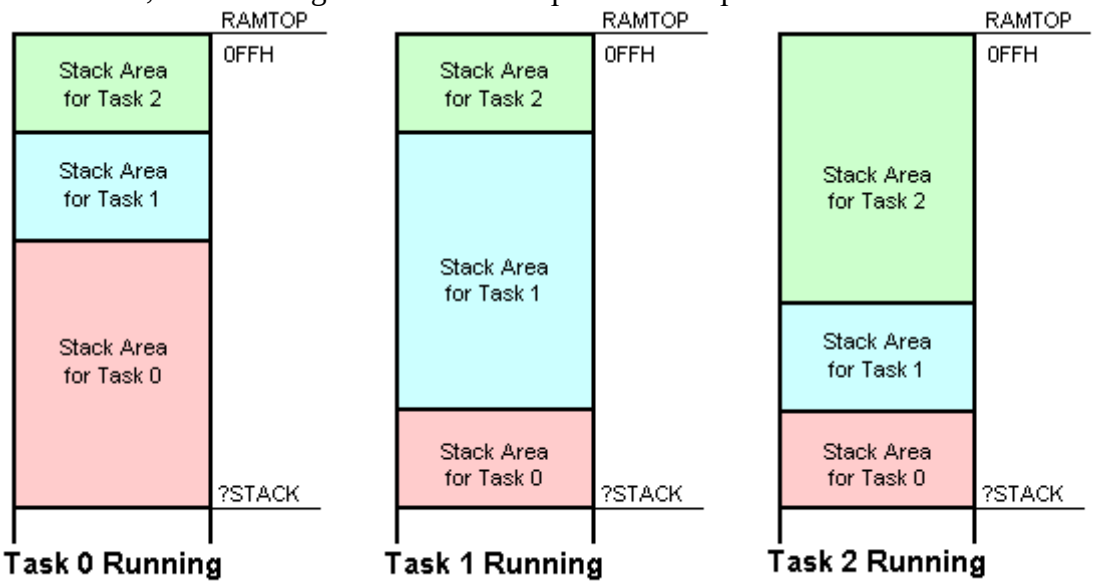
Obrázok 97, Task management – prepínanie úloh v RTX166



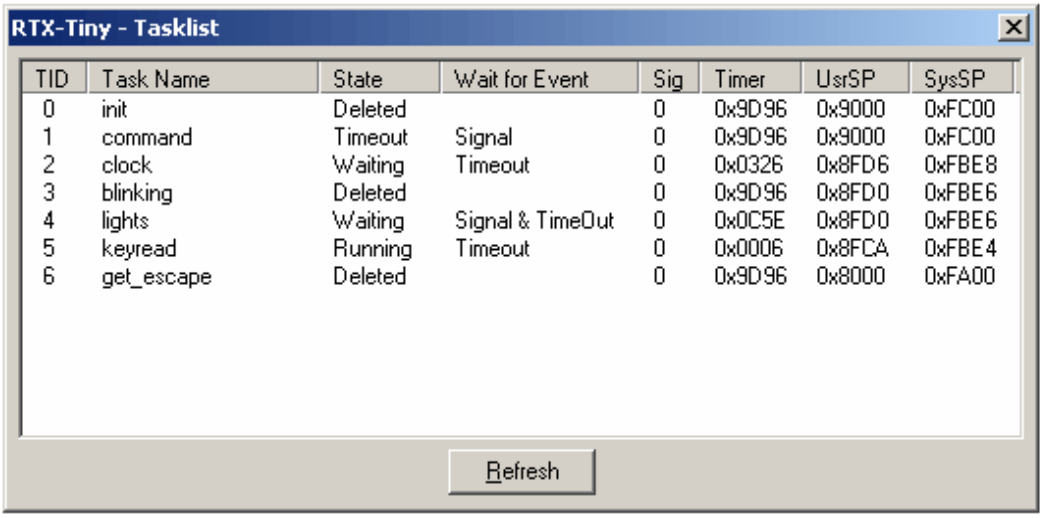
Obrázok 98, Stack management – alokácia pamäťového priestoru v RTX166

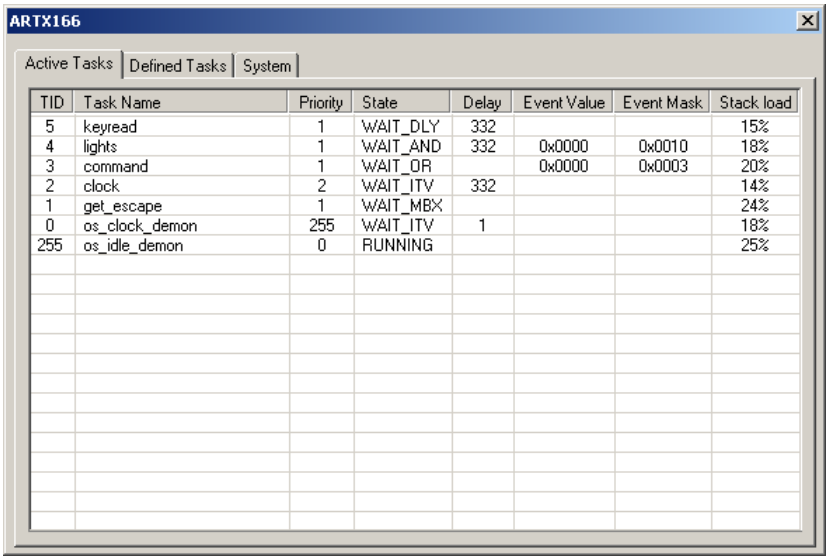


Obrázok 99, Stack management – alokácia pamäťového priestoru v RTX51

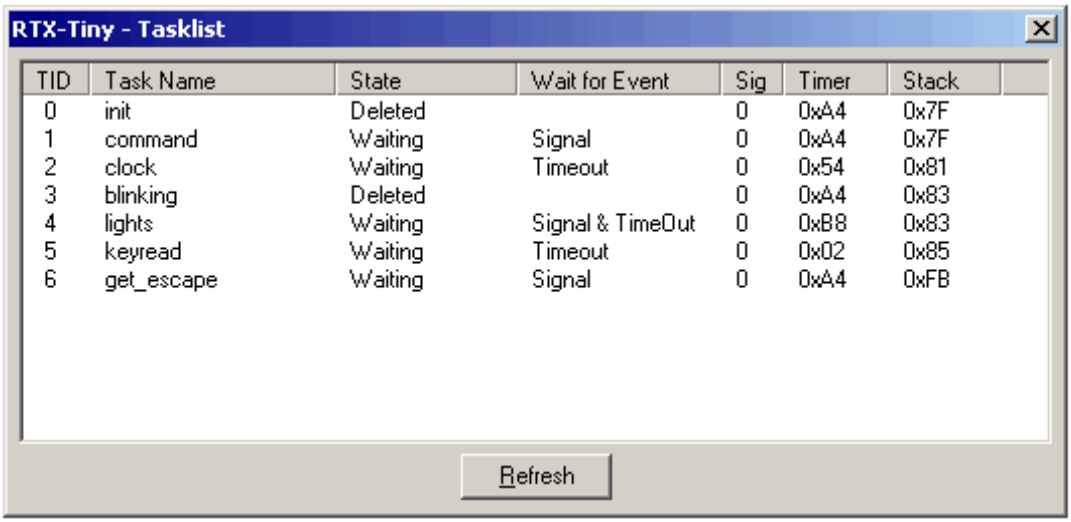


Obrázok 100, Debugging – Stav úloh a systémových prostriedkov RTX166 Tiny a AR166

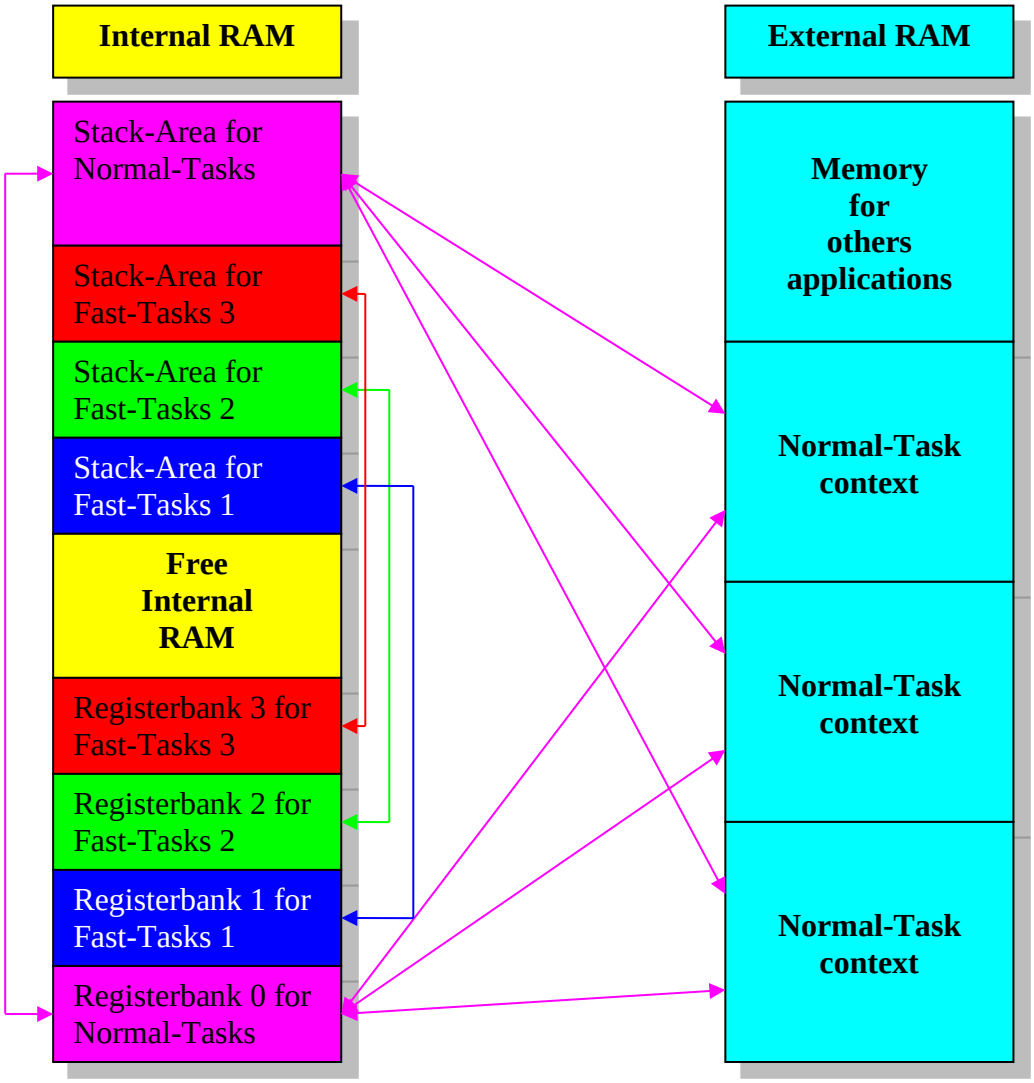




Obrázok 101, Debugging – Stav sputených úloh a systémových prostriedkov RTX51 Tiny



Obrázok 102, Typy úloh a alokácia pamäťového priestoru v RTX51 Full



Obrázok 102, Typy úloh a alokácia pamäťového priestoru v RTX51 Full farebne znázorňuje stratégiu používania jednotlivých registrových bánk RB0 až RB3 s RTX51 Full. Pre úlohy s prioritou 0 až 2 (fialová farba) je používaná výhradne banka registrov RB0. Všetky úlohy štandardne zdieľajú banku registrov RB0. Banky registrov RB1 a RB3 môže využiť len úloha deklarovaná ako fast-task s najvyššou prioritou 3. Každá úloha s najvyššou prioritou musí mať vlastnú banku registrov a nie je možné zdieľať viacerými úlohami rovnakú banku registrov. Ak vyššie uvedené nebude dodržané program nebude korektne pracovať.

8.9. Vzájomná komunikácia medzi úlohami v RTOS

Komunikácia v RTX51 môže byť vykonávaná pomocou:

- Fronty správ *(message queue).*
- Semaforom.
- Binárnym semaforom.
- Prírastkovým semaforom.

Fronta správ je v podstate druh paketu *(bloku dát)*, ktorý je odovzdávaný iným úlohám. Task môže oznámiť očakávanie správy a okamžite prechádza do stavu BLOCKED. Ak je prijatá očakávaná správa a má práve najvyššiu prioritu tak je prerušený práve vykonávaný task a riadenie je odovzdané task-u očakávajúceho správu. Správa môže byť v závislosti na výrobcovi RTOS pevnú, alebo premenlivú dĺžku. RTX51 umožňuje definovanie obidvoch typov správ.

Semafor umožňuje svojou funkciou zdieľanie systémových zdrojov viacerým úlohám bez ich vzájomnej kolízie. Výborným príkladom je súčasný prístup k sériovému rozhraniu UART dvomi úlohami. V prípade ak jedna úloha vysiela dáta cez sériové rozhranie, musí byť druhá úloha blokovaná, aby nevysielala na rovnaký sériový kanál. V prípade plne duplexného sériového kanála mikroprocesora 8051 je možné, aby jedna úloha vysielala na rozhranie a druhá úloha prijímala údaje zo sériového rozhrania. Ale súčasná funkcia vysielania a príjmu dát viacerými úlohami je blokovaná použitím semafora.

Binárny semafor[98] na rozdiel od predchádzajúceho semafora umožňuje rezervovať systémové zdroje CPU len pre jeden task. Môžu teda nadobúdať stav TRUE, alebo FALSE.

Prírastkový semafor oproti vyššie uvedených semaforov majú možnosť definovať len určitý prípustný počet súčasne pristupujúcich taskov k zdieľaným perifériám CPU. Každý task vyžadujúci použitie semafora zníži stav semafora o **1**. Každý použitý semafor zasa zvýši túto hodnotu o **1**. V prípade ak je semafor použitý maximálnym počtom taskov, prechádza do **0** a ďalší task žiadajúci semafor prechádza do stavu BLOCKED, pokiaľ iný task požadovaný semafor neuvoľní.

[98] Použitý v RTX51 Full

Vyššie uvedené typy semaforov použité napr. v RTX51 sa môžu od použitého operačného systému výrazne líšiť viď. (Burkhard, 2003).

9 Systémové požiadavky na operačný systém RTX51

9.1. RTX51 Tiny

- Maximálne je možné deklarovať 16 úloh.
- V podstate je to časť funkcií verzie RTX51 Full.
- Vhodná pre jedno čipové mikropočítače bez externej pamäti dát a programu.
- Nepodporuje preemptívny[99] scheduler – plánovač.
- Nepodporuje vzájomnú komunikáciu medzi úlohami pomocou správ.
- Nie je možné alokovať spoločne zdieľané dátové oblasti v RAM.

Hardwarové požiadavky na RTX51 Tiny verzie 2.02:

- 7 Byte pamäti dát RAM.
- 900 Byte v pamäti programu CODE.
- RTX51 Tiny[100] vyžaduje 3*[počet úloh] nepriamo adresovateľnej pamäti IDATA.

9.2. RTX51 Full

- Maximálne je možné deklarovať 256 úloh, z toho môže byť 19 úloh aktívnych.
- Štandardné úlohy môžu pracovať s prioritou 0 až 2, *(max. 16 úloh)*.
- Tzv. Fast-úlohy pracujú s najvyššou prioritou 3, *(max. 3 úlohy)*.
- Umožňuje Round-Robin, alebo preemptívny scheduler (len RTX51 Full[101]).
- Poskytuje až 4 úrovne priorít číslovaných od 0 po 3.
- Pracuje paralelne s prerušovacím systémom 8051.
- Využíva MAILBOX[102] na prenos signálov a správ medzi úlohami *(max. 8)*.
- Použitie max. 8 binárnych semaforov pre zdieľanie periférií v 8051.
- Je možné alokovať max. 16 spoločných pamäťových priestorov.
- Úloha môže aktívne čakať na prerušenie, časový interval, signál, alebo správu od inej úlohy, alebo prerušenia.
- Maximálne oneskorenie na prerušenie je menej ako 50 cyklov mikroprocesora.
- Možnosť zvoliť použitie časovača T0, T1, T2 pre jadro RTX51.
- Nastaviteľný systémový časovač od 1000 do 40000 cyklov.
- Existencia rôznych pamäťových modelov[103] SMALL, COMPACT a LARGE.

---

[99] Preemptívny – voľne preložený ako násilný, k prepnutiu úlohy dôjde pri požiadavke plánovača pri obslúžení vzniknutej udalosti v ľubovoľnej časti programu s RTOS.

[100] RTX51 Tiny je po menšej úprave použiteľný aj pre malé mikropočítače napr. AT89C4051 s minimálnym množstvom 4kB pamäte programu.

[101] RTX51 Full je robustný RTOS pre náročné aplikácie. Vyžaduje min. 1kB externej RAM a približne 100 Byte na deklarovaný task plus premenné použité v tasku.

[102] Mailbox prijíma a odosiela 16 bitovú hodnotu medzi jednotlivými úlohami v operačnom systéme RTX51 Full. Takýmto spôsobom je programátor schopný zdieľať obsahy jednotlivých premenných, alebo štruktúr samostatných taskov bez ich poškodenia a tak zabezpečíme ich aktuálnosť a konzistenciu. V RTX51 Tiny nie je implementovaný.

[103] Záleží na použitom mikroprocesore a jeho internej alebo externej pamäti. Pri použití funkcie printf() môže pri modeli SMALL vzniknúť nedostatok internej pamäti RAM, pretože funkcia potrebuje k svojej činnosti alokovať

Tabuľka 8, Dĺžka trvania funkcií operačného systému RTX51

Function	Description	CPU Cycles
isr_recv_message †	Receive a message (call from interrupt).	71 (with message)
isr_send_message †	Send a message (call from interrupt).	53
isr_send_signal	Send a signal to a task (call from interrupt).	46
os_attach_interrupt †	Assign task to interrupt source.	119
os_clear_signal	Delete a previously sent signal.	57
os_create_task	Move a task to execution queue.	302
os_create_pool †	Define a memory pool.	644 (size 20 * 10 bytes)
os_delete_task	Remove a task from execution queue.	172
os_detach_interrupt †	Remove interrupt assignment.	96
os_disable_isr †	Disable 8051 hardware interrupts.	81
os_enable_isr †	Enable 8051 hardware interrupts.	80
os_free_block †	Return a block to a memory pool.	160
os_get_block †	Get a block from a memory pool.	148
os_send_message †	Send a message (call from task).	443 with task switch
os_send_signal	Send a signal to a task (call from tasks).	408 with task switch 316 with fast task switch 71 without task switch
os_send_token †	Set a semaphore (call from task).	343 with fast task switch 94 without task switch
os_set_slice †	Set the RTX-51 system clock time slice.	67
os_wait	Wait for an event.	68 for pending signal 160 for pending message

† These functions are available only in RTX-51 Full.

Zdroj: (Keil Elektronik GmbH, a iní, 2002)

Tabuľka 9, Technické údaje RTX51

Description	RTX-51 Full	RTX-51 Tiny
Number of tasks	256; max. 19 tasks active	16
RAM requirements	40 .. 46 bytes DATA 20 .. 200 bytes IDATA (user stack) min. 650 bytes XDATA	7 bytes DATA 3 * <task count> IDATA
Code requirements	6KB .. 8KB	900 bytes
Hardware requirements	timer 0 or timer 1	timer 0
System clock	1000 .. 40000 cycles	1000 .. 65535 cycles
Interrupt latency	< 50 cycles	< 20 cycles
Context switch time	70 .. 100 cycles (fast task) 180 .. 700 cycles (standard task) depends on stack load	100 .. 700 cycles depends on stack load
Mailbox system	8 mailboxes with 8 integer entries each	not available
Memory pool system	up to 16 memory pools	not available
Semaphores	8 * 1 bit	not available

Zdroj: (Keil Elektronik GmbH, a iní, 2002)

cca. 40Byte, ktoré nikde nevypisuje ako alokované a program potom vykonáva skok do častí pamäte programu kde sa nenachádza žiadny program.

Hardwarové požiadavky na RTX51 Full 7.0:

- 40 až 46 Byte pamäti dát DATA.
- 900 Byte v pamäti programu CODE.
- 20 až 200 Byte v pamäti dát IDATA.
- min. 650 Byte v externej pamäti dát XDATA.
- 6 až 8 kB pamäti programu CODE.

9.3. Pamäťový priestor RTX51

Štandardná harwardská architektúra mikroprocesora 8051 je schopná adresovať 64kB pamäti programu a dát. V niektorých aplikáciách je vyžadované väčšie množstvo pamäti dát, alebo programu a rieši sa to pridaním niekoľkých adresných vodičov k existujúcim, ktoré vytvoria v podstate virtuálny pamäťový priestor. Tento spôsob správy pamäti je nazývaný ako ***xBANKING*** a umožňuje rozšíriť pamäťový priestor až na 16MB.[104]

Jeho funkciu je možné vysvetliť ako ***n*** segmentov pamäti RAM, alebo ROM pripojených „paralelne“ k mikroprocesoru 8051, pričom generovanie výberového signálu pre konkrétnu pamäť je zabezpečená výberovými signálmi $\overline{\boldsymbol{PSEN}}$, ***ALE***, $\overline{\boldsymbol{RD}}$, $\overline{\boldsymbol{WR}}$ a doplňujúcimi adresnými vodičmi na porte **P1** potrebných pre **xBANKING**. Linker **BL51** vytvorí ***n*** úsekov pamätí programu ku ktorým priradí ešte nevyhnutnú obsluhu pamäťového priestoru programu a dát.

Maximálne môže RTX51 Tiny[105] a Full obsluhovať až 64 bánk programu a dát *(t.j. 4MB)* pričom každá pamäť môže mať maximálne 2^{16} byte t.j. 64kB. Pri použití moderných derivátov mikroprocesorov rady 51 je možné používať až 256 bánk programu a dát čo predstavuje *(16MB)* pamäťového priestoru pre aplikácie.

Obrázok 103, Nastavenie premennej RAMTOP pre RTX51 Tiny

```
83    ;
84    ;  Define the highest RAM address used for CPU stack
85    RAMTOP       EQU  07FH     ; default is address (256-1)
86    ;
87    FREE_STACK   EQU  20h  ; default is 20 bytes free space on stack
88    ;                                ; the value 0 disables stack checking
```

[104] Túto pamäť môžeme obsluhovať pomocou portu P1, na ktorom dochádza k výberu aktívnej n-tej fyzickej pamäti RAM, alebo ROM. Pamäť RAM je už potom považovaná za xBank RAM.

[105] Pre jednočipové mikroprocesory ktoré majú len 128B pamäti RAM je nutné nastaviť v **conf_tny.a51** RAMTOP na hodnotu 07Fh, ináč RTX51 Tiny nefunguje správne. Pre procesory 8051 ktoré majú 256B internej pamäti RAM je RAMTOP nastavená na hodnotu 0FFh.

10 Operačný systém reálneho času RTX51

10.1. Vzájomná komunikácia a synchronizácia medzi úlohami v RTX51

RTX51 poskytuje tri spôsoby ktoré povoľujú individuálnej úlohe synchronizovať sa a komunikovať s ďalšími úlohami:

- **SIGNALS** sú v podstate riadiace signály neobsahujúce informácie o zdrojovej úlohe a ich úlohou je synchronizovať vykonávanie jednotlivých úloh.
- **MESSAGES** sú dáta s pevnou, alebo premenlivou dĺžkou, ktoré sú prenášané cez tzv. *mailbox* a slúžia na vzájomnú výmenu dát medzi úlohami. Každý mailbox je schopný prijať až 8 správ[106] v poradí ako boli prijaté.
- **SEMAPHORES** sú jednoduchým mechanizmom, ktorý je schopný zdieľať systémové zdroje CPU. Semafory sa používajú vtedy, ak minimálne dve úlohy používajú súčasne jedno zariadenie[107]. Vhodným príkladom je napríklad sériové RS232, CAN, SPI, alebo I^2C rozhranie. Vzájomne sa teda blokuje prístup druhej úlohy k rozhraniu ak s ním pracuje prvá úloha. V RTX51 Full je možné použiť len binárne semafory.

10.2. Mechanizmus prepínania úloh v RTX51

RTX51 rozoznáva 4 základné stavy:

- **READY** Všetky úlohy ktoré nie sú v stave ***BLOCKED*** alebo ***RUNNING*** a sú pripravené k spusteniu sú v stave ***READY***.
- **RUNNING** V stave ***RUNNING*** sa nachádza úloha ktorú práve vykonáva CPU.
- **BLOCKED** V tomto stave sa nachádza úloha ktorá aktívne čaká na výskyt udalosti *(timeout, message, semaphore, alebo signal)*.
- **SLEEPING** Tento stav nadobudne úloha ktorá je síce deklarovaná, ale nebola ešte spustená. Zároveň tento stav nadobudne úloha ktorá bola operačným systémom ukončená. Úloha sa môže nachádzať len v jednom z vyššie uvedených stavov a zmena z jedného do druhého stavu je charakterizovaná ako skoková.

[106] Správa t.j. **Messages** je premenná typu **int**, alebo **unsigned int**. Niekedy je zbytočne a nezmyselne vypisované hlásenie RTX51 o nesprávnom použitom údajovom type. Pritom sú v podstate tieto typy v RTX51 rovnocenné. V prípade, že bude vypisované hlásenie o typovej nekompatibilite, je potrebné vyskúšať oba údajové typy.

[107] Ako príklad môžeme považovať funkciu **printf()**, ktorá je použitá v dvoch rôznych task-och. Z pohľadu RTX51 sa ale jedná o „súčasný“ prístup do rovnakej procedúry. Pokiaľ táto funkcia nie je deklarovaná ako reentrantná, dôjde k poškodeniu environmentu funkcie **printf()** a tým analogicky aj k nefunkčnosti funkcie.

10.3. Riadenie úloh pomocou udalostí v RTX51

Úloha môže aktívne čakať na výskyt jednej udalosti, alebo kombinácie udalostí ktoré sa vyskytnú v priebehu vykonávania programu. Pokiaľ jedna úloha čaká na udalosť, ostatné úlohy môžu byť v stave ***READY*** a môžu byť prepnuté do stavu ***RUNNING***.

Riadenie pomocou nižšie uvedených udalostí je možné pomocou systémovej funkcie RTX51 ***os_wait(K_EVENT,TIME,NULL)***[108], kde **parameter *K_EVENT*** umožňuje definovať typ očakávanej udalosti.

Zoznam udalostí pre funkciu *os_wait():*

- **Semaphore**[109]Len RTX51 Full, používa sa pre obsluhu zdieľaných systémových zdrojov napr. volanie funkcie **printf()** z viacerých taskov.
- **Signal** Používa sa na vzájomnú synchronizáciu jednotlivých úloh.
- **Interrupt**[110] Len RTX51 FULL, úloha môže čakať na hardwarové prerušenie.
- **Message**[111] Len RTX51 FULL, používa sa pre výmenu správ medzi úloh.
- **Time-out** Preruší vykonávanie úlohy na definovaný počet hodinových cyklov systémového časovača RTX51 a odovzdá riadenie ďalšej úlohe – task-u.
- **Interval** Len RTX51 Tiny[112], je ekvivalentný *timeout-u* ale systémový časovač nie je po pretečení resetovaný a tým generuje veľmi presné časové úseky v programe.

V programe je možné využiť aj kombináciu vyššie uvedených udalostí v jednom príkaze a to napríklad: ***os_wait(K_TMO+K_SIG,100,0)***, kde RTX51 očakáva príjem signálu od inej úlohy po dobu 100 cyklov systémového časovača. K_TMO je v tomto prípade „zbytočná“, pretože K_SIG je nastavená tak, že po 100 cykloch časovača sa ukončuje automaticky príjem čakania na signál a správa sa ukončuje akoby sa aktivoval „timeout“ s udalosťou K_TMO. Ak

[108] RTX51 Full oproti RTX51 Tiny neobsahuje **os_task_switch()** a preto sa používa funkcia **os_wait(K_TMO,0,0)**.
[109] Nie je možné nastaviť kombináciu v rovnaký task na súčasný príjem správy z mailbox-u selektoru MSG_EVENT (0-7) a token-u pomocou selektoru SEM_EVENT semaforu (8-15).
[110] Prerušenie na túto udalosť je vhodné nastaviť lepšie pomocou funkcie **os_attach_interrupt(n)**, ako pomocou **os_enable_isr(n),** kde **n = (0..31)** pretože funkcia os_attach_interrupt(n) nastaví kompletne príslušný register IE a nepotrebujeme napísať obsluhu t.j. handler pre dané prerušenie **n**.
[111] Odoslanie správy sa vykonáva pomocou funkcie **os_send_message(mbx,var,100)** odovzdáva obsah premennej **var** príslušnému MailBox-u **mbx** a čaká na doručenie 100 cyklov. Prijatie správy je v tvare **os_wait(K_TMO+K_SIG+K_MBX+mbx,200,&vars)**, kde sa obsah správy prijatej cez mailbox **mbx** uloží na adresu premennej **&vars**.
[112] Nezodpovedá to úplne skutočnosti ktorá je uvedená v príručke programátora, pretože tento typ udalosti v **os_wait(K_IVL,100,NULL)** funguje aj v RTX51 Full a umožňuje generovanie veľmi presných časových intervalov ktoré sú odvodené od hardware systémového časovača T0, T1, alebo T2.

použijeme ***os_wait(K_IVL+K_SIG,100,0)*** a preruší sa čakanie na udalosť iným typom udalosti ako je „timeout“ je nutné použiť procedúru ***os_reset_interval(100)***, ktorá vykoná opätovné nastavenie hardware časovača RTX51. Obrázok 104 ukazuje neštandardné použitie udalosti typu „timeout“, ktorá generuje presné časové intervaly a zároveň očakáva udalosť typu „signal“. V prípade ukončenia čakania pomocou udalosti „signal“ je podľa príručky RTX51 nevhnutné ukončiť pomocou funkcie ***os_reset_interval(100)*** čakanie, aby nedošlo ešte opätovne k zbytočnému vyvolaniu udalosti „timeout“. Pri krokovaní programu sme zistili, že aj napriek použitému príkazu ***os_reset_interval(100)*** nedôjde k resetu hardware časovača a vyvolá sa zbytočne ešte navyše bezdôvodne „timeout“ udalosť. Obrázok 105 zobrazuje rovnaký task v ktorom je použitý ***os_wait(K_TMO+K_SIG,50,0)*** ktorý pracuje správne, lebo nepotrebuje resetovať hardware časovača RTX51 a prijatím udalosti typu „signal“ ukončí korektne task.

Obrázok 104, Nesprávne ošetrenie udalosti typu "interval"

```
void Clock(void) _task_ CLOCK
{
 while(1)
  {
   switch(os_wait(K_IVL|K_SIG,100,NULL))
    {
     case TMO_EVENT : i++; break;
     case SIG_EVENT : Counter++; os_reset_interval(100); break;
    }
  }
}
```

Obrázok 105, Správne ošetrenie udalosti typu "timeout"

```
void System(void) _task_ SYSTEM
{
 os_create_task(CLOCK);
 os_create_task(TICKS);
 while(1)
  {
   switch(os_wait(K_TMO|K_SIG,50,NULL))
    {
     case TMO_EVENT : break;
     case SIG_EVENT : break;
    }
  }
}
```

Príklad využitia *os_wait(event_selector,timeout,0)* v kombinácii s inými udalosťami

c:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\I2c_DSW.c

```
 86  void Adc(void) _task_ ADC _priority_ 3
 87  {
 88   signed char RtxCompletion;
 89   os_attach_interrupt(10);
 90   while(1)
 91    {
 92     Watchdog;
 93     switch(RtxCompletion=os_wait(K_TMO+K_INT+K_SIG,10,NULL))
 94      {
 95       case SIG_EVENT  : Voltage=ADC_Conversion(2); break;
 96       case TMO_EVENT  : ADCON|=(ADCS+2);
 97       case INT_EVENT  : ADCON^=ADCI; Voltage=Conversion(); break;
 98      }
 99    }
100  }
```

10.4. Prerušovací podsystém RTX51

Obsluha prerušenia je jednou z hlavných a najdôležitejších úloh, ktoré musí RTX51 výborne ovládať. Mnoho operačných systémov práve stroskotá pri obsluhe viacnásobného príjmu prerušení v reálnom čase. **Prerušenie sa v žiadnom prípade nesmie stratiť!**

RTX51 podporuje dve rozdielne kategórie úloh:

1. Standard-task:

- Vyžadujú viac času na prepnutie medzi úlohami.
- Všetky úlohy zdieľajú spoločnú banku registrov ***RB0*** a zásobník ***STACK***.
- Sú prerušiteľné fast-task úlohami a štandardnými prerušeniami z C51.
- Môžu sa vzájomne prerušovať podľa vopred deklarovaných pravidiel.
- Maximálne 16[113] aktívnych úloh v programe, *(256 neaktívnych)*.

2. Fast-task:

- Používané pre rýchlu reakciu na výskyt udalosti.
- Úlohy majú najvyššiu úroveň priority 3.
- Obsahujú vlastný separátny (oddelený) zásobník ***STACK***.
- Používajú fyzicky oddelené banky registrov ***RB1*** *až* ***RB3*** pre *fast-tasks*[114].
- Môžu byť prerušené len výskytom štandardného prerušenia HW[115].
- Maximálne môžu byť v systéme definované 3 *fast-task* úlohy.
- Možnosť využiť pre prerušenie **PFI**[116] (Power Fail Interrupt)

[113] Súčasne je možné spustiť v RTX81 Full 16+3 t.j. 19 taskov.
[114] Prepnutie úlohy s prioritou 3 trvá max. 50 cyklov mikroprocesora, musí byť použitá reg. banka **3** pomocou direktívy **#pragma REGISTERBANK (3)** ako to ukazuje vyššie uvedený príklad. Štandardné úlohy s prioritou **0 až 2** využívajú banku registrov **0**, fast-task úlohy vyžadujú každá samostatne použitú banku registrov (1 až 3).
[115] Niektorého prerušenia od hardware 8051.
[116] Niektoré procesory PFI nepodporujú, viď manuál C51 a Cx51 na www.keil.com.

Príklad úlohy typu Fast-Task:

```
84    #pragma  REGISTERBANK  (3)
85
86    void  Adc (void ) _task_  ADC  _priority_  3
87    {
88     signed  char  RtxCompletion ;
89     os_attach_interrupt (10 );
90     while (1)
91       {
92        Watchdog ;
93        switch (RtxCompletion =os_wait (K_TMO +K_INT +K_SIG , 10 , NULL ))
94         {
95          case  SIG_EVENT  : //Voltage=ADC_Conversion(2); break;
96          case  TMO_EVENT  : ADCON |=( ADCS +2);
97          case  INT_EVENT  : ADCON ^=ADCI ; Voltage =Conversion (); break ;
98         }
99       }
100   }
```

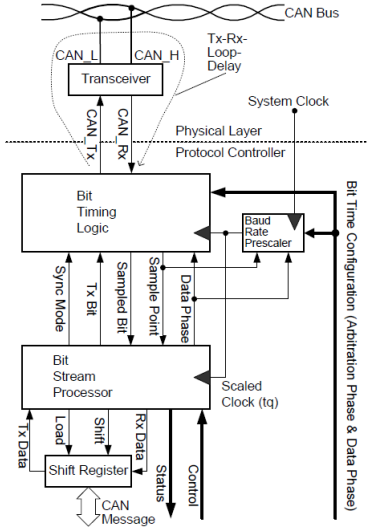
11 Komunikácia RTX51 s periférnym rozhraním CAN

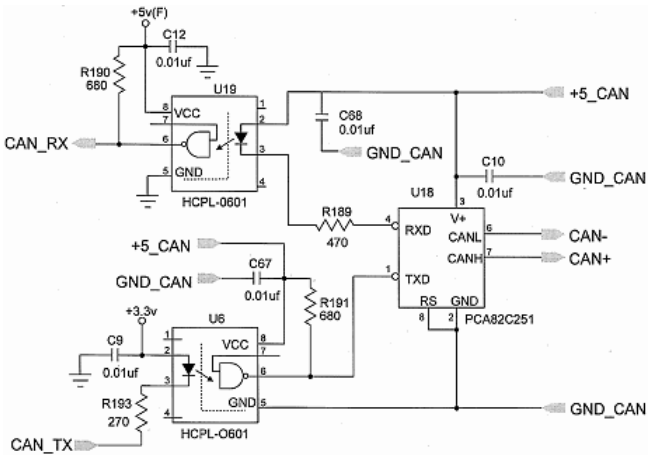
Komunikačné rozhranie CAN definovala a štandardizovala firma BOSCH v roku 1986. Využitie sa našlo hlavne v automobilovom priemysle, kde sa vďaka nej znížila hmotnosť použitých vodičov na komunikáciu s jednotlivými komponentami umiestnených vo vozidle.

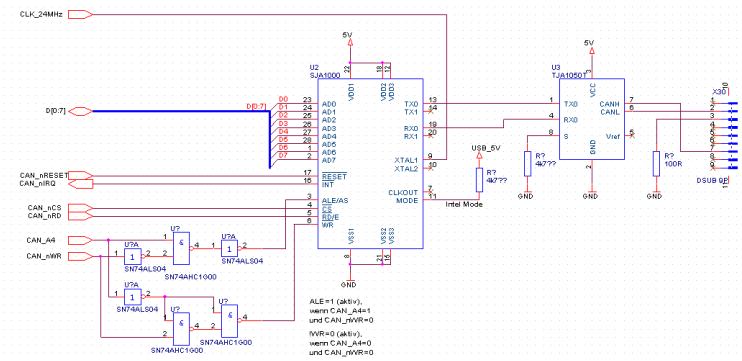
11.1. Vlastnosti CAN

- ✓ sériový typ komunikácie pre aplikácie v reálnom čase,
- ✓ rýchlosť prenosu dát až do 1,0 Mbs^{-1} (terminátor 124Ω na oboch koncoch vedenia),
- ✓ extrémne vysoká odolnosť voči chybám pri prenose dát,
- ✓ predurčená na použitie v automobile,
- ✓ použitie v priemyselnej automatizácii a pri riadení aplikácií,
- ✓ definovaný priemyselný štandard **ISO11898**[117].

Obrázok 106, Komunikácia CAN s fyzickou vrstvou CAN Bus

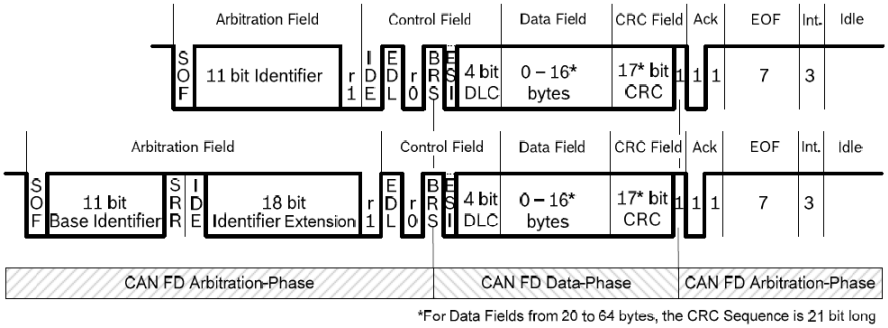






117 http://www.semiconductors.bosch.de/media/pdf/canliteratur/can_fd.pdf

Obrázok 107, Formát CAN správ



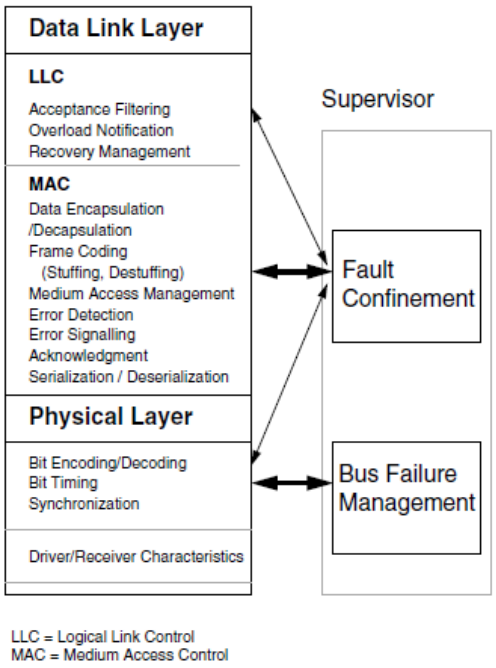
11.2. Komunikačný model CAN

Komunikačný model rozhrania CAN je veľmi podobný komunikačnému modelu TCP/IP v sieti Ethernet. Maximálna dĺžka medzi dvomi komunikujúcimi zariadeniami v CAN na ktorej je schopná komunikovať rýchlosťou 1,0 Mbs^{-1} je cca. 40m, rýchlosťou 125kbps až na vzdialenosť 500m.

Každý uzol vyžaduje:

- ✓ hostiteľský procesor,
- ✓ CAN radič,
- ✓ vysielač (TxD, RxD).

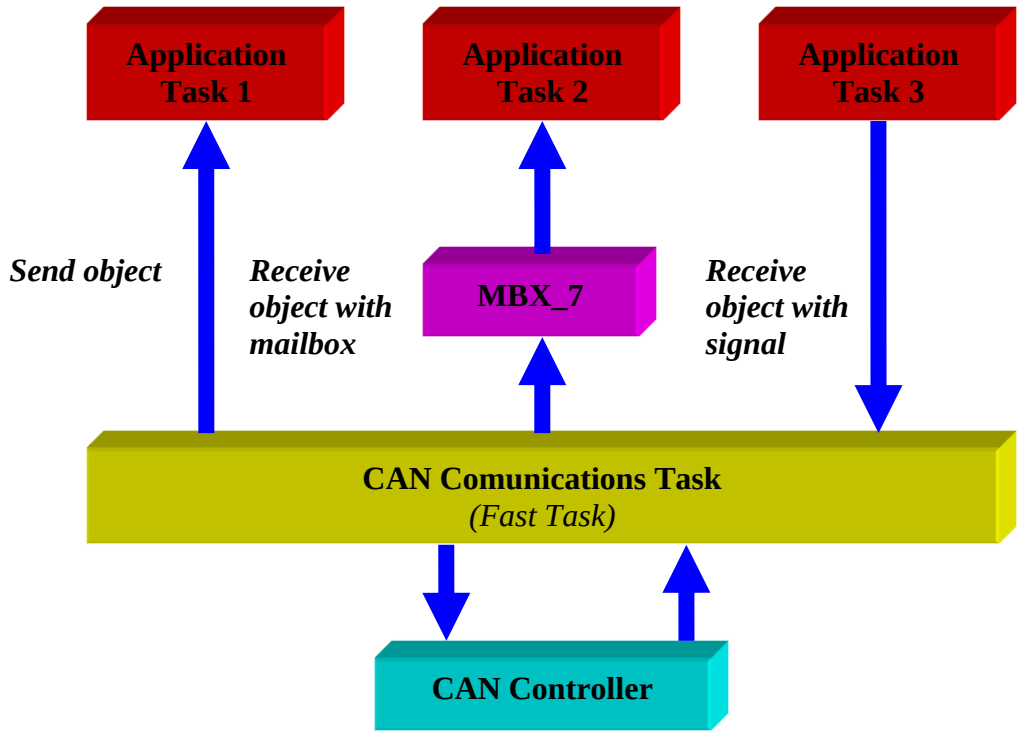
Obrázok 108, Základný koncept CAN



Tabuľka 10, Podpora[118] RTOS od firmy KEIL pre rozhranie CAN

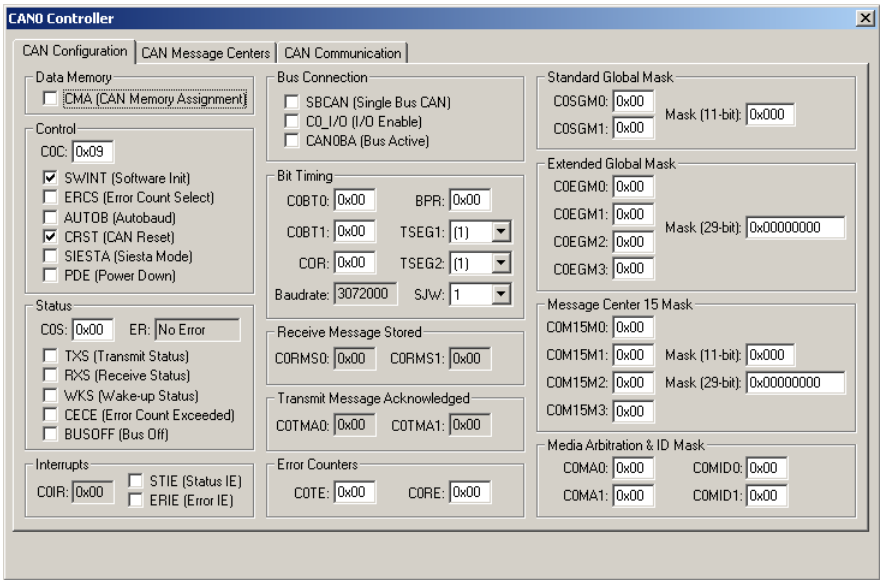
Part Number CAN Devices Supported	RTX51 Real-Time Kernel	RTX166 Real-Time Kernel
Infineon 81C90/81C91 (Stand-alone CAN controller)	✔	
Infineon C505C (8051-based Micro with CAN)	✔	
Infineon C515C (8051-based Micro with CAN)	✔	
Infineon C167CR (C16x-based Micro with CAN)		✔
Infineon C164CI (C16x-based Micro with CAN)		✔
Intel 82526/82527 (Stand-alone CAN controller)	✔	
Philips 82C200 (Stand-alone CAN controller)	✔	
Philips 8xC592 (8051-based Micro with CAN)	✔	
Philips 8xCE598 (8051-based Micro with CAN)	✔	

Obrázok 109, Spôsob komunikácie aplikácie RTOS a rozhraním CAN

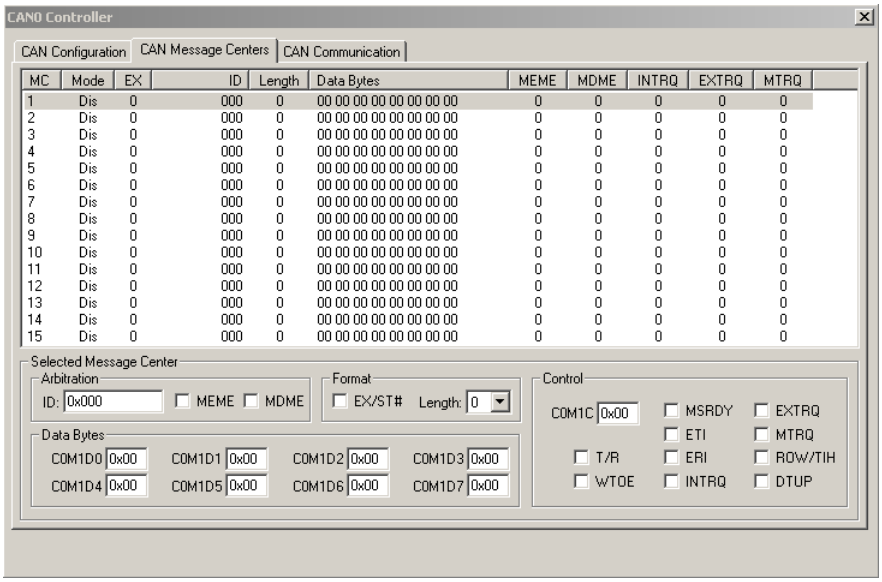


[118] http://www.keil.com/uvision/db_sim_prf_can.asp, RTX51 Full verzie 7.0 podporuje navyše mimoriadne populárny obvod SJA1000T od firmy Philips Semiconductor. Bližšie informácie o obvode SJA1000T nájdete na http://www.nxp.com/documents/application_note/AN97076.pdf.

Obrázok 110, CAN Controller v µVision4



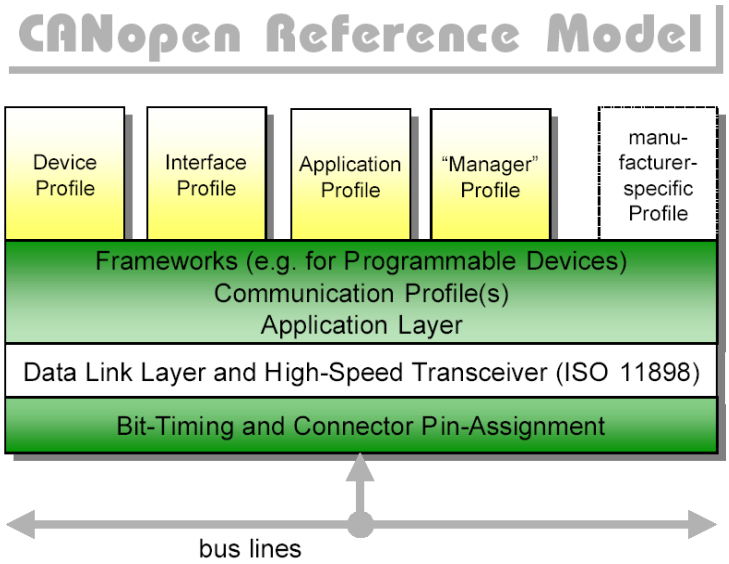
Obrázok 111, CAN Message Center v µVision4



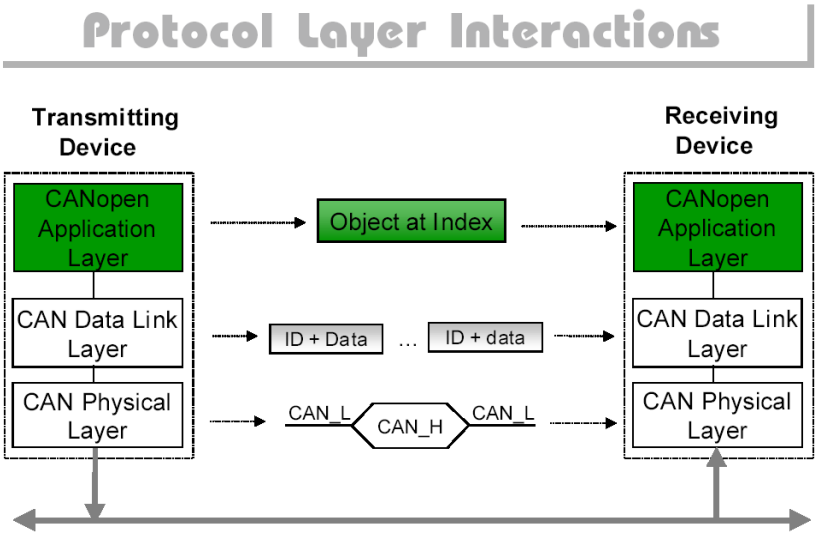
Obrázok 112, Generovanie[119] CAN správ v prostredí µVision4

```
SIGNAL void sendCAN (float secs)  {
  while (1)  {
    CAN0ID = 0x4510;          // CAN message ID
    CAN0L  = 2;               // message length 2 bytes
    CAN0B0 = (info & 0xFF);   // message data byte 0
    CAN0B1 = (info >> 8);     // message data byte 1
    CAN0IN = 2;               // send CAN message with 29-bit ID
    swatch (secs);            // delay for specified time
    info++;                   // increment info value
  }
}

>sendCAN (0.5);              // invoke sendCAN function
```

Obrázok 113, Otvorený referenčný model zbernice CAN

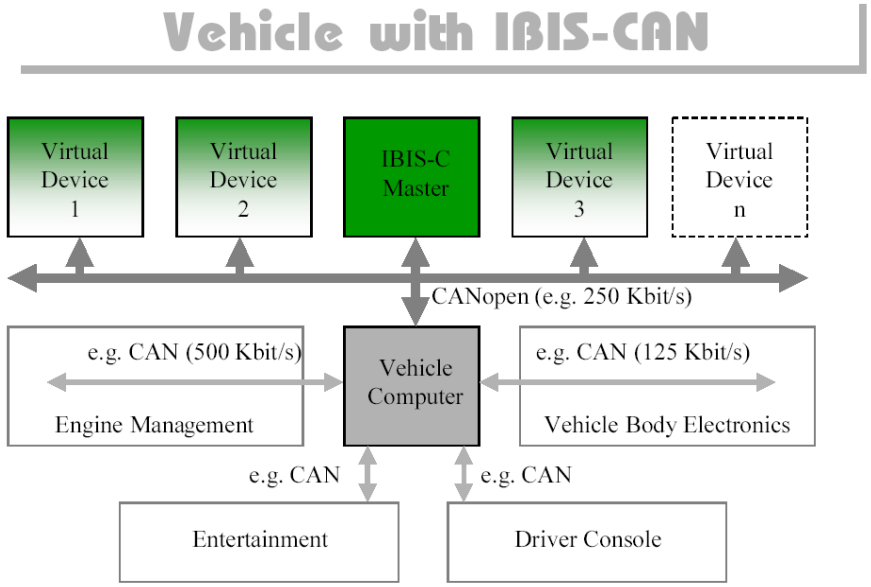


Obrázok 114, Vrstvový model zbernice CAN

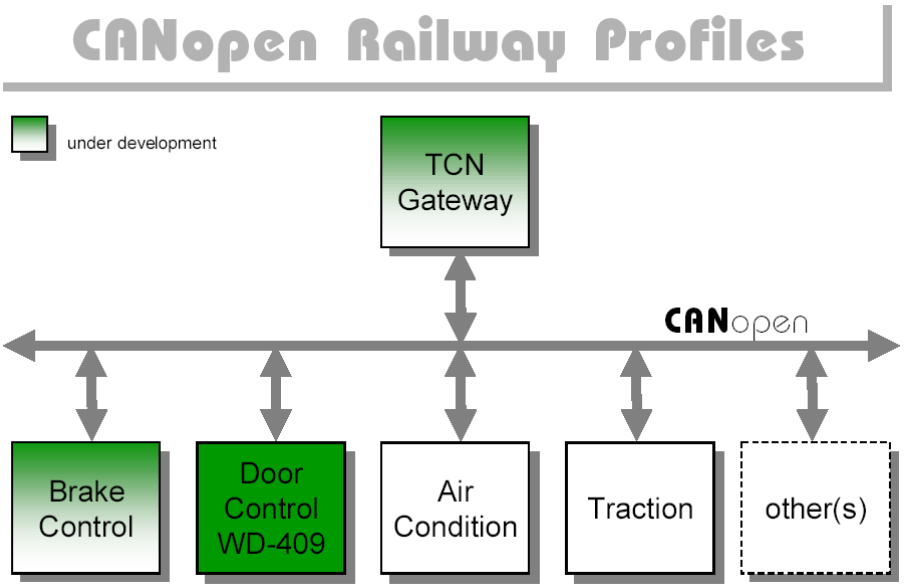


[119] Program generovania CAN správ musí byť uložený v súbore debugger.ini odkiaľ umožňuje vytvárať jednoduché CAN správy.

Obrázok 115, Použitie zbernice CAN v automobilovom priemysle



Obrázok 116, Použitie zbernice CAN v elektrickej trakcii



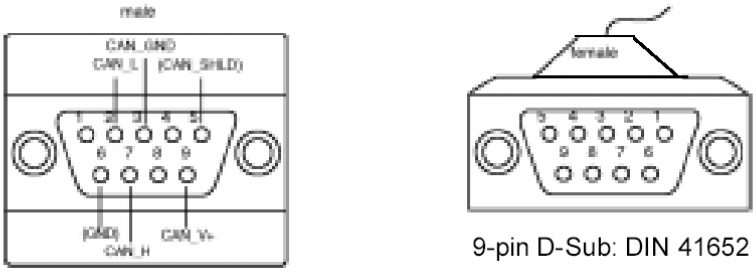
Tabuľka 11, Charakteristické vlastnosti zbernice CAN

DC Characteristics

Bus Length	Bus Cable		Termination Resistance	Max. Baudrate
	Length-Related Resistance	Bus-Line Cross-Section		
0 .. 40 m	70 mΩ/m	0.25 mm² .. 0.34 mm² AWG23, AWG22	124 Ω (1%)	1 Mbit/s at 40 m
40 .. 300 m	<60 mΩ/m	0.34 mm² .. 0.6 mm² AWG22, AWG20	127 Ω (1%)	500 Kbit/s at 100 m
300 .. 600 m	<40 mΩ/m	0.5 mm² .. 0.6 mm² AWG20	150 Ω to 300 Ω	100 Kbit/s at 500 m
600 m .. 1 km	<26 mΩ/m	0.75 mm² .. 0.8 mm² AWG 18	150 Ω to 300 Ω	50 Kbit/s at 1k m

Obrázok 117, Rozloženie vývodov zbernice CAN

CiA DS-102 Pin Assignment



9-pin D-Sub: DIN 41652

Pin	Signal	Description
1	-	Reserved
2	CAN_L	CAN_L bus line dominant low
3	CAN_GND	CAN Ground
4	-	Reserved
5	(CAN_SHLD)	Optional CAN Shield
6	GND	Optional Ground
7	CAN_H	CAN_H bus line dominant high
8	-	Reserved
9	(CAN_V+)	Optional CAN external supply

Podrobné informácie o zbernici CAN[120] je možné získať www.can-cia.de.

[120] http://www.interfacebus.com/CAN-Bus-Description-Vendors-Canbus-Protocol.html

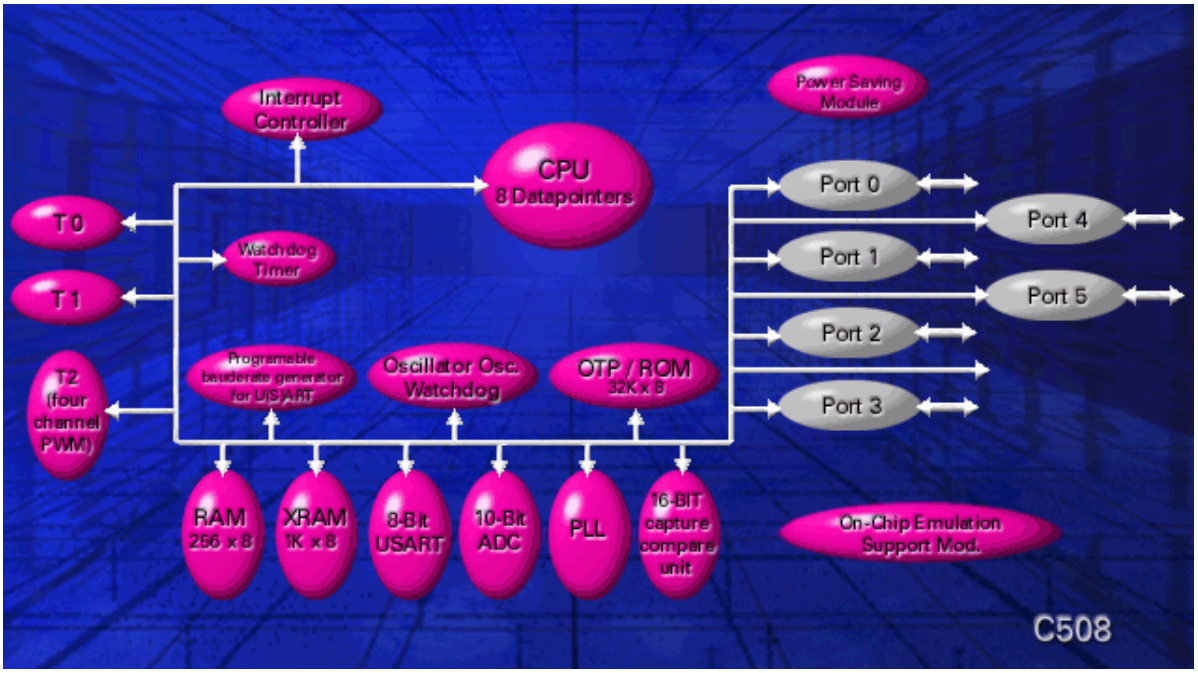
Tabuľka 12, Zoznam funkcií CAN v prostredí RXT51

CAN Function	Description
can_bind_obj	Bind an object to a task; task is started when object is received.
can_def_obj	Define communication objects.
can_get_status	Get CAN controller status.
can_hw_init	Initialize CAN controller hardware.
can_read	Directly read an object's data.
can_receive	Receive all unbound objects.
can_request	Send a remote frame for the specified object.
can_send	Send an object over the CAN bus.
can_start	Start CAN communications.
can_stop	Stop CAN communications.
can_task_create	Create the CAN communication task.
can_unbind_obj	Disconnect the binding between a task and an object.
can_wait	Wait for reception of a bound object.
can_write	Write new data to an object without sending it.

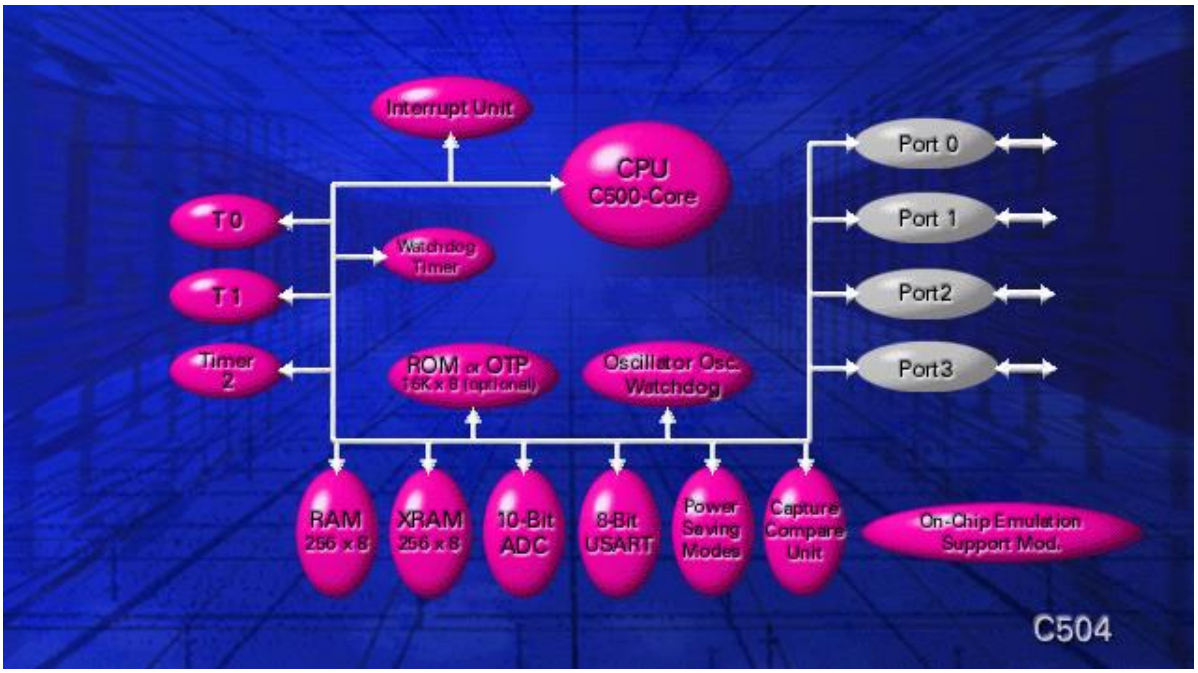
12 Moderné a perspektívne architektúry mikroprocesorov

12.1. 8 bitová architektúra

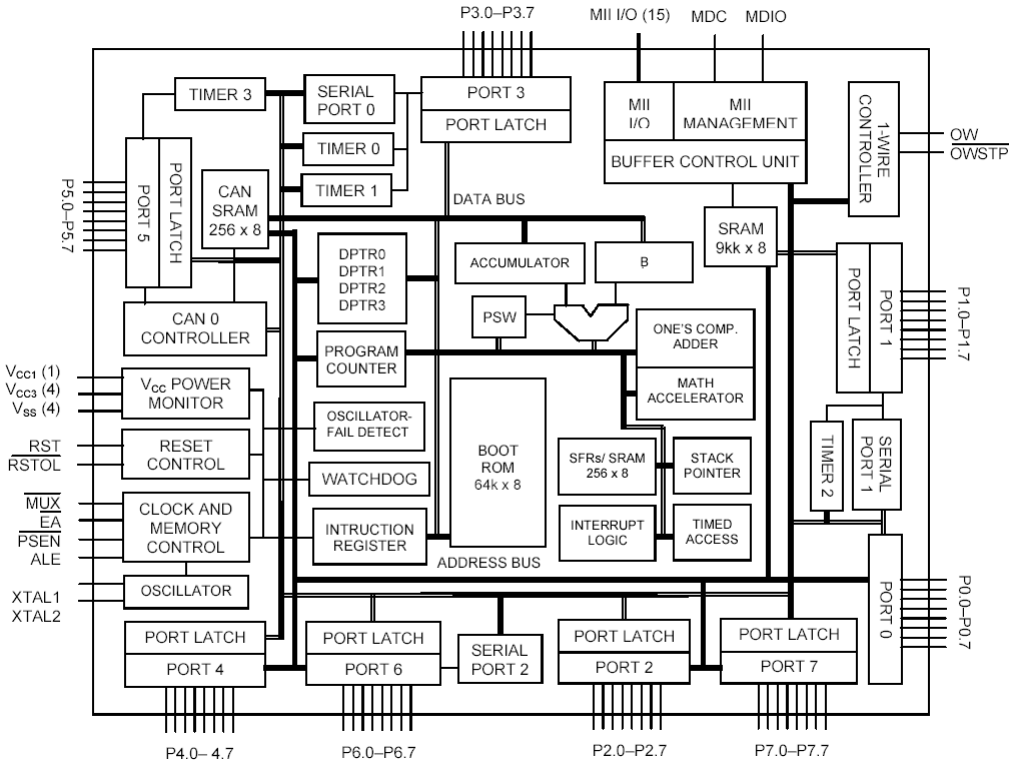
Obrázok 118, Architektúra 8051 – C508 od firmy INFINEON



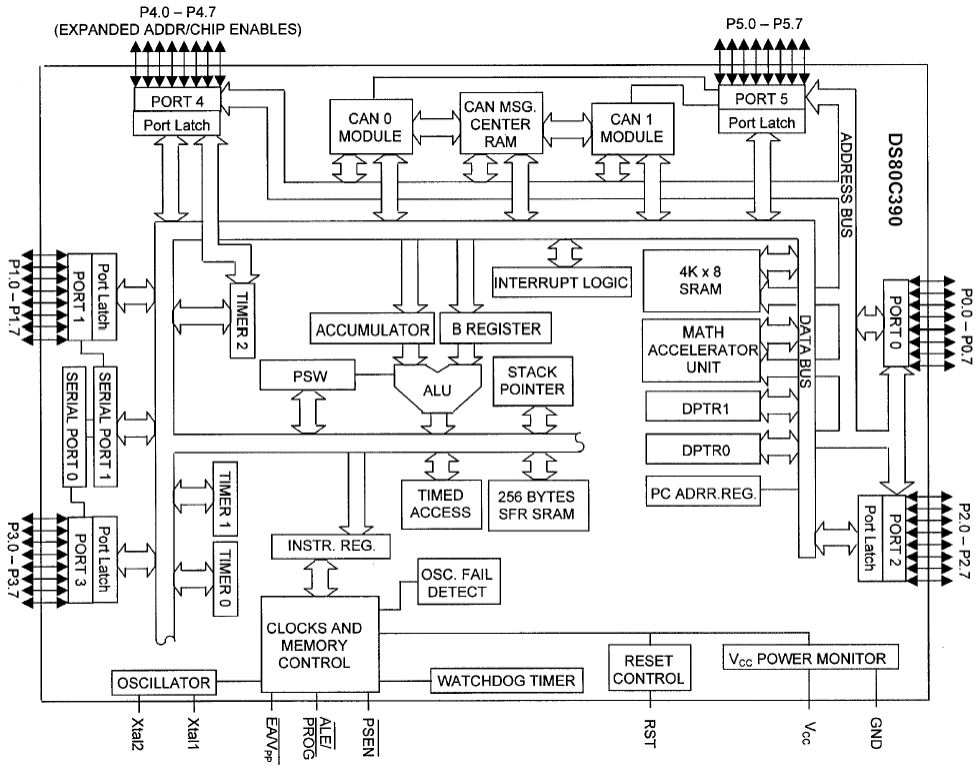
Obrázok 119, Architektúra 8051 – C504 od firmy INFINEON



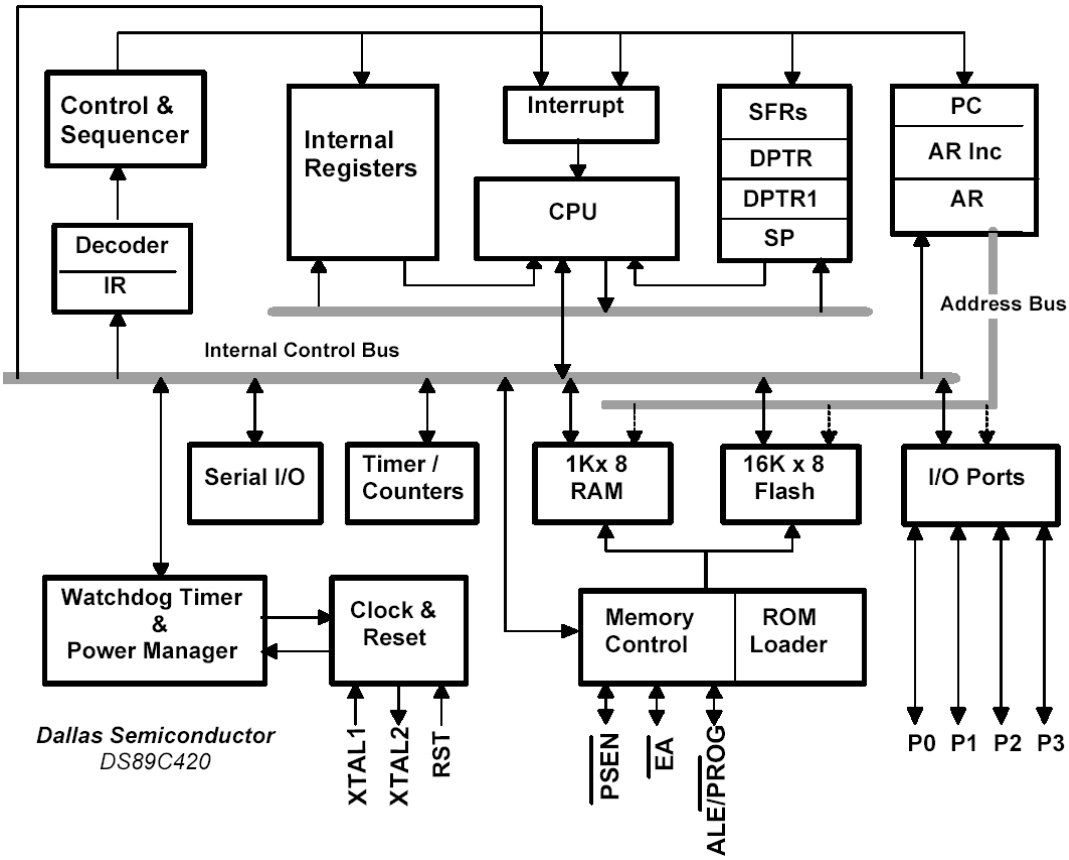
Obrázok 120, Ultra High-Speed 8051 - DS80C400 od firmy Dallas pre sieťové aplikácie



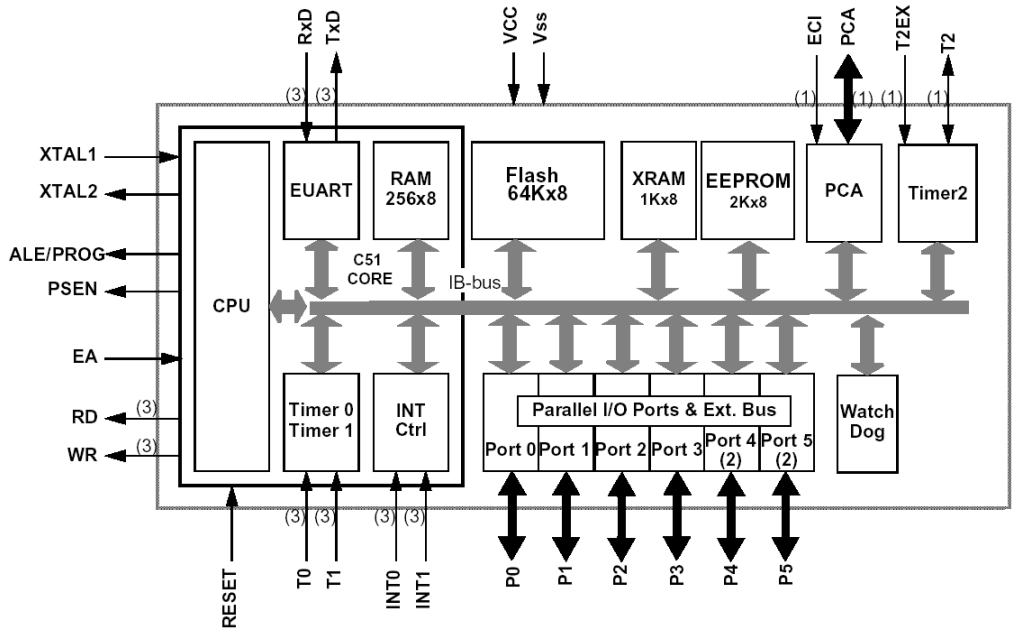
Obrázok 121, Ultra High-Speed 8051 – DS80C390 od firmy Dallas s CAN



Obrázok 122, High-Speed architektúra 8051 – DS89C420 od firmy Dallas Semiconductor

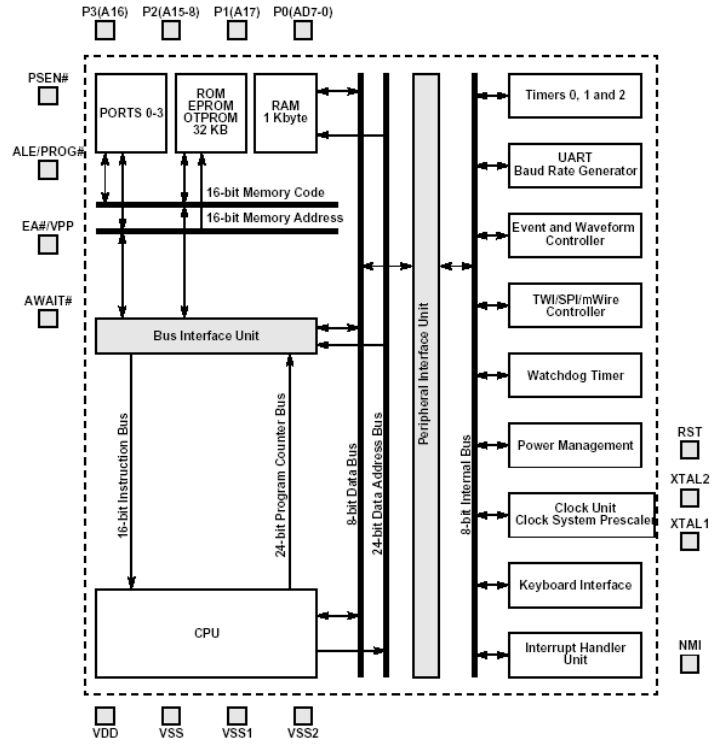


Obrázok 123, High-Speed architektúra 8051 – AT89C51RD2 od firmy ATMEL

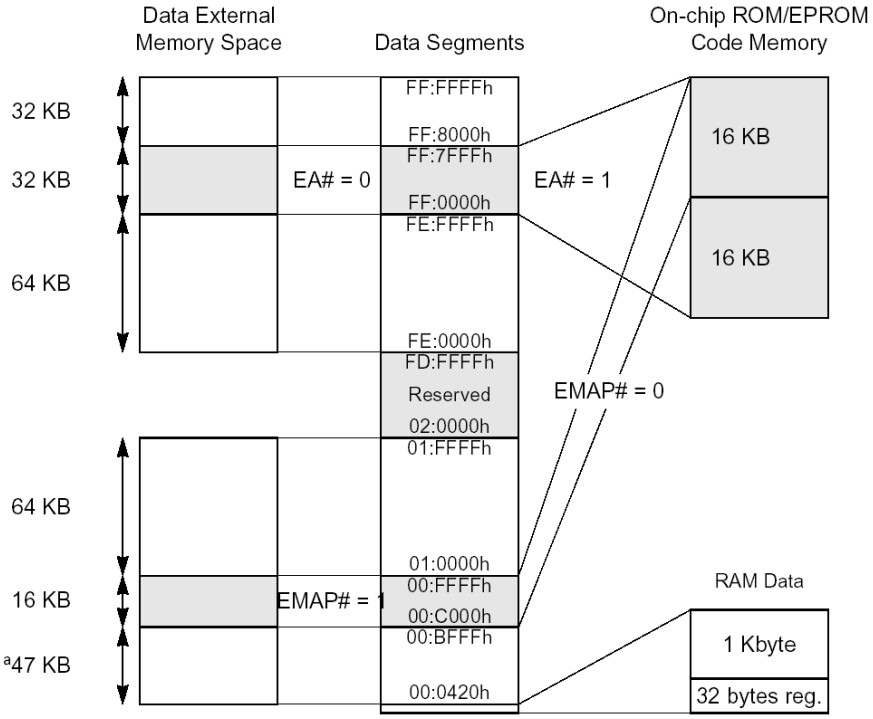


12.2. 16 bitová architektúra

Obrázok 124, High-Speed architektúra 80251 – TSC8x251G2D od firmy ATMEL

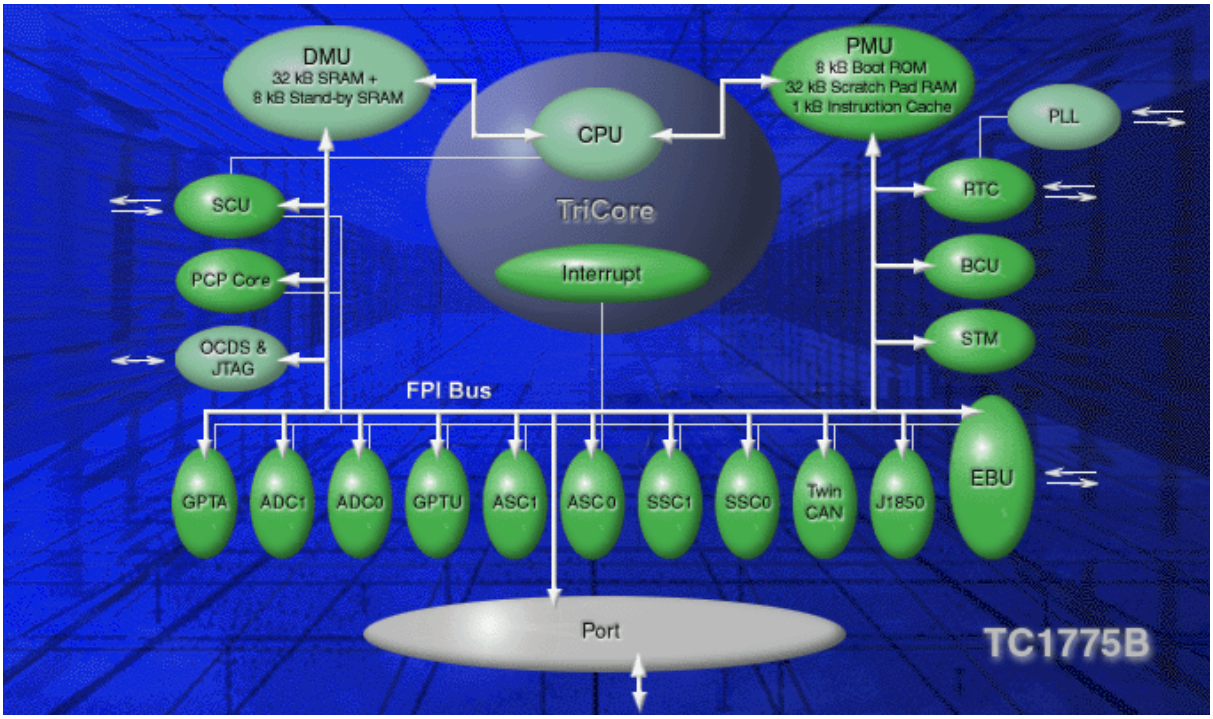


Obrázok 125, Pamäťový model architektúry 80251 – TSC8x251G2D od firmy ATMEL

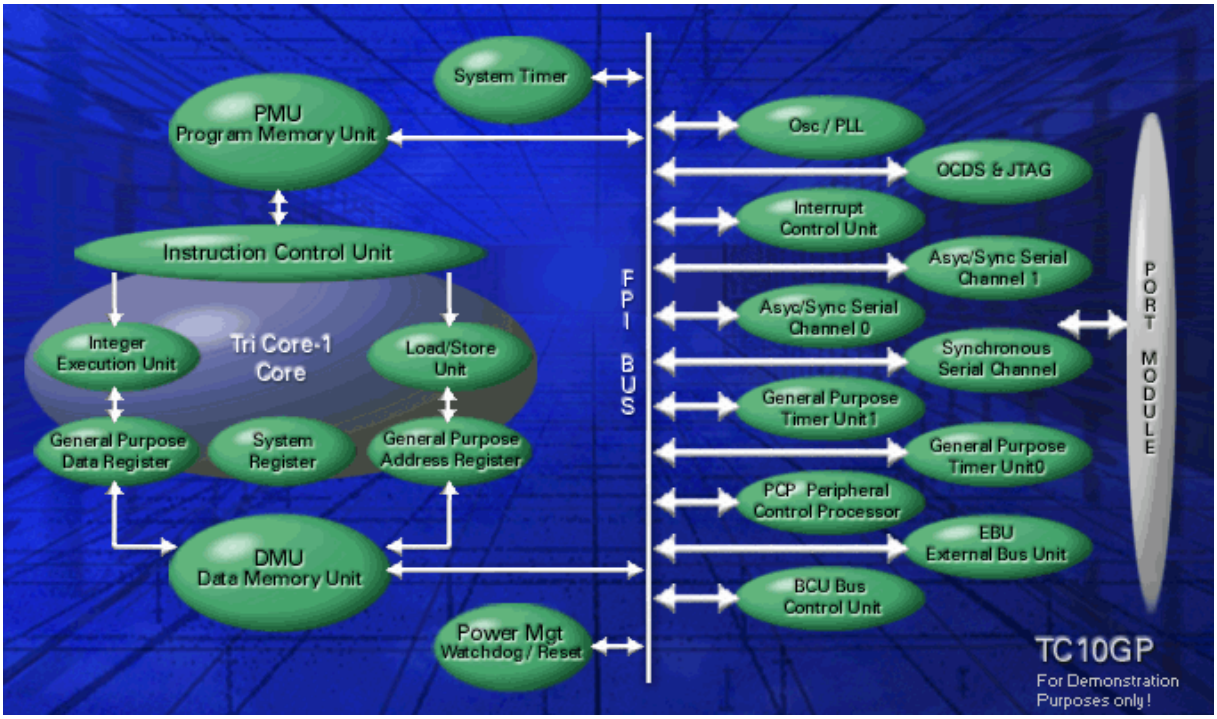


12.3. 32 bitová architektúra

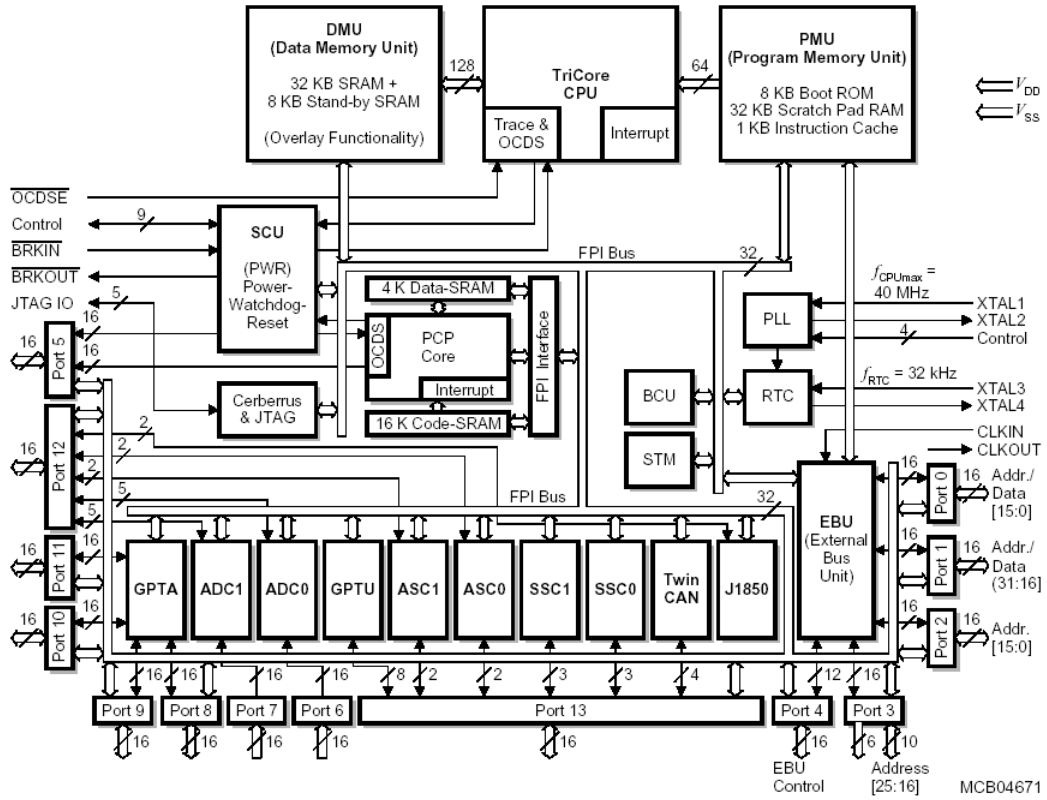
Obrázok 126, Architektúra Tri Core 32 bit. DSP TC1775B od firmy INFINEON



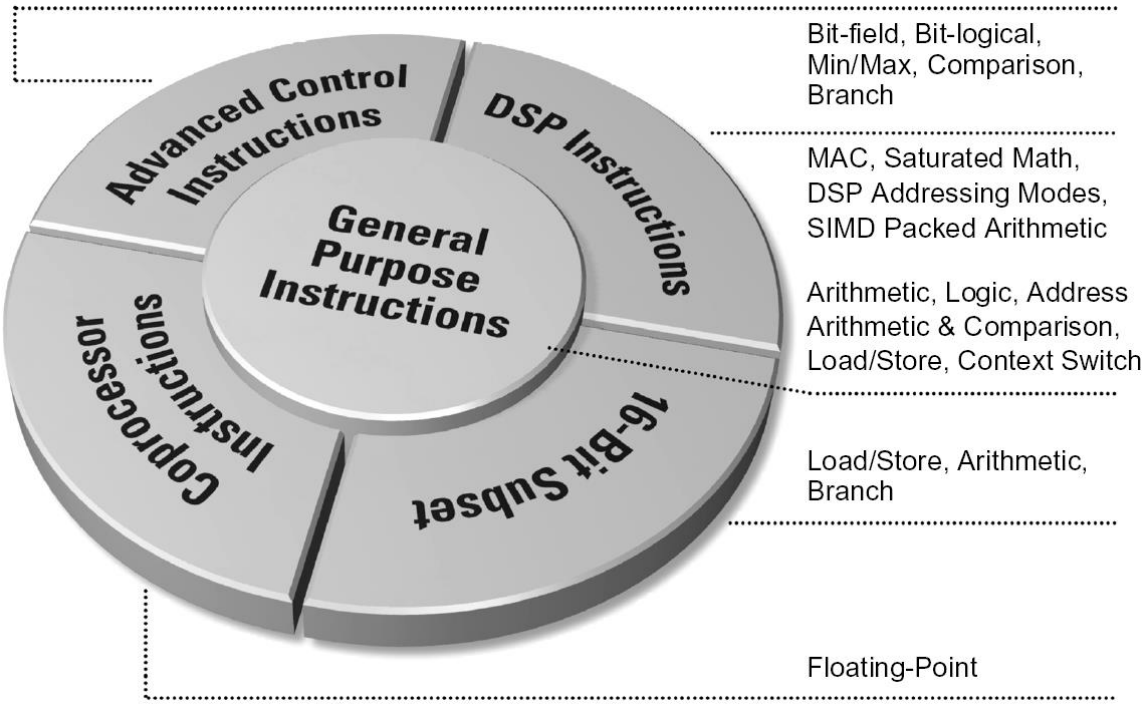
Obrázok 127, Architektúra Tri Core 32 bit. DPS TC10GP od firmy INFINEON



Obrázok 128, Architektúra 32 bit. mikroprocesora TC1775B od firmy INFINEON



Obrázok 129, Architektonické vlastnosti architektúry Tri Core od firmy INFINEON



12.4. Arduino UNO a Raspberry PI

Platforma Arduino[121] a Raspberry PI je vítanou alternatívou pre tých, ktorí sa chcú oboznámiť s programovaním aplikácii s mikroprocesormi. Cenová relácia týchto rôznorodých zariadení je niekoľko jednotiek až desiatok EUR podľa konfigurácie. Nevýhodou tejto platformy je, že k nej neexistuje operačný[122] systém, ktorý by umožňoval vykonávať viac úloh naraz. Programovať[123] je možné pomocou jazyka C, Python, alebo Java Script-ov.

Obrázok 130, Doska plošného spoja Arduino UNO

Obrázok 131, Raspberry PI

[121] http://fritzing.org, Aplikácia Fritzing je špecializovaný software na komplexnú automatizáciu návrhu elektronického dizajnu mikroprocesorových platforiem na báze Arduino, alebo Raspberry PI. Ponúka návrh a simuláciu na "breadboard-e" v reálnom čase s možnosťou využiť knižnicu prídavných súčiastok na veľmi vysokej úrovni.

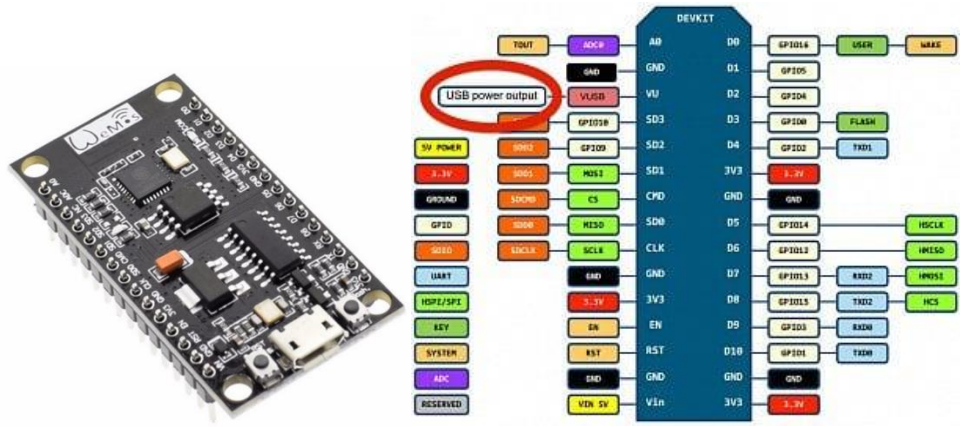
[122] Na https://github.com/feilipu/Arduino_FreeRTOS_Library je k dispozícii operačný systém od Richard Berry info@freertos.org, ktorý využíva WDT časovač s rozlíšením 15ms a je určený na nenáročné nasadenie v praxi.

[123] Packet Tracer umožňuje zápis programu pomocou jazyka C, Java Script a Python s následnou vizuálnou simuláciou.

12.5. NodeMCU

Na trhu je už dlhší čas zaujímavá architektúra pod obchodným názvom NodeMCU[124] Lua 3 Wifi[125], ktorá je určená pre nenáročné aplikácie IoT s konektivitou do internetu. Táto architektúra obsahuje osvedčený a nenáročný modul ESP8266 s 32Mb pamäti FLASH s čipovou súpravou CH340G pre USB sériové rozhranie, 11x GPIO s výstupným prúdom 12mA pri +3,3V a všetky okrem GPIO 16 majú možnosť PWM modulácie, čo postačuje aj pre náročnejšie aplikácie. Výhodu je kompatibilita s Arduino prostredím[126], čo umožňuje využiť už existujúce knižnice a API. Niektoré varianty Node MCU nie sú +5V napäťovo tolerantné, preto dochádza v extrémnych prípadoch k prehriatiu čipu a tým k nesprávnej funkcii. I keď ESP8266 má len jeden analógový vývod, na ktorý je možné priviesť napätie v rozsahu 0 až +1V, štandardne používa napätie +3,3V čo vyhovuje väčšine aplikácií. Keďže výpočtovú jednotku procesora využíva aj modul Wifi, stáva sa, že pri vysokom zaťažení procesora na dlhý čas Wifi „zamrzne“. NodeMCU naviac podporuje štandardné rozhrania ako OneWire, I2C, SPI, UART, ADC ktoré programátorovi podstatným spôsobom rozširujú aplikačné možnosti. Variant dosky V2 aj s prevodníkom CH340 si ľahko pomýlite s V3 Lolin. Variant V3, ktorý je väčší ako V2 a má nápis Lolin, variant V2 má na spodnej strane nápis Amica.

Obrázok 132, NodeMCU s ESP8266 a rozmiestnením vývodov



[124] Výrobcom je firma **Espressif Systems** ktorá túto platformu predstavila už 30.12.2013. V Európe sa dostáva do povedomia až v roku 2014 keď bol integrovaný do dosky. Srdce ESP8266 obsahuje 32 bitový, jednojadrový RISC procesor Xtensa LX 106 s frekvenciou 80MHz a možnosťou pretaktovania až na 160MHz. NodeMCU je typicky Open-Source platforma, ktorá umožňuje programátorom vytvárať projekty nielen pre oblasť automatizácie, ale je určená prioritne na jednoduché projekty ktoré počítajú s využitím internetu ako prenosového média.

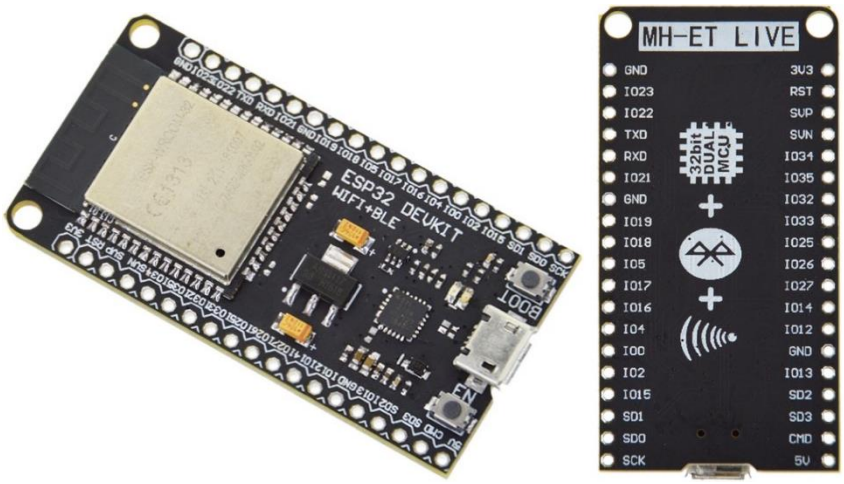
[125] Podporuje štandard **802.11 b/g/n** s frekvenciou 2,4GHz, pričom v pásme **802.11 b** má zisk až 25dB.

[126] Do Arduino prostredia je potrebné nainštalovať v menu prostredia „Manager Additional Board“ link http://arduino.esp8266.com/stable/package_esp8266com_index.json a následne vybrať príslušný hardware zo zoznamu.

12.6. ESP32

Vývojová doska[127] pod označením ESP32 obsahuje oproti predchádzajúcej architektúre NodeMCU s ESP8266 niekoľko podstatných vylepšení. Okrem Wifi obsahuje už aj Bluetooth verzie 4.2 s podporou BLE (Bluetooth Low Energy), dva výkonné procesory s pamäťou SRAM o veľkosti 512kB. Obsahuje 36 GPIO vývodov a bola pridaná podpora zberníc I^2C, SPI a UART.

Obrázok 133, Modul ESP32 DualCore

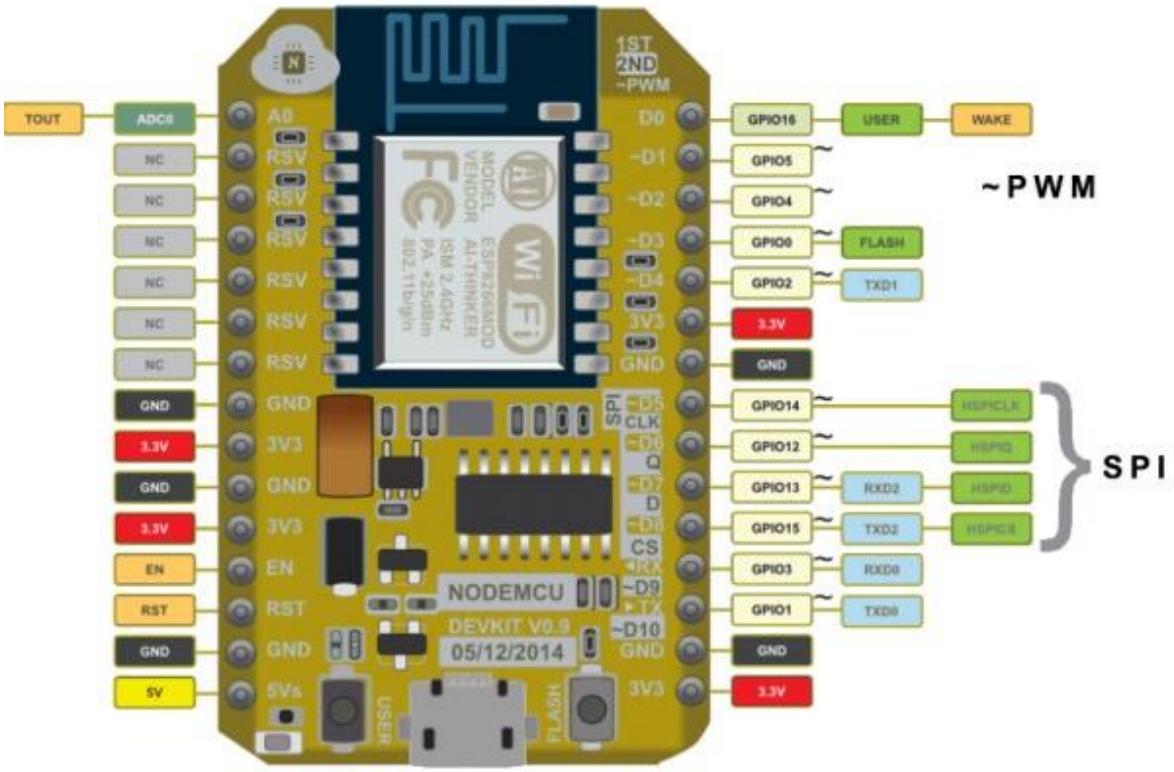


Obrázok 134, Porovnanie vlastností ESP8266 a ESP32

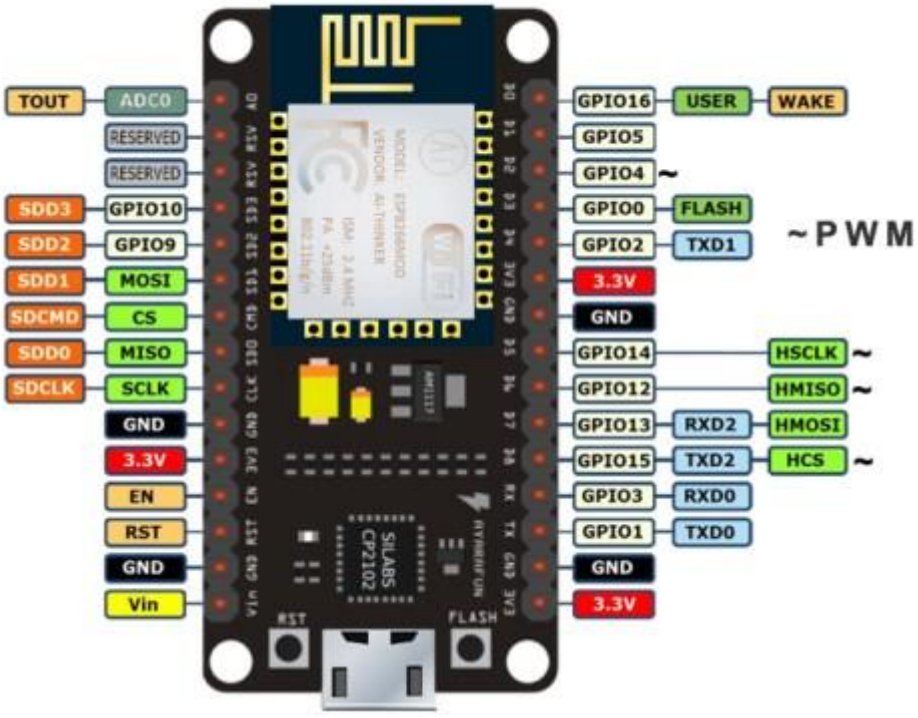
Specifications	ESP8266	ESP32
MCU	Xtensa® Single-Core 32-bit L106	Xtensa® Dual-Core 32-bit LX6 600 DMIPS
802.11 b/g/n Wi-Fi	Yes, HT20	Yes, HT40
Bluetooth	None	Bluetooth 4.2 and below
Typical Frequency	80 MHz	160 MHz
SRAM	160 kBytes	512 kBytes
Flash	SPI Flash , up to 16 MBytes	SPI Flash , up to 16 MBytes
GPIO	17	36
Hardware / Software PWM	None / 8 Channels	1 / 16 Channels
SPI / I2C / I2S / UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	None	1
Ethernet MAC Interface	None	1
Touch Sensor	None	Yes
Temperature Sensor	None	Yes
Working Temperature	-40℃ ~ 125℃	-40℃ ~ 125℃

Obrázok 135, NodeMCU v0.9

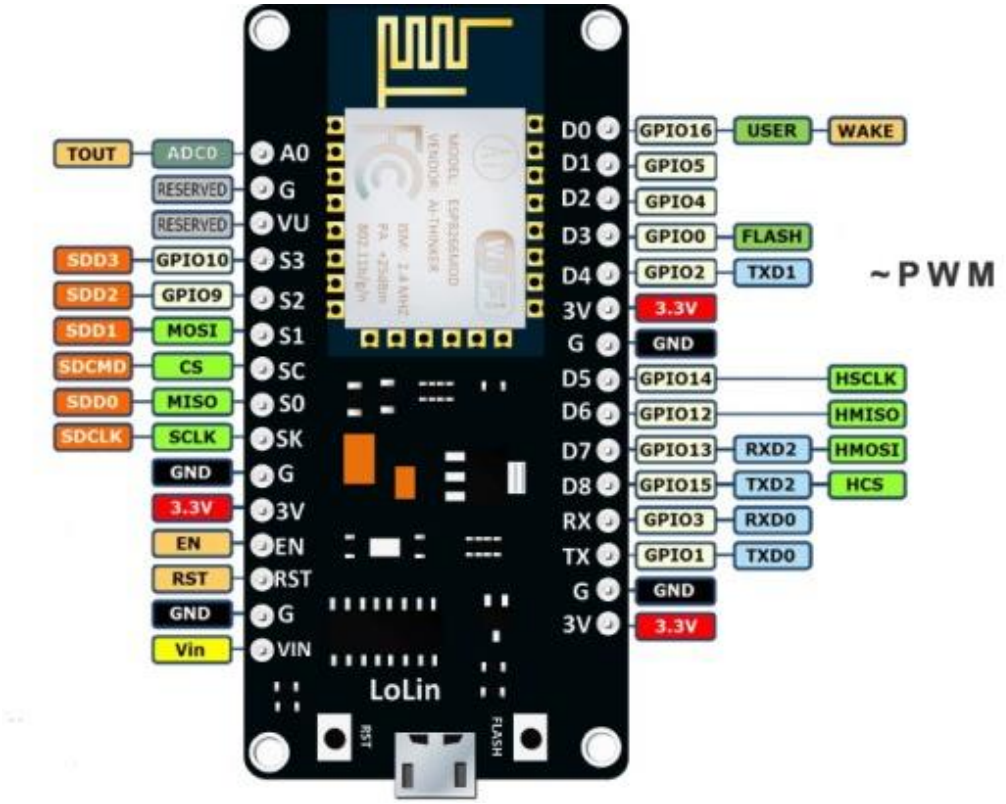
[127] Integráciu dosky ESP32 do prostredia Arduino je možné získať na https://github.com/espressif/arduino-esp32.git



Obrázok 136, NodeMCU v1.0



Obrázok 137, NodeMCU v3.0



Obrázok 138, Rozmiestnenie vývodov ESP32

ESP32 Dev Board PINMAP

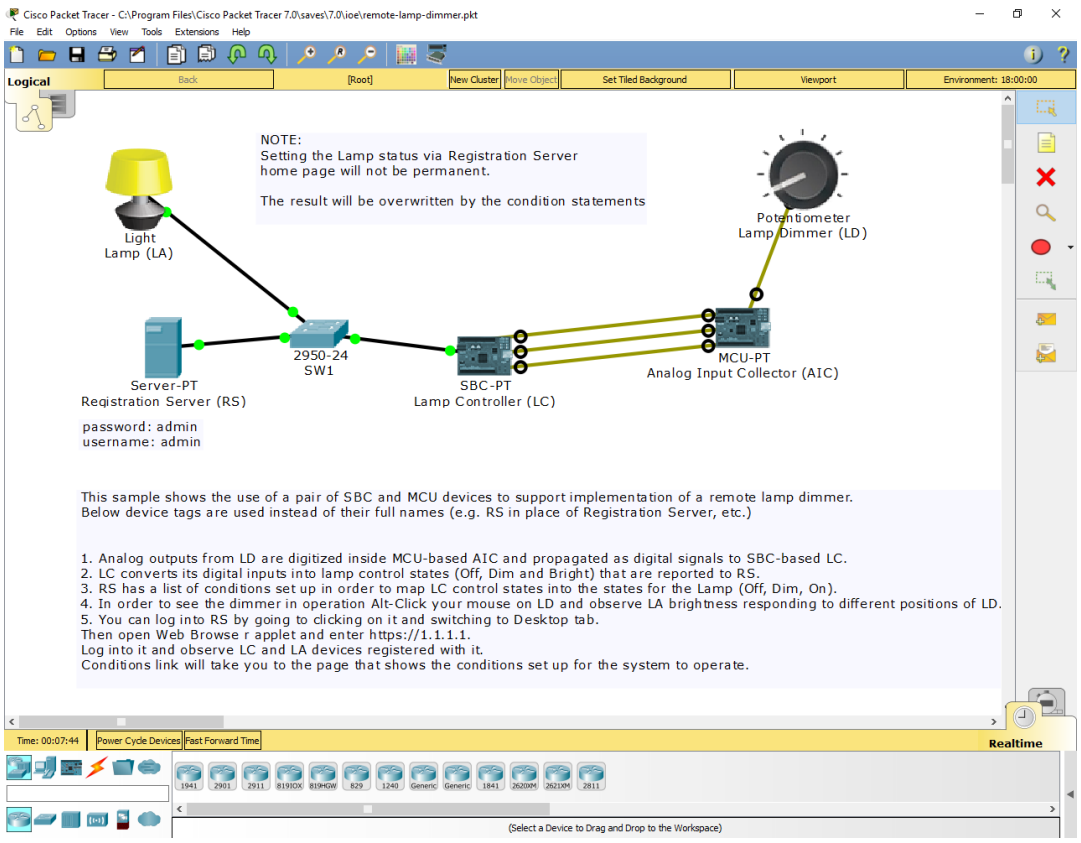
					3.3V		GND						
(pu)				RESET	EN		GPIO23	VSPI MOSI					SPI MOSI
SVP			ADC0		GPI 36		GPIO22						Wire SCL
SVN			ADC3		GPI 39		GPIO1	TX0					Serial TX
			ADC6		GPI 34		GPIO3	RX0					Serial RX
			ADC7		GPI 35		GPIO21						Wire SDA
		TOUCH9	ADC4		GPIO32		GND						
		TOUCH8	ADC5		GPIO33		GPIO19	VSPI MISO					SPI MISO
DAC1			ADC18		GPIO25		GPIO18	VSPI SCK					SPI SCK
DAC2			ADC19		GPIO26		GPIO5	VSPI SS				(pu)	SPI SS
		TOUCH7	ADC17		GPIO27		GPIO17						
	TMS	TOUCH6	ADC16	HSPI SCK	GPIO14		GPIO16						
(pd)	TDI	TOUCH5	ADC15	HSPI MISO	GPIO12		GPIO4		ADC10	TOUCH0		(pd)	
					GND		GPIO0	BOOT	ADC11	TOUCH1		(pu)	
	TCK	TOUCH4	ADC14	HSPI MOSI	GPIO13		GPIO2		ADC12	TOUCH2		(pd)	
				FLASH D2	GPIO9		GPIO15	HSPI SS	ADC13	TOUCH3	TDO	(pu)	
				FLASH D3	GPIO10		GPIO8	FLASH D1					
				FLASH CMD	GPIO11		GPIO7	FLASH D0					
					5V		GPIO6	FLASH SCK					

12.7. Simulačný software - simulátory

V poslednom čase je možné sledovať trend postupného prechodu od architektúr založených na jadre 8051, alebo 80166 na modernejšiu a perspektívnejšiu architektúru ARM. Tomu sa musí prispôsobiť aj výrobca software ktorého snahou je vytvoriť univerzálny software[128] pre tvorbu aplikácií. Jedným z prvých simulátorov je aj Packet Tracer, ktorý sa donedávna používal len na simuláciu zariadení umiestnených v lokálnej počítačovej sieti. Od verzie Packet Tracer 7.0 je možnosť pripojenia modulov Arduino UNO, alebo Raspberry PI a ich programová a vizuálna simulácia.

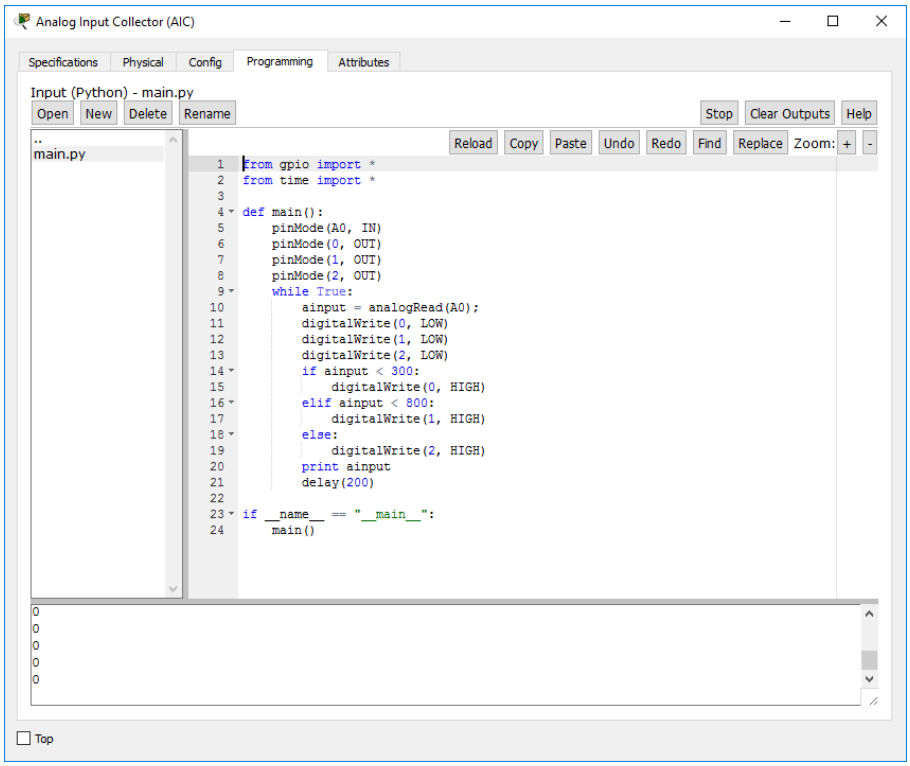
Pripojením týchto modulov do siete Ethernet 802.3, alebo 802.II dostaneme typické zariadenie IoT[129] s omnoho širším spektrom využitia v praxi. Keďže sa jedná o jednoduchý mikroprocesorový modul s prístupom do internetu umožňujeme výmenu údajov medzi zariadením a serverom.

Obrázok 139, Packet Tracer 7.0

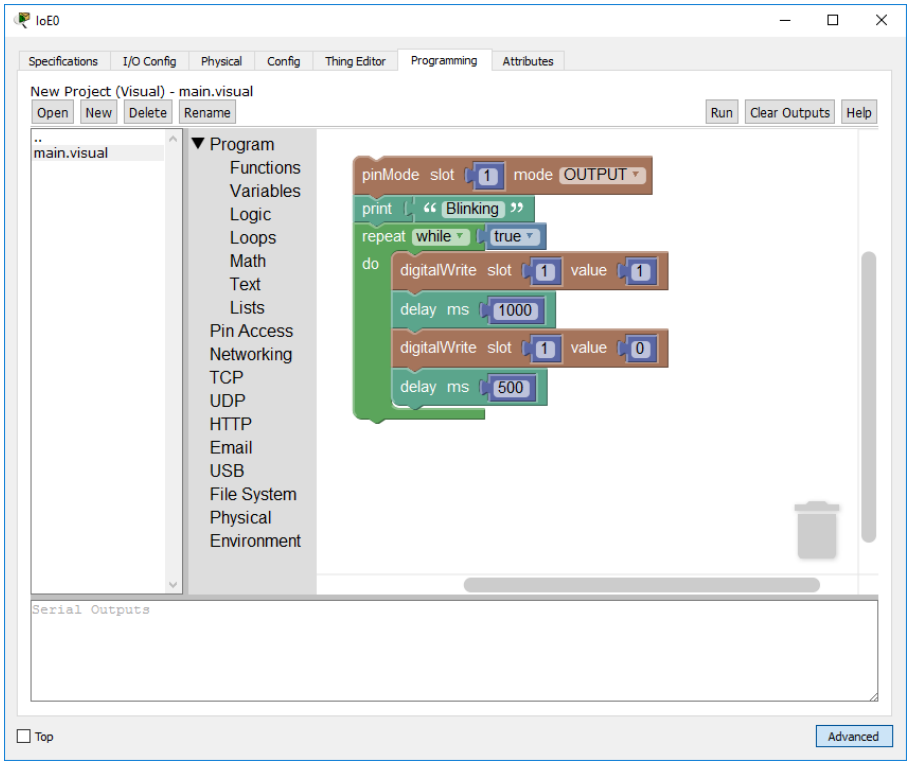


Obrázok 140, Ukážka zápisu programu v prostredí Packet Tracer 7.0

[128] Používa sa programovací jazyk Python, alebo Java Script
[129] IoT – Internet of Things (Internet vecí)



Obrázok 141, Vizuálny zápis programu pomocou štruktúrogramu



13 Zoznam funkcií RTX51

Detailnejší opis funkcií[130] a programovacích techník (Dogan, 2000) operačného systému RTX51 je uvedený v (Keil Elektronik GmbH, et al., 2002), alebo v (Keil Embedded Development, 2016).

13.1. Inicializácia operačného systému RTX51

- *os_start_system(task_number)*

13.2. Funkcie riadenia úloh

- *os_create_task(task_number)*
- *os_delete_task(task_number)*
- *os_running_task_id()*

13.3. Funkcie pre obsluhu prerušení

- *os_attach_interrupt(interrupt)*
- *os_detach_interrupt(interrupt)*
- *os_enable_isr(interrupt)*
- *os_disable_isr(interrupt)*
- *os_wait(event_selector,timeout,0)*
- *os_set_int_masks(ien0,ien1,ien2)*
- *os_reset_int_masks(ien0,ien1,ien2)*

13.4. Funkcie pre synchronizáciu úloh

- *os_send_signal(task_number)*
- *os_wait(event_selector,timeout,0)*
- *os_clear_signal(task_number)*
- *isr_send_signal(task_number)*

[130] Zoznam funkcií operačného systému sú definované v hlavičkovom súbore **RTX51.H**, alebo **RTX51tny.h**

13.5. Funkcie pre výmenu správ medzi úlohami

- *os_send_message(mailbox,message,timeout)*
- *os_wait(event_selector,timeout,*message)*
- *isr_send_message(mailbox,message)*
- *isr_recv_message(mailbox,*message)*

13.6. Funkcie pre zdieľanie systémových zdrojov CPU

- *os_send_token(semaphore)*[131]
- *os_wait(event_selector,timeout,0)*

13.7. Funkcie pre správu externej pamäti

- *os_create_pool(block_size,*memory,mem_size)*
- *os_get_block(block_size)*
- *os_free_block(block_size,*block)*

13.8. Funkcie systémového časovača

- *os_set_slice(timeslice)*
- *os_wait(event_selector,timeout,0)*

13.9. Kontrolné a ladiace funkcie

- *os_check_tasks(*table)*
- *os_check_task(task_number,*table)*
- *os_check_mailboxes(*table)*
- *os_check_mailbox(mailbox,*table)*
- *os_check_semaphores(*table)*
- *os_check_semaphore(semaphore,*table)*
- *os_check_pool(block_size,*table)*

[131] RTX51 Full neumožňuje súčasné použitie **message** a **token** v programe. V súbore **rtxsetup.inc** je zapnutá podpora t.j. rezervovanie pamäte oboch spôsobov komunikácie **?RTX_SEMAPHORE_SUPPORT EQU 1**, a **?RTX_MAILBOX_SUPPORT EQU 1**.

14 Programovanie aplikácií s využitím RTX51 Tiny a RTX51 Full

14.1. Problematika práce s RTX51 Tiny

V tejto kapitole nie je možné dať jednoznačný návod, ako má programátor postupovať pri tvorbe programu s využitím operačného systému RTX51, pretože to je závislé na schopnostiach programátora správne algoritmovať program. Zlému programátorovi po nevykonanom rozbore riešeného problému a nesprávneho algoritmovania nepomôže ani najnovšia, najrýchlejšia a najlepšia verzia akéhokoľvek vývojového nástroja. V tomto prípade bude platiť zlaté programátorské pravidlo, ktoré hovorí: *"Program je práve taký šikovný a dobrý ako jeho programátor"*. Hlavnou úlohou RTX51 Tiny je vykonávanie užívateľom definovaných časových intervalov jednotlivých úloh, takže sa nám môže zdať, že mikroprocesor nevykonáva inú činnosť a len čaká. ***"Ponáhľa sa preto aby čakal, a čaká preto, aby sa potom ponáhľal"***.

14.2. Praktické skúsenosti s RTOS RTX51 Tiny

Po skúsenostiach s RTX51 Tiny, môžem odporučiť tým, ktorý sa rozhodnú pracovať s daným RTOS, že je vhodný práve do jednoduchých, rýchlych a nenáročných aplikácií, ktoré nevyžadujú, aby bol mikroprocesorový systém vybavený externou pamäťou dát. Z hľadiska nárokov na užívateľa vyžaduje tento systém RTOS väčšie nároky na schopnosti programátora. Užívateľ musí byť schopný napísať vlastné rutiny obsluhy prerušenia t.j. ich reakcie na výskyt prerušenia, nemôže používať zložitejšie údajové typy a štruktúry bez prítomnosti externej pamäti, absencia výmeny správ medzi úlohami (*mailbox, semaphore)* atď.

Prepnutie[132] úloh je závislé na polohe a veľkosti zásobníka môže sa pohybovať v rozsahu 100 až 700 cyklov mikroprocesora. Odozva na prerušenie je v prípade RTX51 Tiny menšia ako 20 strojových cyklov[133] mikroprocesora.

Pri modifikovaných architektúrach 8051 môžu byť isté rozdiely, pretože používajú odlišné časovania[134] samotného jadra mikroprocesora. S vyššie uvedenými skutočnosťami musí počítať programátor, pretože mu umožňujú znížiť reakčné časy na vzniknuté udalosti.

---

[132] Pri tasku s prioritou ***_priority_ 3*** je prepnutie vykonané za 50 strojových cyklov mikroprocesora.

[133] Odozva na zaregistrované prerušenie je závislá na nastavení parametra LONG_USR_INTR v súbore **conf_tny.a51**, kde hodnota LONG_USR_INTR=00h je rýchlejšia a LONG_USR_INTR=01h pomalšia odozva na spracovanie prerušenia podľa polohy zásobníka.

[134] Interný delič delí frekvenciu oscilátora v pomere 1:3, 1:4, 1:6. Štandardné architektúry mikroprocesora 8051 delia frekvenciu oscilátora v pomere 1:12. UltraHighSpeed architektúry DS89C420 až DS89C450 pracujú s frekvenciou oscilátora 1:1.

RTX51 Tiny je možné spustiť aj na mikroprocesoroch ktoré majú 128 Byte pamäti RAM, pričom musíme hodnotu RAMTOP uloženú v **conf_tny.a51** zmeniť na hodnotu 7Fh. Aby RTX51 Tiny fungoval správne nesmie výsledné alokovanie aplikácií pamäti v internej RAM prekročiť hodnotu cca. 75 Byte. V prípade ak nepoužívame prerušenia táto hodnota nesmie prekročiť 78 Byte RAM, ináč operačný systém RTX51 Tiny nedokáže správne prepínať úlohy navzájom a dochádza čiastočnej nefunkčnosti niektorých úloh čo je pomerne ťažké odhaliť. Jednoduchšia situácia nastáva vtedy ako alokácia pamäti dát prekročí 80 Byte, vtedy už dochádza k viditeľnej nefunkčnosti programu s RTX51 Tiny. Pri RTX51 Full je situácia odlišná, pretože už pracujeme s externou pamäťou dát a pre správnu funkčnosť si vyžaduje RTX51 cca. 900 Byte pamäti dát. Zároveň treba počítať aj cca. 100 Byte pamäti na každý task a zapuzdrené premenné.

14.3. Problematika práce s RTX51 Full

RTX51 Full je celkovo robustnejší a sofistikovanejší operačný systém oproti RTX51 Tiny. Obsahuje detailne prepracovaný plánovač, ktorý umožňuje „násilne“ ukončiť úlohu čo tento operačný systém zaraďuje do kategórie RTOS[135]. Z vyššie uvedeného vyplývajú aj celkovo vyššie nároky na hardware, čo pri dnešných cenách za jednotlivé komponenty ako sú pamäť RAM, alebo EPROM nie je problémom. Veľkou výhodou RTX51 Full je možnosť spracovať súčasne viac požiadaviek (Timeout, Interval, Signal, Interrupt, Mailbox, Semaphore) a ich kombinácie. Ďalšou výhodou je možnosť využiť funkcie správy dynamického prideľovania pamäti pomocou funkcie **os_create_pool()**, čo v konečnom dôsledku umožňuje tvoriť úzko špecifické aplikácie pracujúce so štruktúrovaným blokmi dát, kde s prípadnou kombináciou mailbox-u je možné adresu štruktúry odoslať ľubovoľnému task-u v rámci operačného systému na ďalšie spracovanie.

[135] RTOS - Real Time Operating System

15 Príklady programov v ASM, C51 a RTX51 Tiny

15.1. Delay v ASM ako funkcia

c:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\Delay.asm

```
1   //-----------------------------------------------------
2   public       _Delay          ;Volanie funkcie s parametrom
3   //-----------------------------------------------------
4   Prog2        segment  code   ;Vytvaram cakaciu slucku cca. 300us pre zapis do radica HD pri 0x80
5                rseg  Prog2
6   //-----------------------------------------------------
7   _Delay:      djnz  r7,_Delay ;Cas do registra R7
8                ret             ;Vratim sa z rutiny podprogramu
9   //-----------------------------------------------------
10               end
11
```

Poznámka: funkcia **Delay(unsigned char)** je volaná[136] s parametrom určujúcim dĺžku oneskorenia s údajovým typom unsigned char. Príklad zápisu funkcie je **Delay(0x80)** čo pri 18,432MHz kryštáli predstavuje oneskorenie cca. 300µs.

15.2. Delay v C51 pomocou cyklu while

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
228  void Delay(unsigned char Dlzka) //reentrant
229  {
230   while((--Dlzka)!=0x00);
231  }
232
```

15.3. Readkey ako funkcia

C:\Omega\data\Dropbox\Záloha\C51\Display\readkey.asm

```
1   //-----------------------------------------------------
2   public       Readkey      ;Volanie funkcie bez parametra
3   //-----------------------------------------------------
4   Prog0        segment  code
5                rseg  Prog0
6   //-----------------------------------------------------
7   Readkey:     jnb  RI,$    ;Cakam na prijem znaku z RS232
8                mov  r7,SBUF ;Ulozim ako parameter do R7
9                mov  r6,#00h ;Vratim vysledok v registroch R6+R7 int
10               clr  RI      ;Zmazem priznak prijmu znaku
11               ret          ;Vratim sa z rutiny podprogramu
12  //-----------------------------------------------------
13               end
14
```

Poznámka: funkcia Readkey[137] odovzdáva výsledok typu *char* volaním Znak=Readkey().

[136] Funkcia **Delay()** je deklarovaná v assembleri ako ***public _Delay***. Podtrhnutie pred názvom funkcie vyžaduje a očakáva vstupné parametre pre funkciu ktorá vracia (**int**) a ktoré pri volaní musia byť umiestnené ako vstupné parametre funkcie do registrov R6 a R7.

[137] **Readkey()** je obdoba funkcie **_getkey()** ktorú používa jazyk C. V samotnej podstate jazyk C odovzdáva pomocou funkcie v skutočnosti vždy len výsledok typu **int**. Údajový typ **char** použitý vo funkcii **Readkey()** je v tomto prípade vlastne pretypovaný **int** na typ **char**. Výsledok assembler-ovskej funkcie je odovzdaný jazyku C pomocou registra R6 a R7. MSB je odovzdané v registri R6 a LSB v R7.

15.4. Reset v jazyku assembler

C:\Omega\data\Dropbox\Záloha\C51\Display\reset.asm

```
1   //***************************************
2   public       Reset
3   extrn  code  (?C_STARTUP)
4   //***************************************
5   Prog_Reset   segment  code
6                rseg  Prog_Reset
7   //---------------------------------------
8   Reset:       clr  ea
9                jmp  ?C_STARTUP
10               ret
11  //---------------------------------------
12               end
```

V jazyku C je možné reset vykonať programovo volaním **((void (code*) (void)) 0x0000) ()**, kde číslo 0x0000[138] je adresa skoku mikroprocesora po vykonaní resetu.

15.5. Gregoriánsky kalendár v C51 ako procedúra

C:\Omega\data\Dropbox\Záloha\C51\clock full 2\Clock Full 2.c

```
241  unsigned char Gregorian (unsigned char Den,Mes,unsigned int Rok)
242  {
243   unsigned char Pocet,den,mes,nazov=Tue;        //Tu som menil na unsigned char, zabera menej
     miesta v pamati
244   unsigned int rok;
245   for (rok=2002; rok<=2099; rok++)
246    {
247     Datum.Tyzden=0x01;
248     Datum.Poradie=0x01;
249     for (mes=Jan; mes<=Dec; mes++)
250      {
251  //    os_wait(K_TMO,1,NULL);
252       Pocet=Pocet_Dni[mes-1];
253       if (mes==Feb) {if (rok%4==0) Pocet=29; else Pocet=28;}
254       for (den=1; den<=Pocet; den++)
255        {
256         if ((Den==den)&&(Mes==mes)&&(Rok==rok))
257         {
258            if (((Rok%4)!=0)&&(Mes>=3)) Datum.Poradie++; //Korekcia mien na prestupnost rokov
259        return (nazov);
260       }
261       if (nazov++>=Sun) {nazov=Mon; Datum.Tyzden++;}
262          if (Datum.Tyzden>53) Datum.Tyzden=0x01;        //Bolo tu toto asi zle: Datum.Tyzden>=53
263          Datum.Poradie++;
264        }
265      }
266    }
267   return (nazov);
268  }
269
270  void Vypocet_Datumu (void)
271  {
272   Datum.Nazov=Gregorian (Datum.Den,Datum.Mes,Datum.Rok);
273  //  Datum.Nazov=Gregor(Datum.Den,Datum.Mes,(Datum.Rok-1900));
274  }
```

Poznámka: Vyššie uvedené procedúra pre výpočet poradového čísla dňa je veľmi podobná ako vo funkcii programu napísaného v assembler-i uvedeného v kapitole 15.6.

[138] Táto hodnota sa môže líšiť podľa typu mikroprocesora. Implicitne je pri mikroprocesore 8051 nastavená na hodnotu 0000h. S vyššie uvedenou adresou súvisí aj tabuľka vektorov prerušenia v súbore **rtxsetup.inc**, kde je potrebné potom v položke **?RTX_INTBASE EQU 0000H** nastaviť novú hodnotu.

15.6. Gregoriánsky kalendár v assembleri ako funkcia

C:\Omega\data\Dropbox\Záloha\C51\clock full 2\KALENDAR.ASM

```
1   public      _Gregor
2   ;-------------------------------------------------------------------------------------------
3   Vars        segment  data
4               rseg  Vars
5   ;-------------------------------------------------------------------------------------------
6   Den :       ds  1
7   Mes :       ds  1
8   Rok :       ds  1
9   Poc_Dni :   ds  1
10  Akt_Den :   ds  1
11  Nas_Den :   ds  1
12  Nas_Mes :   ds  1
13  Nas_Rok :   ds  1
14  ;-------------------------------------------------------------------------------------------
15  Tabs        segment  code
16              rseg  Tabs
17  ;-------------------------------------------------------------------------------------------
18  Tab1 :      db  00h,31d,29d,31d,30d,31d,30d,31d,31d,30d,31d,30d,31d   ;Priestupny rok ...
19  Tab2 :      db  00h,31d,28d,31d,30d,31d,30d,31d,31d,30d,31d,30d,31d   ;Normalny rok ...
20  ;-------------------------------------------------------------------------------------------
21  Progs0      segment  code
22              rseg  Progs0
23  ;-------------------------------------------------------------------------------------------
24  _Gregor :   mov  Nas_Den,r7           ;r7+r6
25              mov  Nas_Mes,r5           ;r5+r4
26              mov  Nas_Rok,r3           ;r3+r2   24.2.2002, udava sa v tvare Rok-1900 !!!
27              mov  Rok,#01h             ;1901, rok 101 je 2001
28              mov  Mes,#01h             ;Januar
29              mov  Den,#01h             ;1.1.1901
30              mov  Akt_Den,#02h         ;1.1.1901 bol Utorok
31  ?PR?Greg :  mov  a,Rok
32              mov  b,#04h
33              div  ab
34              mov  a,b
35              cjne  a,#00h,PreNo
36  PreYes :    mov  dptr,#Tab1           ;Pocet dni v mesiaci pre rok
37              jmp  PreEnd
38  PreNo :     mov  dptr,#Tab2           ;Pocet dni v mesiaci pre rok
39  ;-------------------------------------------------------------------------------------------
40  PreEnd :    mov  a,Mes
41              movc  a,@a+dptr           ;Z TabX vyberiem spravny pocet dni v mesiaci
42              mov  Poc_Dni,a
43              mov  a,Den
44              mov  r0,#Akt_Den          ;Nepriamo adresujem premennu Akt_Den
45              jmp  Cyklus
46  ?PR?XXX :   inc  @r0
47              inc  Den
48              mov  a,Den
49              cjne  @r0,#08h,Cyklus
50              mov  @r0,#01h
51  ;-------------------------------------------------------------------------------------------
52  Cyklus :    push  acc
53              mov  a,Den
54              cjne  a,Nas_Den,?PR?L111  ;Tu mozem zastavit na aktualnom datume XX.XX.?PR?XXXX !!!
55              mov  a,Mes
56              cjne  a,Nas_Mes,?PR?L111
57              mov  a,Rok
58              cjne  a,Nas_Rok,?PR?L111
59              pop  acc                  ;Ak sa dostanem tu, nasiel som akt. datum
60              mov  r7,Akt_Den
61              ret                       ;Ukoncujem funkciu
62  ;-------------------------------------------------------------------------------------------
63  ?PR?L111 :  pop  acc
64              cjne  a,Poc_Dni,?PR?XXX   ;Tu mozem zistit den pripadajuci na 31.1.1901 !!!
65              inc  Akt_Den              ;Prechadzam do dalsieho mesiaca, tak aktualizujem aj Akt_Den
66              mov  r0,#Akt_Den          ;Pracujem s nepriamou adresaciou, pretoze je to teraz vyhodne
67              cjne  @r0,#08h,?PR?YYY    ;Aby nahodou nepridalo osmy den ....
68              mov  Akt_Den,#01h         ;Ak je osmy den, potom nastav na 1
69  ?PR?YYY :   mov  Den,#01h
70              mov  r0,#Mes
71              inc  @r0                  ;Zvysim o jeden mesiac
72              cjne  @r0,#13d,?PR?Greg   ;Uz som presiel 12 mesiac, musim pretocit na 1 mesiac
73              mov  @r0,#01h             ;Nastavim 1 mesiac
74              inc  Rok
75              jmp  ?PR?Greg
76  ?PR?Cykl :  call  ?PR?Greg
77              inc  Rok                  ;Ak som doteraz nenasiel idem na dalsi rok
78              jmp  ?PR?Cykl
79              end
```

Poznámka: funkcia **unsigned char Gregor(unsigned char, unsigned char, unsigned char)** vracia číslo v rozsahu 1 až 7 predstavujúce deň v týždni. Od pondelka do nedele.

15.7. Kvadratická rovnica

V základnom tvare je kvadratická rovnica vyjadrená ako:

$$a \cdot x^2 + b \cdot x + c = 0$$

Príklad výpočtu:

Koeficienty kvadratickej rovnice sú dané: a=1, b=1, c=1.
Riešenie je možné pomocou diskriminantu od ktorého závisí či rovnica bude mať jeden, dva, korene v oblasti reálnych čísel, alebo v prípade záporného diskriminantu riešenie v obore komplexných čísel:

$$D = b^2 - 4 \cdot a \cdot c$$

$$D = b^2 - 4 \cdot a \cdot c = 1^2 - 4 \cdot 1 \cdot 1 = -3$$

$$x_{1,2} = \frac{-b \pm \sqrt{\mathrm{D}}}{2 \cdot a} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Ak $D > 0$ potom riešením budú dva korene v obore reálnych čísel

$$x_{1,2} = \frac{-b \pm \sqrt{\mathrm{D}}}{2 \cdot a}$$

Ak $D = 0$ potom riešením bude jeden koreň v obore reálnych čísel

$$x_1 = -\frac{b}{2 \cdot a}$$

Ak $D < 0$ potom riešenie bude dvojica koreňov v obore komplexných čísel

$$x_{1,2} = \frac{-b \pm \sqrt{\mathrm{D}}}{2 \cdot a} = \frac{-b \pm i \cdot \sqrt{|D|}}{2 \cdot a} = \frac{-1 \pm i \cdot \sqrt{|-3|}}{2 \cdot 1} = \frac{-1 \pm i \cdot \sqrt{3}}{2} = -\frac{1}{2} \pm \frac{\sqrt{3}}{2} \cdot i$$

$$x_1 = -\frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i$$

$$x_2 = -\frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i$$

Kontrola správnosti výpočtu:

$$(x - x_1) \cdot (x - x_2) = 0$$

$$\left(x + \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i\right) \cdot \left(x + \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i\right) = 0$$

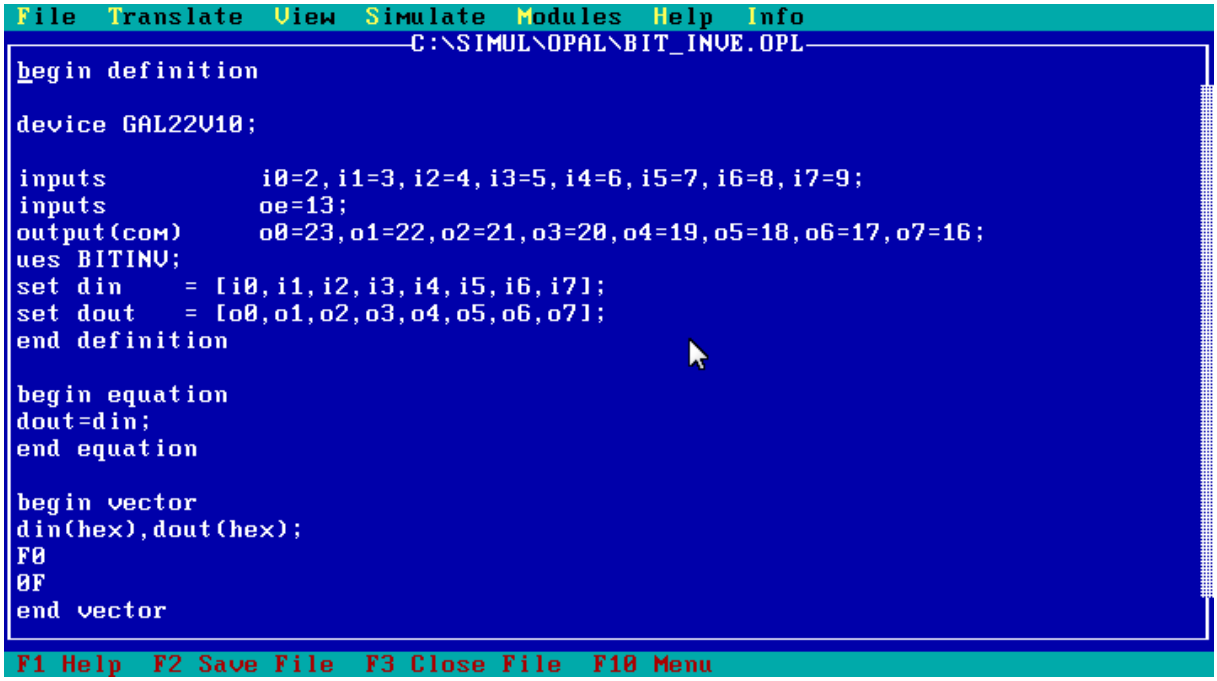
Obrázok 142, Program pre výpočet kvadratickej rovnice v rovine komplexných čísel

```
#include <reg51.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct _complex
{
 float re,im;
}
complex;              //Deklaracia premennej komplexne cislo ...

typedef struct _coeficient
{
 float ka,kb,kc;
}
coeficient;

float D;              //Diskriminant
coeficient c;         //Koeficienty a,b,c
complex x1,x2;        //Vysledky

void main(void)
{
 TI=1;
 c.ka=1.0;
 c.kb=1.0;
 c.kc=1.0;
 while(1)
  {
    D=c.kb*c.kb-4*c.ka*c.kc;
   if(D>0)
    {
     x1.re=(-c.kb+sqrt(D))/(2*c.ka);
     x2.re=(-c.kb-sqrt(D))/(2*c.ka);
     x1.im=x2.im=0;
    }
   if(D==0)
    {
     x1.re=x2.re=-c.kb/(2*c.ka);
     x1.im=x2.im=0;
    }
   if(D<0)
    {
     x1.re=-c.kb/(2*c.ka);
     x1.im=+sqrt(abs(D))/(2*c.ka);
     x2.re=-c.kb/(2*c.ka);
     x2.im=-sqrt(abs(D))/(2*c.ka);
    }
   printf("%+f%+fi\r\n",x1.re,x1.im);
   printf("%+f%+fi\r\n",x2.re,x2.im);
  }
}
```

15.8. Bitový „invertor“ portu

V praxi je niekedy potrebné medzi sebou zameniť jednotlivé významové bity portu medzi sebou. Na internetových stránkach[139] diskusných skupín je možné nájsť množstvo spôsobov ako zameniť bity medzi sebou. Niektoré využívajú metódy výpočtu[140] a ďalšie využívajú tabuľkové spôsoby spracovania údajov. Uvedený program využíva metódu výberu a náhradu dát z tabuľky, čím dosahujeme vysokú efektivitu a rýchlosť[141] oproti výpočtovým metódam. Pri vstupných hodnotách „abcdefgh” dostaneme na výstupe „hgfedcba“.

Obrázok 143, Bitový invertor s EPLD



Programovateľné logické polia tvoria v elektronike významnú skupinu obvodov, ktorých funkcia sa dá programovo meniť. Väčšinou plnia funkciu kombinačných, alebo sekvenčných logických obvodov. Modernejšie obvody MAPL128[142] už dokážu vytvoriť aj zložitejšie elektronické konštrukcie napr. arbiter ktorý riadi asynchrónne prístup dvoch procesorov do jednej pamäti RAM.

[139] http://www.keil.com/forum/16883/
[140] http://8052.com/forum/read/90817
[141] Čas spracovania algoritmu výberu z tabuľky trvá približne 11 cyklov mikroprocesora, čo pri 12MHz kryštáli predstavuje typicky približne t=11,00µs čo je oproti iným technikám minimálne dvojnásobné urýchlenie. Ak by sme požadovali rýchlosti pod t=5,00ns je potrebné využiť programovateľné hradlové pole napr. GAL22V10 ktoré je schopné tento bitový invertor vytvoriť pomocou programovateľných logických hradiel AND a OR.
[142] Texas Instruments, www.ti.com

Obrázok 144, Bitový invertor portu

```
#include <reg52.h>
#include <stdio.h>
#include <stdlib.h>

extern unsigned char inverter1(unsigned char);
extern unsigned char inverter2(unsigned char);
extern unsigned char inverter3(unsigned char);

void Ovl(void)
{
  int i;
  i+=1;
}

unsigned char code Table[]=
{
 0x00,0x80,0x40,0xC0,0x20,0xA0,0x60,0xE0,
 0x10,0x90,0x50,0xD0,0x30,0xB0,0x70,0xF0,
 0x08,0x88,0x48,0xC8,0x28,0xA8,0x68,0xE8,
 0x18,0x98,0x58,0xD8,0x38,0xB8,0x78,0xF8,
 0x04,0x84,0x44,0xC4,0x24,0xA4,0x64,0xE4,
 0x14,0x94,0x54,0xD4,0x34,0xB4,0x74,0xF4,
 0x0C,0x8C,0x4C,0xCC,0x2C,0xAC,0x6C,0xEC,
 0x1C,0x9C,0x5C,0xDC,0x3C,0xBC,0x7C,0xFC,
 0x02,0x82,0x42,0xC2,0x22,0xA2,0x62,0xE2,
 0x12,0x92,0x52,0xD2,0x32,0xB2,0x72,0xF2,
 0x0A,0x8A,0x4A,0xCA,0x2A,0xAA,0x6A,0xEA,
 0x1A,0x9A,0x5A,0xDA,0x3A,0xBA,0x7A,0xFA,
 0x06,0x86,0x46,0xC6,0x26,0xA6,0x66,0xE6,
 0x16,0x96,0x56,0xD6,0x36,0xB6,0x76,0xF6,
 0x0E,0x8E,0x4E,0xCE,0x2E,0xAE,0x6E,0xEE,
 0x1E,0x9E,0x5E,0xDE,0x3E,0xBE,0x7E,0xFE,
 0x01,0x81,0x41,0xC1,0x21,0xA1,0x61,0xE1,
 0x11,0x91,0x51,0xD1,0x31,0xB1,0x71,0xF1,
 0x09,0x89,0x49,0xC9,0x29,0xA9,0x69,0xE9,
 0x19,0x99,0x59,0xD9,0x39,0xB9,0x79,0xF9,
 0x05,0x85,0x45,0xC5,0x25,0xA5,0x65,0xE5,
 0x15,0x95,0x55,0xD5,0x35,0xB5,0x75,0xF5,
 0x0D,0x8D,0x4D,0xCD,0x2D,0xAD,0x6D,0xED,
 0x1D,0x9D,0x5D,0xDD,0x3D,0xBD,0x7D,0xFD,
 0x03,0x83,0x43,0xC3,0x23,0xA3,0x63,0xE3,
 0x13,0x93,0x53,0xD3,0x33,0xB3,0x73,0xF3,
 0x0B,0x8B,0x4B,0xCB,0x2B,0xAB,0x6B,0xEB,
 0x1B,0x9B,0x5B,0xDB,0x3B,0xBB,0x7B,0xFB,
 0x07,0x87,0x47,0xC7,0x27,0xA7,0x67,0xE7,
 0x17,0x97,0x57,0xD7,0x37,0xB7,0x77,0xF7,
 0x0F,0x8F,0x4F,0xCF,0x2F,0xAF,0x6F,0xEF,
 0x1F,0x9F,0x5F,0xDF,0x3F,0xBF,0x7F,0xFF,
};

unsigned char i;

void main(void)
{
 while(1)
  {
   i=0x33;
   P1=Table[i];
   P1=inverter1(i);
   P1=inverter2(i);
   P1=inverter3(i);
  }
}
```

15.9. Funkcie bitového invertoru portu v assembleri

```
1   public      _inverter1,_inverter2,_inverter3,_Tabulka
2   ;------------------------------------------------------
3   prog0       segment code
4               rseg prog0
5   ;------------------------------------------------------
6   _inverter1: mov a,r7        ;Funkcia invertuje bity R7 systemom R7.7 na R7.0, R7.6 na R7.1 ...
7               mov c,acc.0
8               mov b.7,c
9               mov c,acc.1
10              mov b.6,c
11              mov c,acc.2
12              mov b.5,c
13              mov c,acc.3
14              mov b.4,c
15              mov c,acc.4
16              mov b.3,c
17              mov c,acc.5
18              mov b.2,c
19              mov c,acc.6
20              mov b.1,c
21              mov c,acc.7
22              mov b.0,c
23              mov r7,b
24              ret
25  ;------------------------------------------------------
26  prog1       segment code
27              rseg prog1
28  ;------------------------------------------------------
29  _inverter2: mov a,r7        ;Funkcia invertuje bity R7 systemom R7.7 na R7.0, R7.6 na R7.1 ...
30              mov c,acc.1
31              rlc a
32              mov acc.2,c
33              mov c,acc.3
34              rlc a
35              mov acc.4,c
36              mov c,acc.5
37              rlc a
38              mov acc.6,c
39              swap a
40              mov r7,a
41              ret
42  ;------------------------------------------------------
43  prog2       segment code
44              rseg prog2
45  ;------------------------------------------------------
46  _inverter3: mov a,r7        ;Funkcia invertuje bity R7 systemom R7.7 na R7.0, R7.6 na R7.1 ...
47              mov dptr,#_Tabulka
48              movc a,@a+dptr
49              mov r7,a
50              ret
51  ;------------------------------------------------------
52  prog3       segment code
53              rseg prog3
54  ;------------------------------------------------------
55  _Tabulka:   db  0x00,0x80,0x40,0xC0,0x20,0xA0,0x60,0xE0
56              db  0x10,0x90,0x50,0xD0,0x30,0xB0,0x70,0xF0
57              db  0x08,0x88,0x48,0xC8,0x28,0xA8,0x68,0xE8
58              db  0x18,0x98,0x58,0xD8,0x38,0xB8,0x78,0xF8
59              db  0x04,0x84,0x44,0xC4,0x24,0xA4,0x64,0xE4
60              db  0x14,0x94,0x54,0xD4,0x34,0xB4,0x74,0xF4
61              db  0x0C,0x8C,0x4C,0xCC,0x2C,0xAC,0x6C,0xEC
62              db  0x1C,0x9C,0x5C,0xDC,0x3C,0xBC,0x7C,0xFC
63              db  0x02,0x82,0x42,0xC2,0x22,0xA2,0x62,0xE2
64              db  0x12,0x92,0x52,0xD2,0x32,0xB2,0x72,0xF2
65              db  0x0A,0x8A,0x4A,0xCA,0x2A,0xAA,0x6A,0xEA
66              db  0x1A,0x9A,0x5A,0xDA,0x3A,0xBA,0x7A,0xFA
67              db  0x06,0x86,0x46,0xC6,0x26,0xA6,0x66,0xE6
68              db  0x16,0x96,0x56,0xD6,0x36,0xB6,0x76,0xF6
69              db  0x0E,0x8E,0x4E,0xCE,0x2E,0xAE,0x6E,0xEE
70              db  0x1E,0x9E,0x5E,0xDE,0x3E,0xBE,0x7E,0xFE
71              db  0x01,0x81,0x41,0xC1,0x21,0xA1,0x61,0xE1
72              db  0x11,0x91,0x51,0xD1,0x31,0xB1,0x71,0xF1
73              db  0x09,0x89,0x49,0xC9,0x29,0xA9,0x69,0xE9
74              db  0x19,0x99,0x59,0xD9,0x39,0xB9,0x79,0xF9
75              db  0x05,0x85,0x45,0xC5,0x25,0xA5,0x65,0xE5
```

```
76                db  0x15,0x95,0x55,0xD5,0x35,0xB5,0x75,0xF5
77                db  0x0D,0x8D,0x4D,0xCD,0x2D,0xAD,0x6D,0xED
78                db  0x1D,0x9D,0x5D,0xDD,0x3D,0xBD,0x7D,0xFD
79                db  0x03,0x83,0x43,0xC3,0x23,0xA3,0x63,0xE3
80                db  0x13,0x93,0x53,0xD3,0x33,0xB3,0x73,0xF3
81                db  0x0B,0x8B,0x4B,0xCB,0x2B,0xAB,0x6B,0xEB
82                db  0x1B,0x9B,0x5B,0xDB,0x3B,0xBB,0x7B,0xFB
83                db  0x07,0x87,0x47,0xC7,0x27,0xA7,0x67,0xE7
84                db  0x17,0x97,0x57,0xD7,0x37,0xB7,0x77,0xF7
85                db  0x0F,0x8F,0x4F,0xCF,0x2F,0xAF,0x6F,0xEF
86                db  0x1F,0x9F,0x5F,0xDF,0x3F,0xBF,0x7F,0xFF
87    ;-----------------------------------------------------
88
89                end
```

15.10. Vzájomná výmena obsahu dvoch premenných bez pomocnej premennej

V programátorskej praxi neraz potrebujeme vymeniť obsah dvoch premenných, pričom je nežiadúce, ak sa pri tom používa pomocná premenná[143].

Obrázok 145, Výmena obsahu premenných bez pomocnej premennej

```
#include <reg51.h>
#include <stdio.h>
#include <stdlib.h>

int x,y;

void main(void)
{
 x=0x1234;
 y=0x5678;
 while(1)
  {
   x=x+y;          //Priprava na vymenu x ...
   y=x-y;          //Priprava na vymenu y ...
   x=x-y;          //Uz su tu vymenene hodnoty x a y navzajom ...
  }
}
```

[143] Táto situácia môže nastať ak je málo pamäti v mikroprocesore.

15.11. Ovládač sériového rozhrania RS232 v C51 bez prerušenia

```
1   /*---------------------------------------------------------------------------
2   HELLO.C
3
4   Copyright 1995-2005 Keil Software, Inc.
5   ---------------------------------------------------------------------------*/
6
7   #include <REG52.H>                 /* special function register declarations    */
8                                      /* for the intended 8051 derivative          */
9
10  #include <stdio.h>                 /* prototype declarations for I/O functions */
11
12
13  #ifdef MONITOR51                           /* Debugging with Monitor-51 needs    */
14  char code reserve [3] _at_ 0x23;           /* space for serial interrupt if      */
15  #endif                                     /* Stop Exection with Serial Intr.    */
16                                             /* is enabled                         */
17
18
19  /*-------------------------------------------------
20  The main C function.   Program execution starts
21  here after stack initialization.
22  -------------------------------------------------*/
23  void main (void)
24  {
25
26  /*-------------------------------------------------
27  Setup the serial port for 1200 baud at 16MHz.
28  -------------------------------------------------*/
29  #ifndef MONITOR51
30      SCON  = 0x50;                  /* SCON: mode 1, 8-bit UART, enable rcvr      */
31      TMOD |= 0x20;                  /* TMOD: timer 1, mode 2, 8-bit reload        */
32      TH1   = 221;                   /* TH1:   reload value for 1200 baud @ 16MHz  */
33      TR1   = 1;                     /* TR1:  timer 1 run                          */
34      TI    = 1;                     /* TI:    set TI to send first char of UART   */
35  #endif
36
37  /*-------------------------------------------------
38  Note that an embedded program never exits (because
39  there is no operating system to return to).   It
40  must loop and execute forever.
41  -------------------------------------------------*/
42    while (1)
43    {
44      P1 ^= 0x01;                    /* Toggle P1.0 each time we print */
45      printf ("Hello World\n");      /* Print "Hello World"  */
46    }
47  }
48
49
50
```

Popis programu: Vyššie uvedený program využíva štandardnú knižnicu na výpis textového reťazca pomocou funkcie printf(“Hello world\n“)[144] reťazec “Hello world” s prenosovou rýchlosťou 19200 b.s^{-1}. Pri použití RTX51 Tiny nie je potrebné vykonať žiadne zmeny

[144] Pri použití funkcie **printf()** musí byť správne inicializované sériové rozhranie RS232 a na záver inicializačnej sekvencie nastavený bit **TI=1**, v opačnom prípade funkcia nebude vôbec fungovať. Bit TI=1 musíme nastaviť len vtedy ak nepoužívame pri prenose dát prerušenie od sériového kanála. V opačnom prípade by sa nám periodicky vyvolávalo prerušenie od sériového kanála na adrese 0023h.

v programe. Jediné obmedzenie[145] použitie funkcie printf() v RTX51 Tiny je podstatné v tom, že nereentrantnú[146] funkciu je možné použiť len v jednom task-u.

[145] Obmedzenie je možné odstrániť povolením overaly-u a zabezpečiť tak viacnásobné volanie funkcie viacerými task-ami s RTX51 Full. V menu **BL51 Misc** povoliť overlay funkcie príkazom: ol(?PR?PRINTF?PRINTF ! *), čo umožní viacnásobný prístup z iných taskov k funkcii printf(). Pritom je treba zabezpečiť, aby nedošlo k súčasnému prístupu úloh k funkcii printf !!!. Pre názornosť uvádzam zápis tzv. viacnásobného použitia overlay-u ktorý je použitý vo viacerých task-och programu s použitím operačného systému RTX51 Full: ol(?PR?_PUT_STRING?CLOCK_FULL_2 ! *,?PR?_PUT_INS?CLOCK_FULL_2 ! *,?PR?PRINTF?PRINTF ! *).

[146] Funkcie ktoré môžu byť volané reentrant-ne t.j. súčasne z viacerých zdrojov využívajú definovaný zásobník pre odkladanie environmentu prostredia funkcie. Bez deklarácie reentrant pri volaní funkcie dôjde k nevyhnutnému poškodeniu vlastných premenných funkcie a tým k nesprávnej funkcii.

15.12. Ovládač 232.NET pre multiprocesorovú komunikáciu

C:\Omega\Data\Dropbox\Záloha\C51\Multiprocesorová komunikácia s 8051\Multiproce

```
  1  /***************************************
  2   * 232.NET multiprocesorova komunikácia *
  3   ***************************************/
  4
  5  #include <reg552.h>
  6  #include <stdio.h>
  7  #include <stdlib.h>
  8  #include <string.h>
  9  #include <rtx51.h>
 10  #include <dsw.h>
 11
 12  #define  Token         '^'   //Malo by to byt 0x5E='^'
 13  #define  MyAddrNet     0x00  //Definujem adresu siete 8051
 14  #define  MyAddr51      0xAB  //Definujem moju adresu, dostanem ju z .ini debuggera-a
 15  #define  MyBroadcast   0xFF  //Definujem Broadcast-s address-u
 16  #define  WaitForEver   0xFF  //Cakam donekonecna ...
 17  #define  MBXSend       0x00  //Cislo mailbox-u kde sa odosielaju data
 18  #define  MBXRecv       0x01  //Cislo mailbox-u kde sa prijimaju data
 19
 20  #define  SYSTEM        0x00
 21  #define  SERIAL        0x01
 22  #define  COM           0x02
 23
 24  extern  void Reset51(void);
 25
 26  #define SizeOfPacketData 32
 27
 28  typedef xdata struct                                    //Musi byt deklarovana a pouzivana
     ako je to uvedene v tomto tvare, inac spravne nefunguje !!!
 29  {
 30   unsigned char Preamble,Dst,Src;
 31   unsigned int Size;
 32   unsigned char Data[SizeOfPacketData];
 33   unsigned char Crc;
 34  }
 35  Packet51;
 36
 37  #define NumMemBlocks  5                                 //Definujem [n] pamatovych blokov
     ...
 38  #define SizeMemBlock   (sizeof(Packet51))
 39  #define SizeMemPool    (NumMemBlocks*(2+SizeMemBlock))  //(Header+sizeof(Packet51)*n)
 40
 41  xdata volatile Packet51 Packet;                         //Definoval som prazdny packet,
     naplnim ho v inom task-u, struktura nesmie byt optimalizovana kompilatorom ...
 42  xdata unsigned char MemoryPool[SizeMemPool] _at_ 0x0000;    //MemoryPool je niekedy lepsie
     definovat na pevnej adrese ale uVision robi problemy pri simulacii ...
 43  xdata Packet51 *TransmitMessage,*ReceivedRS232;         //Ukazovatel na prijate znaky z
     mailbox-u a na prijate znaky z RS232 a obsahujem prijaty packet z RS232 ...
 44  xdata unsigned int *MessageToSend,*MessageToRecv;       //Rozne nadefinovane buffer-e
     vysielaneho a prijimaneho packet-u
 45
 46  enum state {OFF=0, ON=1};
 47  enum lists {preamblepkt=0, dstpkt=1, srcpkt=2, sizeofpkthi=3, sizeofpktlo=4, headerpkt=5,
     crcpkt=(sizeof(Packet51)-1), endpkt=sizeof(Packet51), sizedatapkt=SizeOfPacketData,
     mempoolpkt=sizeof(MemoryPool)};
 48
 49  void Com(void) _task_ COM _priority_ 0
 50  {
 51   signed char RtxCompletion;
 52   while(1)
 53    {
 54     switch(RtxCompletion=os_wait(K_TMO+K_SIG+K_INT,WaitForEver,NULL))
 55      {
 56       case TMO_EVENT :
 57       case SIG_EVENT : RtxCompletion=0x00;break;          //Prisli mi data z RS232 ...
 58       case INT_EVENT : break;
 59      }
 60    }
 61  }
 62
 63  unsigned char Check_CRC(unsigned char *Block) reentrant  //Volam rutinu z viacerych
     taskov ...
 64  {
 65   unsigned int i;
 66   unsigned char tmp;
```

Page 1

C:\Omega\Data\Dropbox\Záloha\C51\Multiprocesorová komunikácia s 8051\Multiproce

```
 67    tmp=0x00;
 68    for(i=0x00; i<(sizeof(Packet51)-1); i++) tmp+=Block[i];        //Scitam vsetky polozky packetu
 69    tmp=~tmp; tmp+=1;
 70    return(tmp);
 71   }
 72
 73   bit recv=OFF,send=OFF;
 74
 75   void Serial(void) _task_ SERIAL _priority_ 0
 76   {
 77    signed char RtxCompletion;
 78    idata unsigned int iin,iout;
 79    data unsigned char Znak;
 80    TMOD|=0x20;
 81    PCON|=0x80;
 82    S0CON|=0x50;
 83    TH1=0xFB;                                                     //Nastavim prenosovu
       rychlost 19200bps
 84    TR1=1; //TI=1;                                                //Je tu potrebne len pre
       printf(), putchar() !!!
 85    if((MessageToSend=os_get_block(SizeMemBlock))!=NULL);         //Blok pamati rezervujem
       pre task ktory je pouzity v SERIAL
 86    if((MessageToRecv=os_get_block(SizeMemBlock))!=NULL) ReceivedRS232=(unsigned
       int)MessageToRecv;  //Adresa prijimacieho buffera v pool-e sa inicializuje ...
 87    RtxCompletion=os_attach_interrupt(4);                         //Simultanny prijem a
       vysielanie je riadene prerusenim od serioveho kanala ...
 88    while(1)
 89     {
 90      switch(RtxCompletion=os_wait(K_TMO+K_SIG+K_INT+K_MBX+MBXSend,WaitForEver,(unsigned int
       xdata*)&TransmitMessage)) //Prijem packetu z mailbox-u, prijimam len adresu na strukturu ...
 91        {
 92         case TMO_EVENT : iin=0x0000;                                          //Ak nepride
       prerusenie od prijmu dat nulujem pocitadlo aby som sa nezacyklil ...
 93                          //memset(ReceivedRS232,NULL,sizeof(Packet51));
       //Vynulujem prijimaci buffer ReceiveRS232 ...
 94                          break;
 95         case MSG_EVENT : //os_send_signal(SERIAL); break;
       //Nepotrebujem pocitat CRC prijateho packetu a predsa to posielam v ramci 8051, aby som poslal
       signal sam sebe na spracovanie ... !!!
 96                          if(Check_CRC((unsigned char
       xdata*)TransmitMessage)==(TransmitMessage->Crc)) os_send_signal(SERIAL); break; //Vypocitam
       CRC prijateho packetu a musia byt rovnake aby som poslal signal sam sebe na spracovanie ... !!!
 97                          break;
 98         case SIG_EVENT : send=ON; iout=preamblepkt;
 99                          S0BUF=(TransmitMessage->Preamble);
100                          os_wait(K_TMO,0,NULL);                          //Aby stihli
       vsetky systemy zaregistrovat pritomnost Tokenu '^' na sieti, maju niekolko ms...
101                          break;
102         case INT_EVENT : if(TI&RI) recv=send=ON;                         //Ak som tu,
       tak mam prijem a vysielanie sucasne co potvrdzuje sucasne prijimanie a vysielanie sucasne  ...
103                          if(RI)
104                           {
105                            RI=0; Znak=S0BUF; recv=ON;                    //Pritomnost
       tokenu signalizuje data na zbernici
106                            if(Znak==Token)
107                             {
108                              iin=0x0000;
109                              memset(ReceivedRS232,NULL,sizeof(Packet51));  //Vynulujem
       prijimaci buffer ReceiveRS232, pretoze obsahuje stare hodnoty ...
110                             }
111                            switch(iin++)
112                             {
113                              case  preamblepkt : ReceivedRS232->Preamble=Znak; break;//Ak je to
       token tak sa nastavim na zaciatok dat ...
114                              case  dstpkt      : ReceivedRS232->Dst=Znak; break;
115                              case  srcpkt      : ReceivedRS232->Src=Znak; break;
116                              case  sizeofpkthi : ReceivedRS232->Size=256*Znak; break;
117                              case  sizeofpktlo : ReceivedRS232->Size+=Znak; break;
118                              case  crcpkt      : ReceivedRS232->Crc=Znak;
119                                                  if(Check_CRC((unsigned char
       xdata*)ReceivedRS232)==(ReceivedRS232->Crc))
120                                                   {
121                                                    recv=OFF;
122
       //RtxCompletion=os_free_block(SizeMemBlock,MessageToRecv);  //Zrusim prijimaci buffer ...
```

C:\Omega\Data\Dropbox\Záloha\C51\Multiprocesorová komunikácia s 8051\Multiproce

```
123                                                 switch(ReceivedRS232->Dst)        //Komu je
    packet urceny, vykonam akciu ???
124                                                  {
125                                                   case  MyAddrNet    : break;
126                                                   case  MyAddr51     :
127                                                   case  MyBroadcast :
    RtxCompletion=os_send_signal(COM);                   //V strukture ReceivedRS232 je prijaty
    packet ...
128                                                                        break;
129                                                   default           :
    memset(ReceivedRS232,NULL,sizeof(Packet51)); break; //Nie su to data pre mna, mazem prijimaci
    buffer ...
130                                                  }
131                                                }
132                                               break;
133                             default           : ReceivedRS232->Data[iin-(headerpkt+1)]=Znak;
    break;   //Pokracujem v prijme dat ...
134                            }
135                           }
136                          if(TI)
137                           {
138                            TI=0;
139                            switch(++iout)
140                             {
141                              case preamblepkt
    :                                                          //Tu by som sa nemal nikdy dostat,
    lebo som uz poslal Preamble ...
142                              case dstpkt       : S0BUF=(TransmitMessage->Dst); break;
143                              case srcpkt       : S0BUF=(TransmitMessage->Src); break;
144                              case sizeofpkthi  : S0BUF=(high(TransmitMessage->Size)); break;
145                              case sizeofpktlo  : S0BUF=(low(TransmitMessage->Size)); break;
146                              case crcpkt       : S0BUF=(TransmitMessage->Crc); break;
147                              case endpkt       :
    //RtxCompletion=os_free_block(SizeMemBlock,MessageToSend);  //RtxCompletion==NULL je OK ak
    uvolnim pamat ...
148                                                 send=OFF;
149                                                 break;//Nevymazavat, musi to tu byt !!!
150                              default           : S0BUF=(TransmitMessage->Data[iout-headerpkt]);
    break; //Poslem vsetky data ...
151                             }
152                            }                    //Odosielanie znaku nemusim riesit, ide
    prerusenim os_attach_interrupt(4)  ...
153       }
154     }
155   }
156
157   void System(void) _task_ SYSTEM _priority_ 0
158   {
159    signed char RtxCompletion;
160    RtxCompletion=os_set_slice(15360);
161    RtxCompletion=os_create_pool(SizeMemBlock,MemoryPool,SizeMemPool);   //Vytvorim MemoryPool v
    pamati XRAM pre datove struktury ...
162    RtxCompletion=os_create_task(SERIAL);
163    RtxCompletion=os_create_task(COM);
164    while(1)
165     {
166
167      if(MessageToSend!=NULL)                                           //Vytvorim packet ...
168       {
169        Packet.Preamble=Token;
170        Packet.Dst=(unsigned char)rand();
171        Packet.Src=(unsigned char)rand();
172        Packet.Size=sizeof(Packet.Data);
173        strcpy(Packet.Data,"!!! Diversant Software !!!\r\n");
174        Packet.Crc=Check_CRC((unsigned char xdata*)&Packet);
175        memcpy(MessageToSend,&Packet,sizeof(Packet));                   //Pamat je rezervovana a
    pozem kopirovat blok na odoslanie ...
176   //     while(recv==ON)  RtxCompletion=os_wait(K_TMO,1,NULL);         //Je prijem dat cez
    RS232 tak sa stop-nem na chvilu s vysielanim ...
177        RtxCompletion=os_send_message(MBXSend,(unsigned int xdata*)MessageToSend,WaitForEver);
    //Posielam adresu datovej struktury cez mailbox ...
178       }
179   //    Packet.Crc=Check_CRC((unsigned char xdata*)MessageToSend);      //Vypocitam CRC
    skopirovaneho packetu a musia byt rovnake !!!
180      RtxCompletion=os_wait(K_TMO,250,NULL);
```

Page 3

```
C:\Omega\Data\Dropbox\Záloha\C51\Multiprocesorová komunikácia s 8051\Multiproce
181   //    RtxCompletion=os_free_block(SizeMemBlock,MessageToSend);        //RtxCompletion==NULL je
      OK ak uvolnim pamat v tasku SERIAL !!!
182   //    os_delete_task(SYSTEM);
183     }
184   }
185
186   void main(void)
187   {
188    signed char RtxCompletion;
189    RtxCompletion=os_start_system(SYSTEM);
190   }
```

V praxi sa v súčasnej dobe čoraz častejšie stretávame s potrebou komunikácie viacerých mikroprocesorov navzájom. Situácia sa nám podstatne zjednodušuje integrovaným sériovým rozhraním RS232, CAN, I^2C, Ethernet, prípadne iných komunikačných rozhraní. Sériový typ komunikácie je oproti paralelným formám podstatne jednoduchšie, pretože používa malý počet komunikujúcich vodičov. V praxi stačia na vytvorenie obojsmernej komunikácie 1 až 2 vodiče. Väčšina mikroprocesorov obsahuje integrované sériové komunikačné rozhranie RS232, preto sme sa rozhodli vytvoriť jednoduchý komunikačný model pre RS232. Ako základ sme si vzali technologicky vyspelejší a náročnejší Ethernet 802.3, ktorý je štandardom v oblasti počítačových sietí. Pre nás je podstatný samotný princíp komunikačného modelu. V tomto modeli má každý komunikujúci unikátnu MAC adresu ktorá ho jednoznačne identifikuje na fyzickej úrovni a IP adresu ktorá znova identifikuje zariadenie na logickej úrovni. K dátam ktoré sa odosielajú sa pripojí na začiatok dátovej oblasti blok ktorý obsahuje zdrojovú a cieľovú adresu. Na koniec tohoto bloku sa pripojí tzv. CRC kontrolný súčet, ktorý chráni prepravovaný blok dát pred poškodením pri transporte od zdroja k cieľu. Rovnaký model je použitý aj v tomto prípade. Zdrojová, cieľová adresa, veľkosť dátovej oblasti, CRC sú kvôli jednoduchej implementácii do 8051 len 8 bitové štruktúry. Podľa potreby je ich možné modifikovať. Takýmto spôsobom je možné vytvoriť komunikáciu nielen medzi dvojicou autonómnych systémov a dokážeme jednoznačne identifikovať zdroj a cieľ komunikácie.

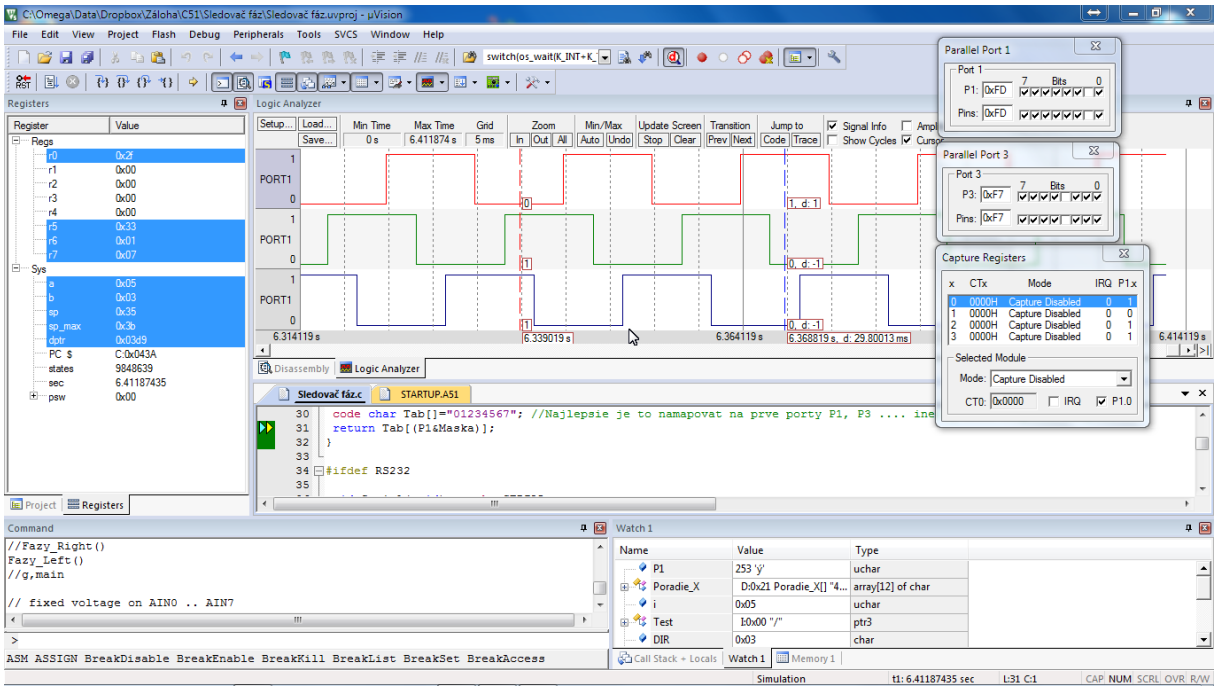
V našom prípade položku "**SFD**", "**EtherType**" nepoužijeme. Využijeme len "**Preamble**" identifikátor začiatku dátového paketu tzv. Tokenom "^" ktorý sme si zadefinovali ako štartovací znak, zdrojovú a cieľovú "**MAC**" adresu, dátovú oblasť "**Payload**" a kontrolný súčet "**FCS**" (Frame Check Sum) t.j. "**CRC**". Túto modifikovanú štruktúru budeme kvôli jednoduchosti programu odosielať pomocou funkcie **putchar()** a prijímať pomocou **getchar()**, prípadne vysielanie a príjem bude pre zvýšenie efektívnosti riadený prerušením. Celý program je napísaný v jazyku C s použitím RTX51 Full. S výhodou využíva oddelené prijímacie a vysielacie registre mikroprocesora **S0CON,** takže pregram je plne duplexný t.j. FDX. Pamäťové štruktúry paketu (**Packet51**) sú staticky umiestnené v pamäťovom priestore **xdata**.

Štruktúra MessageToSend a ReceivedMessage sú umiestnené do pamäti pomocou výkonných funkcií operačného systému RTX51 Full **os_get_block()**, ktorý je už alokovaný pomocou funkcie **os_create_pool()**. Odoslanie packetu sa uskutoční pomocou funkcie RTX51 **os_send_message()**. Možno sa niekomu môže zdať použitie funkcie **os_send_message()** ako zbytočné, ale funkcia obsahuje tzv. Buffer organizovaný ako (**FIFO F**irst-**I**n **F**irst-**O**ut) ktorý môže naraz zachytiť a spracovať až 8 správ v poradí ako prišli. Ako prvá je teda spracovaná tá správa ktorá čaká najdlhšie v rade, ktorá prišla ako prvá.
V štruktúre **ReceivedRS232** t.j. **ReceivedRS232->Data** sú prijaté dáta cez RS232 na vlastnú adresu zariadenia. Štruktúra **TransmitMessage** t.j. **TransmitMessage->Data** obsahuje odosielané dáta na príslušnú adresu v sieti **232.NET**.

15.13. Elektronický sledovač fáz s mikroprocesorom riadený prerušením

Srdcom elektronického sledovača fáz je mikroprocesor 80C552 na ktorý je privedená trojica signálov SL1 až SL3 ktoré predstavujú galvanicky oddelené signály TTL v 3-fázovej sieti. Tieto signály sú privedené na P1.0 až P1.2, ktoré sú pripojené interne na záchytnú jednotku CAPTURE mikroprocesora 80C552. Pripojenie na hardware záchytnej jednotky je vykonané trojicou taskov s názvom Phase1 až Phase3 pomocou funkcie operačného systému RTX51 Full **os_attach_interrupt(n)**. Vyššie uvedené task-y sú nastavené na zmenu úrovne, takže pri zmene úrovne záchytná jednotka automaticky generuje prerušenie ktoré je signalizované bitom CT0 až CT2. Poradie fáz je uložené v tabuľke a v pravidelných intervaloch sú stavy na porte P1 porovnávané s hodnotami v tabuľke. V prípade výpadku fázy je do 20ms automaticky vygenerovaná chyba pomocou internej premennej ERR=0 ktorá je mapovaná do portu P3.2 a zároveň sa nastaví premenná DIR=0. V prípade správnej funkcie fáz je DIR=2 pre pravotočivý sled a DIR=3 pre ľavotočivý sled fáz. Rovnako sa nastavia premenné LEDR a LEDL mapované do portu P3.3 a P3.4, kde je sled signalizovaný nastavením[147] do logickej úrovne 1. Obrázok 147 zobrazuje zjednodušený variant sledovača je možné zostrojiť s jedno-čipovým mikroprocesorom AT89LP4052..

Obrázok 146, Simulácia programu sledovača fáz

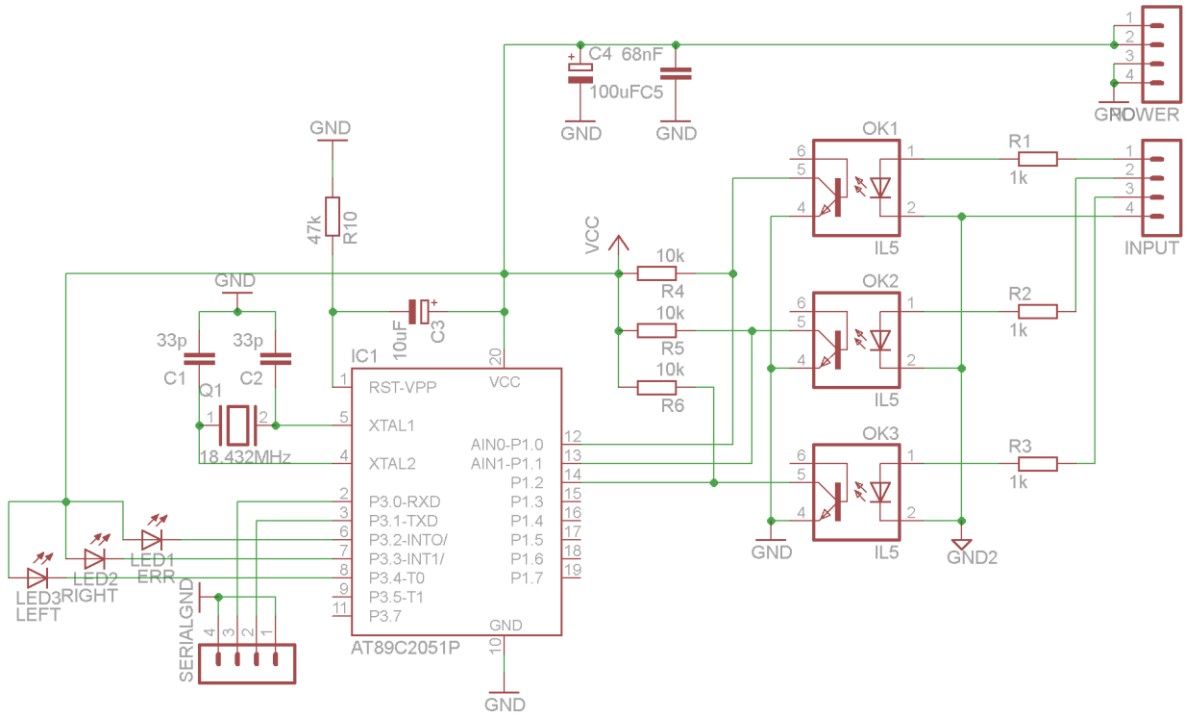


[147] Mikroprocesor 80C552 nemá posilňovač portu a maximálne je možné jeden vývod portu P1, alebo P3 zaťažiť prúdom približne I=1mA. Preto musíme použiť dodatočne na rozsvietenie príslušnej LED diódy TTL hradlo.

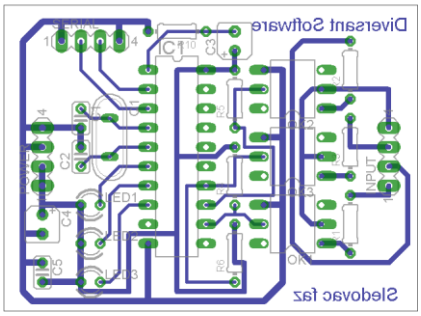
Video 1, Sledovač fáz s 80C552 riadený prerušením - simulácia

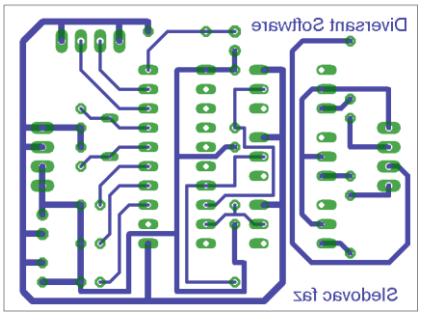
https://www.youtube.com/watch?v=Z6tmBQb8BRU

Obrázok 147, Schéma zapojenia sledovača fáz s AT89LP4052



Obrázok 148, Doska plošného spoja sledovača fáz s AT89LP4052





C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Sledovač fáz.c

```
#include <reg552.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <rtx51.h>

#define  MASK    0x07

#define  SYSTEM  0
#define  ERRORS  1
#define  SERIAL  2
#define  PHASE1  3
#define  PHASE2  4
#define  PHASE3  5

typedef enum {NO_OK=0, YES_OK=1, DIR_R=2, DIR_L=3} Chyby;
typedef enum {OFF=0, ON=1} Stavy;
typedef enum {R=0, L=1} Smery;

char DIR;              //Vysledok zistenia faz ...
sbit ERR  = P3^2;      //ERR=0 je chyba, ERR=1 je OK
sbit LEDR = P3^3;      //LED signalizujuca otacanie sa faz RIGHT
sbit LEDL = P3^4;      //LED signalizujuca otacanie sa faz LEFT

                  //123456   123456
code char *Smer[]={"645132","623154"};
data char Poradie_X[12];
code char *Message[]={"NOT_OK","OK","RIGHT","LEFT"};

unsigned char Port(unsigned char Maska)
{
 code char Tab[]="01234567"; //Najlepsie je to namapovat na prve porty P1, P3 .... ine by boli komplikovanejsie ...
 return Tab[(P1&Maska)];
}

void Serial(void) _task_ SERIAL
{
 PCON|=0x80;
 S0CON=0x50;
 TMOD|=0x20;
 TH1=TL1=0xFB;
 TR1=1;
 TI=1;
 printf("System Ready ...\r\n");
 while(1)
  {
   switch(os_wait(K_TMO+K_SIG,250,NULL))
    {
     case TMO_EVENT : printf("Smer otacania faz = %s\r\n",Message[DIR]);break;
     case SIG_EVENT : printf("Chyba %02bd\r\n",DIR); break;
    }
  }
}

void Errors(void) _task_ ERRORS
{
 ERR=OK;
 while(1)
  {
   switch(os_wait(K_SIG,0xFF,NULL))
    {
     case SIG_EVENT : ERR=NO_OK;
                      DIR=NO_OK;
                      os_send_signal(SERIAL);
                      break;
     case TMO_EVENT : ERR=YES_OK; break;                    //Nulujem WDT, inac by to skoncilo chybou
    }
  }
}

unsigned char i = 0;

void Reload(void)
```

Page 1

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Sledovač fáz.c

```
 74  {
 75   if(i>=sizeof(Poradie_X))
 76    {
 77     i=0x00;
 78     os_send_signal(SYSTEM);
 79    }
 80  }
 81
 82  void Phase1(void) _task_ PHASE1 _priority_ 0
 83  {
 84   TM2CON=0x01;
 85   CTCON|=0x02+0x01;
 86   os_attach_interrupt(6);
 87   while(1)
 88    {
 89     switch(os_wait(K_INT+K_TMO,30,NULL))                 //Cakam na fazu A a zachytavam obsahy
    zachytnych registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
 90      {
 91       case INT_EVENT : CTI0=0;
 92                        Reload();
 93                        Poradie_X[i++]=Port(MASK); break;    //Prisla hrana fazy, tak vykonam
    toto ...
 94       case TMO_EVENT : os_send_signal(ERRORS); break;    //Ak sa tu dostanem neprisla hrana
    fazy do TIME tak vyvolam chybu ...
 95      }
 96    }
 97  }
 98
 99  void Phase2(void) _task_ PHASE2 _priority_ 0
100  {
101   TM2CON=0x01;
102   CTCON|=0x08+0x04;
103   os_attach_interrupt(7);
104   while(1)
105    {
106     switch(os_wait(K_INT+K_TMO,30,NULL))                 //Cakam na fazu B a zachytavam obsahy
    zachytnych registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
107      {
108       case INT_EVENT : CTI1=0;
109                        Reload();
110                        Poradie_X[i++]=Port(MASK); break;    //Prisla hrana fazy, tak vykonam
    toto ...
111       case TMO_EVENT : os_send_signal(ERRORS); break;    //Ak sa tu dostanem neprisla hrana
    fazy do TIME tak vyvolam chybu ...
112      }
113    }
114  }
115
116  void Phase3(void) _task_ PHASE3 _priority_ 0
117  {
118   TM2CON=0x01;
119   CTCON|=0x20+0x10;
120   os_attach_interrupt(8);
121   while(1)
122    {
123     switch(os_wait(K_INT+K_TMO,30,NULL))                 //Cakam na fazu C a zachytavam obsahy
    zachytnych registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
124      {
125       case INT_EVENT : CTI2=0;
126                        Reload();
127                        Poradie_X[i++]=Port(MASK); break;    //Prisla hrana fazy, tak vykonam
    toto ...
128       case TMO_EVENT : os_send_signal(ERRORS); break;    //Ak sa tu dostanem neprisla hrana
    fazy do TIME tak vyvolam chybu ...
129      }
130    }
131  }
132
133  void System(void) _task_ SYSTEM _priority_ 0
134  {
135   char *Test;
136   os_set_slice(1536);
137   os_wait(K_TMO,20,NULL);
138   #ifdef RS232
139   os_create_task(SERIAL);
```

Page 2

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Sledovač fáz.c

```
140    #endif
141    os_create_task(PHASE1);
142    os_create_task(PHASE2);
143    os_create_task(PHASE3);
144    os_create_task(ERRORS);                                  //Spustim Watchdog na pritomnost faz ....
145    while(1)
146     {
147      os_wait(K_SIG,0xFF,NULL);
148      if((Test=strstr(Poradie_X,Smer[R]))!=NULL) DIR=DIR_L; //Smer otacania sa faz je vlavo
149      if((Test=strstr(Poradie_X,Smer[L]))!=NULL) DIR=DIR_R; //Smer otacania sa faz je vpravo
150      switch(DIR)                                             //Nastavim OK, alebo ERROR
151       {
152        case DIR_R : LEDR=ON; LEDL=OFF; ERR=OK; break;
153        case DIR_L : LEDL=ON; LEDR=OFF; ERR=OK; break;
154        default    : LEDL=LEDR=OFF; ERR=NO_OK; break;
155       }
156     }
157   }
158
159   void main(void)
160   {
161    os_start_system(SYSTEM);
162   }
```

Page 3

15.14. Elektronický sledovač fáz s mikroprocesorom a priamym meraním frekvencie

Konštrukcia vyššie uvedeného elektronického zariadenia je rovnaká ako v kapitole 15.13, pričom sa upravil len samotný podprogram[148] obsluhy prerušenia záchytnej jednotky CAPTURE. V premennej **Delta** je uložený počet impulzov medzi dvomi prerušeniami jednej záchytnej jednotky a vyjadruje jednu polovicu periódy sieťového napätia. Záchytná jednotka je v systéme 80C552 napájaná frekvenciou $f_{CAPTURE} = \frac{1}{12} \cdot f_{XTAL}$ t.j. $\frac{1}{12}$ frekvencie oscilátora.

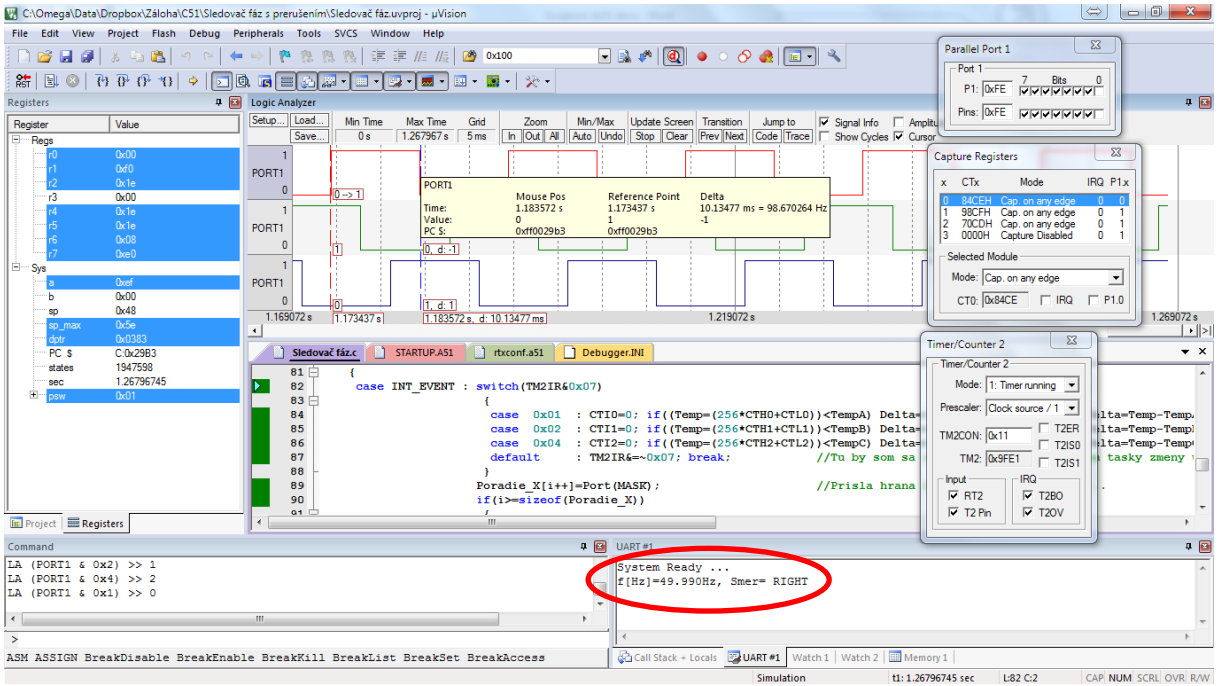
Výslednú veľkosť frekvencie napätia v sieti je možné vypočítať ako:

$$f = \frac{f_{XTAL}}{24 \cdot (256 \cdot CTH0 + CTL0)} \qquad \text{[Hz; Hz, 1, 1]} \qquad (6.)$$

fXTAL je frekvencia kryštálu použitého v konštrukcii mikroprocesorovej jednotky

CTH0 a ***CTL0*** sú registre mikroprocesora záchytnej jednotky CAPTURE

Obrázok 149, Priame meranie frekvencie mikroprocesorom 80C552 riadený prerušením



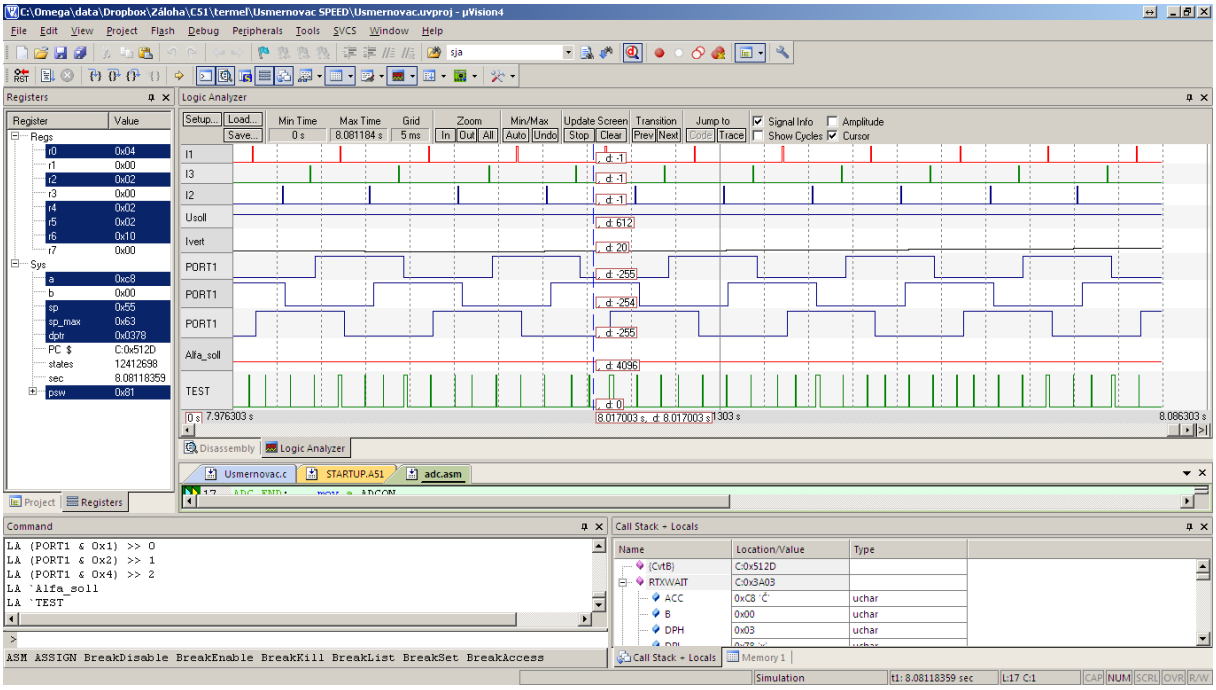
[148] Odporúčanie pre prax: Task **PHASE** by mal mať najvyššiu úroveň priority (1 až 3), aby sa prednostne a načas zabezpečila obsluha prerušení od záchytnej jednotky. V opačnom prípade by dochádzalo k nesprávnym meraniam a tým skresleniu výsledných hodnôt.

15.15. Tyristorový regulátor – fázové riadenie usmerňovača

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
233   #pragma  REGISTERBANK  (3)
234
235   sbit  I1  = P4^0;
236   sbit  I3  = P4^1;
237   sbit  I2  = P4^2;
238
239   void  Impuls (void) _task_  IMPULS  _priority_  3
240   {
241    #define  Length  75                                          //60*1.366=85us pre dlzku
      impulzu !!! ak nemam funkciu Delay reentrantnu
242    #define  Dlzka  (Length/1.366)                              //Aby som nemusel stale
      prepocitavat na skutocnu dlzku impulzu ...
243    os_attach_interrupt (11);
244    os_attach_interrupt (12);
245    os_attach_interrupt (13);
246    while (1)
247     {
248      Watchdog;
249      switch (os_wait (K_TMO+K_INT,K*100,NULL))
250       {
251        case  INT_EVENT  : switch (TM2IR&0x70)                   //RTX51 HANDLER prepne do cca.
      110us na tento task zovsadial !!!
252                           {
253                        case   0x10   : CMI0=0; I1=1; Delay (Dlzka); I1=0; break;  //Generujem 100us
      impulz I1 pre SL1 pre podmienku CMH0+CML0=TMH2+TML2
254                        case   0x20   : CMI1=0; I3=1; Delay (Dlzka); I3=0; break;  //Generujem 100us
      impulz I3 pre SL3 pre podmienku CMH1+CML1=TMH2+TML2
255                        case   0x40   : CMI2=0; I2=1; Delay (Dlzka); I2=0; break;  //Generujem 100us
      impulz I2 pre SL2 pre podmienku CMH2+CML2=TMH2+TML2
256                        default    : TM2IR&=~0x70; break;                  //Ak som tu tak z
      nevysvetlitelnych dovodov prisli niektore, alebo vsetky dve-tri prerusenia od komparatora !!!
      Stane sa to pri poruchach na silovych fazach a vtedy nesmie generovat impulz !!! ...
257                           }
258                  break;
259     case  TMO_EVENT  : os_send_message (ERROR_MAIL,IMPULS,TIME); break;
260     }
261     }
262   }
```

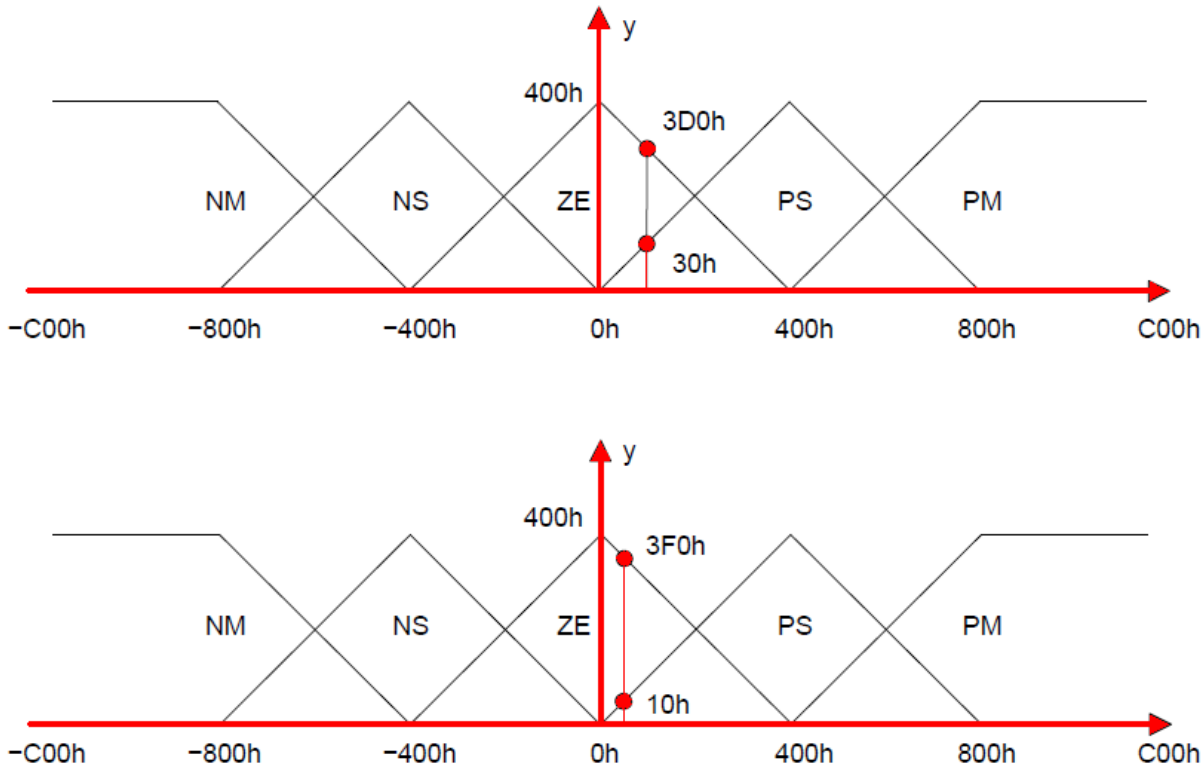
Obrázok 150, Generovanie zapínacích impulzov - simulácia



15.16. Fuzzy regulátor v C

Teória Fuzzy regulácie je známa približne od roku 1980. Najväčší pokrok v tejto oblasti dosiahli Japonci, ktorí sa snažili v čo najväčšej miere implementovať jej princípy do praxe. V princípe ide o regulovanie procesu na základe slovného opisu problému. Najlepším príkladom je napríklad šoférovanie v autoškole. Inštruktor v autoškole neučí žiaka tak, že mu prikáže pri mierne pravotočivej zákrute otočiť volant riadenia o 35,6° vpravo, ale prikáže mu pootočiť volant mierne vpravo. Pri prudkej zákrute mu prikáže otočiť volant na pravý doraz doprava. To sú zjednodušené princípy Fuzzy regulácie. Fuzzy algoritmy nie sú extra matematicky zložité a pre navrhovanie úplne stačia aj stredoškolské znalosti matematiky. Tieto algoritmy bývajú veľmi robustné a odolné proti rôznym chybám hodnôt a šumom. Ich veľkou nevýhodou je, že vďaka tomu môžu byť niekedy pomerne nepresné. Ak sa chceme tomu vyhnúť tak je ich potrebné spojiť s lineárnym regulátorom. Bližšie informácie môžete nájsť v (Novák, 2000).

Obrázok 151, Zaradenie hodnoty k príslušnosti prvku danej množiny



Nastavenie Fuzzy regulátora je v praxi komplikovanejšie ako klasického PID regulátoru, pretože Fuzzy regulátor je svojou charakteristickou štruktúrou značne nelineárny. Je to spôsobené tým, že pracujeme s viacerými množinami príslušnosti do ktorých zaraďujeme spracovávané hodnoty. Existuje množstvo metód na nastavenie týchto regulátorov.

Fuzzy regulátor obsahuje:

- Fuzzyfikátor
- Inferečný mechanizmus[149]
- Defuzzyfikátor

[149] Inferenčný mechanizmus obsahuje aj tzv. bázu pravidiel IF THEN ELSE, ktorý jednoznačne zaradí príslušné hodnoty do stupňa príslušnosti k danej množne.

Obrázok 152, Fuzzy regulátor v jazyku C s RTX51

C:\Omega\data\Dropbox\Záloha\C51\fuzzy\Fuzzy.c

```
1  //*****************************************************************************************************
   *********
2  // Header: FUZZY regulator s RTX51 pre taviace pece
3  // File Name: FUZZY.C
4  // Author: Ing. Eduard Jadron, Jilemnickeho 3999/37, 03601 Martin, Slovak Republic
5  // Date: 29.12.2005
6  //*****************************************************************************************************
   *********
7
8  #include <reg552.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <rtx51.h>
12 #include <dsw.h>
13
14 #define REGULATOR 0
15 #define ACTION    1
16
17 int Error,dError,LastError;
18
19 #define SIZE  5   //Velkost inferencneho bloku !!!
20
21 int X1[SIZE],X2[SIZE],Y[SIZE],Output;
22
23 int Soll, Vert;
24
25 //typedef enum {NM,NS,ZE,PS,PM} State;
26
27 #define NM  0 // Negative medium
28 #define NS  1 // Negative small
29 #define ZE  2 // Zero equal
30 #define PS  3 // Positive small
31 #define PM  4 // Positive medium
32
33 const unsigned char InferenceTable[SIZE][SIZE] =
34 {    /*NM,NS,ZE,PS,PM*/ //dError X2[]
35 /*NM*/ {PM,PM,PM,PS,ZE}, //NM   E
36 /*NS*/ {PM,PM,PS,ZE,NS}, //NS   r
37 /*ZE*/ {PM,PS,ZE,NS,NM}, //ZE   r
38 /*PS*/ {PS,ZE,NS,NM,NM}, //PS   o
39 /*PM*/ {ZE,NS,NM,NM,NM} //PM   r
40 };                //    X1[]
41
42 #define K 0x18
43
44 const int OutputFunc[SIZE] = {-2*K,-K,+0x00,+K,+2*K};
45
46 unsigned int ADC(unsigned char Channel)    //Rutina trva 55us pri 18.432MHz
47 {
48  #define ADCS  0x08
49  #define ADCI  0x10
50  #define ADEX  0x20
51  unsigned int x;
52  ADCON=Channel;                              //Vyber kanalu pre prevod 0..7
53  ADCON|=ADEX+ADCS;                           //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
54  while((ADCON&ADCI)==0x00);                  //Cakam na ukoncenie prevodu
55  ADCON^=ADCI;                                //Nulujem priznak AD prevodnika
56  x=ADCH*0x04;                                //Normovanie na (int)
57  switch(ADCON&(0x80+0x40))
58   {
59    case 0x40   : x+=0x01; break;
60    case 0x80   : x+=0x02; break;
61    case 0xC0   : x+=0x03; break;
62   }
63  return(x);
64 }
65
66 typedef enum {SCALE=0x0400,SK0=+0x0000,SK1=+0x0400,SK2=+0x0800} Osi;
67
68 void Fuzzification(int Value, int *Data) //reentrant
69 {
70  unsigned char i;
71  for(i=0x00; i<SIZE; i++) Data[i]=0x00;
72  if(Value<-SK2) Data[NM]=SCALE;
```

Page1

C:\Omega\data\Dropbox\Záloha\C51\fuzzy\Fuzzy.c

```
73     else if (Value <- SK1 )
74     {
75      Data [NM ]= SCALE -( Value +SK2 );
76      Data [NS ]= Value +SK2 ;
77     }
78     else if (Value <+ SK0 )
79     {
80      Data [NS ]= SCALE -( Value +SK1 );
81      Data [ZE ]= Value +SK1 ;
82     }
83     else if (Value <+ SK1 )
84     {
85      Data [ZE ]= SCALE -Value ;
86      Data [PS ]= Value ;
87     }
88     else if (Value <+ SK2 )
89     {
90      Data [PS ]= SCALE -( Value -SK1 );
91      Data [PM ]= Value -SK1 ;
92     }
93     else  Data [PM ]= SCALE ;
94    }
95
96    void  FuzzyInference (int  *X1 , int  *X2 , int  *Y) //reentrant
97    {
98     int  min [SIZE ];
99     int  max , Temp ;               //Tu musi byt v pripade RTX51 a LARGE modelu "data int max" s
      optimalizaciou Loop rotation - 6
100    unsigned  char  i,j,maxpos ;
101    for (i=0x00 ; i<SIZE ; i++)  Y[i]= 0x00 ;  //Vymazem vystupny vektor Y[]
102    for (i=0x00 ; i<SIZE ; i++)
103     {
104      for (j=0x00 ; j<SIZE ; j++)
105       if (X1 [i]< X2 [j])  min [j]= X1 [i]; else  min [j]= X2 [j];
106       max =min [0];
107       maxpos =0x00 ;
108      for (j=0x01 ; j<SIZE ; j++)
109       if (max <min [j])
110        {
111       max =min [j];
112       maxpos =j;
113      }
114      Temp =Y[InferenceTable [i][ maxpos ]];
115      if (Temp <max )    Y[InferenceTable [i][ maxpos ]]+= max ;
116      if (Temp >SCALE )  Y[InferenceTable [i][ maxpos ]]= SCALE ;
117     }
118   }
119
120   int  Defuzzification (int  *Y) //reentrant
121   {
122    unsigned  char  i;
123    int  ReturnVal =0, SumY =0;
124    for (i=0x00 ; i<SIZE ; i++)
125     {
126      SumY += Y[i];
127      ReturnVal += Y[i]* OutputFunc [i];
128     }
129    return ((long ) ReturnVal << 2)/ SumY ; //Skalujem cislom 4
130   }
131
132   void  Regulator (void ) _task_  REGULATOR  _priority_  0
133   {
134    os_set_slice (xtal (18.432E6 )/ 10 );
135    PWMP =0x20 ;
136    Soll =Vert =0x0000 ;
137    while (1)
138     {
139   //   Vert=ADC(0);
140      if (++ Vert >= 1024 )  Vert =0;
141      Soll =ADC (2);
142      LastError =Error ;
143      Error =( Vert -Soll )<< 1;
144      dError =( Error -LastError );
145      Fuzzification (Error , X1 );
```

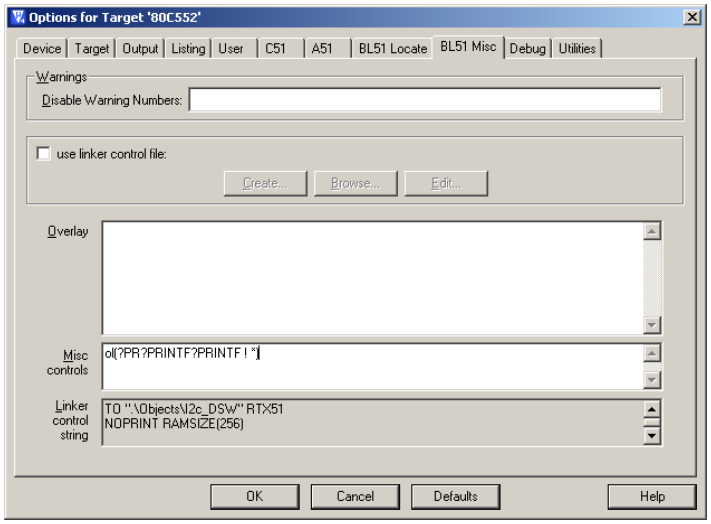
C:\Omega\data\Dropbox\Záloha\C51\fuzzy\Fuzzy.c

```
146       Fuzzification (dError ,X2 );
147       FuzzyInference (X1 ,X2 ,Y);
148       Output =Defuzzification (Y);
149    //    PWM0=0x80-Output;
150       PWM0 =PWM1 =/*0x80-*/ Output ;
151       os_wait (K_TMO ,2,NULL );
152      }
153    }
154
155    void  main (void )
156    {
157     os_start_system (REGULATOR );
158    }
```

15.17. Použitie „overlay“ v RTX51 Full

RTX51 a každý operačný systém je nesmierne mocným nástrojom ktorý umožňuje programátorovi vytvoriť vysoko efektívny programový kód. Ak si uvedomíme, že sa jedná o prostredie, kde je možné spustiť viac procesov súčasne, čoskoro nastane problém so súčasným prístupom z viacerých úloh t.j. taskov k jednej funkcii, alebo procedúre. Najlepším príkladom je práve funkcia **printf()**. Neraz sa vyskytne v praxi požiadavka vypísať práve pomocou funkcie **printf()** do prenosového kanálu z dvoch úloh súčasne. Táto funkcia nie je reentrantná a tak jej spustenie z dvoch úloh nie je možné, pretože sa prepíšu jej interné premenné, čo spôsobí jej nekorektnú funkciu. Obrázok 153 ukazuje ako je možné využívať rovnakú funkciu z viacerých úloh v spolupráci so semaformi.

Obrázok 153, Použitie a zápis overlay-u v µVision



15.18. Použitie „overlay“ direktívy pri zdieľaní funkcií

Ak niektorá používaná funkcia v prostredí RTX51 nie je reentrantná, tak jej súčasné spustenie z dvoch rôznych úloh spôsobí nesprávnu funkciu. Obrázok 154 názorne ukazuje ako je možné využívať rovnakú funkciu z viacerých úloh v spolupráci so semaformi a ukazuje jednoduchý príklad ošetrenia zápisu súčasného prístupu[150] z viacerých taskov na jednu funkciu. Pritom je nutné zabezpečiť, aby nedošlo k prepnutiu z jednej úlohy vykonávajúcou rovnakú funkciu druhou úlohou vykonávajúcou tú istú funkciu. Tento problém je možné riešiť pomocou semaforov. Práve binárny semafor zabezpečí, aby súčasne do rovnakej funkcie ktorá nie je

[150] V prípade reentrant-ných funkcií, alebo procedúr ktoré majú k dispozícii alokovanú dostatočne veľkú vlastnú pamäť pre „vnorenie“ by nemalo dôjsť k nesprávnej funkcii pri súčasnom využívaní jednej funkcie viacerými úlohami.

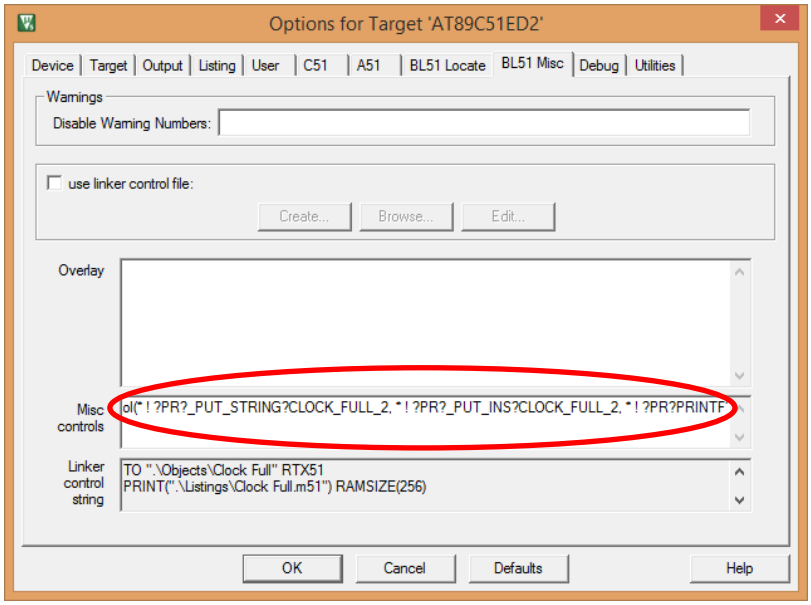
reentrantná nepokúsili sa pristúpiť viaceré úlohy a nespôsobili tak prípadné prepísanie environmentu samotnej funkcie. V prípade že sa tak stane, „prepnutá“ funkcia bude obsahovať neplatné hodnoty. Linker µVision pri súčasnom prístupe viacerých taskov do jednej funkcie vypíše varovné hlásenie, ktoré musíme ošetriť použitím štandardných direktív.

Príklad zápisu overlay-u pri viacnásobnom použití jednej funkcie rôznymi taskmi v RTX51:

a. ol(?PR?_PUT_STRING?CLOCK_FULL_2 ! *,?PR?_PUT_INS?CLOCK_FULL_2 ! *,?PR?PRINTF?PRINTF ! *)
b. ol(* ! ?PR?_PUT_STRING?CLOCK_FULL_2, * ! ?PR?_PUT_INS?CLOCK_FULL_2, * ! ?PR?PRINTF?PRINTF)[151]

Z vyššie uvedeného je vidieť, že v zdrojovom súbore s názvom **clock_full_2.c** je opakovane volaná funkcia **put_string(), put_ins() a printf()** z viacerých taskov.

Obrázok 154, Ošetrenie viacnásobne volaných funkcií v µVision5



[151] Rovnaký zápis, len iným spôsobom overlay direktívy uvedený BL51Misc v Options for Target a položke Misc Controls.

Obrázok 155, Chybové hlásenia pri súčasnom prístupe viacerých úloh

```
Rebuild target 'AT89C51ED2'
compiling Clock Full 2.c...
assembling EE_ASM.ASM...
assembling RTXCONF.A51...
assembling STARTUP.A51...
assembling KALENDAR.ASM...
linking...
*** WARNING L15: MULTIPLE CALL TO SEGMENT
    SEGMENT: ?PR?PRINTF?PRINTF
    CALLER1: ?PR?DISPLAY?CLOCK_FULL_2
    CALLER2: ?PR?KEYBOARD?CLOCK_FULL_2
*** WARNING L15: MULTIPLE CALL TO SEGMENT
    SEGMENT: ?PR?_PUT_STRING?CLOCK_FULL_2
    CALLER1: ?PR?DISPLAY?CLOCK_FULL_2
    CALLER2: ?PR?KEYBOARD?CLOCK_FULL_2
*** ERROR L107: ADDRESS SPACE OVERFLOW
    SPACE:   XDATA
    SEGMENT: ?RTX?TASKCONTEXT?10
    LENGTH:  0032H
Program Size: data=96.5 xdata=1789 code=18907
Target not created.
```

Obrázok 156, Výpis linkera L51 po kompilácii programu s použitím overlay direktív

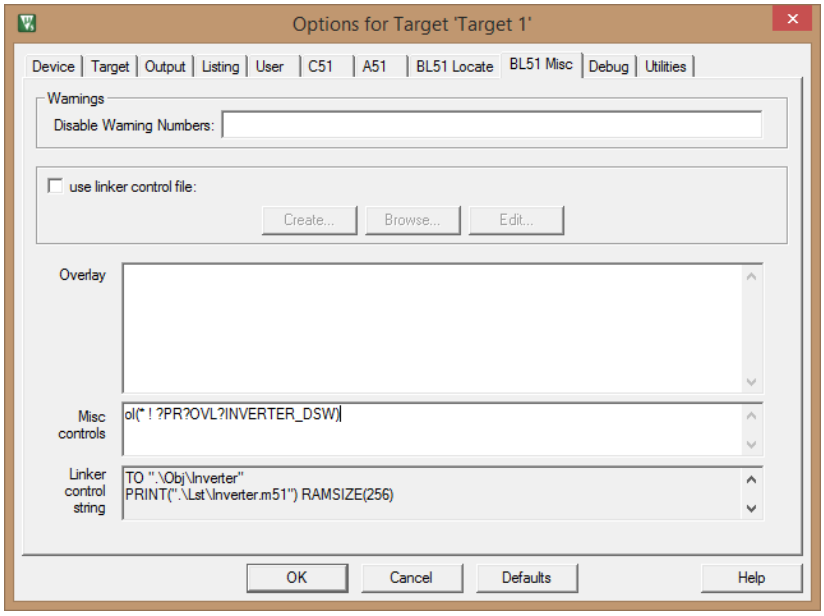
```
Rebuild target 'AT89C51ED2'
compiling Clock Full 2.c...
assembling EE_ASM.ASM...
assembling RTXCONF.A51...
assembling STARTUP.A51...
assembling KALENDAR.ASM...
linking...
Program Size: data=90.4 xdata=1764 code=18907
creating hex file from ".\Objects\Clock Full"...
".\Objects\Clock Full" - 0 Error(s), 0 Warning(s).
```

V praxi sa môže stať, že sa v programe nachádza funkcia, ktorá je deklarovaná, ale nie je volaná. Programátor môže ošetriť tento stav pomocou „overlay“ direktívy, čím sa odstráni zbytočné chybové hlásenie s kódom L16[152].

Funkcia, ktorá nie je volaná z hlavného programu spôsobí generovanie chybového hlásenia, ktoré je možné potlačiť použitím direktívy overlay-a ol(* ! ?PR?_MENO_FUNKCIE), kde ?PR? je určenie segmentu v ktorom sa nachádza funkcia za ktorú nasleduje meno funkcie.

[152] *** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS SEGMENT: ?PR?MYFUNC?X. Uvedené chybové hlásenie je signálom, že uvedená funkcia, alebo procedúra ktorá je deklarovaná v programe nie je vôbec volaná. Toto chybové hlásenie je možné jednoducho programátorsky odstrániť vložením „overlay“ direktívy prostredia µVision5 ol(* ! ?PR?NAME_FUNCTION?NAME_SOURCE_FILE) do BL51Misc v okne Options for Target. Chybové hlásenie sa už nebude pri preklade programu vypisovať a linker potvrdzuje, že preklad programu bol úspešný.

Obrázok 157, Príklad zápisu ošetrenia nevolanej funkcie



Príklad: v súbore **inverter_dsw.c** je deklarovaná funkcia **(void)Ovl(void)**, ktorá v celom programe neobsahuje funkčné volanie. Na odstránenie chybového hlásenia o nevolanom segmente použijeme direktívu **ol(* ! ?PR?OVL?INVERTER_DSW)**[153], ktorá automaticky zabezpečí, že linker nebude generovať chybové hlásenie.

Obrázok 158, Výpis programu s chybovým hlásením o nevolanom segmente

```
Rebuild target 'Target 1'
assembling STARTUP.A51...
compiling inverter_dsw.c...
assembling inverter_asm.asm...
linking...
*** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS
    SEGMENT: ?PR?OVL?INVERTER_DSW
Program Size: data=12.0 xdata=0 code=363
creating hex file from ".\Obj\Inverter"...
".\Obj\Inverter" - 0 Error(s), 1 Warning(s).
```

Obrázok 159, Výpis programu bez chybového hlásenia o nevolanom segmente

```
Rebuild target 'Target 1'
assembling STARTUP.A51...
compiling inverter_dsw.c...
assembling inverter_asm.asm...
linking...
Program Size: data=12.0 xdata=0 code=363
creating hex file from ".\Obj\Inverter"...
".\Obj\Inverter" - 0 Error(s), 0 Warning(s).
```

[153] Je možné použiť aj direktívu s veľkými písmenami **OL(* ! ?PR?OVL?INVERTER_DSW)**

15.19. Ovládač RS232 s využitím I/O buffer-a riadeného prerušením

c:\Keil\C51\Rtx51\Examples\Trafic\SERIAL.C

```
/*------------------------------------------------------------------------------*/
/*       putbuf:  write a character to SBUF or transmission buffer              */
/*------------------------------------------------------------------------------*/
putbuf (char c) {
  if (!sendfull) {                    /* transmit only if buffer not full   */
    if (!sendactive && !sendstop) {   /* if transmitter not active:         */
      sendactive = 1;                 /* transfer the first character direct*/
      SBUF = c;                       /* to SBUF to start transmission      */
    }
    else {                            /* otherwize:                         */
      outbuf[oend++ & (OLEN-1)] = c;  /* transfer char to transmission buffr*/
      if (((oend ^ ostart) & (OLEN-1)) == 0)  sendfull = 1;
    }                                 /* set flag if buffer is full         */
  }
}
/*------------------------------------------------------------------------------*/
/*       putchar:  interrupt controlled putchar function                        */
/*------------------------------------------------------------------------------*/
char putchar (char c) {
  if (c == '\n') {                    /* expand new line character:         */
    while (sendfull) {                /* wait for transmission buffer empty */
      otask = os_running_task_id ();  /* set output task number             */
      os_wait (K_SIG, 0, 0);          /* RTX-51 call: wait for signal       */
      otask = 0xff;                   /* clear output task number           */
    }
    putbuf (0x0D);                    /* send CR before LF for <new line>   */
  }
  while (sendfull) {                  /* wait for transmission buffer empty */
    otask = os_running_task_id ();    /* set output task number             */
    os_wait (K_SIG, 0, 0);            /* RTX-51 call: wait for signal       */
    otask = 0xff;                     /* clear output task number           */
  }
  putbuf (c);                         /* send character                     */
  return (c);                         /* return character: ANSI requirement */
}
/*------------------------------------------------------------------------------*/
/*       _getkey:  interrupt controlled _getkey                                 */
/*------------------------------------------------------------------------------*/
char _getkey (void) {
  while (iend == istart) {
    itask = os_running_task_id ();    /* set input task number              */
    os_wait (K_SIG, 0, 0);            /* RTX-51 call: wait for signal       */
    itask = 0xff;                     /* clear input task number            */
  }
  return (inbuf[istart++ & (ILEN-1)]);
}
/*------------------------------------------------------------------------------*/
/*       serial:  serial receiver / transmitter interrupt                       */
/*------------------------------------------------------------------------------*/
serial () interrupt 4 using 1 {       /* use registerbank 1 for interrupt   */
  unsigned char c;
  bit   start_trans = 0;
  if (RI) {                           /* if receiver interrupt              */
    c = SBUF;                         /* read character                     */
    RI = 0;                           /* clear interrupt request flag       */
    switch (c) {                      /* process character                  */
      case CTRL_S:
        sendstop = 1;                 /* if Control+S stop transmission     */
        break;

      case CTRL_Q:
        start_trans = sendstop;       /* if Control+Q start transmission    */
        sendstop = 0;
        break;

      default:                        /* read all other characters into inbuf*/
        if (istart + ILEN != iend) {
          inbuf[iend++ & (ILEN-1)] = c;
        }
                                      /* if task waiting: signal ready      */
        if (itask != 0xFF) isr_send_signal (itask);
        break;
    }
  }
  if (TI || start_trans) {            /* if transmitter interrupt           */
    TI = 0;                           /* clear interrupt request flag       */
    if (ostart != oend) {             /* if characters in buffer and        */
      if (!sendstop) {                /* if not Control+S received          */
        SBUF = outbuf[ostart++ & (OLEN-1)];      /* transmit character      */
        sendfull = 0;                 /* clear 'sendfull' flag              */
                                      /* if task waiting: signal ready      */
        if (otask != 0xFF) isr_send_signal (otask);
      }
    }
    else sendactive = 0;              /* if all transmitted clear 'sendactive*/
  }
}
```

V novom ovládači pre sériové rozhranie RS232 boli použité nové užívateľom napísané funkcie ***putchar()*** a ***_get_key()*** ktoré nahrádzajú implicitné funkcie z knižnice.

15.20. Univerzálny ovládač komunikačného I²C rozhrania

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\I2C_Driver.C

```
#include  <reg552.h>

#define  Dlzka  6

extern  void  Delay (unsigned  char );

void  I2C_Start (void )
{
 SCL =SDA =1;      //AGSI drivers vyzaduje !!!, Zacalo davat dobre vysledky s AGSI drivers
 Delay (Dlzka );   //AGSI drivers vyzaduje !!!, Zacalo davat dobre vysledky s AGSI drivers
 SDA =0;
 Delay (Dlzka );
 SCL =0;
 Delay (Dlzka );
}

void  I2C_Bit (unsigned  char  Data )
{
 SDA =Data ; Delay (Dlzka ); SCL =1; Delay (Dlzka ); SCL =0; Delay (Dlzka );
}

unsigned  char  I2C_ReadBit (void )
{
 unsigned  char  Result ;
 #ifndef  SIMUL
  SDA =1;          //Zacalo davat dobre vysledky ....
 #endif
 Delay (Dlzka );
 SCL =1;
 Delay (Dlzka );
 Result =SDA ;
 Delay (Dlzka );
 SCL =0;
 Delay (Dlzka );
 return (Result );
}

void  I2C_Stop (void )
{
 SDA =0; Delay (Dlzka ); SCL =1; Delay (Dlzka ); SDA =1; Delay (Dlzka );
}

unsigned  char  I2C_Read (unsigned  char  doACK )
{
 unsigned  char  i, Result =0;
 for (i=0x00 ; i<0x08 ; i++)
  {
   Result <<= 1;
   Result |=( I2C_ReadBit ()& 0x01 );
  }
 if (doACK == 0)  I2C_Bit (1);   /* No ACK */  else  I2C_Bit (0);   /* Do the ACK */
 return (Result );
}

unsigned  char  I2C_SendByte (unsigned  char  Data )
{
 unsigned  char  i;
 for (i=0x00 ; i<0x08 ; i++)
  {
   if (Data &0x80 )  I2C_Bit (1);  else  I2C_Bit (0);
   Data <<= 1;
  }
 return (I2C_ReadBit ());
}

unsigned  char  I2C_SendAddr (unsigned  char  address )
{
 I2C_Start ();
 return (I2C_SendByte (address ));
}
```

Používané obvody[154] pre meranie teploty je možné nastaviť na 9 až 13[155] bitový prevod teploty.

[154] DS1631 je obvodovo a protokolovo čiastočne rovnaký ako DS1624 a DS1621. Obvod DS1631 má príkaz spustenia prevodu teploty 0x51 a u DS1621 – DS1624 je 0xEE.

[155] Len pri obvode DS1624. Tento obvod je možné simulovať v prostredí µVision4 s AGSI ako DS1621.

15.21. Prístup do pamäti E²PROM 256Byte v DS1624 cez I²C rozhranie.

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

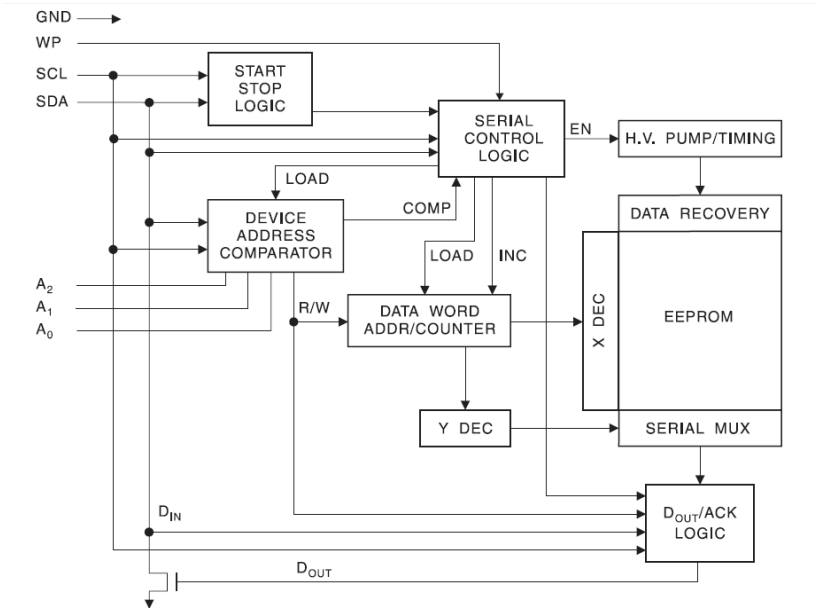
```
172   void Write_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char Data)
      //Funguje zapis a aj citanie do DS1624
173   {
174    I2C_SendAddr (Adresa+WRITE);
175    I2C_SendByte (ACCESS_MEMORY);
176    I2C_SendByte (EEprom_Addr);
177    I2C_SendByte (Data);
178    I2C_Stop ();
179    os_wait (K_TMO,EEPROM_WAIT,NULL);    //Cakam 10ms cez RTX51 Full
180   }
181
182   unsigned char Read_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr)  //Funguje
      zapis a aj citanie do DS1624
183   {
184    register unsigned char LSB;
185    I2C_SendAddr (Adresa+WRITE);
186    I2C_SendByte (ACCESS_MEMORY);
187    I2C_SendByte (EEprom_Addr);
188    I2C_SendAddr (Adresa+READ);
189    LSB=I2C_Read (0);
190    I2C_Stop ();
191    return (LSB);
192   }
193
194   void Write_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
      Dlzka, unsigned char *Buffer)
195   {
196    #ifndef RTX51
197     register unsigned char Counter=60;
198    #endif
199    register unsigned char i;
200    for (i=0x00; i<Dlzka; i++)
201     {
202      Write_Eprom_DS1624 (Adresa,EEprom_Addr+i,Buffer[i]);  //Funguje zapis a aj citanie do DS1624
203     }
204   }
205
206   void Read_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
      Dlzka, unsigned char *Buffer)
207   {
208    register unsigned char i;
209    for (i=0x00; i<Dlzka; i++)
210     {
211      Buffer[i]=Read_Eprom_DS1624 (Adresa,EEprom_Addr+i);   //Funguje zapis a aj citanie do DS1624
212     }
213   }
```

Pri zápise do E²PROM[156] je treba mať na pamäti, že je nutné počkať cca. 10ms pri 20°C (50ms max.) na ukončenie zápisového cyklu do pamäti E²PROM. Až prijatí I2C_Stop() sa skutočne uskutoční zápis z internej pamäti RAM obvodu DS1624 do E²PROM.

[156] Pamäť je možné použiť aj v jednoduchom i sekvenčnom zápisovom cykle.

15.22. Pamäť E²PROM s rozhraním I²C

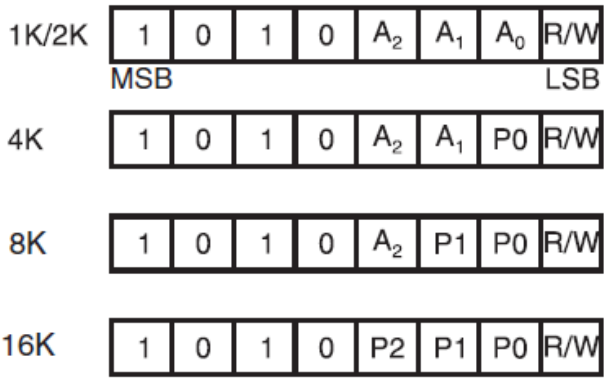
15.22.1. Bloková organizácia pamäti AT24C32/64.



Charakteristické vlastnosti pamäte AT24C32.

- vyrobená CMOS technológiou,
- organizácia[157] 4096 slov s 8 bitmi,
- obojsmerné I²C rozhranie,
- napájacie napätie od 1,8V do 5,5V,
- 10^6 zápisových cyklov, doba uloženia 100 rokov.

Obrázok 160, Tvorba adresy E²PROM pamäti AT24C01 až AT24C16



[157] Pre AT24C01 až AT24C16 je potrebné odoslať len 8 bitovú adresu cez I²C v režime čítania a zápisu. Ovládač je zhodný pre E²PROM pamäti radu AT24C01 až AT24C64, prípadne po malých úpravách blokového zápisu na 64Byte aj pamäť 24C256. Adresa v tomto prípade je tvorená kombináciou adresy obvodu a výberových bitov P0, P1 a P2, ktoré predstavujú tzv. stránku pamäti. Pamäť M24256 nie je odporúčaná pre nové aplikácie. Technické informácie o pamäti E²PROM 24C256 sú http://www.techdesign.be/projects/datasheet/24LC256.pdf. AT24C32 a vyššie už vyžadujú 16 bitovú adresu na zbernici pri zápise a čítaní z/do pamäti.

15.22.2. Funkcie zápisu a čítania pre AT24C01 až AT24C16 E²PROM

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\AT24Cxx.c

```
#include <reg552.h>
#include <rtx51.h>
#include <stdio.h>
#include <stdlib.h>
#include <dsw.h>

#define READ   1
#define WRITE  0

#define EEPROM_WAIT 1

extern void I2C_Start (void);
extern void I2C_Stop (void);
extern void Delay (unsigned char Dlzka);
extern void I2C_SendAddr (unsigned char Data);
extern void I2C_SendByte (unsigned char Data);
extern unsigned char I2C_Read (unsigned char);

void Write_Eprom_AT24Cxx (unsigned char Adresa, unsigned int EEprom_Addr, unsigned char Data)
{
 unsigned char subaddr;
 subaddr=(high (EEprom_Addr))*2;                //Pre pamati AT24C01 az AT24C16
 I2C_SendAddr (Adresa+subaddr+WRITE);
 I2C_SendByte (low (EEprom_Addr));
 I2C_SendByte (Data);
 I2C_Stop ();
 os_wait (K_TMO,EEPROM_WAIT,NULL);          //Cakam 10ms cez RTX51 Full
}

unsigned char Read_Eprom_AT24Cxx (unsigned char Adresa, unsigned int EEprom_Addr)
{
 unsigned char LSB,subaddr;
 subaddr=(high (EEprom_Addr))*2;                //Pre pamati AT24C01 az AT24C16
 I2C_SendAddr (Adresa+subaddr+WRITE);
 I2C_SendByte (low (EEprom_Addr));
 I2C_SendAddr (Adresa+READ);
 LSB=I2C_Read (0);
 I2C_Stop ();
 return (LSB);
}
```

15.22.3. Funkcie zápisu a čítania pre AT24C32 4kB E²PROM

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\AT24C32.c

```
void Write_Eprom_AT24C32 (unsigned char Adresa, unsigned int EEprom_Addr, unsigned char Data)
{
 I2C_SendAddr (Adresa+WRITE);
 I2C_SendByte (high (EEprom_Addr));          //Pre AT24C08 tento riadok vymazat, je to 1kB pamat
 I2C_SendByte (low (EEprom_Addr));
 I2C_SendByte (Data);
 I2C_Stop ();
 os_wait (K_TMO,EEPROM_WAIT,NULL);          //Cakam 10ms cez RTX51 Full
}

unsigned char Read_Eprom_AT24C32 (unsigned char Adresa, unsigned int EEprom_Addr)
{
 unsigned char LSB;
 I2C_SendAddr (Adresa+WRITE);
 I2C_SendByte (high (EEprom_Addr));              //Pre AT24C08 tento riadok vymazat, je to 1kB pamat
 I2C_SendByte (low (EEprom_Addr));
 I2C_SendAddr (Adresa+READ);
 LSB=I2C_Read (0);
 I2C_Stop ();
 return (LSB);
}
```

Pri použití simulácie I²C sériovej pamäti E²PROM pomocou AGSI ovládača je nevyhnutné nastaviť správne komunikačné vodiče[158] SDA a SCL. Pri pomalších zariadeniach je potrebné vložiť čakacie cykly ktoré zabezpečia požadované časovanie obvodov. Zápis a čítanie[159] je potom vykonané cez I²C komunikačný protokol. Komunikačné rutiny sú vytvorené ako univerzálne pomocou 2 komunikačných vrstiev. Prvá vrstva tvorená programom I2C_Driver.c je nízkoúrovňový hardware-ový ovládač I²C protokolu, nad ktorým je už druhá vrstva predstavujúca software-ový ovládač I²C zariadenia.

15.22.4. Funkcie blokového zápisu a čítania pre AT24C32 4kB E²PROM

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\AT24C32.c

```
41  #define  SizeOfEpromBlock  32                    //Velkost stranky 32Byte u AT24C32 pre blokovy
    prenos
42
43  void  Write_Block_Eprom_AT24C32 (unsigned  char  Adresa , unsigned  char  EEprom_Page , unsigned  char  *
    Buffer )
44  {
45   #ifndef  RTX51
46    unsigned  char  Counter =60 ;
47   #endif
48   unsigned  char  i;
49   I2C_SendAddr (Adresa +WRITE );
50   I2C_SendByte (high (EEprom_Page *SizeOfEpromBlock ));         //Pre AT24C08 tento riadok vymazat,
    je to 1kB pamat
51   I2C_SendByte (low (EEprom_Page *SizeOfEpromBlock ));
52   for (i=0x00 ; i<SizeOfEpromBlock ; i++)
53    {
54     I2C_SendByte (Buffer [i]);
55     os_wait (K_TMO ,EEPROM_WAIT ,NULL );            //Cakam 10ms cez RTX51 Full
56    }
57   I2C_Stop ();
58  }
59
60  void  Read_Block_Eprom_AT24C32 (unsigned  char  Adresa , unsigned  char  EEprom_Page , unsigned  char  *
    Buffer )
61  {
62   unsigned  char  i;
63   I2C_SendAddr (Adresa +WRITE );
64   I2C_SendByte (high (EEprom_Page *SizeOfEpromBlock ));         //Pre AT24C08 vymazat, je to 1kB pamat
65   I2C_SendByte (low (EEprom_Page *SizeOfEpromBlock ));
66   I2C_SendAddr (Adresa +READ );
67   for (i=0x00 ; i<SizeOfEpromBlock ; i++)                      //Prenasam 32Byte blok udajov s pACK
68    {
69     switch (i)
70      {
71       case  (SizeOfEpromBlock -1)   : Buffer [i]= I2C_Read (0); break ;    // nACK pre ukoncenie
72       default                       : Buffer [i]= I2C_Read (1); break ;    //pACK pre udrziavanie
73      }
74    }
75   I2C_Stop ();
76  }
```

[158] Komunikačný hardware I²C je v súčasnosti už nevyhnutnou súčasťou niektorých mikroprocesorov napr. Philips 80C552 a sú mapované na porty P1.6 (SCL) a P1.7 (SDA). Ich použitie je možné zaznamenať v širokej oblasti spotrebnej elektroniky.

[159] Čítať z pamäti E²PROM cez AGSI ovládač je možné len hodnoty ktoré boli zapísané priamo cez sériový komunikačný kanál do I²C pamäti. V prípade, že chceme prečítať hodnoty ktoré sme tam vložili len cez klávesnicu pomocou integrovaného „EditBox-u“ v prostredí Keil µVision5, tak nedostaneme žiadne hodnoty, pretože vložené hodnoty sa automaticky neprepíšu do sériovej pamäti E²PROM v systéme AGSI ovládača pamäti v prostredí µVision.

15.23. AD prevodník v 80C552

15.23.1. Ovládač pre 10 bitový A/D prevodník v 80C552 v assembleri

C:\Omega\Data\Dropbox\Záloha\C51\AD in C\ADC Converter.asm

```
;-------------------------------------------------------------------------------
Repeat              macro   Num
                        anl a,#Num
                        rl a
                        rl a
                    endm
;-------------------------------------------------------------------------------
public              _ADC_CONVERSION         ;ADC_Conversion(unsigned char Channel)
public              ?PR?CONVERSION          ;Conversion(unsigned char Channel)
;-------------------------------------------------------------------------------
ADCON               data 0C5h
ADCH                data 0C6h
;-------------------------------------------------------------------------------
ADCS                bit acc.3               ;Spustim prevod AD prevodnika
ADCI                bit acc.4               ;Kontrolujem ci je prevod ukonceny ADCON.4...
MASKA               equ 11000000B           ;D[6..7] ADCH alebo ADCON[6..7]
;-------------------------------------------------------------------------------
Prog_ADC_Converter  segment code
                    rseg Prog_ADC_Converter
;-------------------------------------------------------------------------------
_ADC_CONVERSION:    mov a,r7                ;Vstupny parameter je cislo kanala pre meranie v R7
                    setb ADCS               ;Nastavim "ADC prevod"
                    xch a,ADCON             ;Obsah A potrebujem do ADCON aby sa AD prevod
spustil
?PR?ADC_END:        mov a,ADCON             ;ADCI signalizuje ukonceny prevod a pritomnost
vysledku
                    jnb ADCI,?PR?ADC_END    ;Je prevod ukonceny ???
                    clr ADCI                ;bit ADCI nastavujem na 0
                    mov ADCON,a
?PR?CONVERSION:     Repeat MASKA            ;LSB vysledku je v D7,D6 v ADCON[6..7]
                    mov r7,a                ;LSB AD prevodu 2 bitov R7=(ADCON[6..7]<<2) ...
                    mov a,ADCH
                    Repeat ~MASKA           ;Ziskavam ADCH[0..5]<<2 & (~MASKA) a scitam ich s R7
                    orl a,r7                ;Scitavam ADCH[2..7] | R7
                    xch a,r7                ;vysledok ADCH[2..7] | (ADCON[6..7]<<2)
                    mov a,ADCH
                    Repeat MASKA            ;Ziskavam ADCH[6..7]&MASKA
                    mov r6,a                ;Vysledok je MSB AD prevodu
                    ret
;-------------------------------------------------------------------------------
;                   cseg at 0053h           ;os_attach_interrupt(10)
;                   push acc
;                   pop acc
;                   reti
;-------------------------------------------------------------------------------
                    end
```

Vykonávanie tejto rutiny trvá[160] 55µs pri 18.432MHz oscilátore. Rovnaký AD prevodník obsahuje aj mikroprocesor DS87C550 od firmy Maxim Integrated.[161] Prevod v tomto prípade je 16 period[162] oscilátora.

[160] S volaním C a odovzdaním parametrov trvá rutina celkovo 63µs.

[161] http://www.maximintegrated.com/, firma ukončila oficiálne jeho výrobu. Pinovo a obvodovo je kompatibilný s mikroprocesorom Philips 80C552. DS87C550 obsahuje navyše ešte port P6 ktorý obsadzuje nepoužité vývody obvodu PLCC.

[162] Podrobný vzťah na výpočet prevodu je uvedený na http://pdfserv.maximintegrated.com/en/ds/DS87C550.pdf.

15.23.2. Ovládač pre 10 bitový A/D prevodník v 80C552 v jazyku C

C:\Omega\Data\Dropbox\Záloha\C51\AD in C\AD in C.c

```
#include <reg552.h>
#include <rtx51.h>
#include <stdlib.h>
#include <stdio.h>
#include <intrins.h>

#define ADCS 0x08
#define ADCI 0x10
#define ADEX 0x20

extern int ADC_Conversion(unsigned char);

int Voltage;

int ADC(unsigned char Channel)     //Rutina trva 55us pri 18.432MHz
{
 register unsigned int x;
 ADCON=Channel;                    //Vyber kanalu pre prevod 0..7
 ADCON|=(ADEX+ADCS);               //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
 while((ADCON&ADCI)==0x00);        //Cakam na ukoncenie prevodu
 ADCON^=ADCI;                      //Nulujem priznak AD prevodnika
 x=ADCH*0x04;                      //Normovanie na (int)
 switch(ADCON&(0x80+0x40))
  {
   case 0x40  : x+=0x01; break;
   case 0x80  : x+=0x02; break;
   case 0xC0  : x+=0x03; break;
  }
 return(x);
}
```

Vykonávanie tejto rutiny trvá[163] 55µs pri 18.432MHz oscilátore.

```
unsigned int ADCC (unsigned char Channel )     //Rutina trva 97us pri 18.432MHz aj s volanim C
{
 ADCON =Channel ;                              //Vyber kanalu pre prevod 0..7
 ADCON |=ADCS ;                                //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
 while ((ADCON &ADCI )== 0x00 );               //Cakam na ukoncenie prevodu cca. 80us
 ADCON ^=ADCI ;                                //Nulujem priznak AD prevodnika
 return (256 *ADCH +(ADCON &0xC0 )>> 6);
}
```

Vykonávanie tejto rutiny trvá 97µs pri 18.432MHz oscilátore.

15.24. Simulácia pamäti E²PROM v µVision

Mikroprocesory AT89C51ED2 a iné majú integrovanú energeticky nezávislú pamäť E²PROM ktorá sa používa na ukladanie konštánt, alebo premenných použitých v programe. Počet zápisových cyklov do tejto pamäti obmedzený technologicky približne na 10^5 zápisových cyklov. Počet čítacích cyklov z pamäti E²PROM je neobmedzený. Program µVison4 dokáže pomocou LX51[164], alebo AX51 pristupovať do tejto pamäti[165] ktorá je deklarovaná ako **far** umiestnená v pamäťovom priestore v rozsahu X:0x020000 - X:0x0207FF.[166]

[163] S volaním C a odovzdaním parametrov trvá rutina celkovo 67µs.
[164] S operačným systémom RTX51 Tiny alebo RTX51 Full po naprogramovaní nepracuje správne a nie je možné kvôli neustálym resetom odladiť napísaný program !!!
[165] Pre správnu činnosť je potrebné zapnúť mód LX51, AX51 prekladača. Pre možnosti simulácie viď. stránku http://www.keil.com/uvision/db_sim_prf_flash.asp
[166] Viď. DS89C390, alebo DS80C400, DS80C410, DS80C411.

Obrázok 161, Použitie E^{2}PROM pamäti v programe

```
/*
 *  This file contains the data defintions that are stored in EEPROM
 */

#ifndef  DECLARE                    // if DECLARE is not defined
  #define  DECLARE   extern         // variables are extern
#endif

struct  sample   {
  char  carray[20];
  int   iarray[20];
  long  larray[20];
};

DECLARE  struct  sample  far  esample1;  // reserve space for esample1 struct
DECLARE  struct  sample  far  esample2;  // reserve space for esample2 struct
DECLARE  char  *         far  etxt_ptr;  // pointer in EEPROM space
```

15.25. Elektronický potenciometer DS1807 s I^{2}C

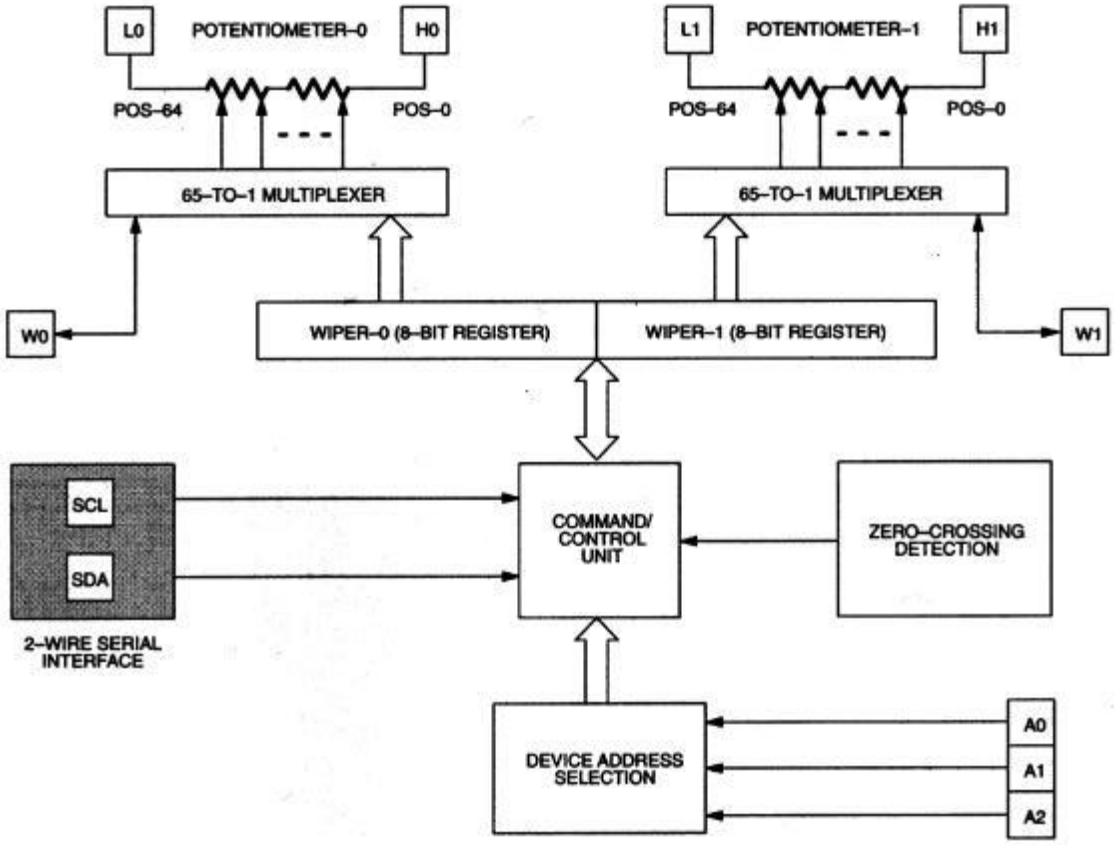
```
#include <reg552.h>
#include <rtx51.h>
#include <stdio.h>
#include <stdlib.h>
#include <dsw.h>

#define  I2C_READ   1
#define  I2C_WRITE  0

#define  EEPROM_WAIT  1

extern  void  Delay(unsigned  char  Dlzka);
extern  void  I2C_SendAddr(unsigned  char  Data,unsigned  char  ACK);
extern  void  I2C_SendByte(unsigned  char  Data);
extern  unsigned  char  I2C_Read(unsigned  char);
extern  void  I2C_Start(void);
extern  void  I2C_Stop(void);

#define  DS1807   0x50                      //CONTROL BYTE t.j. Idendifikacne cislo - adresa DS1807
#define  POT0     0xA9                      //COMMAND
#define  POT1     0xAA                      //COMMAND
#define  POTALL   0xAF                      //COMMAND

void  Write_DS1807(unsigned  char  Adresa,  unsigned  int  Data)
{
 #ifndef  RTX51
  register  unsigned  char  Counter=60;
 #endif
 I2C_SendAddr(Adresa,I2C_WRITE);
 I2C_SendByte(POT0);
 I2C_SendByte(high(Data));                  //Data pre POT1
 I2C_SendByte(low(Data));                   //Data pre POT0
 I2C_Stop();
 os_wait(K_TMO,EEPROM_WAIT,NULL);
}

unsigned  int  Read_DS1807(unsigned  char  Adresa)
{
 register  unsigned  char  LSB,MSB;
 I2C_SendAddr(Adresa,I2C_READ);
 MSB=I2C_Read(1);                           //Data pre POT1
 LSB=I2C_Read(0);                           //Data pre POT0
 I2C_Stop();
 return(MSB*0x100+LSB);
}
```

Elektronický duálny logaritmický odporový potenciometer[167] DS1807 sa používa v spotrebnej elektronike nielen na reguláciu hlasitosti, ale aj ako elektronický potenciometer ovládaný po

[167] DS1807 je 6-bitový.

rozhraní I²C. Jeden krok v tomto prípade tu predstavuje 1dB a 64 krok až 90dB útlm[168]. Súčasne môže pracovať na rovnakej I²C zbernici až 8 obvodov.

Obrázok 162, Vnútorná schéma DS1807



[168] Funkcia MUTE.

15.26. Funkcie I²C pre DS1631 v jazyku C51

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
#include <rtx51.h>
#include <reg552.h>
#include <stdio.h>
#include <stdlib.h>
#include <dsw.h>

#define K (1.0/256.0)

#ifndef READ
  #define READ 1
#endif
#ifndef WRITE
  #define WRITE 0
#endif

#define EEPROM_WAIT 1

extern void Delay(unsigned char);

extern void I2C_Start(void);
extern void I2C_Stop(void);
extern unsigned char I2C_Read(unsigned char);
extern unsigned char I2C_SendByte(unsigned char);
extern unsigned char I2C_SendAddr(unsigned char);
extern void Write_TH_Register(unsigned char, unsigned int);
extern void Write_TL_Register(unsigned char, unsigned int);
extern unsigned int Read_TH(unsigned char);
extern unsigned int Read_TL(unsigned char);

#ifdef DS1631                          //DS1631 je maximálne 12bitový teplomer
 #define START_CONVERT     0x51        //DS1631 I2C_START_CONVERT = 0x51, DS1624 I2C_START_CONVERT = 0xEE
#endif

#ifdef DS1624                          //DS1624 je maximálne 13bitový teplomer
 #define START_CONVERT     0xEE        //DS1631 I2C_START_CONVERT = 0x51, DS1624 I2C_START_CONVERT = 0xEE
 #define ACCESS_MEMORY     0x17        //DS1624
#endif

#define STOP_CONVERT       0x22
#define ACCESS_TH          0xA1
#define ACCESS_TL          0xA2
#define READ_TEMPERATURE   0xAA
#define DS1631_COMMAND     0xAC
#define DS1624_COMMAND     0xAC
#define COMMAND_DONE       0x80
#define COMMAND_1SHOT      0x01
#define POR                0x54
#define READ_COUNTER       0xA8        //DS1621
#define READ_SLOPE         0xA9        //DS1621

#define AddrDS1631A  0x90                //Adresa pre DS1621,DS1624,DS1631 - tzv. CONTROL_BYTE
#define AddrDS1624A  0x90                //Adresa pre DS1621,DS1624,DS1631 - tzv. CONTROL_BYTE
#define AddrAT24c32  0xA0                //Adresa pre AT24C08 az AT24C64 - tzv. CONTROL_BYTE
#define AddrDS1807A  0x50                //Adresa pre DS1807A - tzv. CONTROL_BYTE

unsigned char Get_Config(unsigned char Adresa)
{
 register unsigned char LSB;
 I2C_SendAddr(Adresa+WRITE);
 I2C_SendByte(DS1631_COMMAND);
 I2C_SendAddr(Adresa+READ);
 LSB=I2C_Read(1);
 I2C_Stop();
 return(LSB);
}

void Write_Config(unsigned char Adresa, unsigned char Data)
{
 I2C_SendAddr(Adresa+WRITE);
 I2C_SendByte(DS1631_COMMAND);
 I2C_SendByte(Data);
 os_wait(K_TMO,EEPROM_WAIT,NULL);    //Cakam 10ms cez RTX51 Full
```

Page1

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
 73     I2C_Stop ();
 74    }
 75
 76    void  Config_DS16xx (unsigned  char  Adresa , unsigned  char  Bits )
 77    {
 78     register  unsigned  char  Config ;
 79     register  unsigned  int  Temp ;
 80     Temp =Read_TH (AddrDS1631A );
 81     Temp =Read_TL (AddrDS1631A );
 82     Write_TH_Register (AddrDS1631A ,0xabcd );
 83     Write_TL_Register (AddrDS1631A ,0x1234 );
 84     Config =Get_Config (Adresa );
 85     Config &=~( 0x08 +0x04 );
 86     switch (Bits )
 87      {
 88       case  9    : break ;
 89       case  10   : Config |= 0x04 ; break ;
 90       case  11   : Config |= 0x08 ; break ;
 91       default    : Config |= 0x0c ; break ;   //Ak je zadana zla hodnota alebo 12 tak nastavim 12 bitov
 92      }
 93     Write_Config (Adresa ,Config );        //Nastavim 12 bitovy prevod
 94    }
 95
 96    unsigned  int  Get_Int_Temp (unsigned  char  Adresa )
 97    {
 98     register  unsigned  char  MSB ,LSB ;
 99     I2C_SendAddr (Adresa +WRITE );
100     I2C_SendByte (START_CONVERT );
101     I2C_Stop ();
102    //-----------------------------------------------------
103     I2C_SendAddr (Adresa +WRITE );
104     I2C_SendByte (STOP_CONVERT );
105     I2C_Stop ();
106    //-----------------------------------------------------
107     I2C_SendAddr (Adresa +WRITE );
108     I2C_SendByte (READ_TEMPERATURE );
109     I2C_SendAddr (Adresa +READ );
110     MSB =I2C_Read (1);      //Vyzaduje sa zo SLAVE-a potvrdenie ACK=0!!!
111     LSB =I2C_Read (0);      //Vyzaduje sa z MASTER-a potvrdenie ACK=1!!!
112     I2C_Stop ();
113    //*************************************************
114     if (MSB >= 0x80 ) return ((MSB *256 +LSB )- 65536 ); else return (MSB *256 +LSB );
       //Je to OK !!!
115    //*************************************************
116    }
117
118    float  Get_Float_Temp (unsigned  char  Adresa )
119    {
120     register  unsigned  char  MSB ,LSB ;
121     I2C_SendAddr (Adresa +WRITE );
122     I2C_SendByte (START_CONVERT );
123     I2C_Stop ();
124    //-----------------------------------------------------
125     I2C_SendAddr (Adresa +WRITE );
126     I2C_SendByte (STOP_CONVERT );
127     I2C_Stop ();
128    //-----------------------------------------------------
129     I2C_SendAddr (Adresa +WRITE );
130     I2C_SendByte (READ_TEMPERATURE );
131     I2C_SendAddr (Adresa +READ );
132     MSB =I2C_Read (1);      //Vyzaduje sa zo SLAVE-a potvrdenie ACK=0!!!
133     LSB =I2C_Read (0);      //Vyzaduje sa z MASTER-a potvrdenie ACK=1!!!
134     I2C_Stop ();           //MSB=0x19; LSB=0x10;  //25.062°C
135    //*************************************************
136     if (MSB >= 0x80 ) return ((float )(MSB *256 +LSB )- 65536 ); else return ((float )(MSB *256 +LSB )* K);    //Je
       to OK !!!
137    //*************************************************   }
138    }
139
140    void  Write_TH_Register (unsigned  char  Adresa , unsigned  int  THReg )
141    {
142     I2C_SendAddr (Adresa +WRITE );
143     I2C_SendByte (ACCESS_TH );           // !!!!!!!!!!!!!!!!!!!!
144     I2C_SendByte (high (THReg ));
```

Page2

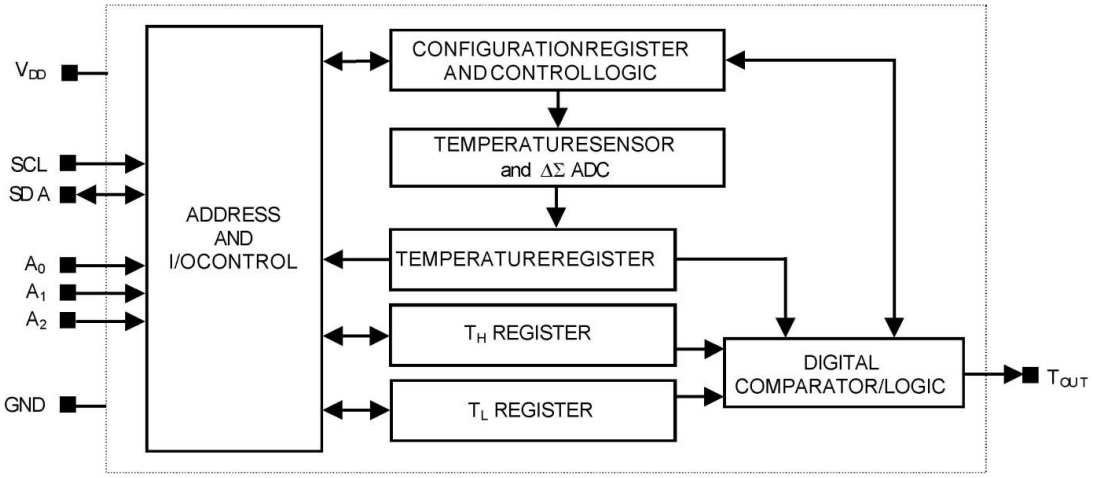
c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
145     I2C_SendByte (low (THReg ));
146     I2C_Stop ();
147     os_wait (K_TMO ,EEPROM_WAIT ,NULL );    //Cakam 10ms cez RTX51 Full
148    }
149
150    void  Write_TL_Register (unsigned  char  Adresa , unsigned  int  TLReg )
151    {
152     I2C_SendAddr (Adresa +WRITE );
153     I2C_SendByte (ACCESS_TL );            // !!!!!!!!!!!!!!!!!!!!
154     I2C_SendByte (high (TLReg ));
155     I2C_SendByte (low (TLReg ));
156     I2C_Stop ();
157     os_wait (K_TMO ,EEPROM_WAIT ,NULL );    //Cakam 10ms cez RTX51 Full
158    }
159
160    unsigned  int  Read_TH (unsigned  char  Adresa )
161    {
162     register  unsigned  char  MSB ,LSB ;
163     I2C_SendAddr (Adresa +WRITE );
164     I2C_SendByte (ACCESS_TH );
165     I2C_SendAddr (Adresa +READ );
166     MSB =I2C_Read (1);
167     LSB =I2C_Read (0);
168     I2C_Stop ();
169     return (MSB *0x100 +LSB );
170    }
171
172    unsigned  int  Read_TL (unsigned  char  Adresa )
173    {
174     register  unsigned  char  MSB ,LSB ;
175     I2C_SendAddr (Adresa +WRITE );
176     I2C_SendByte (ACCESS_TL );
177     I2C_SendAddr (Adresa +READ );
178     MSB =I2C_Read (1);
179     LSB =I2C_Read (0);
180     I2C_Stop ();
181     return (MSB *0x100 +LSB );
182    }
183
184    #ifdef  DS1624
185
186    void  Write_Eprom_DS1624 (unsigned  char  Adresa , unsigned  char  EEprom_Addr , unsigned  char  Data )
       //Funguje zapis a aj citanie do DS1624
187    {
188     I2C_SendAddr (Adresa +WRITE );
189     I2C_SendByte (ACCESS_MEMORY );
190     I2C_SendByte (EEprom_Addr );
191     I2C_SendByte (Data );
192     I2C_Stop ();
193     os_wait (K_TMO ,EEPROM_WAIT ,NULL );    //Cakam 10ms cez RTX51 Full
194    }
195
196    unsigned  char  Read_Eprom_DS1624 (unsigned  char  Adresa , unsigned  char  EEprom_Addr )   //Funguje
       zapis a aj citanie do DS1624
197    {
198     register  unsigned  char  LSB ;
199     I2C_SendAddr (Adresa +WRITE );
200     I2C_SendByte (ACCESS_MEMORY );
201     I2C_SendByte (EEprom_Addr );
202     I2C_SendAddr (Adresa +READ );
203     LSB =I2C_Read (0);
204     I2C_Stop ();
205     return (LSB );
206    }
207
208    void  Write_Block_E2prom_DS1624 (unsigned  char  Adresa , unsigned  char  EEprom_Addr , unsigned  char
       Dlzka , unsigned  char  *Buffer )
209    {
210     #ifndef  RTX51
211      register  unsigned  char  Counter =60 ;
212     #endif
213     register  unsigned  char  i;
214     for (i=0x00 ; i<Dlzka ; i++)
215      {
```

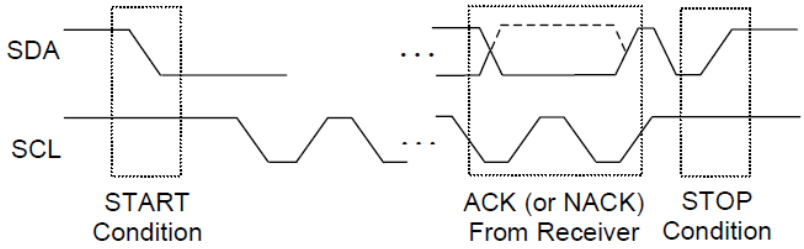
c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
216       Write_Eprom_DS1624 (Adresa ,EEprom_Addr +i,Buffer [i]);   //Funguje zapis a aj citanie do DS1624
217      }
218    }
219
220    void  Read_Block_E2prom_DS1624 (unsigned  char  Adresa , unsigned  char  EEprom_Addr , unsigned  char
       Dlzka , unsigned  char  *Buffer )
221    {
222     register  unsigned  char  i;
223     for (i=0x00 ;  i<Dlzka ;  i++)
224      {
225       Buffer [i]= Read_Eprom_DS1624 (Adresa ,EEprom_Addr +i);    //Funguje zapis a aj citanie do DS1624
226      }
227    }
228
229    /*
230
231    #define  SizeRamBlock       8              //Velkost stranky je 8-Byte u DS1624 pre blokovy prenos
       ...
232    #define SizeOfEpromBlock   256            //Velkost pamati EEPROM ...
233
234    void Write_Block_Eprom_DS1624(unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
       *Buffer)
235    {
236     register unsigned char i,ram_cache=0x01;
237     I2C_SendAddr(Adresa+WRITE);
238     I2C_SendByte(ACCESS_MEMORY);
239     I2C_SendByte(EEprom_Addr);
240     for(i=0x00; i<SizeOfEpromBlock; i++)    //Maximalne prenesiem blokovo 8-byte a I2C_Stop()
       spusti jeho prepis z RAM do EEPROM !!!
241      {
242       I2C_SendByte(Buffer[i]);
243       if(ram_cache++/SizeRamBlock)
244       {
245        ram_cache=0x01;
246        I2C_Stop();
247        os_wait(K_TMO,EEPROM_WAIT,NULL);     //Cakam 10ms cez RTX51 Full
248       }
249      }
250     if((ram_cache!=0x01)&(i!=0x00))  I2C_Stop();         //je to dobre ???
251    }
252
253    void Read_Block_Eprom_DS1624(unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
       *Buffer)
254    {
255     register unsigned char i;
256     I2C_SendAddr(Adresa+WRITE);
257     I2C_SendByte(ACCESS_MEMORY);
258     I2C_SendByte(EEprom_Addr);
259     I2C_SendAddr(Adresa+READ);
260     for(i=0x00; i<SizeOfEpromBlock; i++)                                  //Prenasam 256Byte blok
       udajov s pACK
261      {
262       switch(i)
263        {
264         case  (SizeOfEpromBlock-1)   : Buffer[i]=I2C_Read(0); break;    // nACK pre ukoncenie
265         default                      : Buffer[i]=I2C_Read(1); break;    // pACK pre udrziavanie
266        }
267      }
268     I2C_Stop();
269    }
270    */
271    #endif
```

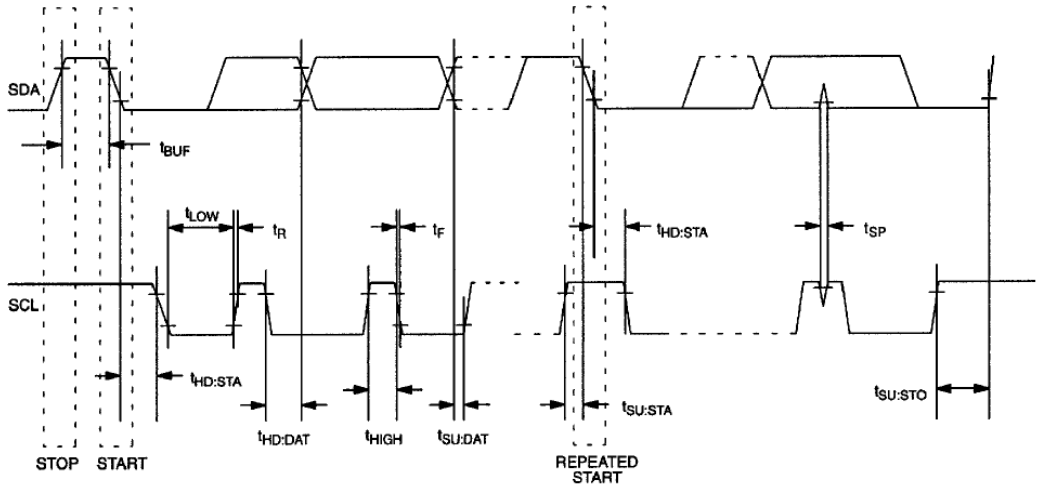
Obrázok 163, Komunikačný model obvodu DS1631 – vnútorné zapojenie[169]



Obrázok 164, Symbolické priebehy napätí na I²C rozhraní pri komunikácii



Obrázok 165, Detailné priebehy napätí na I²C rozhraní pri komunikácii



[169] http://datasheets.maximintegrated.com/en/ds/DS1631-DS1731.pdf

Obrázok 166, Časové závislosti na I²C rozhraní DS1631

Tabuľka 13, Výstupný formát výsledku prevodu z DS1631

	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
MS Byte	S	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	bit 7	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
LS Byte	2^{-1}	2^{-2}	2^{-3}	2^{-4}	0	0	0	0

Tabuľka 14, Výsledok 12-bitového prevodu z DS1631

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0111 1101 0000 0000	7D00h
+25.0625	0001 1001 0001 0000	1910h
+10.125	0000 1010 0010 0000	0A20h
+0.5	0000 0000 1000 0000	0080h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1000 0000	FF80h
-10.125	1111 0101 1110 0000	F5E0h
-25.0625	1110 0110 1111 0000	E6F0h
-55	1100 1001 0000 0000	C900h

Tabuľka 15, Konfiguračný register DS1631

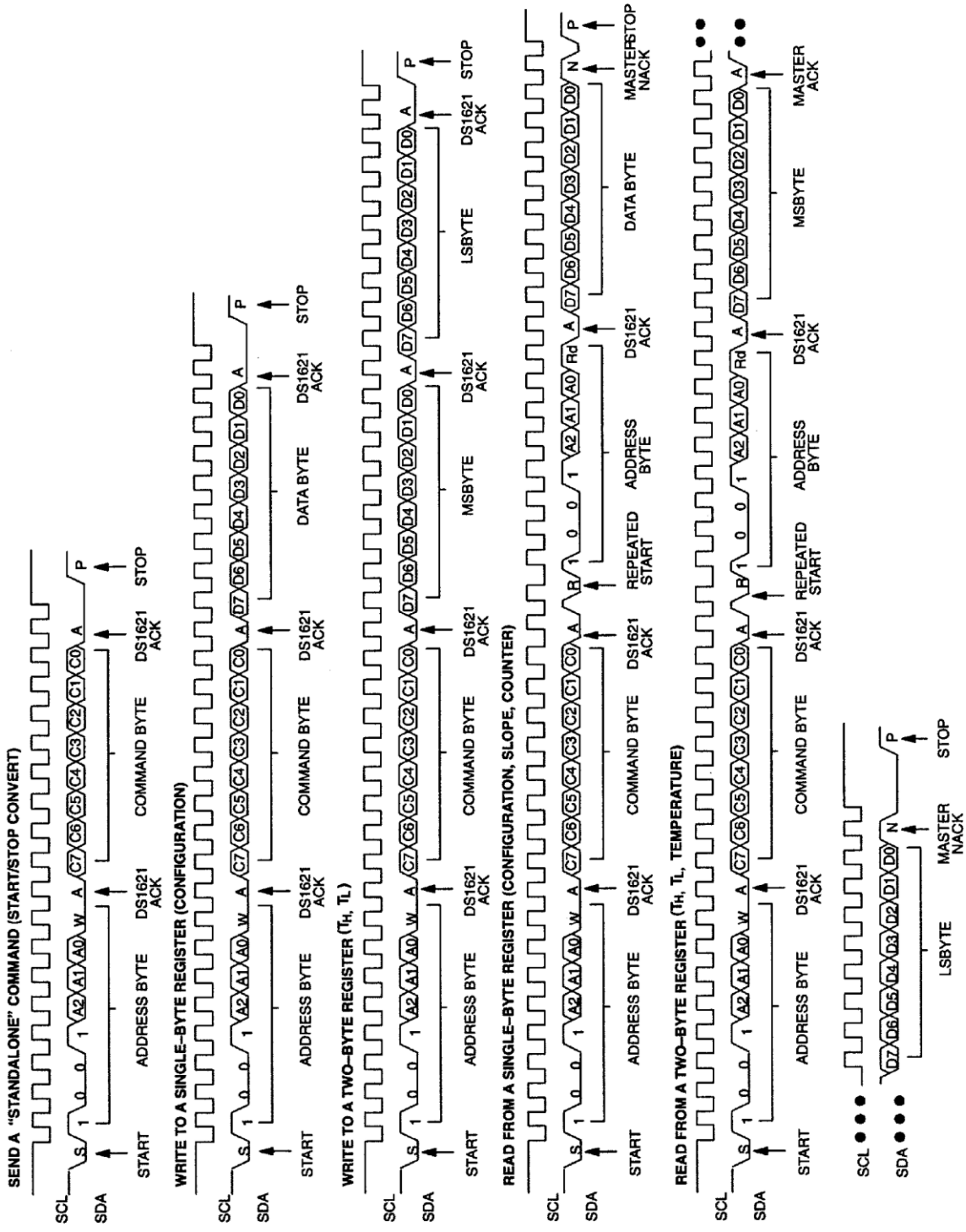
MSb	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	LSb
DONE	THF	TLF	NVB	R1	R0	POL*	1SHOT*

*NV (EEPROM)

Tabuľka 16, Nastavenie presnosti prevodu teploty v DS1631

R1	R0	RESOLUTION (BIT)	CONVERSION TIME (MAX)
0	0	9	93.75ms
0	1	10	187.5ms
1	0	11	375ms
1	1	12	750ms

Obrázok 167, Časové závislosti na I²C rozhraní DS1621



DS1621 je obvodovo a protokolovo zhodný ako DS1624 a je ho možné simulovať v prostredí µVision4.

15.27.Zoznam funkcií pre I²C rozhranie

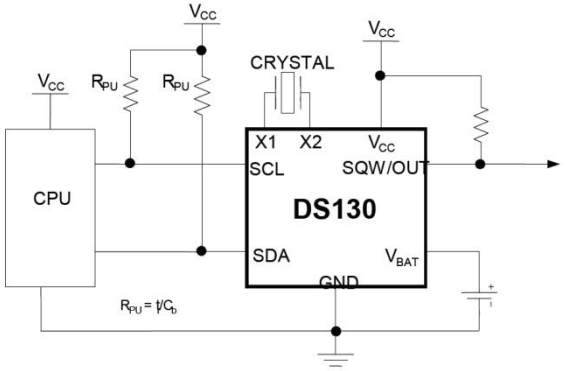
c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\I2C_by_DSW.c

```
21  //*************************************************************************************************************
    **********************************
22  extern void Delay (unsigned char );
23  extern void I2C_Start (void );
24  extern void I2C_Stop (void );
25  extern unsigned char I2C_Read (unsigned char doACK );
26  extern unsigned char I2C_SendByte (unsigned char Data );
27  extern unsigned char I2C_SendAddr (unsigned char Addr );
28  #ifdef BLOCK
29  extern unsigned char i2c_writeblock (unsigned char address ,unsigned char *barr ,int length );
30  extern unsigned char i2c_readblock (unsigned char address ,unsigned char *barr ,int length );
31  extern unsigned char i2c_writereadblock (unsigned char address ,unsigned char *barr1 ,int length1 ,
    unsigned char *barr2 ,int length2 );
32  #endif
33  //*************************************************************************************************************
    **********************************
34  extern unsigned char Get_Config (unsigned char Adresa );
35  extern void Write_Config (unsigned char Adresa , unsigned char Data );
36  extern void Config_DS16xx (unsigned char Adresa , unsigned char Bits );
37  extern unsigned int Get_Int_Temp (unsigned char Adresa );
38  extern float Get_Float_Temp (unsigned char Adresa );
39  extern void Write_TH_Register (unsigned char , unsigned int );
40  extern void Write_TL_Register (unsigned char , unsigned int );
41  extern unsigned int Read_TH (unsigned char );
42  extern unsigned int Read_TL (unsigned char );
43  #ifdef DS1624
44   extern void Write_Eprom_DS1624 (unsigned char Adresa , unsigned char EEprom_Addr , unsigned char
    Data ); //Pre DS1624
45   extern unsigned char Read_Eprom_DS1624 (unsigned char Adresa , unsigned char EEprom_Addr );
                //Pre DS1624
46   extern void Write_Block_Eprom_DS1624 (unsigned char Adresa , unsigned char EEprom_Addr , unsigned
    char *Buffer );
47   extern void Read_Block_Eprom_DS1624 (unsigned char Adresa , unsigned char EEprom_Addr , unsigned
    char *Buffer );
48   extern void Write_Block_E2prom_DS1624 (unsigned char Adresa , unsigned char EEprom_Addr , unsigned
     char Dlzka , unsigned char *Buffer );
49   extern void Read_Block_E2prom_DS1624 (unsigned char Adresa , unsigned char EEprom_Addr , unsigned
    char Dlzka , unsigned char *Buffer );
50  #endif
51  //*************************************************************************************************************
    **********************************
52  extern void Write_Eprom_AT24C32 (unsigned char , unsigned int , unsigned char );
53  extern unsigned char Read_Eprom_AT24C32 (unsigned char , unsigned int );
54  extern void Write_Block_Eprom_AT24C32 (unsigned char Adresa , unsigned char EEprom_Addr , unsigned
    char *Buffer );
55  extern void Read_Block_Eprom_AT24C32 (unsigned char Adresa , unsigned char EEprom_Addr , unsigned
    char *Buffer );
56  //*************************************************************************************************************
    **********************************
57  extern void Write_DS1807 (unsigned char Adresa , unsigned int Data );
58  extern unsigned int Read_DS1807 (unsigned char Adresa );
59  //*************************************************************************************************************
    **********************************
60  extern void Init_Serial (void );
61  //*************************************************************************************************************
    **********************************
62  extern int ADC_Conversion (unsigned char );
63  //*************************************************************************************************************
    **********************************
64
```

15.28. Obvod DS1307 - Real Time Clock

Špecializovaný obvod DS1307 je použiteľný ako hodinový obvod v aplikáciách s mikroprocesormi. Pomocou I²C rozhrania je možné pristupovať k vnútorným registrom obsahujúce čas[170], dátum a aj pamäti RAM o veľkosti 56 Byte. Vývod č. 7 v puzdre DIP s názvom SQW/OUT je možné nastaviť na periodické generovanie frekvencie až do 1Hz. Obvod DS1307 k svojej činnosti vyžaduje len kryštál 32.768kHz a je zálohovaný typicky 3V batériou CR2031. Pre svoju správnu funkciu vyžaduje napájacie napätie 5V a v režime zálohy napätie 3V. Odber prúdu v režime na záložnú batériu je typicky 0,5µA a je ho možné minimalizovať vypnutím oscilátora hodín t.j. nastavením bitu CH[171] na 1. Údaje sú v tomto režime výrobcom zálohované po dobu minimálne 10 rokov.

Obrázok 168, Schéma zapojenia DS1307+



Tabuľka 17, Vnútorné registre DS1307

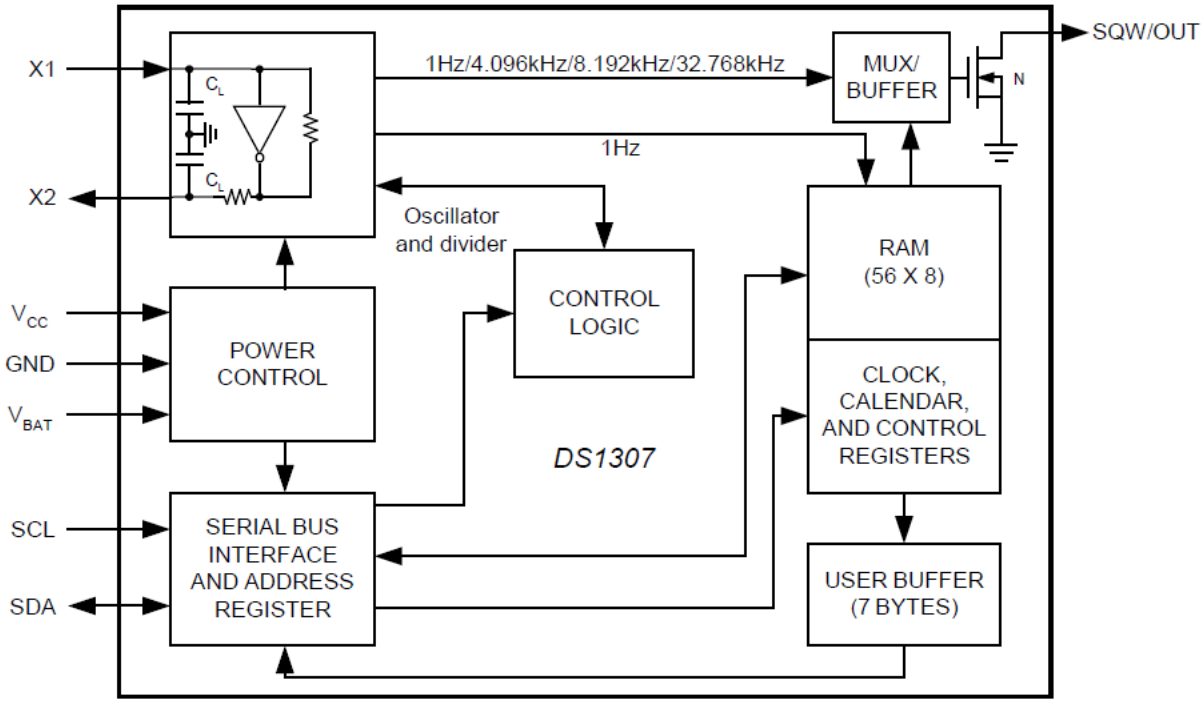
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

[170] Vnútorné registre obvodu DS1307 času a dátumu vyžadujú zápis a čítanie údajov tzv. BCD formáte. Zápis do vnútornej pamäti RAM je síce tiež v BCD formáte, ale principiálne je úplne jedno či je číslo zapísané v BCD, alebo v dekadickom tvare.

[171] Po prvom pripojení napájacieho napätia je nastavený bit CH=1, takže oscilátor hodín je zastavený!

Obrázok 169, Vnútorná schéma zapojenia DS1307

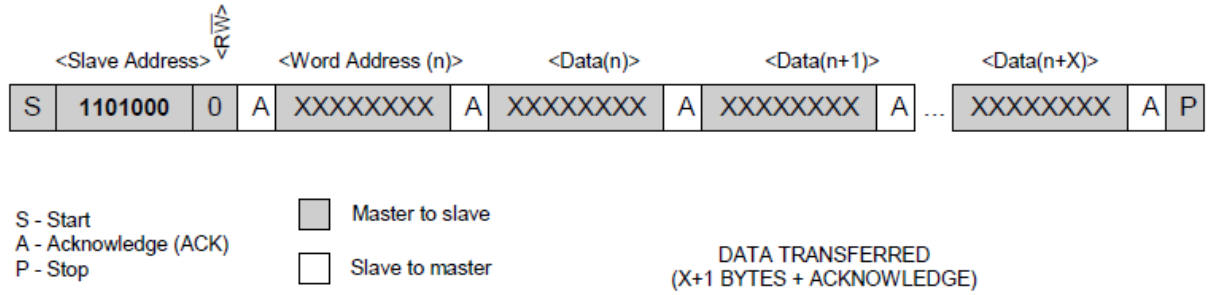


Obrázok 170, Prevod čísla na BCD/DEC formát

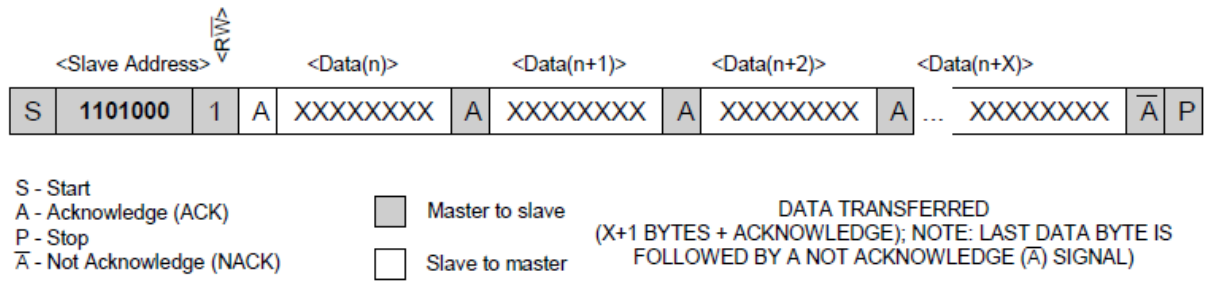
c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\BCD DEC Convert\BCD DEC convert.c

```
1   #include <reg51.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   unsigned char BCD (unsigned char Data )
6   {
7    return ((Data /10 *16 )+( Data %10 ));
8   }
9
10  unsigned char DEC (unsigned char Data )
11  {
12   return ((Data /16 *10 )+( Data %16 ));
13  }
```

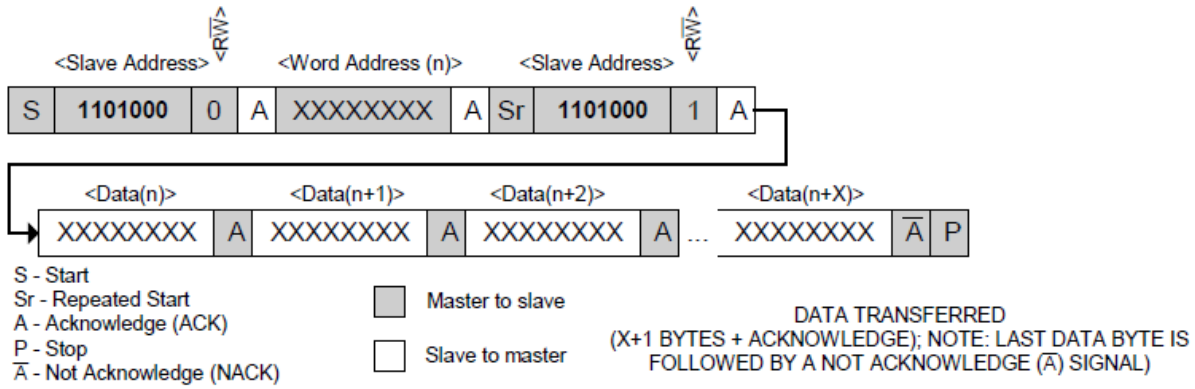
Obrázok 171, Zápis dát v režime Slave Receiver Mode (Blokový mód)



Obrázok 172, Čítanie dát v režime Slave Transmitter Mode (Blokový mód)



Obrázok 173, Kombinované čítanie dát Slave Receive and Transmit (po byte)



Obrázok 174, Komunikačné rutiny pre DS1307 pomocou I²C.

C:\Omega\data\Dropbox\Záloha\C51\DS1307\DS1307+.c

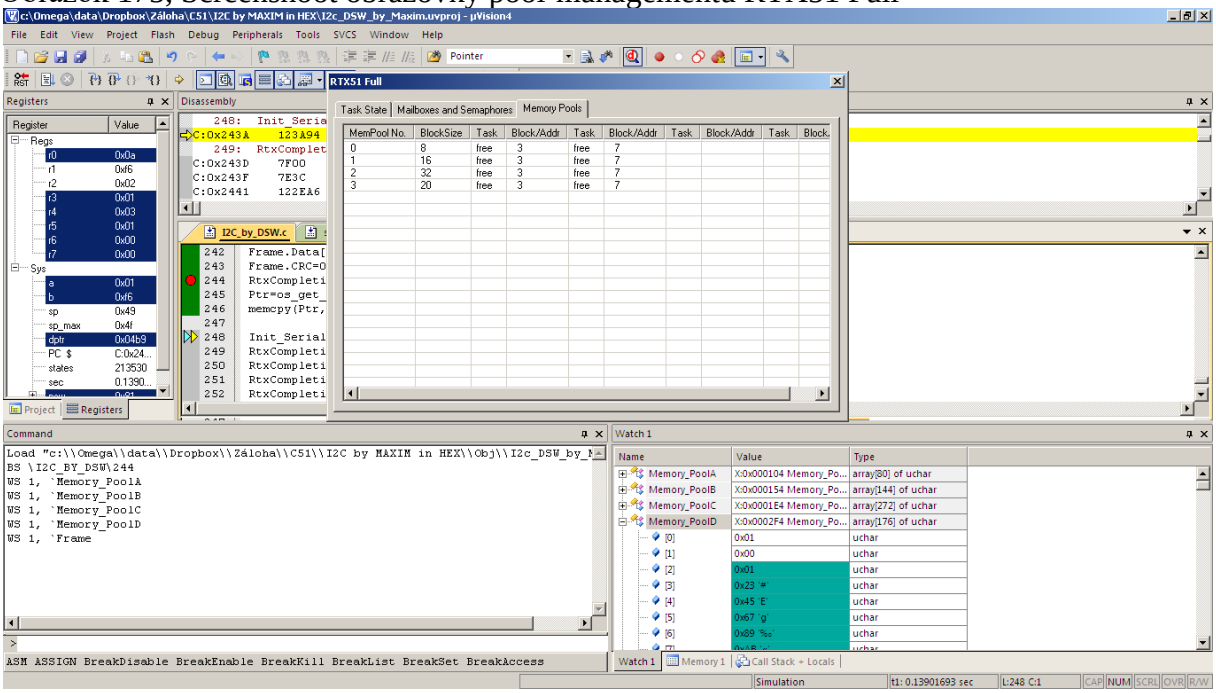
```
#include <rtx51.h>
#include <reg51.h>
#include <stdio.h>
#include <stdlib.h>
#include <dsw.h>
#include <adresy.inc>

#ifndef READ
  #define READ 1
#endif
#ifndef WRITE
  #define WRITE 0
#endif

#define EEPROM_WAIT 1

extern void Delay(unsigned char);

extern void I2C_Start(void);
extern void I2C_Stop(void);
extern unsigned char I2C_Read(unsigned char);
extern unsigned char I2C_SendByte(unsigned char);
extern unsigned char I2C_SendAddr(unsigned char);

#define Radix10 10
#define Radix16 16

unsigned char Get_Time(unsigned char Adresa, unsigned char Addr)
{
 register unsigned char LSB,time;
 I2C_SendAddr(Adresa+WRITE);
 I2C_SendByte(Addr);
//-------------------------------------------------
 I2C_SendAddr(Adresa+READ);
 LSB=I2C_Read(0);    //1 - Vyzaduje sa zo SLAVE-a potvrdenie ACK=0!!!
 I2C_Stop();         //0 - Vyzaduje sa z MASTER-a potvrdenie ACK=1!!!
//*************************************************
 time=LSB;           //Vraciam vysledok v BCD formate
 switch(Addr)
  {
   case Addr_Sec   :
   case Addr_Min   : LSB=(((time&0x70)>>4)*10)+(time&0x0F);  break;
   case Addr_Hod   : LSB=(((time&0x30)>>4)*10)+(time&0x0F);  break;
   case Addr_Day   : LSB=(time&0x07);  break;
   case Addr_Den   : LSB=(((time&0x30)>>4)*10)+(time&0x0F);  break;
   case Addr_Mes   : LSB=(((time&0x10)>>4)*10)+(time&0x0F);  break;
   case Addr_Rok   : LSB=(((time&0xF0)>>4)*10)+(time&0x0F);  break;
  }
 return(LSB);        //Vraciam vysledok v DEC formate
//*************************************************  }
}

void Set_Time(unsigned char Adresa, unsigned char Addr, unsigned char Data)
{
 register unsigned char time;
 time=Data;          //Zadavam vysledok v BCD formate
  switch(Addr)
  {
   case Addr_Sec   :
   case Addr_Min   : Data=(((time/10)&0x07)<<4)|(time%10); break;
   case Addr_Hod   : Data=(((time/10)&0x03)<<4)|(time%10); break;
   case Addr_Day   : Data=(time&0x07); break;
   case Addr_Den   : Data=(((time/10)&0x03)<<4)|(time%10); break;
   case Addr_Mes   : Data=(((time/10)&0x01)<<4)|(time%10); break;
   case Addr_Rok   : Data=(((time/10)&0x0F)<<4)|(time%10); break;
  }
 I2C_SendAddr(Adresa+WRITE);
 I2C_SendByte(Addr);
//-------------------------------------------------
 I2C_SendByte(Data);  //Vraciam vysledok v BCD formate
 I2C_Stop();
}

/*
```

Page1

15.29. Management pamäti v RTX51 Full

RTX51 disponuje výkonným nástrojom pre správu externej pamäti RAM v podobe niekoľkých mocných príkazov uvedených v kapitole 13.7. Nasledujúca časť programu demonštruje ako je možné využiť príkazy správy pamäti v praxi pri tvorbe aplikácií. Tieto príkazy sú vhodné pre použitie pri tvorbe aplikácií využívajúcej veľké množstvo častí pamäti tzv. buffer-ov pre vysielanie, alebo príjem údajov. Organizáciu pamäti definovanú v „pool-e“ si možno predstaviť ako jednoduchú štruktúru typu jednorozmerného poľa[172], kde prvok poľa predstavuje ďalšiu štruktúru s definovaným počtom údajového typu, kde na začiatku je header t.j. popisovač o veľkosti 2 byte informujúci o stave definovaného pool-block. Uvedené funkcie je možné použiť len pri procesoroch s pripojenou externou pamäťou dát XRAM. Vo vyššie uvedenom príklade je sú definované 3 jednoduché pamäťové pool oblasti a jedna zložitejšia štruktúra s názvom „Frame“. Keďže sa jedná o zložitejšiu štruktúru pre skopírovanie jej obsahu do definovaného pool-u je nutné použiť funkciu memcpy().

Obrázok 175, Screenshoot obrazovky pool-managementu RTX51 Full



[172] Definované ako pole štruktúr.

Obrázok 176, Správa pamäti v RTX Full cez pool-management

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\I2C_by_DSW.c

```
205  struct Ramec {long int Destination,Source; unsigned char Data[10]; int CRC;};
206  struct Ramec Frame={0x1111,0x2222,1,2,3,4,5,6,7,8,9,10,0x1234};
207  #define Number_Blocks 8
208  #define Block_SizeA 8                              //Velkost suvisleho bloku dat 08 byte
209  #define Block_SizeB 16                             //Velkost suvisleho bloku dat 16 byte
210  #define Block_SizeC 32                             //Velkost suvisleho bloku dat 32 byte
211  #define Block_SizeD sizeof(Frame)                  //Velkost datovej struktury Frame
212  #define Pool_SizeA (Number_Blocks*(2+Block_SizeA)) //Definujem Number_Blocks blokov
213  #define Pool_SizeB (Number_Blocks*(2+Block_SizeB)) //Definujem Number_Blocks blokov
214  #define Pool_SizeC (Number_Blocks*(2+Block_SizeC)) //Definujem Number_Blocks blokov
215  #define Pool_SizeD (Number_Blocks*(2+Block_SizeD)) //Definujem Number_Blocks blokov
216
217  xdata unsigned char Memory_PoolA[Pool_SizeA];
218  xdata unsigned char Memory_PoolB[Pool_SizeB];
219  xdata unsigned char Memory_PoolC[Pool_SizeC];
220  xdata unsigned char Memory_PoolD[Pool_SizeD];
221  xdata int *Pointers[Number_Blocks];                          //Pointre jednotlivych
     blokov ...
222
223  void Init(void) _task_ INIT _priority_ 0
224  {
225   signed char RtxCompletion;
226   unsigned char i;
227   int *Ptr;
228   Watchdog;
229   i=0x00;
230   RtxCompletion=os_create_pool(Block_SizeA,Memory_PoolA,Pool_SizeA);    //Definujem "pool" oblast
     kde mam data ...
231   Ptr=os_get_block(Block_SizeA);                                       //Ziaskam pointer na
     premennu v alokovanom pool-e
232   *Ptr=0x1234;                                                         //Naplnim premennu v
     alokovanej pamati ....
233   Pointers[i++]=Ptr;                                                   //Ulozim do pola pointrov
     ...
234   while((Ptr=os_get_block(Block_SizeA))!=0)                            //Ziskavam pointre
     jednotlivych blokov ...
235    {
236     Pointers[i++]=Ptr;
237     *Ptr=0xEDDE;
238    }
239    RtxCompletion=os_create_pool(Block_SizeB,Memory_PoolB,Pool_SizeB);    //Definujem "pool"
     oblast kde mam data ...
240   Ptr=os_get_block(Block_SizeB);                                       //Ziaskam pointer na
     premennu v alokovanom pool-e
241   *Ptr=0x5678;                                                         //Naplnim premennu v
     alokovanej pamati ....
242   RtxCompletion=os_create_pool(Block_SizeC,Memory_PoolC,Pool_SizeC);    //Definujem "pool" oblast
     kde mam data ...
243   Ptr=os_get_block(Block_SizeC);                                       //Ziaskam pointer na
     premennu v alokovanom pool-e
244   *Ptr=0xABCD;                                                         //Naplnim premennu v
     alokovanej pamati ....
245   os_free_block(Block_SizeC,Ptr);                                      //Uvolnim pamat premennej
     Ptr v pool-e ...
246
247   Frame.Source=0x89ABCDEF;
248   Frame.Destination=0x01234567;
249   Frame.Data[0]=0xED;
250   Frame.CRC=0xABCD;
251   RtxCompletion=os_create_pool(Block_SizeD,Memory_PoolD,Pool_SizeD);
252   Ptr=os_get_block(Block_SizeD);
253   memcpy(Ptr,&Frame,Block_SizeD);
254   i=0x00;
255   while((os_free_block(Block_SizeA,Pointers[i++]))==0);                 //Uvolnujem jednotlive
     bloky ...
256   Init_Serial();
257   RtxCompletion=os_set_slice(xtal(XTAL));
258   RtxCompletion=os_create_task(THERMO);
259   RtxCompletion=os_create_task(ADC);
260   RtxCompletion=os_delete_task(INIT);
261  }
```

Page 1

15.30. Management pamäti – odovzdávanie premenných a štruktúr mailbox-om

C:\Omega\data\Dropbox\Záloha\C51\Odovzdávanie štruktúry mailboxom s displayom OK\Odovzdávanie štr

```
#include <INC\reg552.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <rtx51.h>

extern void Init_Serial (void);
extern int ADC_Conversion (unsigned char Channel);

extern unsigned char Riadok1,Riadok2;
extern void LCD_Init (void);
extern void LCD_Write_String (char*);
extern void LCD_Goto (unsigned char);

#ifdef SPEED
 #include <AD\tabulka.inc>
#endif

//-------------------------------------------------------------
#define WaitForEver 0xFF
//-------------------------------------------------------------
#define SYSTEM  0
#define CLOCK   1
#define DISPLAY 2
#define ADC     3
//-------------------------------------------------------------
#define MBXS    0x00                                    //Cislomailbox-u
#define MBXC    0x01                                    //Cislomailbox-u
//-------------------------------------------------------------
extern xdata unsigned char Buffer[BufferSize+1];
//-------------------------------------------------------------
enum Den {Mon=1,Tue=2,Wed=3,Thu=4,Fri=5,Sat=6,Sun=7};
enum Mes {Jan=1,Feb=2,Mar=3,Apr=4,May=5,Jun=6,Jul=7,Aug=8,Sep=9,Oct=10,Nov=11,Dec=12};
//-------------------------------------------------------------
typedef xdata struct                                    //XDATA je velmi dolezite, lebo IDATA, DATA nefunguje OK ...
{
 unsigned char Hod,Min,Sek,Den,Mes,Rok;
 int Voltage;
 float Teplota,V;
}Time;
//-------------------------------------------------------------
#define NumMemBlocks  100                               //Definujem n pamatovych blokov ...
#define SizeMemBlock  sizeof(Time)
#define SizeMemPool   NumMemBlocks*(2+SizeMemBlock)
//-------------------------------------------------------------
xdata unsigned char Pool[SizeMemPool];                  //POOL je niekedy lepsie definovat na pevnej adrese ale uVision robi problemy pri simulacii ...
xdata Time Array = {6,15,0,28,4,2013-1900,0x3FF,+23.0,1.23};
xdata Time *ReceivedMessage;                            //Adresa struktury prijatej spravy ...
xdata unsigned int *MessageToSend;                      //Adresa struktury vysielanej spravy ...
//-------------------------------------------------------------

void Clock(void) _task_ CLOCK _priority_ 1
{
 signed char RtxCompletion;
 unsigned int *Ptr;                                                 //Ukazovatel v pamati na odosielanu strukturu ...
 RtxCompletion=os_create_pool(SizeMemBlock,Pool,SizeMemPool);       //Vytvorim pool v pamati XRAM pre datove struktury ...
 while(1)
  {
   switch(os_wait(K_IVL+K_SIG+K_MBX+MBXC,100,NULL))                 //Cakam na rozne udalosti ...
    {
     case SIG_EVENT :
     case TMO_EVENT : if(++Array.Sek==60)
                       {
                        Array.Sek=0;
                        if(++Array.Min==60)
                         {
                          Array.Min=0;
                          if(++Array.Hod==24) Array.Hod=0;
                         }
                       }
```

C:\Omega\data\Dropbox\Záloha\C51\Odovzdávanie štruktúry mailboxom s displayom OK\Odovzdávanie štr

```
71                          os_send_signal (ADC );
72                          if ((MessageToSend =os_get_block (SizeMemBlock ))== NULL )
73                           {
74                            os_delete_task (CLOCK); /*Error handling*/
                                                    //Rezervujem dynamicku memory ....
75                           };
76                          Ptr =memcpy (MessageToSend ,&Array ,sizeof (Array));
                                                  //Skopirujem strukturu Array do struktury MesageToSend
    a necham incicializovany ukazovatel ...
77                          RtxCompletion =os_send_message (MBXS ,(unsigned int )MessageToSend ,WaitForEver
    );             //Posielam len adresu dat ulozenych v strukture mailboxom ...
78                          if ((RtxCompletion =os_free_block (SizeMemBlock ,MessageToSend ))!= NULL )
79                           {
80                            os_delete_task (CLOCK); /*Error handling*/
                                                    //Uvolnim dynamicku memory ....
81                           }
82                          break ;
83        case MSG_EVENT  : break ;
84       }
85     }
86   }
87
88   void Display (void ) _task_ DISPLAY _priority_ 0
89   {
90    signed char RtxCompletion ;
91    LCD_Init ();
92    while (1)
93     {
94      switch (RtxCompletion =os_wait (K_SIG +K_TMO +K_INT +K_MBX +MBXS ,WaitForEver ,(unsigned int xdata *)&
    ReceivedMessage ))
95       {
96        case TMO_EVENT    :
97        case INT_EVENT    : //RtxCompletion=0;
98        case SIG_EVENT    :
99        case MSG_EVENT    : sprintf (Buffer ,"%02bd:%02bd:%02bd,V=%2.2fV\r\n" ,ReceivedMessage ->Hod ,
    ReceivedMessage ->Min ,ReceivedMessage ->Sek ,ReceivedMessage ->V);
100                           printf ("%s" ,Buffer );
101                           LCD_Goto (Riadok2 );
102                           LCD_Write_String (Buffer );
103                           LCD_Goto (Riadok1 );
104                           strcpy (Buffer ,"DSWSoftware\r\n" );
105                           LCD_Write_String (Buffer );
106                           break ;
107       }
108     }
109  }
110
111  void Adc (void ) _task_ ADC _priority_ 0
112  {
113   signed char RtxCompletion ;
114   TCON |= 0x01 +0x04 ;            //Spustim /INT0 a /INT1 hranou a nie urovnou !!!
115   os_attach_interrupt (0);     //Pripojim sa na prerusenie /INT0
116   os_attach_interrupt (2);     //Pripojim sa na prerusenie /INT1
117   while (1)
118    {
119     switch (RtxCompletion =os_wait (K_TMO +K_SIG +K_INT ,WaitForEver ,NULL ))
120      {
121       case INT_EVENT  :
122       case SIG_EVENT  :
123       case TMO_EVENT  : Array .Voltage =ADC_Conversion (0);
124                         #ifdef  SPEED
125                          Array .V=Tab [Array .Voltage ];
126                         #else
127                          Array .V=( 5.0 /1024.0 )*Array .Voltage ; break ; //Prevediem AD hodnotu na int
    a float ...
128                         #endif
129       }
130     }
131  }
132
133  void System (void ) _task_ SYSTEM _priority_ 0
134  {
135   signed char RtxCompletion ;
136   RtxCompletion =os_set_slice (15360 );        // 15360=10ms os_wait, 1536=1ms os_wait
```

C:\Omega\data\Dropbox\Záloha\C51\Odovzdávanie štruktúry mailboxom s displayom OK\Odovzdávanie štr

```
137     RtxCompletion=os_create_task(ADC);
138     RtxCompletion=os_create_task(CLOCK);
139     RtxCompletion=os_create_task(DISPLAY);
140     Init_Serial();
141     while(1)
142      {
143       RtxCompletion=os_wait(K_TMO,50,NULL);
144       os_delete_task(SYSTEM);
145      }
146   }
147
148   void main(void)
149   {
150    signed char RtxCompletion;
151    RtxCompletion=os_start_system(SYSTEM);
152   }
```

V praxi je niekedy potrebné inej úlohe odovzdať[173] nielen jednoduchú premennú, ale aj zložitejšiu štruktúru. Vyššie uvedený program ukazuje možnosť ako je možno odovzdať štruktúru a jej obsah ďalšej úlohe. Pomocou zápisu v jazyku C môžeme prijať pomocou mailbox-u aj pomerne zložitú štruktúru[174]. Štruktúru je možné chápať v podstate ako premennú v Pascal-e známu ako typ pole t.j. „array“ ktorá je alokovaná v priestore pamäti externej pamäti označovanej ako XDATA.

Alokovanie pamäti pre štruktúru je možné pomocou príkazu os_create_pool() t.j. RtxCompletion=os_create_pool(SizeMemBlock,Pool,SizeMemPool), kde **SizeMemBlock** predstavuje veľkosť potrebnej pamäti štruktúry, ukazovateľ **Pool** už predstavuje samotnú štruktúru uloženú v externej pamäti ktorá môže byť už aj ako **n** počet blokov a **SizeMemPool** udáva celkovú výslednú veľkosť samotnej štruktúry.

switch(RtxCompletion=os_wait(K_SIG+K_TMO+K_INT+K_MBX+MBXS,WaitForEver,(unsigned int xdata*)&ReceivedMessage)); Výsledky odovzdané štruktúrou môžeme potom použiť v príkaze printf(), alebo sprintf() čo je vyjadrené v nasledujúcom zápise:

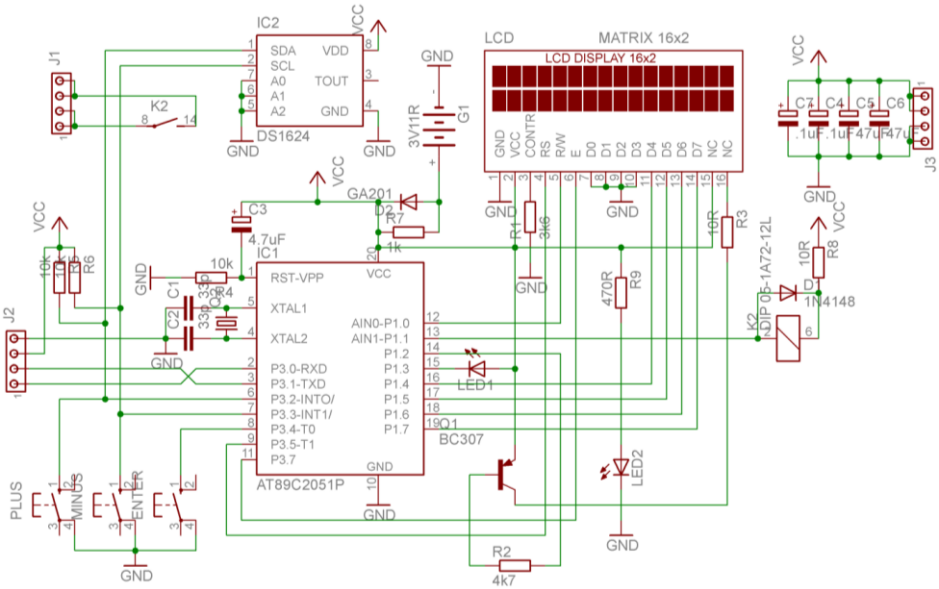
sprintf(Buffer,"%02bd:%02bd:%02bd,V=%2.2fV\r\n",ReceivedMessage->Hod,ReceivedMessage->Min,ReceivedMessage->Sek,ReceivedMessage->V);

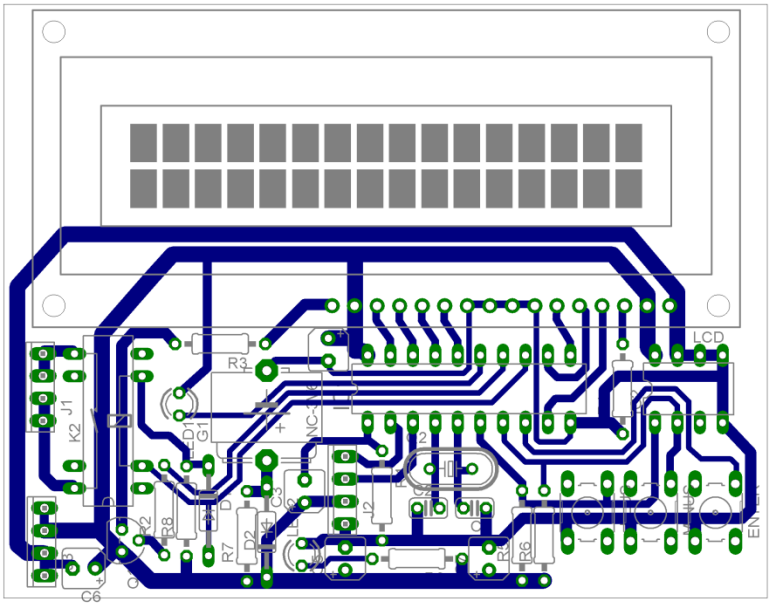
[173] RTX51 umožňuje odovzdať pomocou premennej mailbox-u maximálne premennú **int**, čo v našom prípade môže byť nielen premenná, ale aj ukazovateľ ktorý umožní odovzdať aj adresy zložitejších štruktúr a objektov.

[174] V µVision4 9.50 je v okne WATCH1 náhodne nesprávne zobrazovaný obsah samotnej štruktúry. V tomto prípade obsahuje štruktúra pri simulácii menej prvkov štruktúry ako je definované a ich obsah je nesprávny. V µVision3 je štruktúra vždy správne zobrazovaná so správnym počtom prvkov štruktúry. Chyba je spôsobená pravdepodobne zlou počiatočnou inicializáciou samotných vlastností objektov programu µVision4. Vo verzii 9.52.0.0 µVision4 sa chyba už nevyskytuje.

15.31. Ovládanie radiča HD44780 LCD display-a s RTX51 Tiny.

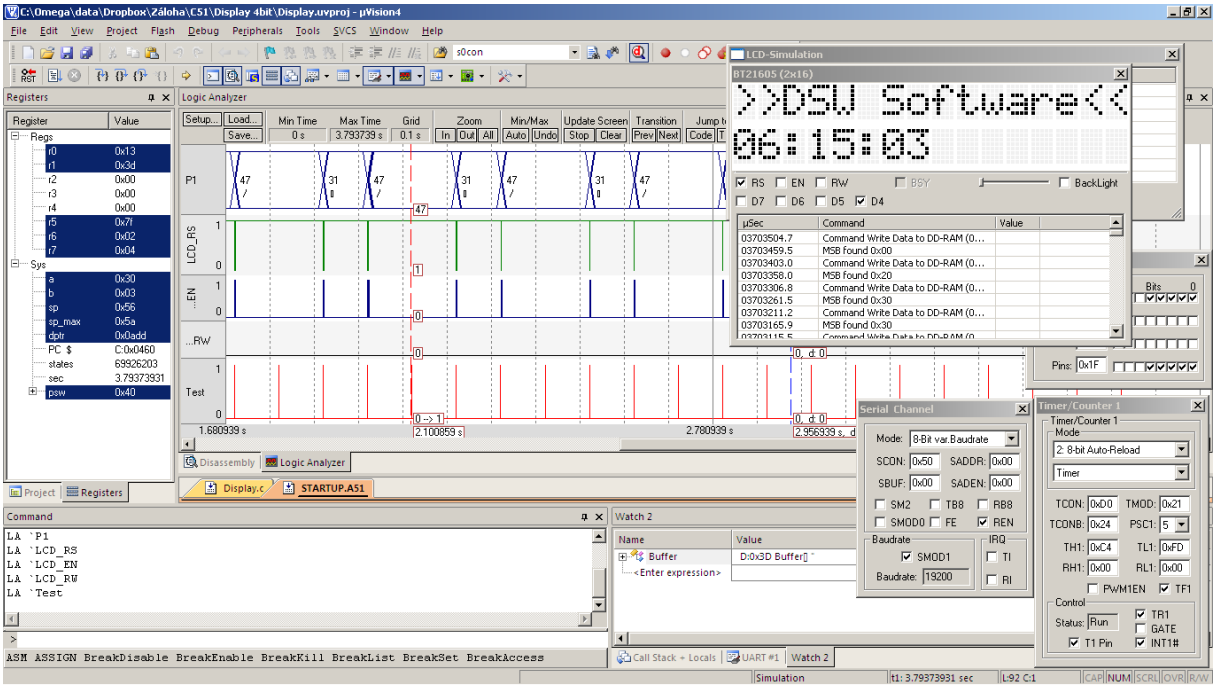
Radiče HD44780 od svojho začiatku sa vzhľadom na svoju jednoduchosť ovládania ihneď stali veľmi obľúbenými prvkami všetkých elektronických zariadení. Programátor môže ovládať[175] radič display-a LCD jednotky z pomocou 4, alebo 8 bitovej zbernice a 3 základných riadiacich signálov RS, RW a E.





[175] Ovládanie display-a cez AGSI ovládače v prostredí µVision5 je možné na všetkých portoch mikroprocesora pomocou 4 bitovej zbernice, pričom musia byť jednotlivé komunikačné vodiče deklarované ako premenná typu bit a nie byte! Napr. správna definícia **sbit LCD_0=P1^0;**. Najčastejšie sa robia chyby ak sa priamo posielajú dáta pomocou príkazov priameho výstupu na port P1=(0x33&0x0F);. Vtedy sa nezobrazujú údaje zapísané priamo do portu P1 na display cez AGSI interface. V súbore **display.lcd** musí byť interným premenným AGSI diplay-a **Dx_NAME** priradená priamo premenná napr. **D4_NAME=LCD_4** t.j. **sbit LCD_4=P4^4** ktorá je použitá v programe a nie priamo port P4, lebo v tomto prípade LCD display nedokáže správne zobrazovať údaje.

Obrázok 177, Simulácia LCD display-a v prostredí Keil µVision4



Na simuláciu LCD sme použili AGSI[176] ovládače nachádzajúce sa voľne dostupné na internete. Jediným obmedzením je možnosť simulácie len jedného LCD radiča pomocou 4 bitovej komunikácie. Navyše všetky dátové vodiče musia byť v programe deklarované ako bitové premenné s veľkosťou jedného bitu tzv. sbit. V prípade použitia rýchlejšej varianty AT89LP4052 namiesto AT89C4051 je nutné ešte porty nastaviť ako obojsmerné[177]. Po resete sú všetky nastavené ako vstupné. Zároveň sa musí v RTX51 Tiny nastaviť konštanta INT_CLOCK na 12-násobne vyššiu hodnotu, pretože AT89LP4052 je procesor tzv. single-cycle, rovnako aj parametre sériového komunikačného rozhrania RS232.

Vyššie uvedená aplikácia prijíma údaje zo sériového rozhrania RS232[178] a prijatý reťazec procesor zobrazí na display-i. Príjem údajov je zabezpečený výskytom tzv. komunikačného tokenu "^" za ktorým nasleduje číslo zobrazovaného riadku (1,2...n) display-a a nakoniec samotný zobrazovaný reťazec ktorý je ukončený znakom "**\n**". V prípade, že bude ukončovací znak chýbať, zobrazí sa maximálne len 16 znakov na jeden riadok LCD. Ostatné prijaté znaky sú ignorované.

[176] http://www.c51.de/c51.de/Dateien/uVision2DLLs.php?Spr=EN
[177] P1M0=P3M0=P1M1=P3M1=0x00;
[178] Signály RXD a TXD na porte P3.0 a P3.1 procesora AT89LP4052.

C:\Omega\Data\Dropbox\Záloha\C51\Display 4bit\Display.c

```
#include <stdio.h>        /* standard I/O .h-file                */
#include <ctype.h>        /* character functions                 */
#include <string.h>       /* string and memory functions         */
#include <stdlib.h>
#include <at89LP4052.h>
#include <rtx51tny.h>
#include <dsw.h>

#include <adresy.inc>

#define SYSTEM  0
#define CLOCK   1
#define SERIAL  2
#define ROTATE  3

#define Port    P1
#define LCD     0xFF    //Inicializacia portu ....
#ifdef x12
 #define  Dlzka   255   //pre 18.432MHz AT89LP4052
#else
 #define  Dlzka   60    //pre 18.432MHz
#endif
#define Riadok1 0x00
#define Riadok2 0x40
sbit  LCD_RW  = P1^0;
sbit  LCD_EN  = P3^7; //Regulator 4.C ma prehodeny RS a E !!!
sbit  LCD_RS  = P1^1;

/*
sbit  LCD_RW  = P3^4;
sbit  LCD_EN  = P3^7; //Regulator 4.C ma prehodeny RS a E !!!
sbit  LCD_RS  = P3^5;
*/
/*
sbit  LCD_RW  = P3^4;
sbit  LCD_EN  = P3^5;
sbit  LCD_RS  = P3^7;
*/
extern void Delay(unsigned char);
extern void Init_Serial(void);

sbit LCD7 = P1^7;
sbit LCD6 = P1^6;
sbit LCD5 = P1^5;
sbit LCD4 = P1^4;

unsigned char Buffer[16+1];

void LCD_Clock(void)
{
 LCD_EN=1;
 #ifdef x12          //Cakaj cca. 50us
  Delay(Dlzka);
  Delay(Dlzka);
  Delay(Dlzka);
 #else               //Cakaj cca. 50us
  Delay(Dlzka);
 #endif
 LCD_EN=0;
}

void LCD_Send(unsigned char Data)
{
 LCD7=(Data&0x80)?1:0;    //Posielam MSB
 LCD6=(Data&0x40)?1:0;
 LCD5=(Data&0x20)?1:0;
 LCD4=(Data&0x10)?1:0;
 LCD_Clock();
 LCD7=(Data&0x08)?1:0;    //Posielam LSB
 LCD6=(Data&0x04)?1:0;
 LCD5=(Data&0x02)?1:0;
 LCD4=(Data&0x01)?1:0;
 LCD_Clock();
}
```

Page 1

C:\Omega\Data\Dropbox\Záloha\C51\Display 4bit\Display.c

```
 75
 76  void LCD_Data(unsigned char Data)
 77  {
 78   LCD_RS=1;              //Pretoze odosielam data !!!
 79   LCD_EN=0;
 80   LCD_RW=0;
 81   LCD_Send(Data);
 82  }
 83
 84  void LCD_Cmd(unsigned char Data)
 85  {
 86   LCD_RS=0;              //Pretoze odosielam prikaz !!!
 87   LCD_EN=0;
 88   LCD_RW=0;
 89   LCD_Send(Data);
 90  }
 91
 92  void LCD_ClrScr(void)
 93  {
 94   LCD_Cmd(0x01);
 95   os_wait(K_TMO,2,NULL);
 96  }
 97
 98  void LCD_Goto(unsigned char Ptr)
 99  {
100   LCD_Cmd(0x80+Ptr);
101   Delay(Dlzka);
102  }
103
104  void LCD_Home(void)
105  {
106   LCD_Cmd(0x02);
107   os_wait(K_TMO,2,NULL);
108  }
109
110  void LCD_Init(void)
111  {
112   LCD_RS=0;
113   LCD_RW=0;
114   LCD_EN=0;
115   #ifdef x12
116    P1M0=P3M0=P1M1=P3M1=0x00;
117   #endif
118   Port=LCD;
119   os_wait(K_TMO,20,NULL);  //Cakaj cca. 20ms
120   LCD_Cmd(0x33);
121   os_wait(K_TMO,5,NULL);
122   LCD_Cmd(0x32);
123   os_wait(K_TMO,5,NULL);
124   LCD_Cmd(0x20+0x08);
125   LCD_Cmd(0x10+0x00);
126   LCD_Cmd(0x08+0x04);
127   LCD_Cmd(0x04+0x02);
128   LCD_ClrScr();
129   LCD_Home();
130  }
131
132  void LCD_Write_String(char *String)
133  {
134   unsigned char i;
135   for(i=0x00; i<LCD_Size; i++)
136    {
137     switch(*String)
138      {
139       case '\n'  :
140       case '\r'  :
141       case '\0'  : *String=' '; break;
142      }
143      LCD_Data(String[i]);
144    }
145   memset(String,' ',LCD_Size);
146  }
147
148  #ifdef SIMUL
```

Page 2

C:\Omega\Data\Dropbox\Záloha\C51\Display 4bit\Display.c

```
149
150   struct Hodiny
151   {
152    unsigned char Hod,Min,Sek,Des;
153   };
154
155   struct Hodiny Cas = {6,15,0,0};
156
157   void Clock(void) _task_ CLOCK
158   {
159    while(1)
160     {
161      os_wait(K_IVL,100,NULL);
162      os_send_signal(SYSTEM);
163      if(++Cas.Des==10)
164       {
165        Cas.Des=0;
166        if(++Cas.Sek==60)
167         {
168          Cas.Sek=0;
169          if(++Cas.Min==60)
170           {
171            Cas.Min=0;
172            if(++Cas.Hod==24)
173             {
174              Cas.Hod=0;
175             }
176           }
177         }
178       }
179     }
180   }
181
182   #endif
183
184   #ifdef INT232
185
186   void IntRS232(void) interrupt 4 //using 3
187   {
188    switch(SCON&0x03)
189     {
190      case 0x01  : RI=0; isr_send_signal(SERIAL); break;
191      case 0x02  : TI=0; break;
192      case 0x03  : TI=RI=0; break;
193     }
194   }
195
196   unsigned char Addr;
197
198   void Serial(void) _task_ SERIAL   //Otestovane a ide OK
199   {
200    bit RECEIVE;
201    register unsigned char i;
202    ES=1;
203    while(1)
204     {
205      os_wait(K_SIG,0xFF,NULL);
206      if(Token==SBUF)
207       {
208        os_wait(K_SIG,0xFF,NULL);
209        switch(Addr=SBUF)
210         {
211          case Adresa1 :
212          case Adresa2 :
213          case Adresa3 :
214          case Adresa4 : RECEIVE=0; i=0x00;              //Prijal som platnu adresu ...
215                         while(!RECEIVE)
216                          {
217                           os_wait(K_SIG,0xFF,NULL);
218                           switch(Buffer[i++]=SBUF)      //Citam SERIAL az pokial nenajdem koniec
    riadku, ako gets() ........
219                            {
220                             case  '\r'  :
221                             case  '\n'  :
```

Page 3

C:\Omega\Data\Dropbox\Záloha\C51\Display 4bit\Display.c

```
222                             case  '\0'  : RECEIVE=1; os_send_signal(SYSTEM);  break;
223                            }
224                           }
225                         break;
226          default      : break;
227         }
228       }
229     }
230   }
231
232   #else
233
234   unsigned char Readkey(void)   //Otestovane a ide OK
235   {
236    while(!RI) os_switch_task();
237    RI=0;
238    return(SBUF);
239   }
240
241   unsigned char Addr;
242
243   void Serial(void) _task_ SERIAL
244   {
245    bit RECEIVE;
246    register unsigned char i;
247    ES=0;
248    while(1)
249     {
250      if(Token==Readkey())
251       {
252        switch(Addr=Readkey())
253         {
254          case Adresa1 :
255          case Adresa2 :
256          case Adresa3 :
257          case Adresa4 : RECEIVE=0; i=0x00;              //Prijal som platnu adresu ...
258                         while(!RECEIVE)
259                          {
260                           switch(Buffer[i++]=Readkey()) //Citam SERIAL az pokial nenajdem koniec
    riadku, ako gets() ........
261                            {
262                             case  '\r'  :
263                             case  '\n'  :
264                             case  '\0'  : RECEIVE=1; os_send_signal(SYSTEM);  break;
265                            }
266                           }
267                         break;
268          default      : break;
269         }
270       }
271     }
272   }
273
274   #endif
275
276   #ifdef ROTATESTRING
277
278   xdata unsigned char Buf[100];
279
280   void Rotate(void) _task_ ROTATE
281   {
282    register unsigned char Znak,Length;
283    while(1)
284     {
285      os_wait(K_TMO,250,NULL);
286      Length=strlen(Buf)-1;
287      Znak=Buf[0];
288      memmove(Buf,Buf+1,Length);
289      Buf[Length]=Znak;
290     }
291   }
292
293   #endif
294
```

Page 4

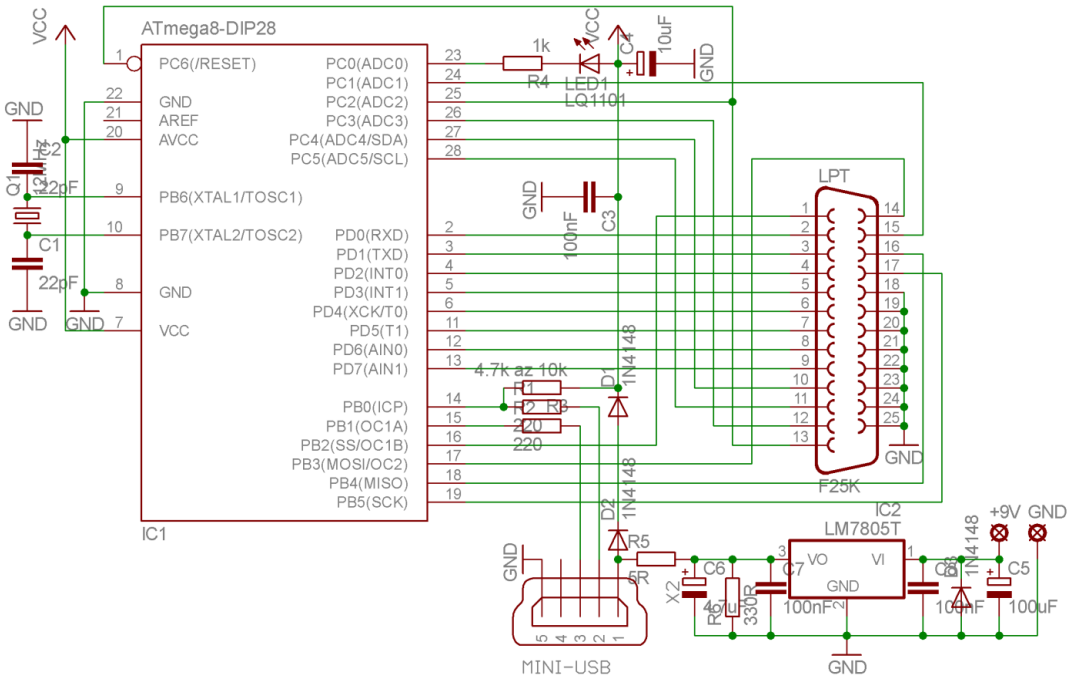
C:\Omega\Data\Dropbox\Záloha\C51\Display 4bit\Display.c

```
295   code unsigned char Test[]="!!!>>>TEST<<<!!!";
296
297   void System(void) _task_ SYSTEM
298   {
299    code unsigned char Logo[]=">>DSW Software<<";
300    #ifdef SIMUL
301     #ifdef DATE
302      code unsigned char *SystemTime[]={__TIME__,__DATE__,__DATE2__,__C51__};
303      sscanf(SystemTime[0],"%02bd:%02bd:%02bd",&Cas.Hod,&Cas.Min,&Cas.Sek);
304     #endif
305    #endif
306    Init_Serial();
307    LCD_Init();
308    #ifdef SIMUL
309     os_create_task(CLOCK);
310    #endif
311    os_create_task(SERIAL);
312    #ifdef ROTATESTRING
313     os_create_task(ROTATE);
314    sprintf(Buf,"%s","Diversant Software (c) 2013, Ing. Eduard Jadron, Stefana Nahalku 18/65, 036
      01 Martin, Slovakia, 0123456789ABCDEF, ");
315    #endif
316    #ifndef SIMUL
317     LCD_Goto(Riadok1);
318     sprintf(Buffer,"%s","Copyright(c)2013");
319     LCD_Write_String(Buffer);
320     LCD_Goto(Riadok2);
321     sprintf(Buffer,"%s",Logo);
322     LCD_Write_String(Buffer);
323    #endif
324    while(1)
325     {
326      switch(os_wait(K_SIG+K_TMO,0xFF,NULL))
327       {
328        case TMO_EVENT : sprintf(Buffer,"%s","Serial Error !!!");    //Neprisli 255ms ziadne udaje
      zo seriovej linky ..
329                         LCD_Goto(Riadok1);
330                         LCD_Write_String(Buffer);
331                         break;
332        case SIG_EVENT :
333                         #ifdef SIMUL
334                         if(Addr++>2) Addr=1;
335                         #endif
336                         switch(Addr)
337                          {
338                           case  Adresa1 : LCD_Goto(Riadok1);
339                                           #ifdef SIMUL
340                                           sprintf(Buffer,"%s",Logo);
341                                           #ifdef ROTATESTRING
342                                           memcpy(Buffer,Buf,sizeof(Buffer)-1);
343                                           #endif
344                                           #endif
345                                           LCD_Write_String(Buffer);
346                                           break;
347                           case  Adresa2 : LCD_Goto(Riadok2);
348                                           #ifdef SIMUL
349
      sprintf(Buffer,"%02bd:%02bd:%02bd",Cas.Hod,Cas.Min,Cas.Sek);
350                                           #endif
351                                           LCD_Write_String(Buffer);
352                                           break;
353                           case  Adresa3 : break;
354                           case  Adresa4 : break;
355                          }
356       }
357     }
358   }
```

15.32. Prevodník portu USB na LPT (USB2LPT)

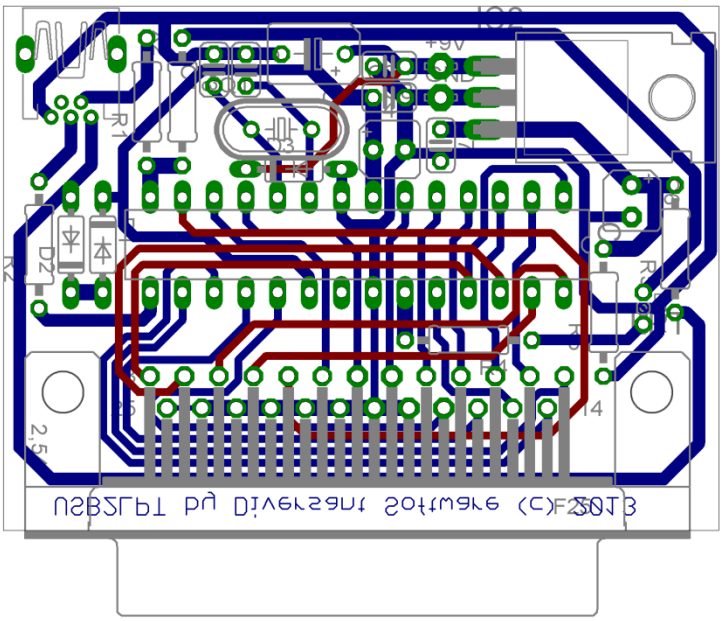
Dnešné moderné počítače alebo notebooky už vo svojej výbave nemajú LPT port známy pod názvom CENTRONICS, alebo IEEE1284. Donedávna slúžil na pripojenie tlačiarní, ale príchodom USB bol vytlačený ako neperspektívne rozhranie. Niektoré aplikácie, ale hlavne hardware-ové zariadenia ho dodnes používajú. Príkladom sú staršie programátory od firmy ELNEC. Na trhu nie sú špecializované prevodníky z USB na LPT a tak je potrebné vytvoriť rozhranie USB2LPT aby bolo možné používať tieto zariadenia aj naďalej. Na trhu síce existujú hotové prevodníky z USB na LPT špeciálne určené pre tlačiarne, ale tieto sú absolútne nevhodné, pretože umožňujú komunikovať len s ovládačmi tlačiarne a operačného systému. Nemajú priradenú komunikačnú adresu portu LPT (378H, 278H, 3BCH) a príslušné IRQx prerušenie. I keď je USB prevádzkované s prenosovou rýchlosťou 480Mbps zmena z režimu čítanie/zápis trvá min. 125µs oproti 1µs v prípade LPT portu. Z tohto dôvodu nie je jednoduché urobiť jednoduchý prevodník ktorý by cez USB obsluhoval LPT port. Jednoducho sa dá podobné zariadenie vyrobiť s pomocou mikroprocesorom ATmega8 a niekoľkých súčiastok. Software vrátane zdrojových programov a hardware je neustále dostupné na stránkach pod názvom USB2LPT.[179]

Obrázok 178, Schéma zapojenia USB2LPT portu



[179] http://www-user.tu-chemnitz.de/~heha/bastelecke/Rund%20um%20den%20PC/USB2LPT/index.html.en

Obrázok 179, Doska plošného spoja USB2LPT portu



15.33. PWM kanál s AT89C4051

Mikroprocesor AT89C4051 neobsahuje PWM kanál ktorý je možné využiť napríklad na ovládanie jasu LED diódy, prípadne na budenie tranzistora ktorý ovláda jednosmerný obvod. Až u novších procesoroch si u firmy ATMEL uvedomili, skutočný význam PWM kanála a od AT89LP4052[180] je už zabudovaná dvojica PWM kanálov. Obrázok 180 upresňuje princíp fungovania PWM kanála. Základ PWM generátora je vytvorený časovačom T0 ktorý je nastavený do 8 bitového „autoreload“ režimu s periodickým vyvolávaním prerušenia po pretečení T0. Premenné t_{ON} a t_{OFF} obsahujú dĺžku intervalov zopnutia tranzistorov LED display-a. Tento spôsob maskovania je ideálne použiteľný na generovanie PWM signálu pre 7-segmentový display. Premenná „Bright“ v programe predstavuje hodnotu PWM veličiny, ktorou sme schopní regulovať jas LED display-a.

[180] Pre zaistenie kompatibility mikroprocesora AT89LP4052 s AT89C4051 je nevyhnutné nastaviť I/O konfiguračné registre P1M0=P3M0=P1M1=P3M1=0x00. Nastavenie je nevyhnutné kvôli výstupnému režimu portu. V prípade, že používame sériové komunikačné rozhranie RS232 je nutné si uvedomiť, že sa jedná o single-cycle procesor ktorý je cca. 12x rýchlejší a tiež musíme nastaviť iné hodnoty generátorov prenosových rýchlostí v TH1=TL1=(0x100-(((long int)XTAL/16*1)/BAUD));

Obrázok 180, Generovanie PWM signálu u AT89C4051 pomocou časovača T0

```
#include <reg51.h>
#include <stdio.h>
#include <stdlib.h>
#include <dsw.h>
#include <adresy.inc>

extern code char Velkost;
extern code unsigned char Tab[];       //Tabulka s 7 segmentovymi cislami

enum {ON=1, OFF=0};

#ifdef BRIGHT

bit Light;                                  //Definujem bit svietenia pre PWM_LED
unsigned char Tmp,Bright,ton,toff;

void PWM0(void) interrupt 1 using 2   //Generujem PWM len s jednym casovacom T0
{
 ET0=0;
 TR0=OFF;               //Vypnem casovac T0
 if(Light)
  {
   TH0=toff;            //Budem generovat periodu vypnutia
   P1=0x00;             //Vypnem zobrazovanie len na niektorych pinoch ...
   Light=OFF;           //Nastavim priznak svietenia
  }
   else
    {
     TH0=ton;           //Nastavim pozadovanu hodnotu jasu
     P1=Tmp;            //Obnovim hodnotu na porte P1 len na niektorych pinoch ...
     Light=ON;          //Nastavim priznak svietenia
    }
 TR0=ON;                //Zapnemcasovac
 ET0=1;
}

void Led(unsigned char Cislo)
{
 TR0=OFF;
 P1=Tmp=Tab[Cislo];     //Prevediem cislo na LED segment
 EA=1;
 ET0=1;                 //Povolim prerusenie od T0
 Light=ON;              //Nastavim priznak svietenia
 ton=0xFF-Bright;       //Nastavim pozadovanu hodnotu jasu
 toff=Bright;           //Budem generovat periodu vypnutia
 TH0=toff;              //Nastavim pozadovanu hodnotu jasu
 TR0=ON;                //Spustimcasovac
}

#else

void Led(unsigned char Cislo)
{
 P1=Tab[Cislo];         //Prevediem cislo na LED segment
}

#endif

```

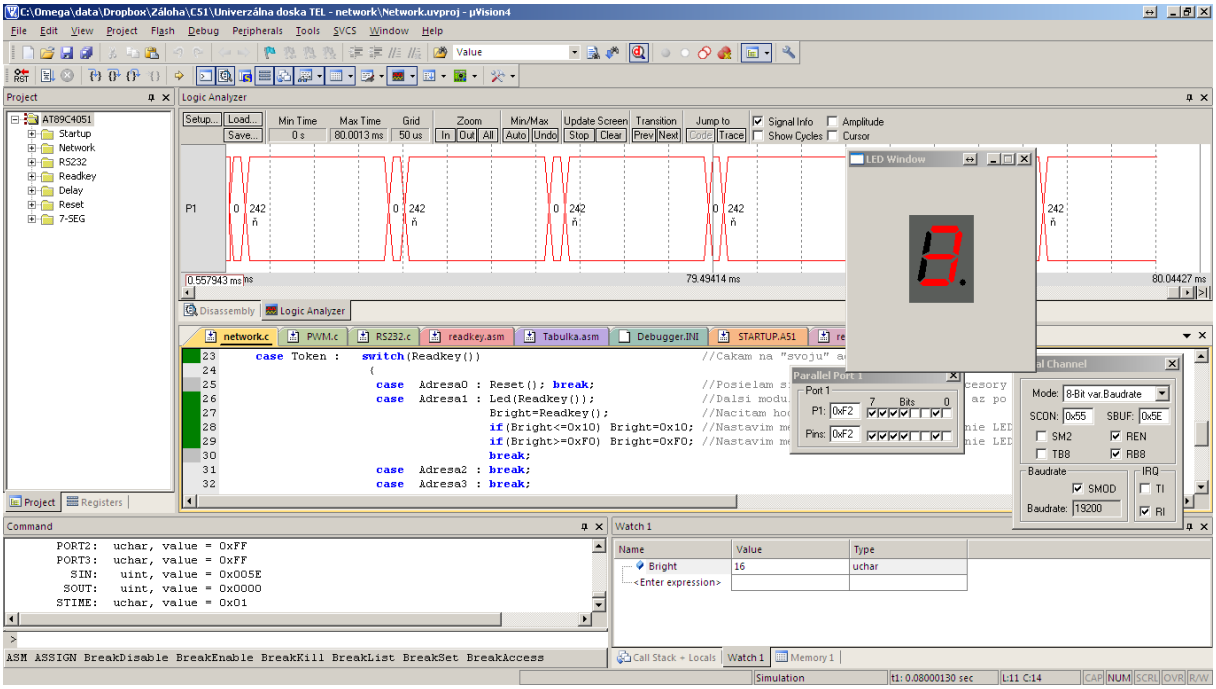
Premenná **Tmp** obsahuje hodnotu ktorá je zapísaná na port P1. Ak by sme čítali port P1 tak by sme čítali priamo vývody mikroprocesora ktoré sú pripojené priamo na bázu tranzistora čo by nám v konečnom dôsledku prečítalo[181] nesprávnu hodnotu pretože zvyšovacie rezistory na vývodoch nám ovplyvnia logickú hodnotu na jednotlivých vývodoch P1.

[181] Pre správnu hodnotu by sme mali čítať po správnosti len záchytný „klopný“ obvod portu P1 mikroprocesora, kde sú zapísané skutočné hodnoty do portu a čítať logické hodnoty nachádzajúce sa na jednotlivých vývodoch portu ktoré môžu byť odlišné ak sa tam nachádza napr. báza tranzistora ktorá logickú hodnotu na výstupe vývodu portu „1“ zmení PN priechodom na „0“.

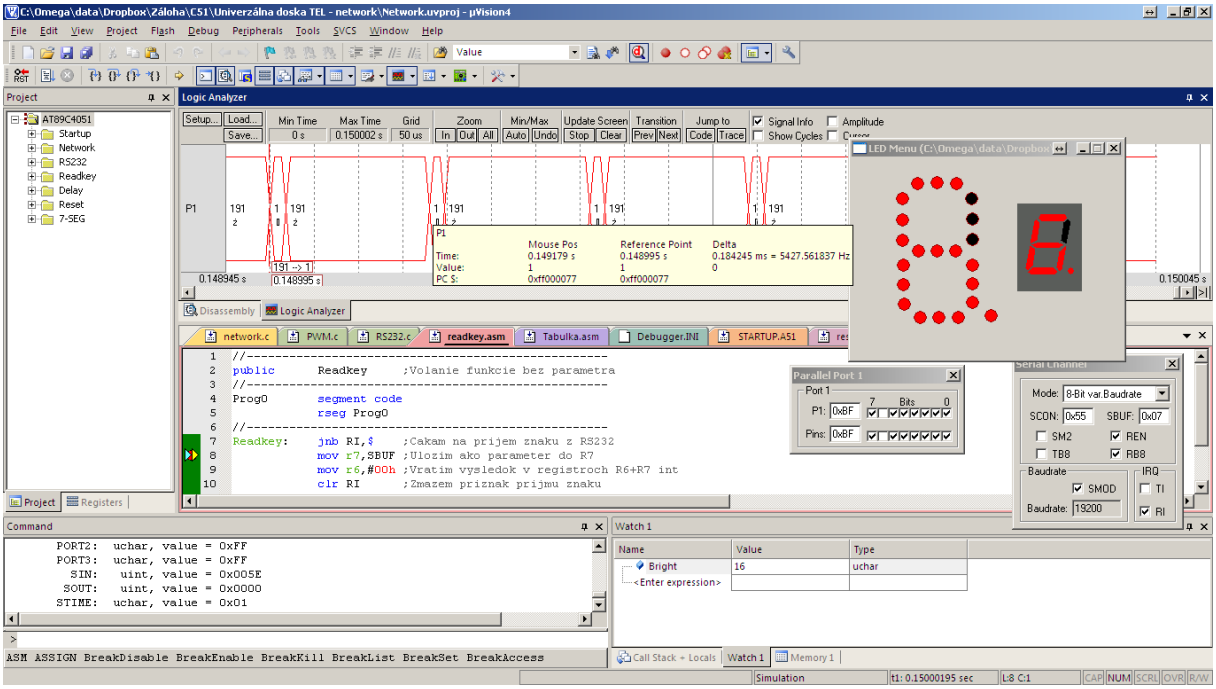
Obrázok 181, Prevodná tabulka 7 seg-mentového displaya v assembleri A51

```
;------------------------------------------------------------------------------
name          Tabulka
;------------------------------------------------------------------------------
public        Tab,Velkost;
;------------------------------------------------------------------------------
Prog0         segment  code  inpage  ;segment umiestneny v ramci jednej 256-byte stranky
              rseg  Prog0
;------------------------------------------------------------------------------
;                .abcdefg      ;Jednotlive segmenty LED display-a
Tab:
              db  01111110b    ;0
              db  00110000b    ;1
              db  01101101b    ;2
              db  01111001b    ;3
              db  00110011b    ;4
              db  01011011b    ;5
              db  01011111b    ;6
              db  01110000b    ;7
              db  01111111b    ;8
              db  01111011b    ;9
              db  01110111b    ;A
              db  00011111b    ;B
              db  00001101b    ;C
              db  00111101b    ;D
              db  01001111b    ;E
              db  01000111b    ;F
              db  01011110b    ;0
              db  00010111b    ;H
              db  00000100b    ;I
              db  00111100b    ;J
              db  00000111b    ;K
              db  00001110b    ;L
              db  01110110b    ;M
              db  00010101b    ;N
              db  00011101b    ;O
              db  01100111b    ;P
              db  00110111b    ;X
              db  00000001b    ;-

;------------------------------------------------------------------------------
Velkost       equ  ($-Tab)          ;Velkost:    db ($-Tab)
;------------------------------------------------------------------------------
              end
```

Obrázok 182, Simulácia PWM kanála so 7-segmentovým displayom a LED display-om



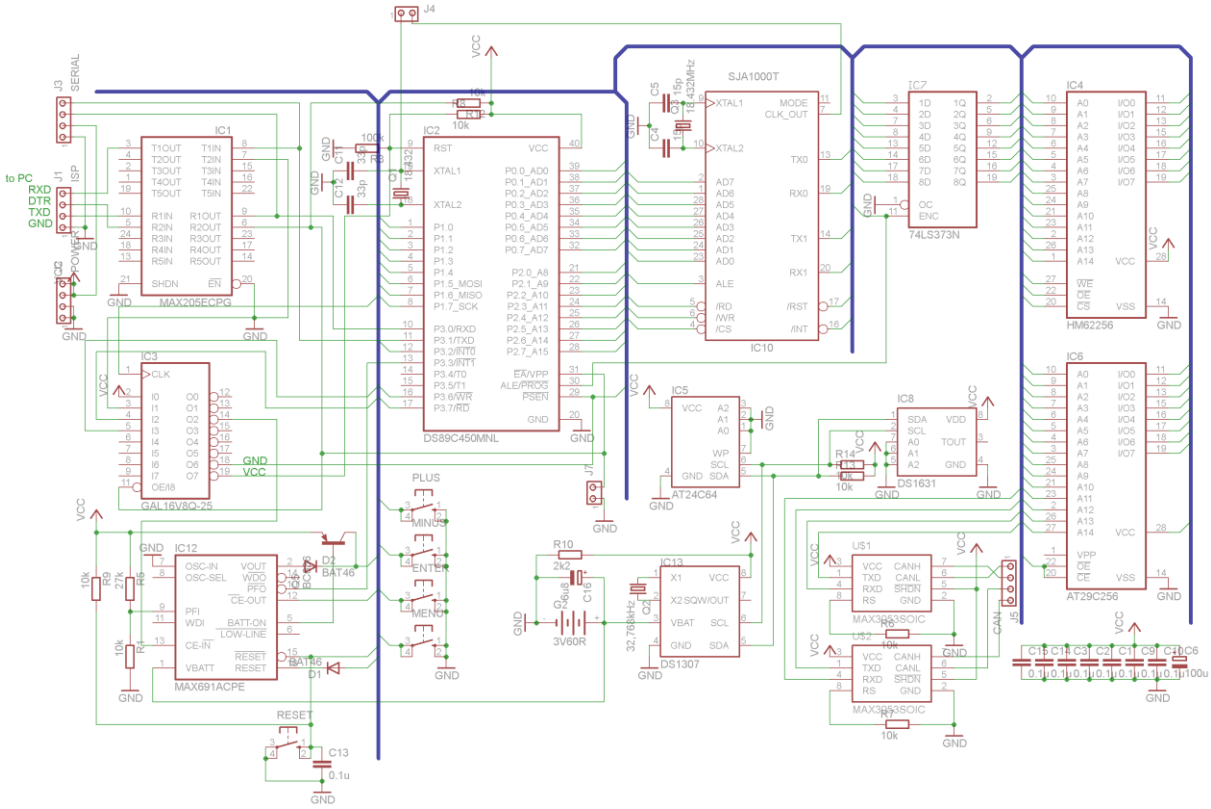
Obrázok 183, Simulácia PWM kanála s LED display-om



15.34. Univerzálne komunikačné rutiny s CAN rozhraním

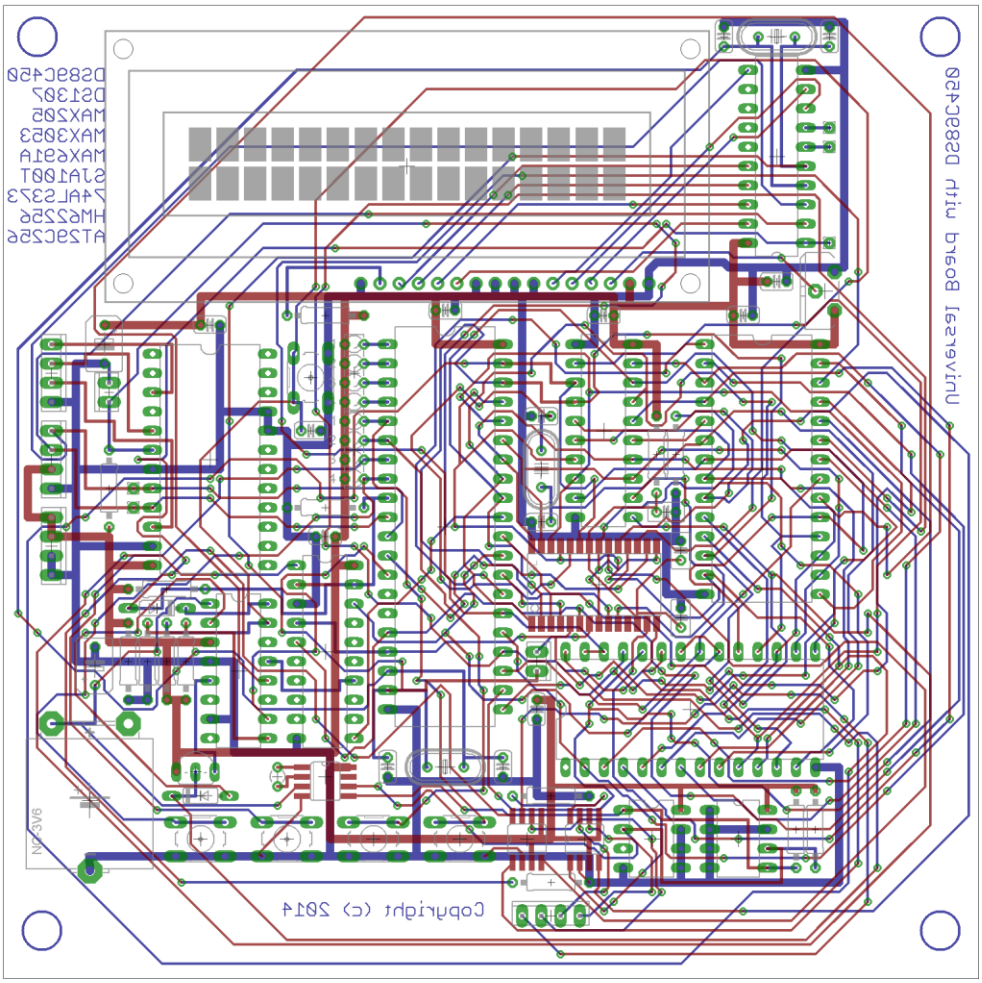
Komunikačné rozhranie CAN definovala v roku 1986 firma Bosch pre potreby automobilového priemyslu. Vďaka jej vynikajúcim vlastnostiam sa dostala aj do spotrebičov každodennej spotreby. Podpora[182] CAN je samozrejmosťou u RTX51 v PK51 pre špeciálne vyvinuté obvody SJA1000T, Philips 82C200, Philips 80592, Siemens c515C, Siemens 81C90 a Intel 82527.

Obrázok 184, Zapojenie univerzálneho modulu s DS89C450 s podporou CAN

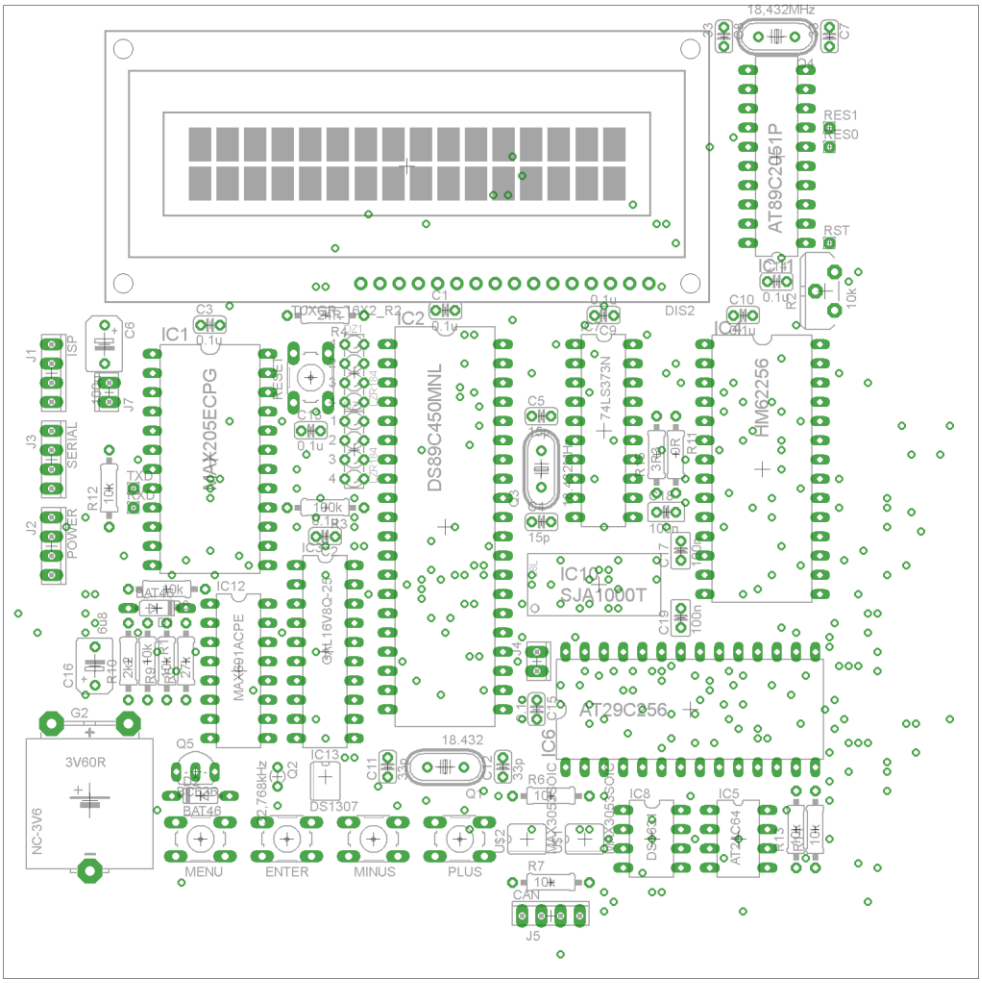


[182] Komunikačné rutiny CAN s RTX51 Full 7.0 nájdete: http://www.keil.com/download/list/c51.htm

Obrázok 185, Doska plošného spoja univerzálneho modulu s podporou CAN



Obrázok 186, Rozmiestnenie súčiastok univerzálneho modulu s podporou CAN



15.35. Ethernet s DS80C400

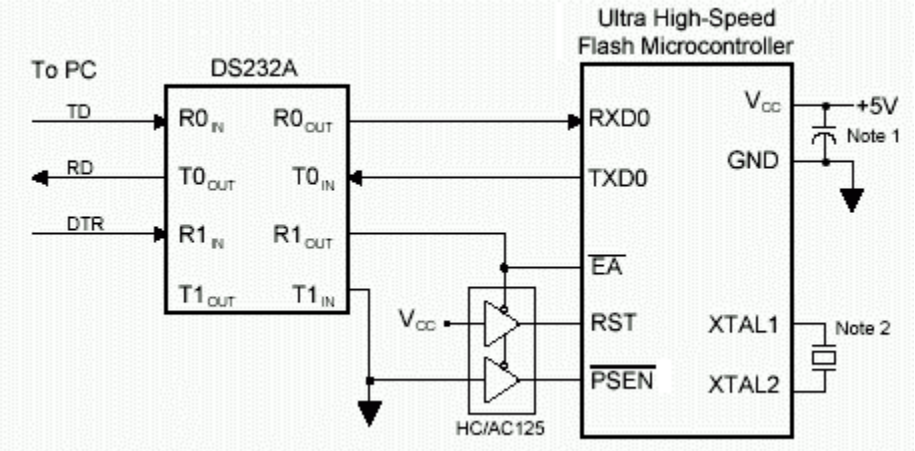
Dnešné moderné mikroprocesory[183] už vo svojom vnútri obsahujú hardware-ovú podporu technológie Ethernet 802.3, ktorá umožňuje navrhnúť ľubovoľné zariadenie s možnosťou komunikácie pomocou protokolov IPv4, alebo IPv6 cez internet. Všetky funkcie Ethernet sú uložené v ROM pamäti procesora a architektúra mikroprocesora umožňuje až 32 súčasných TCP spojení cez sieť. Mikroprocesor obsahuje integrovaný interface IEEE 802.3 MII[184]. Maximálna pracovná frekvencia je 75MHz. Maximálne je možné obslúžiť mikroprocesorom pomocou 24-bit adresy až 16MB pamäti RAM. Štyri DPTR registre umožňujú rýchly prístup k dátam uloženým v XRAM. Spoločne s integrovanou hardware-ovou matematickou jednotkou užívateľ získava k dispozícii výkonný mikroprocesor pre ľubovoľnú aplikačnú oblasť.

15.36. ISP rozhranie pre procesory

Problematika programovania súčiastok je pre každého pracovníka v oblasti mikroprocesorovej techniky dostatočne známa. Niečo iné je napísať program pre procesor a iné je naprogramovať skompilovaný program do procesora. Tu už je nevyhnutne potrebné vo väčšine prípadov vlastniť špecializovaný programátor[185] ktorý je schopný zapísať program do hardware súčiastky. Pokiaľ je nutné programovať súčiastku priamo v aplikácii bez potreby jej odpojenia tak nastáva problém ak súčiastka nepodporuje možnosť programovania cez ISP[186] rozhranie. Rozhranie ISP je pripojené cez externé súčiastky k PC pomocou sériového rozhrania RS232 a komunikuje pomocou software MTK[187]. Mikroprocesory ktoré sú programovateľné priamo zapojené v aplikácii cez ISP sa používajú napr. v PLC.

[183] http://www.maximintegrated.com/en/products/digital/microcontrollers/DS80C400.html
[184] http://www.maximintegrated.com/en/products/interface/transceivers/78Q2123.html
[185] www.elnec.sk
[186] http://www.maximintegrated.com/app-notes/index.mvp/id/3262
[187] v niektorých prípadoch najmä pri použití 64 bitového operačného systému Windows 7 už MTK nefunguje správne a pri pokuse o komunikáciu s mikroprocesorom napr. DS89C450 vypisuje nezmyselné chybové hlásenie a neumožní s ním vykonať žiadnu činnosť. Najlepším riešením ako sa tomu vyhnúť je použiť program Hyperterminal, ktorý je súčasťou operačného systému Windows a pomocou stláčania kláves dokážeme jednoducho naprogramovať mikroprocesor. Príkaz napr. CTRL+K vymaže obsah FLASH, CTRL+D výpis obsahu FLASH, CTRL+L naprogramuje cez integrovaný „bootloader“ súbor v Intel HEX formáte do pamäti FLASH. http://www.maximintegrated.com/products/microcontrollers/software/dev_tool_software/mtk/MTKbeta.zip,

Obrázok 187, ISP pre procesory DS89C450



Obrázok 184 v kapitole 15.34 už obsahuje integrované obvodové zapojenie súčiastok ISP rozhrania pre programovanie procesora DS89C450 v zapojení. Prevodník logických úrovní z CMOS na RS232[188] je tvorený obvodom MAX205 a 74AC125 je nahradený kvôli jednoduchosti a dostupnosti programovateľným obvodom GAL16V8[189] ak uvažujeme prepojenie priamo s PC a programom MTK.

V prípade programovania obvodu pomocou sériového rozhrania tvoreného prevodníkom USB-TTL s čipom CH340G je nutné zabezpečiť inštaláciu správnych ovládačov virtuálneho portu COM do operačného systému. V tomto prípade nie je potrebné mať ďalší prevodník, pretože pracujeme s +5V TTL/CMOS logikou akú používa aj mikroprocesor a nehrozí tak zničenie obvodu. Integrovaný bootloader[190] v DS89C450 umožňuje preniesť pomocou programu Hyperterminal po sériovej linke súbor programu v Intel Hex formáte a zapísať ho do FLASH pamäti mikroprocesora.

[188] Napäťové úrovne RS232 na PC sú v rozsahu ±15V.

[189] GAL16V8 sa už neodporúča používať v nových aplikáciách a nahrádza sa ekvivalentným obvodom GAL20V10, prípadne iným PLD obvodom.

[190] Bootloader mikroprocesora DS89C450 s výhodou používa „autobaud rate“ režim, aby bolo možné použiť široké spektrum frekvencií kryštála XTAL mikroprocesora pri programovaní v cieľovej aplikácii.

15.37. Práca s ukazovateľom v jazyku C

Jazyk C disponuje nesmierne výkonným systémom typizovaných a netypizovaných ukazovateľov, ktorý sa nenachádza v iných programovacích jazykoch napr. Pascal, Basic Fortran a iné. Príklady[191] ktoré sú uvedené v obrázku tejto kapitoly uvádzajú jednotlivé použitia daných pointerov[192] v príslušných konštrukčných volaniach funkcií a procedúr bez parametra alebo s parametrom a nenachádzajú sa obvykle detailne opísané v literatúre (Hrubý, 2009).

Obrázok 188, Pointer v jazyku C

```
#include <reg52.h>
#include <stdio.h>
#include <stdlib.h>

int Counter;

void Int_EX0(void) interrupt 0 using 1
{
 ++Counter;
}

void Start(unsigned int i)
{
 i++;
}

long Xyz(int i, int j, int k)
{
 return(i+j+k);
}

void Func_caller(long (code *fptr)(unsigned int))
{
 unsigned char i;
 for(i=0; i<3; i++)
  {
   (*fptr)(i);    //Volanie funkcie Start(i) s odovzdanim hodnoty ...
  }
}

void (*function_ptr1) (void); //Ukazovatel na funkciu bez parametra ...

void (*function_ptr2) (int i, char j, char ch); //Ukazovatel na funkciu s parametrom, v APNT_129 je jednen parameter long ktory sa nedokaze odovzdat cez parametre ...

void function_ptr (int i, char j, char ch)
{
 i=j+ch;
}

long x;

void main(void)
{
 while(1)
  {
   function_ptr1=Int_EX0;
   function_ptr2=Xyz(0x1234,0x1234abcd,'a');
   (*function_ptr) (0x1234,'a','b');
//*****************************************************************************************************************
   Func_caller(Start);                //Volanie funkcie Start(i) s odovzdanim hodnoty ...
   ((void (code*) (void)) 0x0003) (); //Volanie funkcie alebo procedury na adrese 0x0003 ...
   x=((long (code*) (int, int, int)) &Xyz) (1,2,3); //Volanie funkcie na adrese &Xyz s 3x parametrami a navratom long  ...
//   x=((long (code*) (int, int, int)) 0x8000) (1,2,3); //Volanie funkcie na adrese 0x8000 s 3x parametrami a navratom long  ...
  }
}
```

[191] Podrobnejšie informácie o pointeroch je možné získať na http://www.keil.com/appnotes/docs/apnt_129.asp

[192] Niekedy uvádzané v literatúre ako smerník.

16 Programovanie aplikácií s využitím RTX51 Full

16.1. Problematika práce s RTX51 Full

Oproti predchádzajúcemu operačnému systému RTX51 Tiny je verzia Full komplexnejším a výkonnejším operačným systémom. Jadro operačného systému je napísané tak, aby umožňovalo programátorovi definovať samostatne fungujúce úlohy s reakciou na požadovaný typ udalosti a zaručeným časom odozvy t.j. odpovede.

Súčasne je RTX51 Full schopný spustiť maximálne 19 úloh, pričom 16 úloh môže pracovať s nižšou úrovňou priority, a tri úlohy je možno deklarovať ako ***fast-tasks*** s dobou reakcie pod 50 cyklov mikroprocesora. Prepnutie štandardnej úlohy je závislé na polohe a veľkosti zásobníka môže sa pohybovať v rozsahu 180 až 700 cyklov mikroprocesora. Naproti tomu prepnutie úlohy s úrovňou priority 3 *(fast task)* sa pohybuje v rozsahu 70 až 100 cyklov. Operačný systém RTX51 Full používa pre svoju činnosť len externú pamäť dát v ktorej si uchováva väčšinu informácií o task-och, informácia o polohe a veľkosti zásobníka.

Jedným obmedzením operačného systému RTX51 Full 5.0 z hľadiska programátora je to, že oblasť pamäti programu v rozsahu 0003H až 0080H kde sa štandardne nachádzajú obslužné programy vektorov prerušení využíva RTX51 pre svoje potreby. Z vyššie uvedeného vyplýva, že programátor je úplne odkázaný na obsluhu prerušenia pomocou štandardných funkcií *os_attach_interrupt(number)* a *os_detach_interrupt(number)*. Až od verzie RTX51 Full 7.0 je možné vytvárať vlastné obslužné rutiny prerušenia. Tieto obslužné rutiny nesmú ale kolidovať s štandardnými prerušovacími (interrupt) funkciami uloženými na absolútnej adrese. Lepšiu predstavu o vektoroch prerušenia používaných pri RTX51 nám poskytne súbor ***RTXCONF.A51*** spolu s ***RTXSETUP.INC***. Súbor ***RTX51.LIB*** by mal byť linkerom pripojený až nakoniec. Pri tomto postupe sa vyhneme zbytočným chybovým hláseniam o kolízii pamäťových priestorov ktoré sú zdieľané vektormi prerušenia.

16.2. Použitie modifikovaných architektúr kompatibilných s 8051 s RTX51 Full

Pri mikroprocesore DS87C550[193] je potrebné pozmeniť časti programu ***RTXCONF.A51***, pretože vektory prerušenia sú zmenené oproti štandardnej architektúre 8051.

Aj napriek tomuto nedostatku aplikácie vytvorené pomocou RTX51 sa vyznačujú vynikajúcimi vlastnosťami: variabilnými spôsobmi reakcií, pevne definovanými odozvami na

[193] http://pdfserv.maximintegrated.com/en/ds/DS87C550.pdf

prerušenie, vynikajúcimi schopnosťami vnútro-taskovej t.j. vnútro-úlohovej komunikácie, odolnosťou voči poruchám použitím separátnych stack–tasks oblastí, možnosť implementácie v rôznych typov mikroprocesorov. Architektúra mikroprocesora DS87C550[194] približne 3x rýchlejšia ako konvenčné architektúry 8051 a prevod AD prevodníka je cca. 16,6µs oproti 50µs u 80C552.

Z bohatej praxe môžeme povedať, že na správnu činnosť RTX51 Full je nevyhnutné vyhradiť v externej pamäti XDATA[195] min. cca. 1kB pre jadro samotného operačného systému, min. 250 Byte +(použité premenné a deklarované dátové štruktúry) na každý task s prioritou 0 až 2. Pre task-y s prioritou 3 je použitá interná pamäť DATA, prípadne IDATA. Pri použití RTX51 Full sa snažíme, aby sme mali k dispozícii dostatočný priestor[196] aj pre zásobník procesora STACK, ktorý by mal mať minimálne aspoň 50 Byte, čo si môžeme skontrolovať vo výpise pri kompilácii programu. Keď si predstavíme, že samotný procesor 80c552 má k dispozícii len 256 Byte internej pamäti, musíme veľmi opatrne deklarovať a používať premenné ktoré sú uložené v DATA a IDATA.

[194] Procesor DS87C550 ktorý je derivátom 80C552 je už k dnešnému dňu 5.4.2014 firmou Maxim Integrated nepodporovaný a nevyrábaný. Bol nahradený perspektívnejšími architektúrami.
[195] Pri použití operačného systému RTX51 Full sa deklarované premenné automaticky umiestňujú do externej pamäti dát XRAM.
[196] Dvojnásobne to platí pri použití reentrant-ných funkcií, kde je spotreba internej pamäte procesora priamo úmerná súčinu počtu volaní reentrant-nej funkcie a veľkosti použitých premenných vo funkcii.

Obrázok 189, Implementácia do rtxconf.a51 o nepodporovaný processor DS87C550

c:\Keil\C51\Examples\rtx550\RTXCONF.A51

```
4052    ELSEIF  (?RTX_CPU_TYPE  = 44) ;Written by Ing. Eduard Jadron
4053
4054          ; Define the number and addresses of the interrupt enable registers.
4055          ; Set the not used registers to the same address as ?RTX_IE
4056          ;
4057          INT_EN_MASK_NUMBER    EQU  2
4058          ?RTX_IE               DATA   0A8H
4059          ?RTX_IEN1             DATA   0E8H
4060          ?RTX_IEN2             DATA   0A8H   ; not used
4061
4062          ; Generate the interrupt entry points supported by the peripherals
4063          ; of the selected CPU type.
4064          ;
4065          IF  (?RTX_SYSTEM_TIMER  = 0)
4066             ; Do NOT include the Timer 0 Vector  (INT-3)
4067             INT_ENTRY       0
4068             INT_ENTRY       1
4069             INT_ENTRY       2
4070    ;        INT_ENTRY       3
4071             INT_ENTRY       4
4072             INT_ENTRY       5
4073             INT_ENTRY       6
4074             INT_ENTRY       7
4075             INT_ENTRY       8
4076             INT_ENTRY       9
4077             INT_ENTRY       10
4078             INT_ENTRY       11
4079             INT_ENTRY       12
4080             INT_ENTRY       13
4081             INT_ENTRY       14
4082             INT_ENTRY       15
4083          ELSEIF  (?RTX_SYSTEM_TIMER  = 1)
4084             ; Do NOT include the Timer 1 Vector  (INT-10)
4085             INT_ENTRY       0
4086             INT_ENTRY       1
4087             INT_ENTRY       2
4088         INT_ENTRY       3
4089             INT_ENTRY       4
4090             INT_ENTRY       5
4091             INT_ENTRY       6
4092             INT_ENTRY       7
4093             INT_ENTRY       8
4094             INT_ENTRY       9
4095    ;        INT_ENTRY       10
4096             INT_ENTRY       11
4097             INT_ENTRY       12
4098             INT_ENTRY       13
4099             INT_ENTRY       14
4100             INT_ENTRY       15
4101          ELSEIF  (?RTX_SYSTEM_TIMER  = 2)
4102             ; Do NOT include the Timer 2 Vector  (INT-15)
4103             INT_ENTRY       0
4104             INT_ENTRY       1
4105             INT_ENTRY       2
4106             INT_ENTRY       3
4107             INT_ENTRY       4
4108             INT_ENTRY       5
4109             INT_ENTRY       6
4110             INT_ENTRY       7
4111             INT_ENTRY       8
4112             INT_ENTRY       9
4113             INT_ENTRY       10
4114             INT_ENTRY       11
4115             INT_ENTRY       12
4116             INT_ENTRY       13
4117             INT_ENTRY       14
4118    ;        INT_ENTRY       15
4119          ENDIF
4120
4121          ; Select the Timer 2 configuration.
4122          ;
4123          ?RTX_TIMER2_TYPE   EQU  ?RTX_T2_STANDARD
4124
4125          ; The following table attaches the interrupt numbers (0..31) to the
```

Page1

c:\Keil\C51\Examples\rtx550\RTXCONF.A51

```
4126          ; corresponding bits in the interrupt enable masks of the specific
4127          ; processor.
4128          ; All three interrupt enable register contents must be defined
4129          ; for every interrupt number (even when the specific processor contains
4130          ; only one interrupt mask).
4131          ; Syntax: DB IE-content, IE1-content, IE2-content
4132          ;
4133          ?RTX?RTX_INT_TO_BIT_TABLE?RTXCONF   SEGMENT   CODE
4134                  RSEG   ?RTX?RTX_INT_TO_BIT_TABLE?RTXCONF
4135
4136          ?RTX_INT_TO_BIT_TABLE_BASE :
4137                  DB  01H , 00H , 00H          ; INT_0,  nINTT0 - INT_EX0
4138                  DB  20H , 00H , 00H          ; INT_1,  SCON1 at P1
4139                  DB  40H , 00H , 00H          ; INT_2,  A/D Analog Converter
4140                  DB  02H , 00H , 00H          ; INT_3,  TF0 - Timer 0
4141                  DB  00H , 01H , 00H          ; INT_4,  INT2/CF0
4142                  DB  00H , 10H , 00H          ; INT_5,  CM0F
4143                  DB  00H , 80H , 00H          ; INT_6,  PFI Power Fail Interrupt
4144                  DB  04H , 00H , 00H          ; INT_7,  nINTT1 - INT_EX1
4145                  DB  00H , 02H , 00H          ; INT_8,  INT3/CF1
4146                  DB  00H , 20H , 00H          ; INT_9,  CM1F
4147                  DB  08H , 00H , 00H          ; INT_10, TF1 - Timer 1
4148                  DB  00H , 04H , 00H          ; INT_11, INT4/CF2
4149                  DB  00H , 40H , 00H          ; INT_12, CM2F
4150                  DB  10H , 00H , 00H          ; INT_13, SCON0 at P0
4151                  DB  00H , 04H , 00H          ; INT_14, INT5/CF3
4152                  DB  00H , 80H , 00H          ; INT_15, TF2 - Timer 2
4153
4154          ; Define the greatest supported interrupt number
4155          ;
4156          ?RTX_MAX_INT_NBR   EQU    15
4157
4158          ; Enter Idle Mode Macro
4159          ; Used whenever entering idle state. Peripherals stay active.
4160          ; Leaved after interrupt.
4161          ;
4162          PCON     DATA     87H
4163          ENTER_IDLE  MACRO
4164                  ORL      PCON , #01H        ; Set idle mode
4165                  ENDM
4166
4167          ; Define the register addresses to set the MSB of the page address.
4168          ;
4169          ?RTX_PAGE_OUT_REG  DATA  0A0H       ; write directly to Port 2
4170
4171    ELSE
```

Obrázok 190, Modifikácia súboru RTXCONF.A51 pre procesor DS87C550

c:\Keil\C51\Examples\rtx550\RTXSETUP.INC

```
;  1. CPU  TYPE
;  ===========
;
;    CPU_TYPE  stands  for  the  desired  microprocessor  type  and  can  be
;    selected  from  the  following  table :
;
;  Manufacturer      Model                CPU_TYPE
;  ------------      -----                --------
;  Acer  Lab         M6759                41
;  Aeroflex  UTMC    UT69RH051            23
;  Atmel             87F51                1
;  Atmel             87F51RC              2
;  Atmel             87F52                2
;  Atmel             89C51                1
;  Atmel             89C52                2
;  Atmel             89C55                2
;  Atmel             89F51                1
;  Atmel             89F52                2
;  Atmel             89LS53               2
;  Atmel             89LS8252             2
;  Atmel             89LV52               2
;  Atmel             89LV55               2
;  Atmel             89S53                2
;  Atmel             89S8252              2
;  Atmel  Wireless   80C31                1
;  Atmel  Wireless   80C31X2              1
;  Atmel  Wireless   80C32                2
;  Atmel  Wireless   80C51                1
;  Atmel  Wireless   80C51RA2             5
;  Atmel  Wireless   80C51RD2             5
;  Atmel  Wireless   80C51U2              5
;  Atmel  Wireless   80C52X2              2
;  Atmel  Wireless   80C54X2              2
;  Atmel  Wireless   80C58X2              2
;  Atmel  Wireless   83/87C51RB2          5
;  Atmel  Wireless   83/87C51RC2          5
;  Atmel  Wireless   83/87C51RD2          5
;  Atmel  Wireless   83/87C51U2           5
;  Atmel  Wireless   83/87C52X2           2
;  Atmel  Wireless   87C51                1
;  Atmel  Wireless   T87C5112             29
;  Atmel  Wireless   T89C51CC01           42
;  Atmel  Wireless   T89C51RD2            5
;  Cygnal            C8051F005/6/7        39  [*2]
;  Cygnal            C8051F015/16/17      39  [*2]
;  Cygnal            C8051F020/21/22/23   43  [*2]
;  Cygnal            C8051F206            40  [*2]
;  Cygnal            C8051F226            40  [*2]
;  Cygnal            C8051F236            40  [*2]
;  Dallas            DS80C310             33
;  Dallas            DS80C320             16  [*3]
;  Dallas            DS80C323             16  [*3]
;  Dallas            DS80C530             17  [*3]
;  Dallas            DS87C520/DS83C520    16  [*3]
;  Dallas            DS87C530             17  [*3]
;  Dallas            DS89C420             16  [*3]
;  Dallas            DS87C550             44  [*2]  [*3???]  Written  by  Ing.  Eduard  Jadron
;  Hynix  (Hyundai)  GMS90C32             2
;  Hynix  (Hyundai)  GMS90C320            2
;  Hynix  (Hyundai)  GMS90C51             1
;  Hynix  (Hyundai)  GMS90C52             2
;  Hynix  (Hyundai)  GMS90C54             2
;  Hynix  (Hyundai)  GMS90C56             2
;  Hynix  (Hyundai)  GMS90C58             2
;  Hynix  (Hyundai)  GMS90L32             2
;  Hynix  (Hyundai)  GMS90L320            2
;  Hynix  (Hyundai)  GMS90L51             1
;  Hynix  (Hyundai)  GMS90L52             2
;  Hynix  (Hyundai)  GMS90L54             2
;  Hynix  (Hyundai)  GMS90L58             2
```

Page1

17 Príklady programov s RTX51 Full

V automatizačnej praxi sú používané viaceré typy regulátorov. Najčastejšie sa stretávame s regulátorom PID, alebo PSD. V prípade PID regulátora to môže byť analógová, alebo číslicová t.j. diskrétna forma. Samotný algoritmus diskrétnej formy regulátora je potom možné zapísať pomocou príkazov programovacieho jazyka.

17.1. Analógový regulátor v diskrétnom tvare

Základný vzťah pre analógový tvar PID regulátora:

$$y(t) = K_c \cdot \left[e(t) + \frac{1}{T_i} \cdot \int e(t)dt + T_d \cdot \frac{de(t)}{dt}\right] \quad (7.)$$

Prenos regulátora PID:

$$W_{(s)} = \frac{U_{(s)}}{E_{(s)}} \quad (8.)$$

Po jednoduchej úprave dostaneme nasledujúci vzťah vhodný pre transformáciu[197]:

$$W_{(s)} = K_c \cdot \left(1 + \frac{1}{T_i \cdot s} + \frac{T_d \cdot s}{\gamma \cdot s + 1}\right) = K_c \cdot \left(1 + \frac{1}{T_i \cdot s} + \frac{T_d}{\gamma + s^{-1}}\right) \quad (9.)$$

Použijeme bilineárny transformačný vzťah na prevod zo spojitej oblasti do diskrétnej:

$$s = \frac{2}{T_s} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} \quad (10.)$$

Použitím predchádzajúcej bilineárnej transformácie dostaneme:

$$W_{(s)} = K_c \cdot \left[1 + \frac{T_s}{2 \cdot T_i} \cdot \frac{1 + z^{-1}}{1 - z^{-1}} + \frac{T_d \cdot (1 - z^{-1})}{\gamma \cdot (1 - z^{-1}) + \frac{T_s}{2} \cdot (1 + z^{-1})}\right] \quad (11.)$$

[197] Pri transformácii matematických výrazov odporúčame použiť špecializovaný software MathCAD Prime 3.1, ktoré výrazne urýchlia výpočet požadovaných výrazov.

Ako uvádza (ir. drs. E.H.W. van de Logt, 2011)[198], po transformácii dostaneme výraz v tvare:

$$\frac{U_{(z)}}{E_{(z)}} = K_c \cdot \left[\frac{\left(1+\frac{T_S}{2T_i}+\frac{2T_d}{T_S+2\gamma}\right)+\left(\frac{\frac{T_S^2}{T_i}-4\gamma-4T_d}{T_S+2\gamma}\right)\cdot z^{-1}+\left(\frac{2\gamma-T_S+\frac{T_S(T_S-2\gamma)}{2T_i}+2T_d}{T_S+2\gamma}\right)\cdot z^{-2}}{1-\left(\frac{4\gamma}{T_S+2\gamma}\right)\cdot z^{-1}-\left(\frac{T_S-2\gamma}{T_S+2\gamma}\right)\cdot z^{-2}} \right] \quad (12.)$$

Z vyššie uvedeného výrazu vyjadríme jednotlivé koeficienty k_0, k_1, k_2, a polynómy p_1, p_2,:

$$k_0 = K_c \cdot \left(1 + \frac{T_S}{2 \cdot T_i} + \frac{2 \cdot T_d}{T_S+2 \cdot \gamma}\right) \quad (13.)$$

$$k_1 = K_c \cdot \left(\frac{\frac{T_S^2}{T_i}-4 \cdot \gamma-4 \cdot T_d}{T_S+2 \cdot \gamma}\right) \quad (14.)$$

$$k_2 = K_c \cdot \left(\frac{2 \cdot \gamma-T_S+\frac{T_S \cdot (T_S-2 \cdot \gamma)}{2 \cdot T_i}+2T_d}{T_S+2 \cdot \gamma}\right) \quad (15.)$$

$$p_1 = \left(\frac{4 \cdot \gamma}{T_S+2 \cdot \gamma}\right) \quad (16.)$$

$$p_2 = \left(\frac{T_S-2 \cdot \gamma}{T_S+2 \cdot \gamma}\right) \quad (17.)$$

Substitúciou vyššie uvedených koeficientov a polynómov dostaneme:

$$\frac{U_{(z)}}{E_{(z)}} = \frac{(1-p_1 \cdot z^{-1}-p_2 \cdot z^{-2})}{(k_0-k_1 \cdot z^{-1}-k_2 \cdot z^{-2})} \quad (18.)$$

$$U_{(z)} \cdot (1 - p_1 \cdot z^{-1} - p_2 \cdot z^{-2}) = E_{(z)} \cdot (k_0 - k_1 \cdot z^{-1} - k_2 \cdot z^{-2}) \quad (19.)$$

Transformáciou predchádzajúceho výrazu v časovej oblasti dostaneme do tvaru:

$$u[k] = p_1 \cdot u[k-1] + p_2 \cdot u[k-2] + k_0 \cdot e[k] + k_1 \cdot e_1[k-1] + k_2 \cdot e_2[k-2] \quad (20.)$$

K rovnakému záveru dospel aj (Bobál, a iní, 2005 s. 77).

[198] Inicializačná rutina PID regulátora ***void init_pid2(pid_params *p)*** obsahuje chybu na riadku *p->k2 += 2.0 * p->ti – p->k_lpf * p->ts / p->ti;* kde je uvedená nesprávne integračná konštanta namiesto derivačnej konštanty, má byť uvedený správne riadok *p->k2 += 2.0 * p->td - p->k_lpf * p->ts / p->ti;*. Vyššie uvedená chyba spôsobí zápis nesprávnej hodnoty do premennej koeficientu **k2**, čo môže spôsobiť nežiadúce odchýlky pri regulácii.

Derivovaním základného PID vzťahu získame tvar:

$$du(t) = K_c \cdot \left(de(t) + \frac{e(t)}{T_i} + T_d \cdot \frac{d^2e(t)}{dt}\right) \quad (21.)$$

Transformáciou do časovej diskrétnej oblasti pomocou spätnej diferencie:

$$u_k = u_{k-1} + K_c \cdot \left[(e_k - e_{k-1}) + \frac{T_s \cdot e_k}{T_i} + \frac{T_d}{T_s} \cdot (e_k - 2 \cdot e_{k-1} + e_{k-2})\right] \quad (22.)$$

Aby sme zabránili nežiadúcim zmenám[199] na výstupe regulátora vplyvom D – zložky, pripojíme na vstup dolno-priepustný filter LPF (Low-Pass Filter).

Prenos filtra LPF je v tvare:

$$H(s) = \frac{1}{\gamma \cdot s + 1} \quad (23.)$$

kde veľkosť filtračnej [200]zložky $\boldsymbol{\gamma}$ bude v zvolenom rozsahu napr. 10% z T_d:

$$\gamma = 0{,}10 \cdot T_d \quad (24.)$$

Ako uvádza (Bobál, a iní, 2005), tak ekvivalentná Z prenosová funkcia regulátora s filtrom je:

$$H(z) = \frac{T_s \cdot (1+z^{-1})}{2 \cdot \gamma \cdot (1-z^{-1}) + T_s \cdot (1+z^{-1})} = \frac{T_s}{(T_s + 2 \cdot \gamma)} \cdot \frac{(1+z^{-1})}{1 + \left(\frac{T_s - 2 \cdot \gamma}{T_s + 2 \cdot \gamma}\right) \cdot z^{-1}} \quad (25.)$$

Funkcia LPF filtra zapísaná v časovo diskrétnej oblasti:

$$lpf_k = \frac{2 \cdot \gamma - T_s}{2 \cdot \gamma + T_s} \cdot lpf_{k-1} + \frac{T_s}{T_s + 2 \cdot \gamma} \cdot (e_k - e_{k-1}) \quad (26.)$$

Po dosadení z predchádzajúcich vzťahov dostávame výraz aplikovateľný pre výpočet:

$$u_k = u_{k-1} + K_c \cdot \left[(e_k - e_{k-1}) + \frac{T_s \cdot e_k}{T_i} + \frac{T_d}{T_s} \cdot (lpf_k - 2 \cdot lpf_{k-1} + lpf_{k-2})\right] \quad (27.)$$

Ako uvádza (Bobál, a iní, 2005 s. 62), môžeme použiť na vstupe filter prvého rádu s časovou konštantou $\mathbf{T_f}$, čo môžeme vyjadriť ako:

$$D(s) = K_p \cdot \frac{T_d \cdot s}{T_f \cdot s + 1} \cdot E(s); \qquad T_f = \frac{T_d}{\alpha}; \qquad \alpha \in \langle 3; 20 \rangle \quad (28.)$$

[199] Vzorkovaním vstupného signálu spôsobujeme tzv. „šum“, ktorý sa vplyvom derivačnej zložky ešte viac zosilňuje a môže na výstupe regulátora nadobudnúť veľké hodnoty.
[200] V praxi je empiricky overená a použiteľná veľkosť filtračnej zložky v rozsahu 3 až 20%.

Diskretizáciou predchádzajúceho vzťahu dostaneme:

$$d(k) = \frac{T_d \cdot d(k-1) + K_p \cdot T_d \cdot \alpha \cdot [e(k) - e(k-1)]}{T_d + \alpha \cdot T_s} \quad (29.)$$

Po Tustinovej transformácii získame nasledujúci tvar:

$$d(k) = \frac{(2 \cdot T_d - \alpha \cdot T_s) \cdot d(k-1) + 2 \cdot K_p \cdot T_d \cdot \alpha \cdot [e(k) - e(k-1)]}{2 \cdot T_d + \alpha \cdot T_s} \quad (30.)$$

Obe predchádzajúce transformácie majú rovnaký tvar, ale s rozdielnymi koeficientami ***a***,***b***:

$$d(k) = \boldsymbol{a} \cdot d(k-1) + \boldsymbol{b} \cdot [e(k) - e(k-1)] \quad (31.)$$

Aproximácie jednotlivých sú stabilné pre všetky situácie $\boldsymbol{T_d > 0}$, ale problém nastáva ak koeficient $\boldsymbol{a < 0}$ keď $T_d < \alpha \frac{T_s}{2}$, čo spôsobí nežiadúce oscilácie pri výpočte, v špeciálnom prípade ak bude $\boldsymbol{T_d << T_s}$. Len výraz po diskretizácii dáva dobré výsledky pre všetky hodnoty derivačnej konštanty $\boldsymbol{T_d}$.

17.2. Prevod PID parametrov regulátora na diskrétny tvar PSD

Štandardný tvar PID regulátora sa udáva v integrálnom, alebo diferenciálnom tvare, ktorý je pre použitie v mikroprocesorovej technike prakticky nepoužiteľný. Ako uvádza (HRUBINA, a iní, 2005) je výhodnejšie používať štandardný PID tvar regulátora, ktorý je možné jednoducho previesť na diskrétny tvar:

$$u(t) = K_P \cdot \left(e(t) + \frac{1}{T_i} \cdot \int_0^t e(\tau) d\tau + T_d \cdot \frac{de(t)}{dt} \right) \quad (32.)$$

Štandardný tvar PID regulátora sa udáva v tvare (ŠVARC, 2002):

$$G_{PID}(s) = r_0 + \frac{r_{-1}}{s} + r_1 \cdot s = r_0 \cdot \left(1 + \frac{1}{\frac{r_0}{r_{-1}} \cdot s} + \frac{r_0}{r_1} \cdot s \right) = K_P \left(1 + \frac{1}{T_i \cdot s} + T_d \cdot s \right) \quad (33.)$$

Tabuľka 18, Výpočet koeficientov prenosu regulátora

	Dopredná obdĺžniková metóda FRM	**Spätná obdĺžniková metóda BRM**	**Spätná lichobežníková metóda TRAP**
d0	$+K_p \cdot \left(1 + \frac{T_d}{T_s}\right)$	$+K_p \cdot \left(1 + \frac{T_s}{T_i} + \frac{T_d}{T_s}\right)$	$+K_p \cdot \left(1 + \frac{T_s}{2 \cdot T_i} + \frac{T_d}{T_s}\right)$
d1	$-K_p \cdot \left(1 - \frac{T_s}{T_i} + \frac{2 \cdot T_d}{T_s}\right)$	$-K_p \cdot \left(1 + \frac{2 \cdot T_d}{T_s}\right)$	$-K_p \cdot \left(1 - \frac{T_s}{2 \cdot T_i} + \frac{2 \cdot T_d}{T_s}\right)$
d2	$+K_p \cdot \left(\frac{T_d}{T_s}\right)$	$+K_p \cdot \left(\frac{T_d}{T_s}\right)$	$+K_p \cdot \left(\frac{T_d}{T_s}\right)$

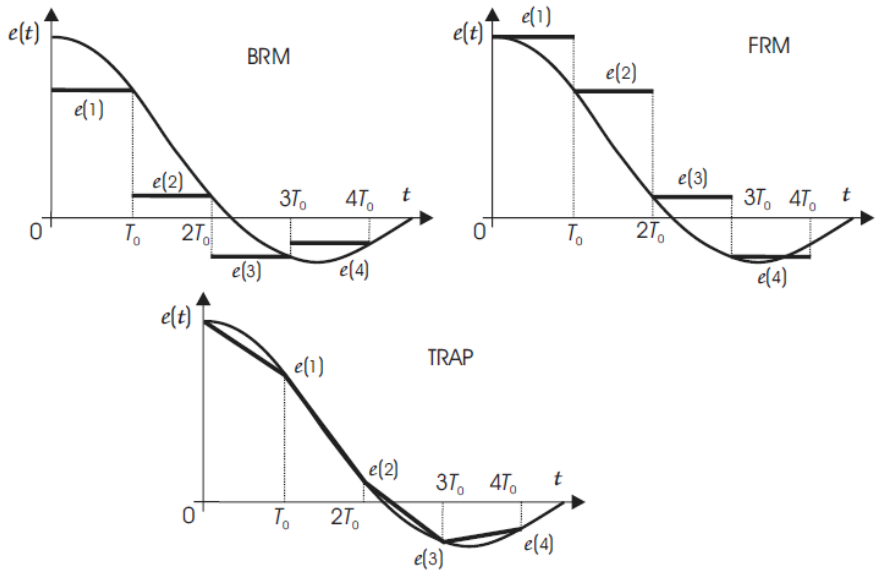
FRM - Forward rectangular method

BRM - Backward rectangular method

TRAP - Trapezoidal method

T_s - perióda vzorkovania [s]

Obrázok 191, Diskretizačné metódy integrálneho prevodu



Prenos PID regulátora v Laplaceovej transformácii v paralelnom[201] tvare je:

$$G_{PID}(s) = P \cdot \left(1 + \frac{I}{P \cdot s} + \frac{D}{P} \cdot s\right) \quad (34.)$$

V knižnici „DSP Software Library“ ktorá je súčasťou CMSIS[202] je výrobcom knižnice implementovaná jednoduchá funkcia PID[203] regulačného algoritmu ktorý bol špeciálne napísaný s použitím SIMD inštrukcií pre dosiahnutie maximálnej rýchlosti výpočtu.

$$y_n = y_{n-1} + A_0 \cdot x_n + A_1 \cdot x_{n-1} + A_2 \cdot x_{n-2} \quad (35.)$$

$$y[n] = y[n-1] + A0 \cdot x[n] + A1 \cdot x[n-1] + A2 \cdot x[n-2]$$

$$\mathrm{A0} = (\mathrm{Kp} + \mathrm{Ki} + \mathrm{Kd})$$

$$\mathrm{A1} = (-\mathrm{Kp}) - (2 * \mathrm{Kd})$$

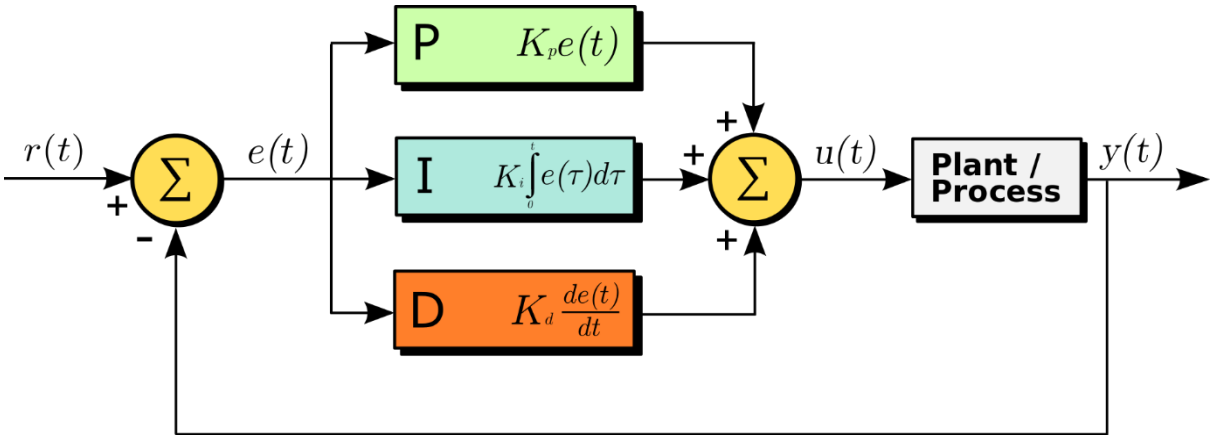
$$A2 = (Kd)$$

K_p - proporcionálna konštanta [-]

T_i - integračná konštanta [s]

T_d - derivačná konštanta [s]

Obrázok 192, PID regulátor implementovaný v CMSIS-DSP knižnici



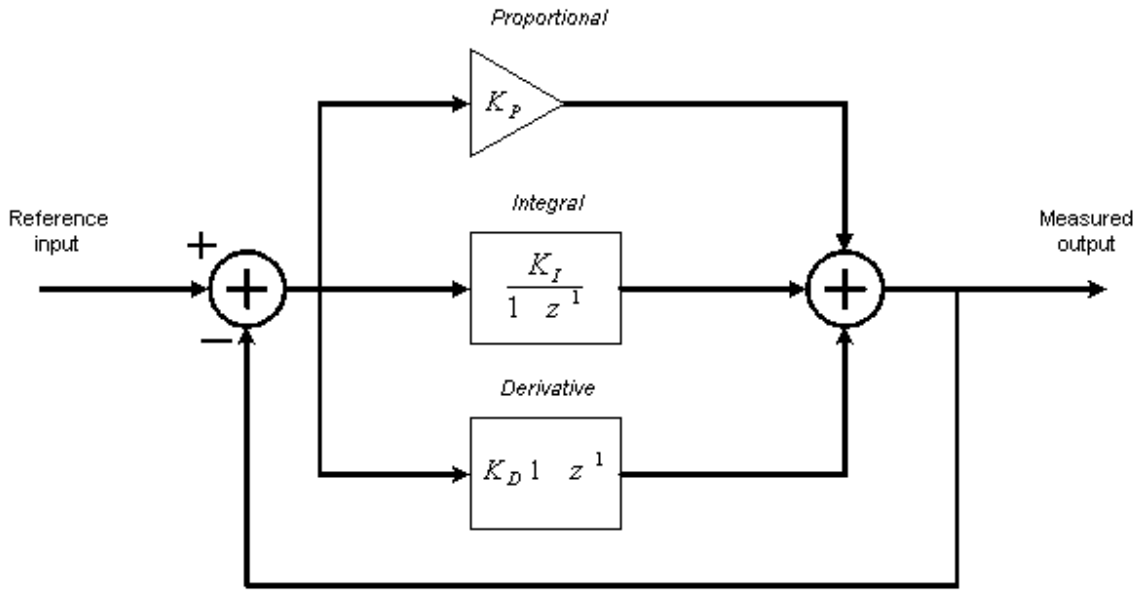
Zdroj obrázku: Wikipedia

[201] Paralelný tvar PID regulátora je „bez interakcie“, keďže sú to 3 samostatné paralelné časti. Interakciou myslíme vzájomné ovplyvňovanie sa jednotlivých zložiek regulátora P, I, D. Zmenou jednej zložky nedôjde k vzájomnému ovplyvneniu, alebo zmeny ostatných zložiek regulátora bez interakcie. V pneumatike, alebo hydraulike sa častejšie stretávame s regulátormi „s interakciou“, kde je v sérii zapojený P, PD, PI regulátor.

[202] https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html

[203] https://www.keil.com/pack/doc/CMSIS/DSP/html/group__PID.html, funkciu **arm_pid_init_f32()** je možné upraviť aj na výpočet PSD regulátora s prepočítanými koeficientami napr. z PID na PSD, alebo iný typ regulátora.

Obrázok 193, Bloková schéma riadenia PID s DSP processorom



Zdroj obrázku: https://keil.com

Obrázok 194, Pôvodný programový kód PID regulátora

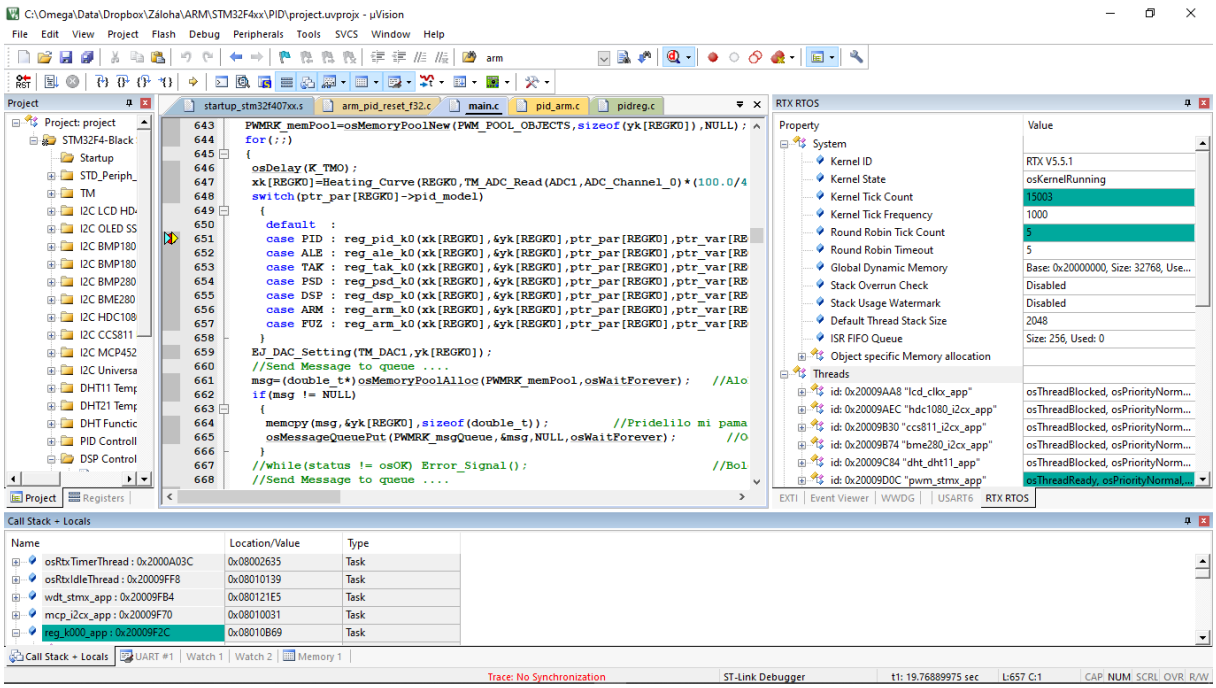
```
typedef struct
{
  float32_t A0;          /**< The derived gain, A0 = Kp + Ki + Kd . */
  float32_t A1;          /**< The derived gain, A1 = -Kp - 2Kd. */
  float32_t A2;          /**< The derived gain, A2 = Kd . */
  float32_t state[3];    /**< The state array of length 3. */
  float32_t Kp;               /**< The proportional gain. */
  float32_t Ki;               /**< The integral gain. */
  float32_t Kd;               /**< The derivative gain. */
} arm_pid_instance_f32;

static __INLINE float32_t arm_pid_f32(arm_pid_instance_f32 * S, float32_t in)
{
  float32_t out;

  /* y[n] = y[n-1] + A0 * x[n] + A1 * x[n-1] + A2 * x[n-2]  */
  out = (S->A0 * in) + (S->A1 * S->state[0]) + (S->A2 * S->state[1]) + (S->state[2]);

  /* Update state */
  S->state[1] = S->state[0];
  S->state[0] = in;
  S->state[2] = out;

  /* return to application */
  return (out);

}
```

Bližšie informácie o ďalších možnostiach využitia DSP Software Library v oblasti ARM procesorov pod označením Cortex-M4[204] uvádza (Trevor, 2013). Knižnica obsahuje jednoduché príklady z rôznych oblastí ktoré sú napísané v jazyku C tak aby sa čo najviac využila výkonná RISC architektúra mikroprocesora ARM. Knižnica obsahuje jednoduché

[204] napr. STM32F407VETx tzv. ARM Cortex-M4

príklady z rôznych oblastí ktoré sú napísané v jazyku C a hlavne v assembleri tak aby sa čo najviac využila výkonná RISC architektúra mikroprocesora ARM.

Obrázok 195, Ukážka prostredia multifunkčného regulátora s RTOS



Obrázok 196, Pôvodná inštancia rutiny pre PID regulátor

```
void arm_pid_init_f32(
  arm_pid_instance_f32 * S,
  int32_t resetStateFlag)
{

  /* Derived coefficient A0 */
  S->A0 = S->Kp + S->Ki + S->Kd;

  /* Derived coefficient A1 */
  S->A1 = (-S->Kp) - ((float32_t) 2.0 * S->Kd);

  /* Derived coefficient A2 */
  S->A2 = S->Kd;

  /* Check whether state needs reset or not */
  if(resetStateFlag)
  {
    /* Clear the state buffer.  The size will be always 3 samples */
    memset(S->state, 0, 3u * sizeof(float32_t));
  }

}
```

Obrázok 197, Modifikovaná inštancia rutiny pre PSD regulátor

```
void arm_psd_init_f32(
  arm_pid_instance_f32 * S,
  int32_t resetStateFlag)
{

  /* Derived coefficient A0 */
  S->A0 = S->Kp;

  /* Derived coefficient A1 */
  S->A1 = S->Ki;

  /* Derived coefficient A2 */
  S->A2 = S->Kd;

  /* Check whether state needs reset or not */
  if(resetStateFlag)
  {
    /* Clear the state buffer.  The size will be always 3 samples */
    memset(S->state, 0, 3u * sizeof(float32_t));
  }

}
```

Obrázok 198, Podprogram PID regulátora z knižnice arm_math.h

```
 static __INLINE float32_t arm_pid_f32(arm_pid_instance_f32 * S, float32_t in)
 {
   float32_t out;

   /* y[n] = y[n-1] + A0 * x[n] + A1 * x[n-1] + A2 * x[n-2]  */
   out = (S->A0 * in) + (S->A1 * S->state[0]) + (S->A2 * S->state[1]) +
(S->state[2]);

   /* Update state */
   S->state[1] = S->state[0];
   S->state[0] = in;
   S->state[2] = out;

   /* return to application */
   return (out);

 }
```

Príklad výpočtu:

Vypočítajte prenos PSD regulátora pri vzorkovaní **T_s=100ms** ak sú vopred známe konštanty **K_p, T_i, T_d** z použitého klasického analógového PID regulátora:

$K_p = 156{,}2$ [-]

$T_i = 230{,}0\ s$

$T_d = 57{,}2\ s$

$$G_{PS}(s) = \frac{d_0 + d_1 \cdot z^{-1}}{1 - z^{-1}}$$

$$G_{PD}(s) = \frac{d_0 + d_2 \cdot z^{-2}}{1 - z^{-1}}$$

$$G_{PSD}(s) = \frac{d_0 + d_1 \cdot z^{-1} + d_2 \cdot z^{-2}}{1 - z^{-1}}$$

Výpočet koeficientov prenosu PSD z PID pomocou spätnej lichobežníkovej metódy:

$$d_0 = +K_p \cdot \left(1 + \frac{T_s}{2 \cdot T_i} + \frac{T_d}{T_s}\right) = 156{,}2 \cdot \left(1{,}00 + \frac{0{,}10}{2 \cdot 230{,}0} + \frac{57{,}5}{0{,}10}\right) = 8{,}997 \cdot 10^4$$

$$d_1 = -K_p \cdot \left(1 - \frac{T_s}{2 \cdot T_i} + \frac{2 \cdot T_d}{T_s}\right) = -156{,}2 \cdot \left(1{,}00 - \frac{0{,}10}{2 \cdot 230{,}0} + \frac{2 \cdot 57{,}5}{0{,}10}\right) = -1{,}798 \cdot 10^5$$

$$d_2 = +K_p \cdot \frac{T_d}{T_s} = 156{,}2 \cdot \frac{57{,}5}{0{,}10} = 8{,}982 \cdot 10^4$$

Po dosadení koeficientov prenosu dostaneme vzťah použiteľný pre program napr. v jazyku C:

$$y_n = y_{n-1} + d_0 \cdot e_n + d_1 \cdot e_{n-1} + d_2 \cdot e_{n-2}$$

Prenos PSD regulátora v z-transformácii bude:

$$G_{PSD}(z) = \frac{d_0 + d_1 \cdot z^{-1} + d_2 \cdot z^{-2}}{1 - z^{-1}} = \frac{8{,}997 \cdot 10^4 - 1{,}798 \cdot 10^5 \cdot z^{-1} + 8{,}982 \cdot 10^4 \cdot z^{-2}}{1 - z^{-1}}$$

Kde **z^{-1}** je spätná, alebo predchádzajúca hodnota operátora vyjadriteľná aj ako:

$$x(k-1) = z^{-1} \cdot x(k) \qquad (36.)$$

Prevod medzi spojitým a diskrétnym systémom môžeme vykonať Tustinovou substitúciou[205]:

$$s = \frac{2}{T_s} \cdot \frac{z-1}{z+1} \tag{37.}$$

alebo spätnou metódou (Backward Rectangular Rule)

$$s = \frac{z-1}{T \cdot z} \tag{38.}$$

alebo doprednou metódou (Forward Rectangular Rule)

$$s = \frac{z-1}{T} \tag{39.}$$

V praxi je potrebné filtrovať derivačnú zložku, čím prenos derivačnej zložky bude mať tvar:

$$D(s) = K_P \cdot \frac{T_d \cdot s}{T_f \cdot s+1} \tag{40.}$$

kde

$$T_f = \frac{T_d}{\alpha} \tag{41.}$$

Obvykle zvolíme **α=10**, čím dosiahneme 10-krát menšiu časovú konštantu filtra oproti dominantnej derivačnej zložke. Pre praktickú realizáciu môže byť **α** v rozsahu 3,0 až 20,0. Prevedením do diskrétnej časovej oblasti dostaneme diferenčný vzťah pre derivačnú zložku:

$$d(k) = \frac{T_d \cdot d(k-1) + K_P \cdot T_s \cdot \alpha [e(k) - e(k-1)]}{T_d + \alpha \cdot T_s} \tag{42.}$$

[205] Používané substitučné metódy pri výpočte parametrov regulátora v diskrétnej oblasti:

$$s = \frac{2}{T_s} \cdot \frac{z-1}{z+1}; \quad s = \frac{z-1}{T \cdot z}; \quad s = \frac{z-1}{T};$$

Tabuľka 19, Voľba periódy vzorkovania **T**s podľa typu regulovaného procesu

Vzorkovacia perióda [s]	**Regulovaný proces**
10,0 µs až 500,0 µs	Presné riadenie a modelovanie, elektrické a energetické systémy, presné riadiace roboty ...
0,5 ms až 20,0 ms	Stabilizácia výkonových dynamických systémov, letové simulátory, rôzne simulačné trenažéry ...
10,0 ms až 100,0 ms	Spracovanie obrazu, virtuálna realita, umelá inteligencia a rozpoznávanie okolia ...
0,5 s až 1,0 s	Monitorovanie a riadenie objektov, riadenie zložitých chemických procesov, riadenie rôznych typov elektrární ...
1,0 s až 3,0 s	Regulácie prietokov kvapalín a plynov ...
1,0 s až 5,0 s	Regulácia tlaku technických kvapalín a plynov ...
5,0 s až10,0 s	Regulácia výšky hladiny technických kvapalín ...
10,0 s až 20,0 s	Regulácia teploty v bytových výmenníkových staniciach ...

Tabuľka 20, Aplikačné a optimálne hodnoty parametrov regulátora, (Balátě, 2003)

Typ regulátora	$G_{PID}(s) = K_p \cdot \left(1 + \frac{1}{T_i \cdot s} + T_d \cdot s\right)$		
	$k_{r*} \equiv r_{0*}$	T_{i*}	T_{d*}
P	$\frac{T_n}{T_u} \cdot \frac{1}{k_s}$	-	-
PI	$0{,}90 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s}$	$3{,}50 \cdot T_u$	-
PD	$1{,}20 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s}$	-	$\frac{1}{4} \cdot T_u$
PID	$1{,}25 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s}$	$2{,}00 \cdot T_u$	$\frac{1}{2} \cdot T_u$

Tabuľka 21, Aplikačné hodnoty parametrov jednotlivých regulátorov, (Balátě, 1996)

Typ regulátora	$G_{PID}(s) = r_0 + \frac{r_{-1}}{s} + r_1 \cdot s$		
	r_0	r_{-1}	r_1
P	$\frac{T_n}{T_u} \cdot \frac{1}{k_s}$	-	-
PI	$0{,}90 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s}$	$0{,}26 \cdot \frac{T_n}{T_u^2} \cdot \frac{1}{k_s}$	-
PD	$1{,}20 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s}$	-	$0{,}30 \cdot T_n \cdot \frac{1}{k_s}$
PID	$1{,}25 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s}$	$0{,}63 \cdot \frac{T_n}{T_u^2} \cdot \frac{1}{k_s}$	$0{,}63 \cdot T_n \cdot \frac{1}{k_s}$

T_u - čas prieťahu [s]

T_n - čas nábehu [s]

k_s - koeficient prenosu sústavy [-]

Tabuľka 22, Nastavenie parametrov diskrétneho regulátora z prechodovej charakteristiky

Typ regulátora	$\boldsymbol{G_{PID}(s) = K_p \cdot \left(1 + \frac{1}{T_i \cdot s} + T_d \cdot s\right)}$		
	$C_p = K = q_0 - q_2$	$C_i = \frac{T}{T_i} = \frac{q_0 + q_1 + q_2}{K}$	$C_d = \frac{T_d}{T} = \frac{q_2}{K}$
P	$\frac{1}{K_p} \cdot \frac{T_n}{T_u + T_n}$	0	0
PS	$\frac{1}{K_p} \cdot \left[\frac{0{,}90 \cdot T_n}{T_u + 0{,}50 \cdot T} - \frac{0{,}13 \cdot T_n \cdot T}{(T_u + 0{,}50 \cdot T)^2}\right]$	$\frac{1}{K_p} \cdot \frac{0{,}27 \cdot T_n \cdot T}{(T_u + 0{,}50 \cdot T)^2}$	-
PSD	$\frac{1}{K_p} \cdot \left[\frac{1{,}20 \cdot T_n}{T_u + T} - \frac{0{,}30 \cdot T_n \cdot T}{(T_u + 0{,}50 \cdot T)^2}\right]$	$\frac{1}{K_p} \cdot \frac{0{,}60 \cdot T_n \cdot T}{(T_u + 0{,}50 \cdot T)^2}$	$\frac{1}{K_p} \cdot \frac{0{,}50 \cdot T_n}{T}$

Riadiaci zásah určíme podľa Takahashiho[206] metódy uvedenej v (HRUBINA, et al., 2005):

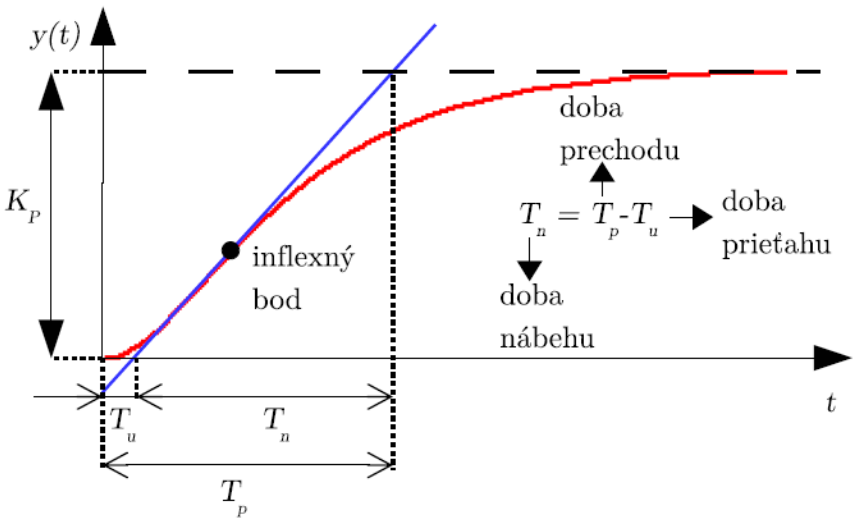
$$u(k) = u(k-1) + \underbrace{k_p}_{C_p} \cdot \left[-y(k) + y(k-1) + \underbrace{\frac{T}{T_i}}_{C_i} \cdot \big(w(k) - y(k)\big) + \underbrace{\frac{T_d}{T}}_{C_d} \cdot \big(2 \cdot y(k-1) - y(k-2) - y(k)\big) \right]$$

$$u(k) = u(k-1) + K_p \cdot \big(-y(k) + y(k-1)\big) + T_i \cdot \big(w(k) - y(k)\big) + T_d \cdot \big(2 \cdot y(k-1) - y(k-2) - y(k)\big)$$

$$k_p = K_p,\ T_i = \frac{k_p \cdot T}{K_i},\ T_d = \frac{K_d \cdot T}{k_p}, \qquad (43.)$$

Dosadením k_p, C_i, C_p získame rekurentný výpočet zásahu regulátora do regulovanej sústavy ktorý je možné jednoduchým spôsobom prepísať do ľubovoľného programovacieho jazyka.

Obrázok 199, Prechodová charakteristika regulovaného procesu



[206] Takahashiho vzťah PSD regulátora zaručuje kvalitné zásahy regulátora v procese aj pri malých zmenách v regulačnej odchýlky. Vzťah $\Delta u(k) = u(k) - u(k-1) = q_0 \cdot e(k) + q_1 \cdot e(k-1) + q_2 \cdot e(k-2)$ známy z predchádzajúcich kapitol je menej citlivý na zmeny $w(k) = e(k) + y(k)$ a poskytuje menej kmitavé, astabilné stavy. Takahashiho regulátor je použiteľný ako robustný PSD regulátor pri adaptívnom type riadenia.

Tabuľka 23, Výpočet parametrov regulátora podľa (ir. drs. E.H.W. van de Logt, 2011)

Metóda + (parameter)	K_c [% .°C^{-1}]	T_i [s]	T_d [s]
Ziegler-Nichols (Open-Loop)	$K_c = 1{,}20 \cdot \left(\frac{1}{\theta \cdot a^*}\right)$	$T_i = 2{,}00 \cdot \theta$	$T_d = \frac{1}{2} \cdot \theta$
Ziegler-Nichols (Open-Loop)	$K_c = 0{,}90 \cdot \left(\frac{1}{\theta \cdot a^*}\right)$	$T_i = 3{,}33 \cdot \theta$	-
Ziegler-Nichols (Closed-Loop)	$K_c = 1{,}20 \cdot \left(\frac{\tau_{hlt}}{K_{hlt} \cdot \theta}\right)$	$T_i = 2{,}00 \cdot \theta$	$T_d = \frac{1}{2} \cdot \theta$
Ziegler-Nichols (Closed-Loop)	$K_c = 0{,}90 \cdot \left(\frac{\tau_{hlt}}{K_{hlt} \cdot \theta}\right)$	$T_i = 3{,}33 \cdot \theta$	-
Cohen-Coon	$K_c = \frac{\tau_{hlt}}{K_{hlt} \cdot \theta} \cdot \left(\frac{\theta}{4 \cdot \tau_{hlt}} + \frac{4}{3}\right)$	$T_i = \theta \cdot \left(\frac{32 \cdot \tau_{hlt} + 6 \cdot \theta}{13 \cdot \tau_{hlt} + 8 \cdot \theta}\right)$	$T_d = \theta \cdot \left(\frac{4{,}0 \cdot \tau_{hlt}}{2 \cdot \theta + 11 \cdot \tau_{hlt}}\right)$
Cohen-Coon	$K_c = \frac{\tau_{hlt}}{K_{hlt} \cdot \theta} \cdot \left(\frac{\theta}{12 \cdot \tau_{hlt}} + \frac{9}{10}\right)$	$T_i = \theta \cdot \left(\frac{30 \cdot \tau_{hlt} + 3 \cdot \theta}{9 \cdot \tau_{hlt} + 20 \cdot \theta}\right)$	-
ITAE-Load	$K_c = 1{,}357 \cdot \frac{1}{K_{hlt}} \cdot \left(\frac{\theta}{\tau_{hlt}}\right)^{-0{,}947}$	$T_i = \left(\frac{\tau_{hlt}}{0{,}842}\right) \cdot \left(\frac{\theta}{\tau_{hlt}}\right)^{+0{,}738}$	$T_d = 0{,}381 \cdot \tau_{hlt} \cdot \left(\frac{\theta}{\tau_{hlt}}\right)^{+0{,}995}$
ITAE-Load	$K_c = 0{,}859 \cdot \frac{1}{K_{hlt}} \cdot \left(\frac{\theta}{\tau_{hlt}}\right)^{-0{,}977}$	$T_i = \left(\frac{\tau_{hlt}}{0{,}674}\right) \cdot \left(\frac{\theta}{\tau_{hlt}}\right)^{+0{,}680}$	-

$\boldsymbol{\Theta}$ - „Death time“ [s]

$\boldsymbol{\tau_{hlt}}$ - časová konštanta HLT systému [s]

$\boldsymbol{K_{hlt}}$ - zosilnenie HLT [°C.%$^{-1}$]

Pri jednotkovom vstupnom skoku je možné tvrdiť pri proporcionálnych sústavách[207], že koeficient prenosu $\mathbf{k_s}$ je daný ustáleným stavom na prechodovej charakteristike. Ak vstupný signál nie je jednotkový skok, tak ako uvádza (Balátě, 2003 s. 129) koeficient prenosu sústavy $\mathbf{k_s}$ bude pomer zmeny regulovanej veličiny $\Delta\mathbf{y}$ k zmene akčnej veličiny $\Delta\mathbf{u}$ v ustálenom stave. (ŠVARC, 2003 s. 47) uvádza $\mathbf{k_s}$ definovanú ako konštantu sústavy s ustálenou hodnotou prechodovej charakteristiky. Obrázok 209 vyššie uvedené tvrdenie názornejšie vysvetľuje na charakteristike proporcionálnej prechodovej aperiodickej sústavy.

[207] Myslíme tým „nekmitavú“ (aperiodickú) proporcionálnu sústavu

Koeficient prenosu regulovanej sústavy vyjadríme v tom prípade ako:

$$k_s = \frac{\Delta y}{\Delta u} \quad (44.)$$

Ak **a*** definujeme ako normalizovaný nárast výstupnej veličiny na jednotkový skok, pričom priemerný nárast **a** je definovaný ako:

$$a = \frac{\Delta T}{\Delta t} \quad (45.)$$

potom je možné zapísať

$$\boldsymbol{a}^* = \frac{\Delta T}{\Delta t} \cdot \frac{1}{\Delta p} = a \cdot \frac{1}{\Delta p} \quad (46.)$$

kde **Δp** [%] je percentuálna zmena na výstupe PID regulátora

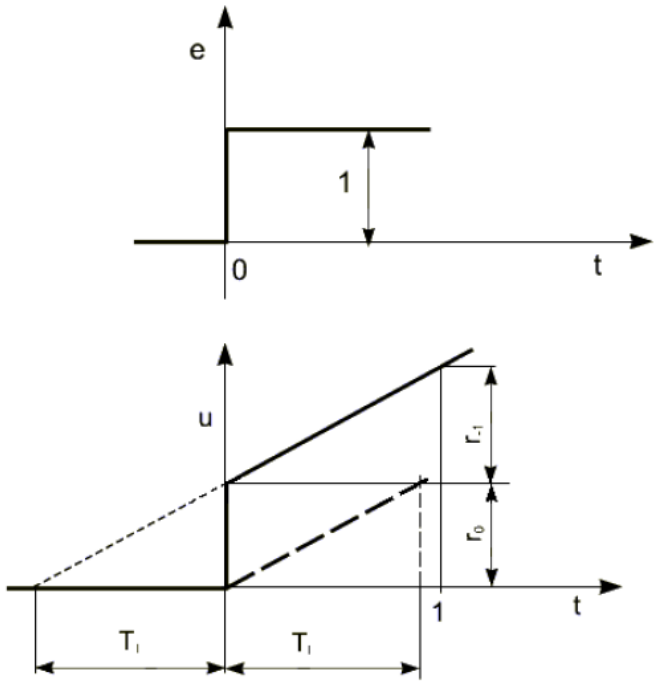
Pásmo proporcionality **pp** [%] udáva o akú hodnotu vyjadrenú v percentách sa musí zmeniť vstupná hodnota regulátora[208], aby sa akčný člen – servopohon prestavil z jednej krajnej polohy do druhej, t.j. z 0% na 100%, alebo naopak.

Integračná časová konštanta $\mathbf{T_i}$ [s] udáva čas ktorý by potreboval čistý integračný regulátor **I** aby prestavil akčný člen do polohy ktorú dosiahne PI regulátor v čase t=0 vplyvom vlastnej proporcionálnej zložky pre vstupný jednotkový skok.

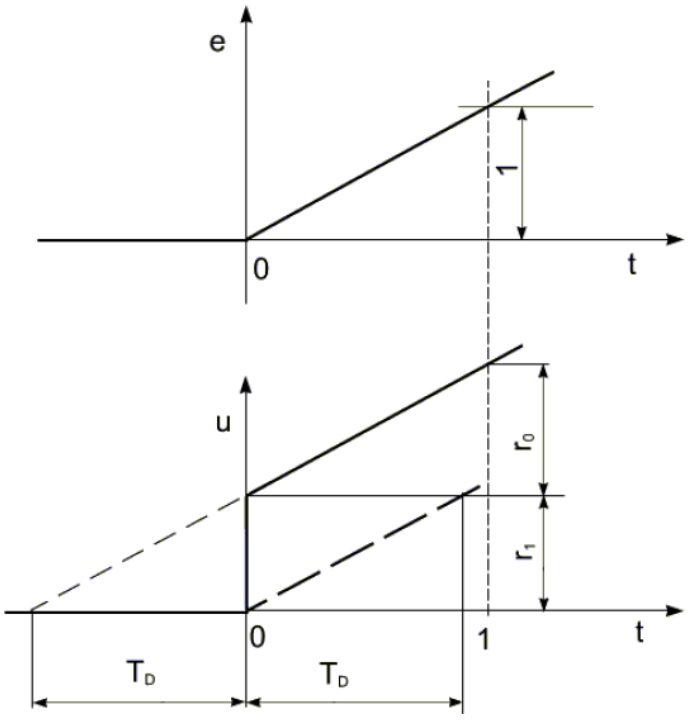
Derivačná časová konštanta $\mathbf{T_d}$ [s] udáva čas, ktorý by potreboval čistý proporcionálny regulátor P, aby prestavil akčný člen do polohy ktorú dosiahne PD regulátor v čase t=0 vplyvom svojej derivačnej zložky.

[208] Meraná veličina, napr. teplota, napätie, prúd, tlak, množstvo ....

Obrázok 200, Prechodová charakteristika PI regulátora



Obrázok 201, Prechodová charakteristika PD regulátora



Tabuľka 24, Charakteristika činnosti spojitých regulátorov podľa (Balátě, 1996)

Regulátor	Vlastnosti regulátoru
P	Je vhodný k regulácii proporcionálnych aj integračných sústav zo zotrvačnosťou 1. rádu a zo strednou časovou konštantou, prípadne s menším dopravným oneskorením. Má trvalú regulačnú odchýlku, ale pomerne dobré stabilné vlastnosti.
I	Je vhodný k regulácii sústav proporcionálnych zo zotrvačnosťou 1. rádu s malou časovou konštantou, bez dopravného oneskorenia. Pracuje bez trvalej regulačnej odchýlky. Nevyhovuje podmienkam stability, pokiaľ reguluje astatický typ regulovanej sústavy.
PI	Je vhodný k regulácii sústav proporcionálnych i integračných zo zotrvačnosťou vyššieho rádu s ľubovoľnými časovými konštantami, s veľkým dopravným oneskorením. Vplyvom I zložky odstraňuje trvalú regulačnú odchýlku, P zložka zlepšuje stabilní vlastnosti regulačného obvodu.
PD	Je vhodný k regulácii sústav proporcionálnych i integračných zo zotrvačnosťou vyššieho rádu zo strednými časovými konštantami, s veľkým dopravným oneskorením. Obsahuje trvalú regulačnú odchýlku. Regulačný obvod s týmto regulátorom má lepšie stabilné vlastnosti než pri použití len samostatného proporcionálneho regulátoru.
PID	PID[209] regulátor je vhodný k regulácii sústav proporcionálnych i integračných zo zotrvačnosťou vyššieho rádu s ľubovoľnými časovými konštantami, s dlhším dopravným oneskorením. Vplyvom I zložky odstraňuje trvalú regulačnú odchýlku, vplyvom D zložky zlepšuje stabilní vlastnosti regulačného obvodu (otáča fázu amplitúdovej fázovej charakteristiky v komplexnej rovine o +90°. Informuje regulátor o zmene regulačnej odchýlky a teda regulátor môže v "predstihu" na tuto zmenu reagovať).

[209] Viac ako v 95% priemyslových regulačných aplikáciách sa používajú kombinované PID regulátory, i keď vo väčšine prípadov s vypnutou derivačnou zložkou. Použitie samostatného „čistého“ I regulátoru je veľmi malé. Pre väčšinu technologických procesov však vyššie uvedené regulátory P, PI úplne postačia.

17.3. Spojité lineárne riadenie s regulátorom

Riadenie, kde je v zapojení použitá spätná väzba sa nazýva regulácia. Reguláciou rozumieme nastavenie technických veličín na požadovanú hodnotu pričom ich udržiavame na požadovanej hodnote aj pri pôsobení poruchovej veličiny. Reguláciu zabezpečuje regulačný obvod.

Regulovaná sústava je samotným objektom regulácie, pričom je regulovaná niektorá z jej veličín.

Regulovaná veličina je veľkosť výstupnej hodnoty z regulovanej sústavy ktorá sa reguláciou udržuje na požadovanej hodnote. Označujeme ju najčastejšie ako ***y***.

Riadiaca veličina *w* je veličina pomocou ktorej nastavujeme hodnotu ktorú má dosahovať regulovaná veličina a predstavuje žiadanú hodnotu regulovanej veličiny.

Regulačná odchýlka e je rozdiel medzi regulovanou t.j. žiadanou a skutočnou hodnotou. V prípade nenulovej regulačnej odchýlky vykoná regulátor požadovaný zásah v regulovanej sústave. Regulačnú odchýlku môžeme vypočítať podľa vzťahu:

$$e = w - y \tag{47.}$$

Akčná veličina u je regulačný zásah do regulačného procesu, aby sa regulačná odchýlka dosahovala nulová, alebo čo najmenšia.

Poruchové veličiny $v_1(t)$, $v_2(t)$, $v_3(t)$… sú nežiadúce pôsobenia na regulovaný proces hlavnou príčinou prečo musíme regulovať. Ich pôsobenie sa uplatňuje v čase a nepredvídateľným spôsobom ovplyvňujú regulačný proces.

Diferenciálna rovnica systému, alebo regulačného členu získame uplatnením matematicko-fyzikálnym opisom vzťahov a zákonov v regulovanom systéme, pričom vyradíme všetky ostatné veličiny okrem vstupných a výstupných veličín.

Prenos regulátora je najčastejšie používaným spôsobom opisu lineárnych regulačných členov a je definovaný ako pomer výstupnej veličiny a vstupnej veličiny vyjadriteľnej Laplaceovým obrazom pri nulových počiatočných podmienkach.

$$G(s) = \frac{L\{y(t)\}}{L\{u(t)\}} = \frac{Y(s)}{U(s)} \quad (48.)$$

$$G(s) = \frac{b_m s^m + \cdots + b_1 s + b_0}{a_n s^n + \cdots + a_1 s + a_0} \quad (49.)$$

Podmienka fyzikálnej realizovateľnosti bude splnená ak stupeň polynómov v čitateli bude menší, alebo rovnaký ako stupeň polynómu v menovateli prenosu **G(s)**.

Príklad

Majme diferenciálnu rovnicu opisujúci systém v nasledujúcom tvare:

$$y''' + 4y'' + 0.5y' + 2y = 6u' + 3u$$

Prepisom do vhodného tvaru vytvoríme prenos systému v tvare:

$$G(s) = \frac{6 \cdot s + 3}{s^3 + 4 \cdot s^2 + 0.5 \cdot s + 2}$$

Zo vzťahu $\boldsymbol{G(s) = \frac{Y(s)}{U(s)}}$ vyplýva, že ak poznáme vstupný signál systému ***u(t)***, alebo jeho obraz ***U(s)*** môžeme z prenosu ***G(s)*** určiť odozvu ***y(t)*** systému na ľubovoľnú zmenu vstupnej veličiny ***u(t)***. Pretože prenos ***G(s)*** je definovaný za predpokladu nulových počiatočných podmienok tak aj obraz odozvy je možné vypočítať len za predpokladu nulových počiatočných podmienok.

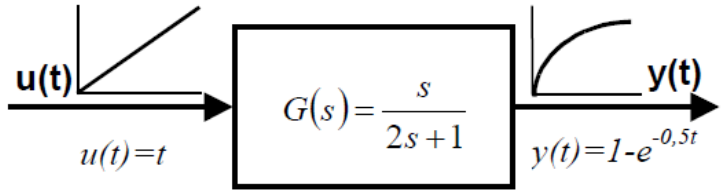
$$Y(s) = G(s) \cdot U(s) \quad (50.)$$

$$y(t) = L^{-1}\{G(s) \cdot U(s)\} \quad (51.)$$

Príklad

Vypočítajte odozvu regulačného členu s uvedeným prenosom na vstupnú veličinu ***u(t),*** ktorá lineárne rastie pre ***t>0***.

Obrázok 202, Odozva regulačného členu s prenosom na vstupnú veličinu



Obraz vstupnej veličiny podľa Laplaceovej transformácie je:

$$U(s) = \frac{1}{s^2}$$

Obraz odozvy môžeme vyjadriť ako:

$$Y(s) = G(s) \cdot U(s) = \left(\frac{s}{2 \cdot s + 1}\right) \cdot \left(\frac{1}{s^2}\right) = \frac{1}{s \cdot (2 \cdot s + 1)} = \frac{1}{2 \cdot s^2 + s} = \left(\frac{1}{s} - \frac{1}{s + 0.5}\right)$$

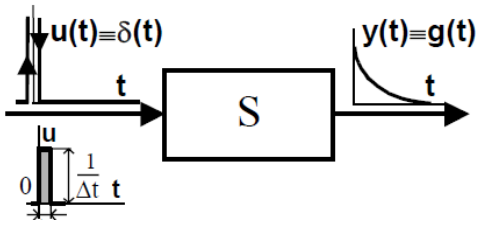
Odozva potom bude:

$$y(t) = L^{-1}\left\{\frac{1}{s} - \frac{1}{s + 0.5}\right\} = 1 - e^{-0.5 \cdot t}$$

17.4. Impulzová funkcia a jej charakteristika

Impulzová funkcia je odozva systému na jednotkový impulz na vstupe systému ***δ(t)*** a jej graf je impulzová charakteristika.

Obrázok 203, Graf impulzovej funkcie



Medzi danou impulzovou funkciou a jej prenosom je vzťah ako medzi originálom a obrazom v Laplaceovej transformácii. Vďaka tomu ju označujeme ako ***g(t)***.

$$g(t) = L^{-1}\{G(s)\}$$

Príklad

Určite impulzovú funkciu a nakreslite impulzovú charakteristiku pre regulačné členy s prenosmi:

$$G(s) = \frac{1.5}{s}$$

$$g(t) = L^{-1}\{G(s)\} = \frac{1.5}{s}$$

$$G(s) = \frac{0.5}{s^2}$$

$$g(t) = L^{-1}\{G(s)\} = 0.5 \cdot t$$

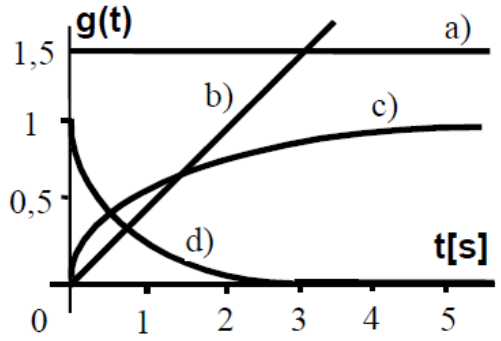
$$G(s) = \frac{1}{s \cdot (2 \cdot s + 1)}$$

$$g(t) = L^{-1}\{G(s)\} = L^{-1}\left\{\frac{1}{s \cdot (2 \cdot s + 1)}\right\} = L^{-1}\left\{\frac{1}{s} - \frac{1}{s + 0.5}\right\} = 1 - e^{-0.5 \cdot t}$$

$$G(s) = \frac{1}{(s + 2)}$$

$$g(t) = L^{-1}\{G(s)\} = L^{-1}\left\{\frac{1}{(s + 2)}\right\} = e^{-2 \cdot t}$$

Obrázok 204, Impulzové charakteristiky funkcie

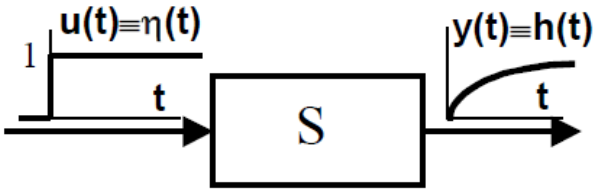


17.5. Prechodová funkcia a jej charakteristika

Prechodová funkcia je odozva systému na jednotkový skok ***η(t)*** na vstupe a označujeme ju ako ***h(t)***. Graf funkcie je prechodová charakteristika. Laplaceov obraz jednotkového skoku je:

$$L\{\eta(t)\} = \frac{1}{s}$$

Obrázok 205, Prechodová funkcia systému na jednotkový skok



$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\}$$

Príklad

Určite prechodovú funkciu a jej prechodovú charakteristiku pre nasledujúce regulačné členy ktorých jednotlivé prenosy sú:

$$G(s) = \frac{1.5}{s}$$

$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = L^{-1}\left\{\frac{1.5}{s^2}\right\} = 1.5 \cdot t$$

$$G(s) = \frac{0.5}{s^2}$$

$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = L^{-1}\left\{\frac{0.5}{s^3}\right\} = 0.25 \cdot L^{-1}\left\{\frac{2}{s^3}\right\} = 0.25 \cdot t^2$$

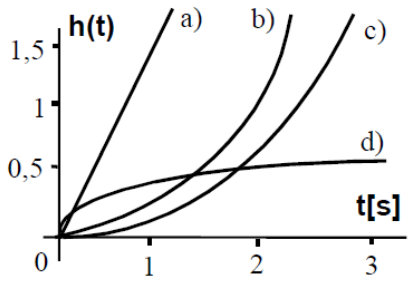
$$G(s) = \frac{1}{s \cdot (2 \cdot s + 1)}$$

$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = L^{-1}\left\{\frac{1}{s^2 \cdot (2 \cdot s + 1)}\right\} = L^{-1}\left\{\frac{1}{s^2} - \frac{2}{s} + \frac{4}{2 \cdot s + 1}\right\} = t - 2 + 2 \cdot e^{-0.5 \cdot t}$$

$$G(s) = \frac{1}{(s + 2)}$$

$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = L^{-1}\left\{\frac{1}{s \cdot (s + 2)}\right\} = L^{-1}\left\{\frac{1}{2 \cdot s} - \frac{1}{2 \cdot (s + 2)}\right\} = 0.5 \cdot (1 - e^{-2 \cdot t})$$

Obrázok 206, Prechodové charakteristiky regulačných členov



Impulzovú funkciu získame derivovaním prechodovej funkcie podľa času.

$$g(t) = \frac{d}{dt}(1.5 \cdot t) = 1.5$$

$$g(t) = \frac{d}{dt}(0.25 \cdot t^2) = 0.5 \cdot t$$

$$g(t) = \frac{d}{dt}(t - 2 + 2 \cdot e^{-0.5 \cdot t}) = 1 - e^{-0.5 \cdot t}$$

$$g(t) = \frac{d}{dt}\left(0.5 \cdot (1 - e^{-2 \cdot t})\right) = e^{-2 \cdot t}$$

Z vyššie uvedených výsledkov je vidno, že impulzové funkcie sú zhodné s z výsledkami z ktorých boli impulzové funkcie určované.

Príklad

Určite prechodové funkcie regulačných členov z nasledujúcich prenosov:

$$G(s) = \frac{s}{2 \cdot s + 1}$$

$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = L^{-1}\left\{\frac{\left(\frac{s}{2 \cdot s + 1}\right)}{s}\right\} = \frac{e^{-\frac{t}{2}}}{2} = 0.5 \cdot e^{-0.5 \cdot t}$$

$$G(s) = \frac{2}{3 \cdot s + 1}$$

$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = L^{-1}\left\{\frac{\left(\frac{2}{3 \cdot s + 1}\right)}{s}\right\} = L^{-1}\left\{\frac{2}{s} - \frac{6}{3 \cdot s + 1}\right\} = -2 \cdot \left(e^{-\frac{t}{3}} - 1\right) =$$

$$= 2 \cdot (1 - e^{-0.33 \cdot t})$$

$$G(s) = \frac{1}{(s+1) \cdot (s+2) \cdot (s+3)}$$

$$h(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = L^{-1}\left\{\frac{\frac{1}{(s+1) \cdot (s+2) \cdot (s+3)}}{s}\right\} =$$

$$= L^{-1}\left\{\frac{1}{s \cdot (s+1) \cdot (s+2) \cdot (s+3)}\right\} = \frac{1}{6} - \frac{e^{-t}}{2} + \frac{e^{-2 \cdot t}}{2} - \frac{e^{-3 \cdot t}}{2} =$$

$$= -\frac{(e^{-t} - 1)}{6}$$

17.6. Stabilita regulačného obvodu

Stabilitou regulačného obvodu myslíme schopnosť ustáliť sa do rovnovážneho stavu po ukončení poruchovej veličiny ktorá ho z rovnovážneho stavu vychýlila. Stabilita je základnou a nevyhnutnou podmienkou správnej funkcie regulačného obvodu.

Regulačný obvod je stabilný ak všetky korene s_1, s_2, ...s_N charakteristickej rovnice sú záporné a v prípade komplexných koreňov majú zápornú reálnu časť komplexného tvaru. Vyššie uvedené znamená, že regulačný obvod je stabilný ak majú všetky korene charakteristickej rovnice záporné reálne časti, alebo ležia v ľavej časti polovice komplexnej charakteristiky.

V prípade, že niektorý z koreňov charakteristickej rovnice leží na imaginárnej osi a žiadny neleží v pravej časti komplexnej polovice, nachádza sa regulačný obvod na hranici stability.

Vychádzame zo štandardného prenosu regulátora ktorý je uvedený v tvare:

$$G_{PID}(s) = K_P\left(1 + \frac{1}{T_i \cdot s} + T_d \cdot s\right) \quad (52.)$$

Štandardný prenos regulátora z predchádzajúceho vzťahu máme vyjadrený v tvare:

$$G_R(s) = 3 \cdot \left(1 + \frac{1}{4 \cdot s} + 8 \cdot s\right) \quad (53.)$$

Štandardný prenos sústavy je vyjadrený v tvare:

$$G_S(s) = \left(\frac{2}{3 \cdot s + 1}\right) \quad (54.)$$

Charakteristická rovnica je vyjadrená ako súčin prenosu sústavy **$G_S(s)$** a prenosu regulátora **$G_R(s)$** a môžeme si vyjadriť túto rovnicu v tvare podielu jednotlivých polynómov.

Môžeme písať výsledný prenos otvorenej slučky:

$$G_0(s) = G_R(s) \cdot G_S(s) = \frac{M(s)}{N(s)} \quad (55.)$$

alebo po úprave dostaneme

$$1 + G_0(s) = 1 + \frac{M(s)}{N(s)} = \frac{M(s)+N(s)}{N(s)} = 0 \quad (56.)$$

Keďže zlomok je nulový keď čitateľ zlomku bude rovný nule, môžeme napísať charakteristickú rovnicu ako súčet polynómov čitateľa a menovateľa rozpojeného obvodu **$G_0(s)$**.

$$M(s) + N(s) = 0$$

Po dosadení získame:

$$G_0(s) = 3 \cdot \left(1 + \frac{1}{4 \cdot s} + 8 \cdot s\right) \cdot \left(\frac{2}{3 \cdot s + 1}\right) = \frac{3 \cdot (32 \cdot s^2 + 4 \cdot s + 1)}{2 \cdot (3 \cdot s + 1) \cdot s} = \frac{96 \cdot s^2 + 12 \cdot s + 3}{2 \cdot s \cdot (3 \cdot s + 1)}$$

$$G_0(s) = \frac{M(s)}{N(s)} = \frac{96 \cdot s^2 + 12 \cdot s + 3}{6 \cdot s^2 + 2 \cdot s}$$

$$M(s) + N(s) = (96 \cdot s^2 + 12 \cdot s + 3) + (6 \cdot s^2 + 2 \cdot s) = 102 \cdot s^2 + 14 \cdot s + 3$$

$$M(s) + N(s) = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 14 \\ 102 \end{bmatrix}$$

Z predchádzajúceho vzťahu vyplýva, že koeficienty **a_0, a_1, a_2 > 0** a teda korene budú záporné, alebo imaginárne, čím je teda splnená základná podmienka stability regulačného obvodu.

Charakteristické korene predchádzajúcej kvadratickej rovnice teda budú:

$$X_{1,2} = \frac{-14 \pm \sqrt[2]{14^2 - 4 \cdot 102 \cdot 3}}{2 \cdot 102} = \frac{-14 \pm \sqrt[2]{196 - 1224}}{2 \cdot 102} = -\frac{7}{102} \pm \frac{\sqrt[2]{257}}{102} \cdot j$$

$$X_{1,2} = \begin{bmatrix} -\frac{7}{102} + \frac{\sqrt[2]{257}}{102} \cdot j \\ -\frac{7}{102} - \frac{\sqrt[2]{257}}{102} \cdot j \end{bmatrix}$$

Ako je vidieť z predchádzajúcich vzťahov, výsledné korene rovnice majú zápornú reálnu časť, preto môžeme tvrdiť, že regulačný obvod je stabilný.

17.7. Experimentálne metódy syntézy regulačných obvodov

Tabuľka 25, Experimentálne nastavenie parametrov regulátora

Regulátor		Regulačný pochod na hranici aperiodicity		Regulačný pochod s 20% preregulovaním a minimálnym časom trvania kmitu	
		Riadenie	Porucha	Riadenie	Porucha
P	K_0	$0{,}3 \cdot \frac{T_n}{T_u}$	$0{,}3 \cdot \frac{T_n}{T_u}$	$0{,}6 \cdot \frac{T_n}{T_u}$	$0{,}6 \cdot \frac{T_n}{T_u}$
PI	K_0	$0{,}35 \cdot \frac{T_n}{T_u}$	$0{,}6 \cdot \frac{T_n}{T_u}$	$0{,}6 \cdot \frac{T_n}{T_u}$	$0{,}7 \cdot \frac{T_n}{T_u}$
	T_i	$1{,}2 \cdot T_u$	$4{,}0 \cdot T_u$	T_u	$2{,}3 \cdot T_u$
PID	K_0	$0{,}6 \cdot \frac{T_n}{T_u}$	$0{,}95 \cdot \frac{T_n}{T_u}$	$0{,}95 \cdot \frac{T_n}{T_u}$	$1{,}2 \cdot \frac{T_n}{T_u}$
	T_i	T_u	$2{,}4 \cdot T_u$	$1{,}35 \cdot T_u$	$2{,}0 \cdot T_u$
	T_d	$0{,}5 \cdot T_u$	$0{,}42 \cdot T_u$	$0{,}47 \cdot T_u$	$0{,}42 \cdot T_u$

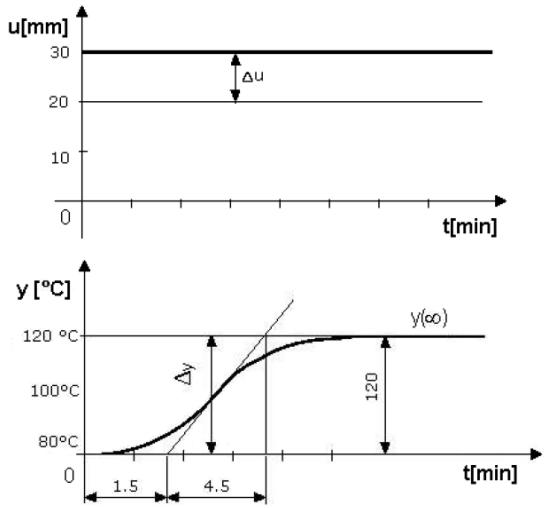
Vyššie uvedené experimentálne nastavenie jednotlivých parametrov regulátora bližšie vysvetľuje (ALEXÍK, a iní, 1993 s. 88).

17.8. Príklad výpočtu parametrov jednoduchého PID regulátora s otvorenou slučkou.

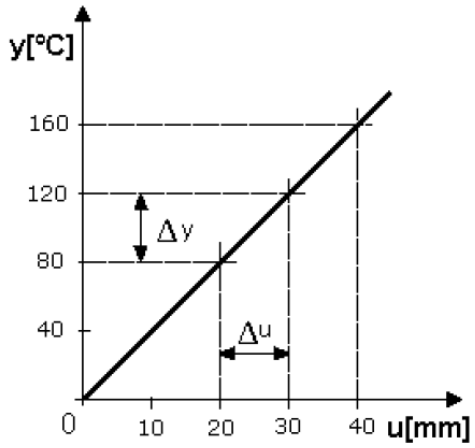
Maximálny uhol otvorenia proporcionálneho servopohonu[210] výmenníkovej stanice je pri maximálnom zdvihu **u=12mm**, **y=120°C**. Nastavte optimálne vzorkovanie[211] $\mathbf{T_s}$ regulátora. Koeficient prenosu (zosilnenia) sústavy $\mathbf{k_s}$ zistíme z existujúcej prechodovej charakteristiky regulovanej sústavy a vypočítame pomocou nasledujúceho vzťahu:

$$k_s = \frac{1}{a_0} = \frac{\Delta y}{\Delta u} = \frac{120°\mathrm{C}}{12\ mm} = 10°\mathrm{C} \cdot mm^{-1}$$

Obrázok 207, Prechodová charakteristika tepelného výmenníku bytovej stanice



Obrázok 208, Statická charakteristika tepelného výmenníku bytovej stanice (Balátě, 1996)



[210] Napr. servopohon AMV 23, prípadne AMV 33 ktorý je používaný v odovzdávacích bytových (výmenníkových) staniciach do maximálne použiteľnej teploty 150°C.

[211] V CMSIS Library je pri PID nastavené implicitné vzorkovanie s periódou T_s=100ms, čo stačí pre väčšinu aplikačných oblastí. V praxi sa osvedčilo použiť minimálne $T_s \leq 0{,}1 \cdot T_i$, prípadne z najviac dominujúcej časovej konštanty.

Činiteľ autoregulácie **a₀** a regulovateľnosť **α** potom bude možné zapísať nasledujúce vzťahy:

$$a_0 = \frac{1}{k_s}$$

$$\propto = \frac{T_u}{T_n}$$

$$k_{r*} = 1{,}25 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s} = 1{,}25 \cdot \frac{4{,}5\ min}{1{,}5\ min} \cdot \frac{1}{10°C \cdot mm^{-1}} = 0{,}375\ mm \cdot °C^{-1}$$

Zosilnenie regulátora musí zabezpečiť, aby pri regulačnej odchýlke Δe=1°C na vstupe regulátora sa prestavil regulačný servopohon o $0{,}375\ mm \cdot °C^{-1}$.

$$T_i = 2{,}00 \cdot T_u = 2{,}00 \cdot 1{,}5\ min = 3{,}00\ min = 180{,}0\ s$$

$$T_d = \frac{1}{2} \cdot T_u = \frac{1}{2} \cdot 1{,}5\ \text{min} = 0{,}75\ min = 45{,}0\ s$$

$$r_0 = K_p = 1{,}25 \cdot \frac{T_n}{T_u} \cdot \frac{1}{k_s} = 0{,}375\ mm \cdot °C^{-1}$$

$$r_{-1} = \frac{r_0}{T_i} = \frac{0{,}375\ mm \cdot °C^{-1}}{3{,}00\ min} = 0{,}125 \cdot mm \cdot min^{-1} \cdot °C^{-1}$$

$$r_1 = r_0 \cdot T_d = 0{,}125 \cdot mm \cdot min^{-1} \cdot °C^{-1} \cdot 0{,}75\ min = 0{,}09375\ mm \cdot °C^{-1}$$

Vypočítame hodnotu **Δyr** ktorá udáva ku akej zmene regulovanej veličiny musí dôjsť pri zosilnení **kr*** [212] aby sa regulačný servopohon prestavil o maximálny zdvih **umax=12,00 mm** t.j. 100% z maximálneho rozsahu. Ak nie je uvedené ináč, tak Ts=100ms.

$$\Delta y_r = \frac{1}{k_{r*}} \cdot u_{max} = \frac{1}{0{,}375\ mm \cdot °C^{-1}} \cdot 12{,}0\ mm = 32{,}00°C$$

Pásmo proporcionality **pp** bude vyjadrené:

$$pp = \frac{\Delta y_r}{\Delta y} \cdot 100\% = \frac{32{,}00°\text{C}}{120°C} \cdot 100\% = 26{,}66\%$$

[212] s hviezdičkou je uvedená veličina s fyzikálnym rozmerom

Proporcionálna konštanta $\mathbf{k_r}$[213] ktorú nastavíme do PID regulátora bude:

$$k_r = \frac{1}{pp} \cdot 100\% = \frac{1}{26{,}66\%} \cdot 100\% = 3{,}75\ [-]$$

$$T_u = \theta = t_1 - t_0$$

$$T_n = t_2 - t_1$$

Konštanty PID regulátora je možné previesť na koeficienty použiteľné u PSD regulátora:

$$d_0 = +K_p \cdot \left(1 + \frac{T_s}{2 \cdot T_i} + \frac{T_d}{T_s}\right) = 3{,}75 \cdot \left(1{,}00 + \frac{0{,}10}{2 \cdot 180{,}0} + \frac{45{,}0}{0{,}10}\right) = 1{,}6913 \cdot 10^3$$

$$d_1 = -K_p \cdot \left(1 - \frac{T_s}{2 \cdot T_i} + \frac{2 \cdot T_d}{T_s}\right) = -3{,}75 \cdot \left(1{,}00 - \frac{0{,}10}{2 \cdot 180{,}0} + \frac{2 \cdot 45{,}0}{0{,}10}\right) = -3{,}378 \cdot 10^3$$

$$d_2 = +K_p \cdot \frac{T_d}{T_s} = 3{,}75 \cdot \frac{45{,}0}{0{,}10} = 1{,}688 \cdot 10^3$$

Použitím koeficientov prenosu dostaneme vzťah použiteľný v regulátore PSD:

$$y_n = y_{n-1} + d_0 \cdot e_n + d_1 \cdot e_{n-1} + d_2 \cdot e_{n-2}$$

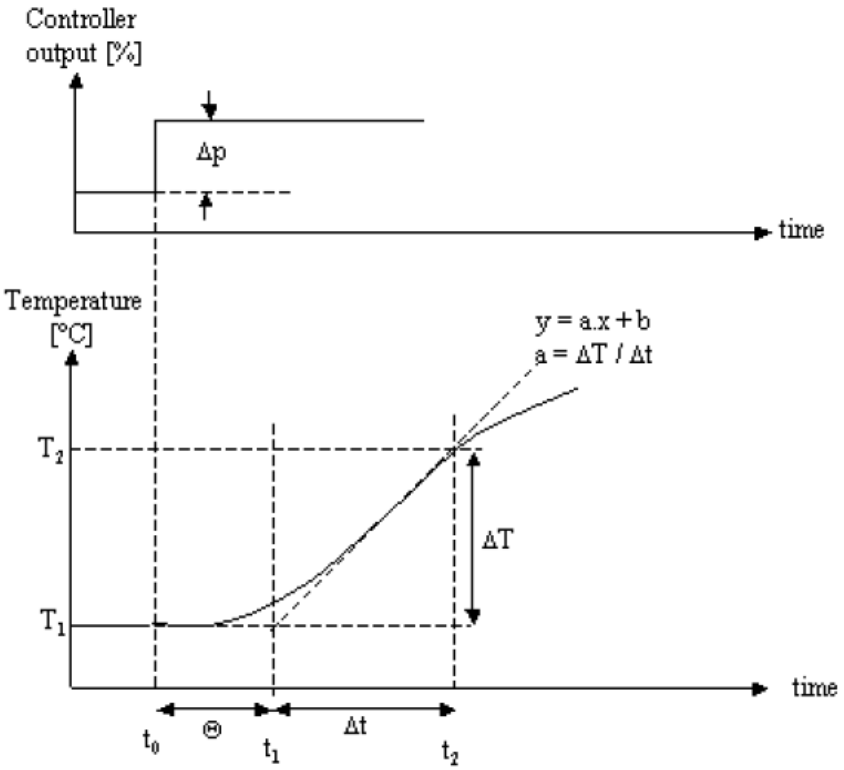
Z – prenos príslušnej funkcie regulátora potom bude:

$$G_R(z) = \frac{d_0 + d_1 \cdot z^{-1} + d_2 \cdot z^{-2}}{1 - z^{-1}} = \frac{1{,}6913 \cdot 10^3 - 3{,}378 \cdot 10^3 \cdot z^{-1} + 1{,}688 \cdot 10^3 \cdot z^{-2}}{1 - z^{-1}}$$

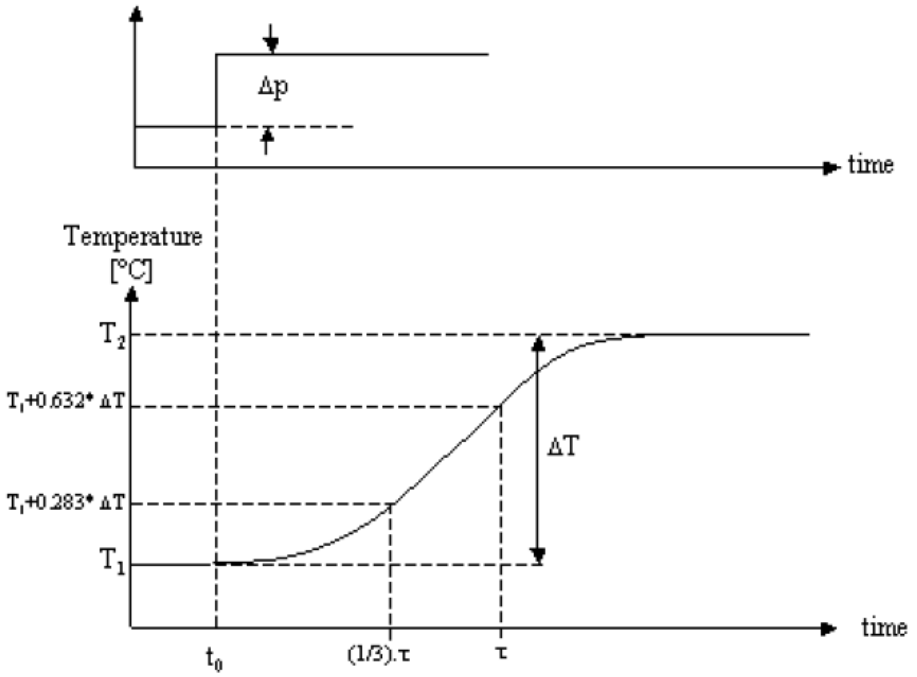
Z vyššie uvedených vzťahov je vidieť, že rovnica spojitého PID regulátora je značne idealizovaná k správaniu sa skutočného PID regulátora. Výpočet pri PSD regulátore nastáva presne podľa diferenčnej rovnice. To v praxi môže spôsobovať veľké problémy, pretože pri tomto type regulátora nedochádza k „prirodzenému“ útlmu veľkých zmien[214] regulačnej odchýlky ktoré sú v prípade klasického PID regulátora absorbované obvodmi realizujúcimi regulátor pomocou pasívnych a aktívnych polovodičových súčiastok.

[213] Veličina je bez fyzikálneho rozmeru, bezrozmerná veličina
[214] Pre obmedzenie je vhodné zaviesť filtrovanie derivačnej zložky

Obrázok 209, Odozva systému ohrevu na akčný zásah



Obrázok 210, Zistenie parametrov prechodovej charakteristiky regulovanej sústavy



17.9. Absolútny a prírastkový diskrétny regulátor PSD

Príklad: Pomocou programovacieho jazyka C51 a RTX51 vytvorte program, ktorý bude vykonávať funkciu regulátora s prírastkovým[215] algoritmom PSD pre reguláciu pohonu jednosmerného motora s cudzím budením s parametrami: **T_i=0,45s, T_d=0,02s, ΔT=20ms, K=0,22, U=240V, I_A=10A**. Žiadaná hodnota prúdu **$I_Ž$** bude vzorkovaná s periódou minimálne **ΔT=20 ms** na porte P5.0 a skutočná hodnota **I_S** na porte P5.1 ako 10 bitová analógová hodnota. Výstupná akčná veličina **y_N** na výstupe regulátora bude zapisovaná do registra PWM0.

Rozbor úlohy: Klasický tvar **PID** algoritmu regulácie je možné vyjadriť vzťahom:

$$y(t) = K_p \cdot \left[e(t) + \frac{1}{T_i} \cdot \int e(t)dt + T_d \cdot \frac{de(t)}{dt}\right] \quad (57.)$$

Absolútny tvar algoritmu regulátora **PSD** je možné zapísať pomocou vzťahu:

$$y_n = K_p \cdot \left[e_n + \frac{1}{T_i} \cdot \sum_{i=1}^{n} \frac{(e_i + e_{i-1})}{2} \cdot \Delta T + \frac{(e_n - e_{n-1})}{\Delta T} \cdot T_d\right] \quad (58.)$$

Prírastkový **PSD** je možné zapísať po jednoduchej úprave predchádzajúceho vzťahu[216]:

$$y_n = y_{n-1} + K_p \cdot \left[\left(1 + \frac{\Delta T}{2 \cdot T_i} + \frac{T_d}{\Delta T}\right) \cdot e_n - \left(1 - \frac{\Delta T}{2 \cdot T_i} + \frac{2 \cdot T_d}{\Delta T}\right) \cdot e_{n-1} + \left(\frac{T_d}{\Delta T}\right) \cdot e_{n-2}\right] \quad (59.)$$

t.j. po zjednodušení

$$y_n = y_{n-1} + A \cdot e_n + B \cdot e_{n-1} + C \cdot e_{n-2} \quad (60.)$$

kde **A, B, C** sú koeficienty prenosu a **e_N** je regulačná odchýlka žiadanej a skutočnej veličiny

$$A = +K_p \cdot \left(1 + \frac{\Delta T}{2 \cdot T_i} + \frac{T_d}{\Delta T}\right) \quad (61.)$$

$$B = -K_p \cdot \left(1 - \frac{\Delta T}{2 \cdot T_i} + \frac{2 \cdot T_d}{\Delta T}\right) \quad (62.)$$

$$C = +K_p \cdot \left(\frac{T_d}{\Delta T}\right) \quad (63.)$$

V prípade, že v prípade v **PSD** regulátore nevyžadujeme **D** zložku, celý vzťah bude mať tvar:

[215] Prírastkový PSD algoritmus oproti absolútnemu PSD „pripočítava“ len jednotlivé prírastky regulačnej odchýlky, čím sa dosahuje vyššia rýchlosť vykonávaniu algoritmu regulátora. Programovo vykonávame v prípade prírastkového PSD len 3 aritmetické súčty a súčiny, pričom absolútny PSD vyžaduje 4 aritmetické súčty a súčiny.
[216] Náhrada PID parametrov bola vykonaná pomocou sumy lichobežníkov pomocou tzv. „spätnej lichobežníkovej metódy“. Pri použití celočíselnej aritmetiky je tento vzťah vypočítaný pri 18,432MHz na mikroprocesore 80C552 za cca. 400µs, pri reálnej aritmetike je tento výpočet ukončený až za 1760µs. S výberom vypočítaných hodnôt z tabuľky uloženej v pamäti programu je tento výpočet ukončený asi za 150µs.

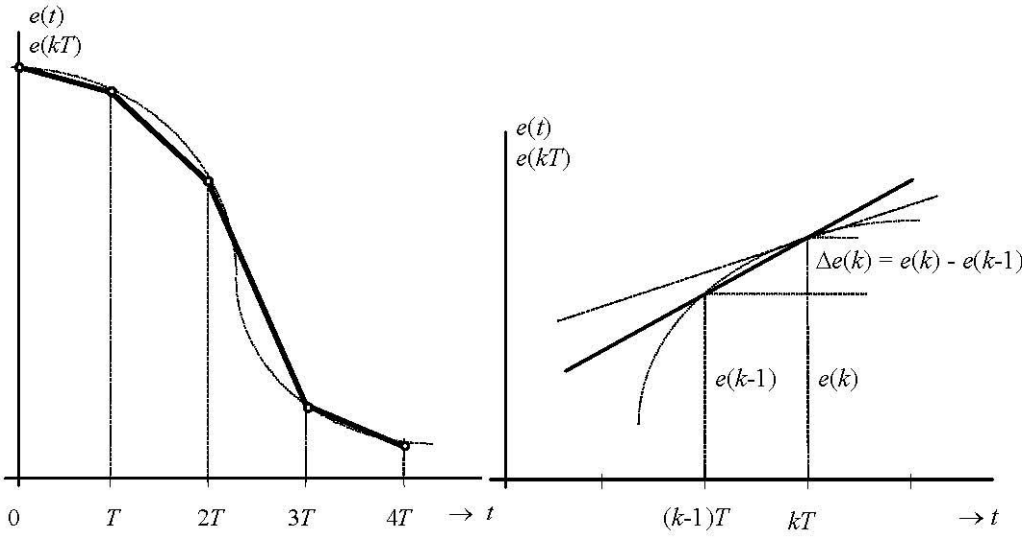
$$y_n = y_{n-1} + A \cdot e_n + B \cdot e_{n-1} \quad (64.)$$

$$A = +K_p \cdot \left(1 + \frac{\Delta T}{2 \cdot T_i} + \frac{T_d}{\Delta T}\right) \quad (65.)$$

$$B = -K_p \cdot \left(1 - \frac{\Delta T}{2 \cdot T_i} + \frac{2 \cdot T_d}{\Delta T}\right) \quad (66.)$$

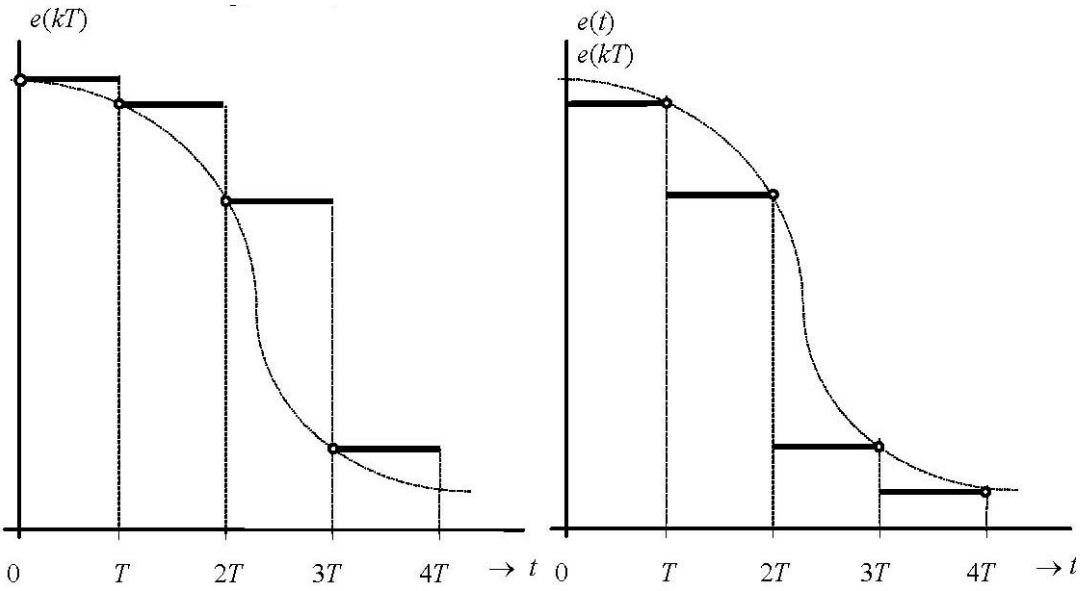
Obrázok 211, Náhrada integrálu sumou lichobežníkov

a. sprava b. zľava



Obrázok 212, Náhrada integrálu sumou obdĺžnikov

a. sprava b. zľava



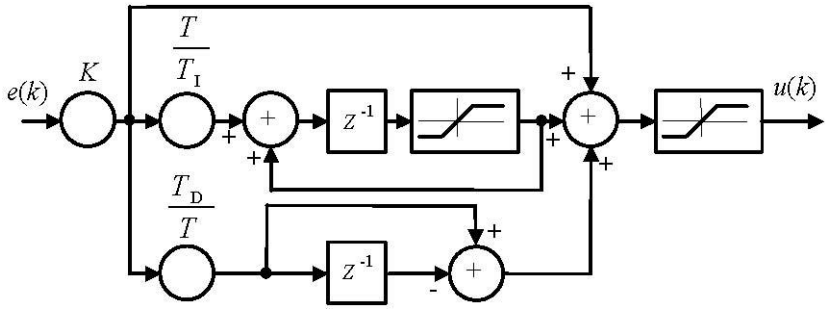
Pri použití náhrady integrálu obdĺžnikmi dostaneme výsledný vzťah:

$$y_n = y_{n-1} + K_p \cdot \left[\left(1 + \frac{T_d}{\Delta T}\right) \cdot e_n - \left(1 - \frac{\Delta T}{T_i} + \frac{2 \cdot T_d}{\Delta T}\right) \cdot e_{n-1} + \left(\frac{T_d}{\Delta T}\right) \cdot e_{n-2}\right] \quad (67.)$$

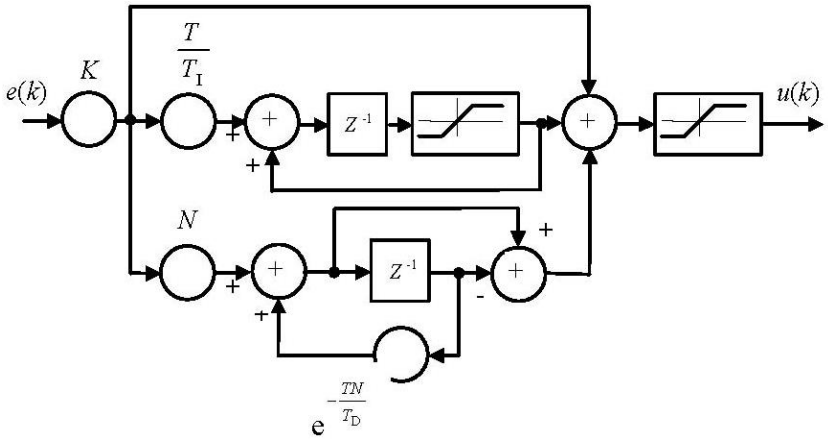
Pri použití náhrady integrálu lichobežníkmi dostaneme výsledný vzťah:

$$y_n = y_{n-1} + K_p \cdot \left[\left(1 + \frac{\Delta T}{2 \cdot T_i} + \frac{T_d}{\Delta T}\right) \cdot e_n - \left(1 - \frac{\Delta T}{2 \cdot T_i} + \frac{2 \cdot T_d}{\Delta T}\right) \cdot e_{n-1} + \left(\frac{T_d}{\Delta T}\right) \cdot e_{n-2}\right] \quad (68.)$$

Obrázok 213, Stavový diagram polohového PSD regulátora



Obrázok 214, Stavový diagram polohového PSD regulátora s filtráciou derivačnej zložky



Zápis programu:

```
#include <c:\keil\c51\inc\reg552.h>
#include <c:\keil\c51\inc\stdio.h>                    /* standard I/O .h-file          */
#include <c:\keil\c51\inc\ctype.h>                    /* character functions           */
#include <c:\keil\c51\inc\string.h>                   /* string and memory functions   */
#include <c:\keil\c51\inc\stdlib.h>
#include <c:\keil\c51\inc\rtx51.h>
#include <c:\keil\c51\inc\absacc.h>
#include <c:\keil\c51\inc\math.h>

#define    INIT              0
#define    COMMAND           1
#define    ANALOG            2
#define    CLOCK             3

xdata float K,Td,Ti,dT,U,Iz,Is;
xdata float a,b,c;
xdata float En[2];
xdata float Yn[2];

#define    CHANNELS          2
idata unsigned int Result_ADC[CHANNELS];

void Command(void) _task_ COMMAND _priority_ 1
{
 Iz=10;
 U=230.0;
 K=0.22;
 Ti=0.45;
 Td=0.2;
 dT=0.02;
 a=K*(1.0+(dT/(2*Ti))+(Td/dT));
 b=K*(1.0-(dT/(2*Ti))+(2*Td/dT));
 c=K*(Td/dT);
 Is=9.99;
 while(1)
  {
   os_wait(K_TMO,2,0);
   os_send_signal(ANALOG);
   Is=(float)Result_ADC[0];
   Iz=(float)Result_ADC[1];
   En[0]=Iz-Is;
   Yn[0]=Yn[1]+a*En[0]-b*En[1]+c*En[2];
   if(Yn[0]>255) Yn=255;
   if(Yn[0]<0) Yn=0;
   PWM0=(unsigned char)Yn;
   Yn[1]=Yn[0];
   En[2]=En[1];
   En[1]=En[0];
  }
}

xdata unsigned int Ticks;

void Clock(void) _task_ CLOCK _priority_ 1
{
 while(1)
  {
   os_wait(K_TMO,100,0);
   Ticks++;
  }
}
```

```
void Init(void) _task_ INIT _priority_ 0
{
 while(1)
  {
   os_set_slice(10000);
   os_create_task(COMMAND);
   os_create_task(CLOCK);
   os_create_task(ANALOG);
   os_delete_task(INIT);
  }
}

#pragma REGISTERBANK (3)

void Analog(void) _task_ ANALOG _priority_ 3
{
 #define ADCS 0x08
 #define ADCI 0x10
 #define ADEX 0x20
 unsigned char ADC_Channel;
 while(1)
  {
   os_wait(K_SIG,0xFF,0);
   for(ADC_Channel=0x00; ADC_Channel<CHANNELS; ADC_Channel++)
    {
     ADCON=ADC_Channel;
                                                                    //Vyberiem kanal ADC0..ADC7 pre prevod
     ADCON|=ADEX+ADCS;                                              //Externe spustanie prevodu+STDAC nast. ADEX=1
     while((ADCON&ADCI)==0x00);                                     //Pockam, pokial neskonci prevod
     ADCON^=ADCI;                                                   //Vysledok prevodu rusim ADCI=0
     Result_ADC[ADC_Channel]=(256*ADCH+(ADCON&0xC0)>>6);            //Vysledok v mV
    }
  }
 os_detach_interrupt(10);
}

#pragma REGISTERBANK (0)

void main(void)
{
 os_start_system(INIT);
}
```

17.10. 8-bitový PSD regulátor

```
#include <reg51.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

signed char KA,KB,KC,En,En1,En2,Yn,Yn1,Iz,Is;

void main(void)
{
 KA=1;
 KB=2;
 KC=3;
 while(1)
  {
   Iz=P1;
   Is=P3;
   En=Iz-Is;
   Yn=Yn1+(KA*En)-(KB*En1)+(KC*En2);
   Yn1=Yn;
   En2=En1;
   En1=En;
  }
}

```

17.11. 32-bitový PSD regulátor

```
#include <reg51.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

float KA,KB,KC,En,En1,En2,Yn,Yn1,Iz,Is,dT;

void Delay(long int Time)
{
 while(--Time!=0);
}

void main(void)
{
 KA=1.0;
 KB=2.0;
 KC=3.0;
 dT=0.5;
 while(1)
  {
   Iz=P1;
   Is=P3;
   En=Iz-Is;
   Yn=Yn1+(KA*En)-(KB*En1)+(KC*En2);
   Yn1=Yn;
   En2=En1;
   En1=En;
   Delay(10000);
  }
}

```

17.12. Jednosmerný impulzový menič v znižovacom zapojení STEP-DOWN

Impulzové meniče sú elektrické prístroje vyznačujúce sa vysokou účinnosťou a malým objemovým koeficientom. Pracovná frekvencia impulzových meničov sa nachádza v rozsahu jednotiek až desiatok kHz, ktorá siaha až do jednotiek MHz. Účinnosti jednotlivých impulzových meničov je závislá na obvodovej topológii meniča – obvodové zapojenie a môže byť v rozsahu od 60% do 99,5%. Návrh impulzového meniča je v porovnaní s lineárnym meničom omnoho komplikovanejší a náročnejší.

Teoretický rozbor impulzového meniča STEP-DOWN (BUCK).

Matematický opis je možné rozdeliť na dve časti:

Spínač S1 zopnutý počas doby **t_1**, takže napätie U_L na indukčnosti vzrastá

$$U_L = L\frac{dI_1}{dt} \tag{69.}$$

Z vyššie uvedeného vyjadríme **I_1**

$$dI_1 = \frac{U_1 - U_2}{L}\cdot t_1 \tag{70.}$$

Spínač S1 rozopnutý počas doby **t_2**, takže napätie U_L na indukčnosti klesá podľa

$$U_L = L\frac{dI_2}{dt} \tag{71.}$$

Z vyššie uvedeného vyjadríme **I_2**

$$-dI_2 = -\frac{U_2}{L}\cdot t_2 \tag{72.}$$

Prúdy **I_1 a I_2** sú vďaka indukčnosti zhodné, takže môžeme písať

$$U_2 \cdot t_2 = \frac{U_1 - U_2}{t_1} \tag{73.}$$

Úpravou vzťahu predchádzajúceho vzťahu získame

$$U_2 = U_1 \cdot \frac{t_1}{t_1 + t_2} = U_1 \cdot \frac{t_1}{T} = U_1 \cdot \delta \tag{74.}$$

Pokiaľ budeme uvažovať účinnosť $\eta = 100\%$ môžeme písať

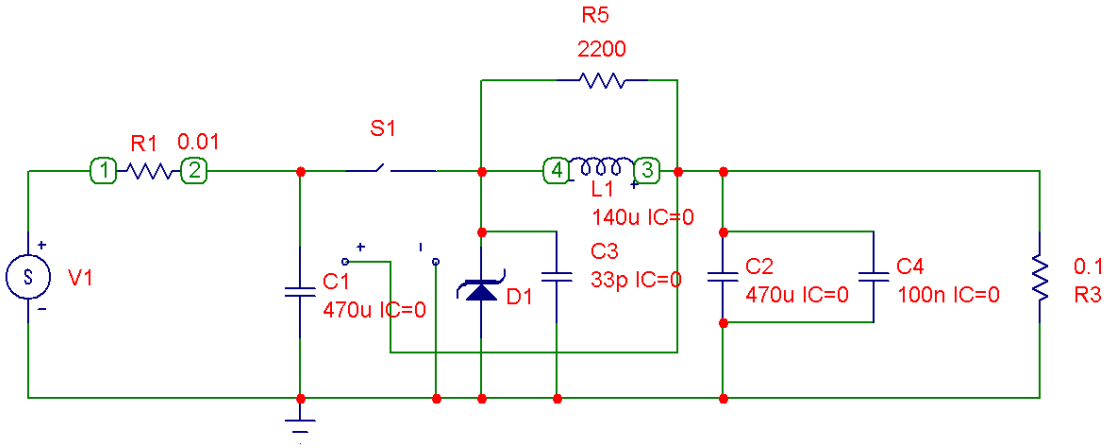
$$U_1 \cdot I_1 = U_2 \cdot I_2$$

(75.)

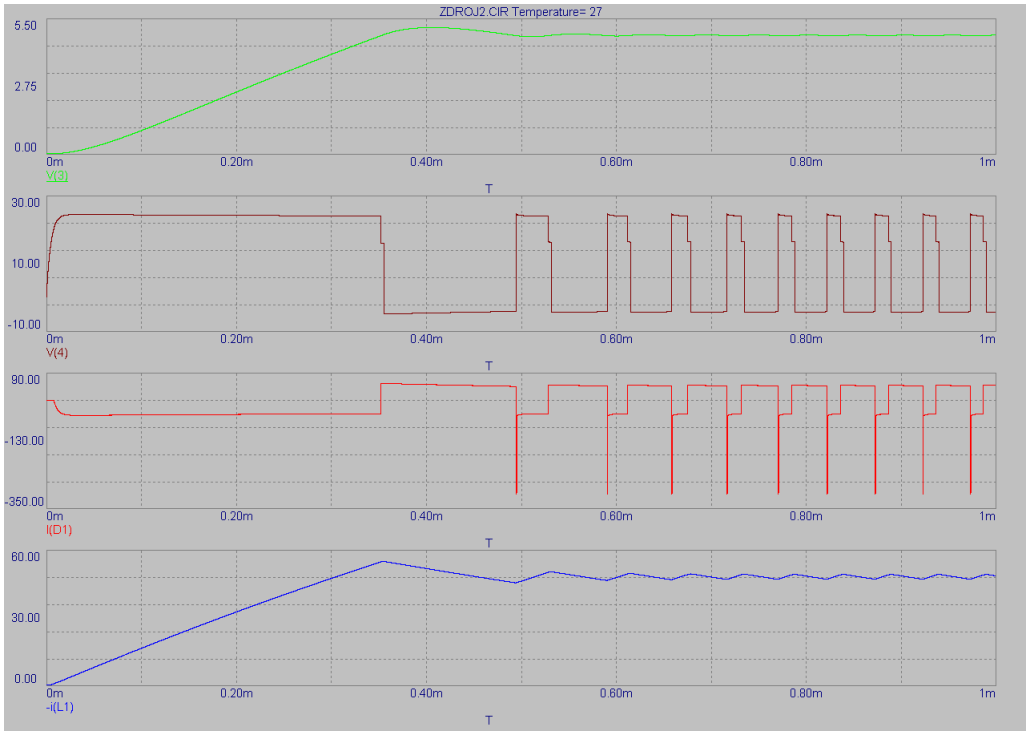
Z čoho vyjadríme výstupný prúd **I2**

$$I_2 = I_1 \cdot \frac{U_1}{U_2} \qquad (76.)$$

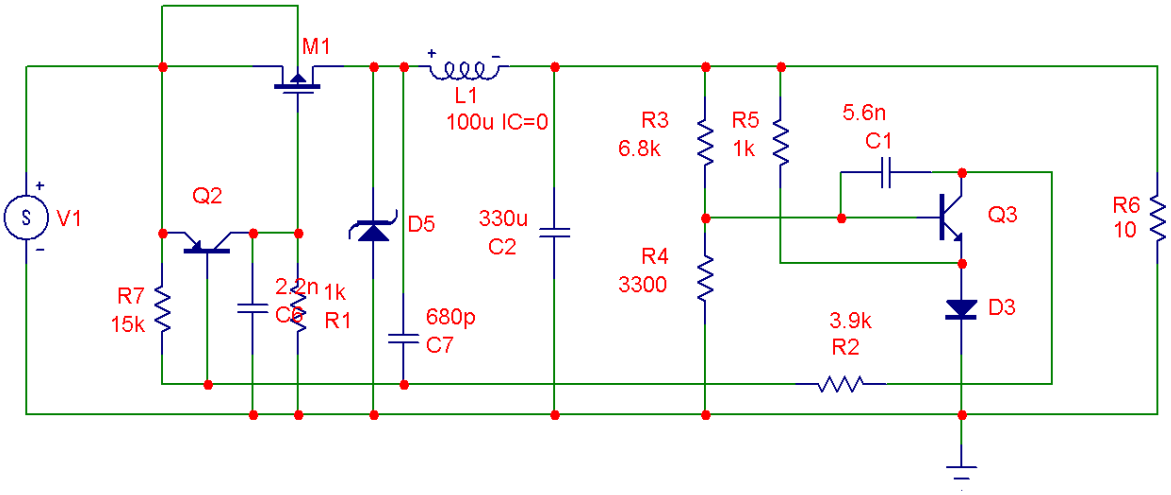
Obrázok 215, Simulačný model impulzového meniča



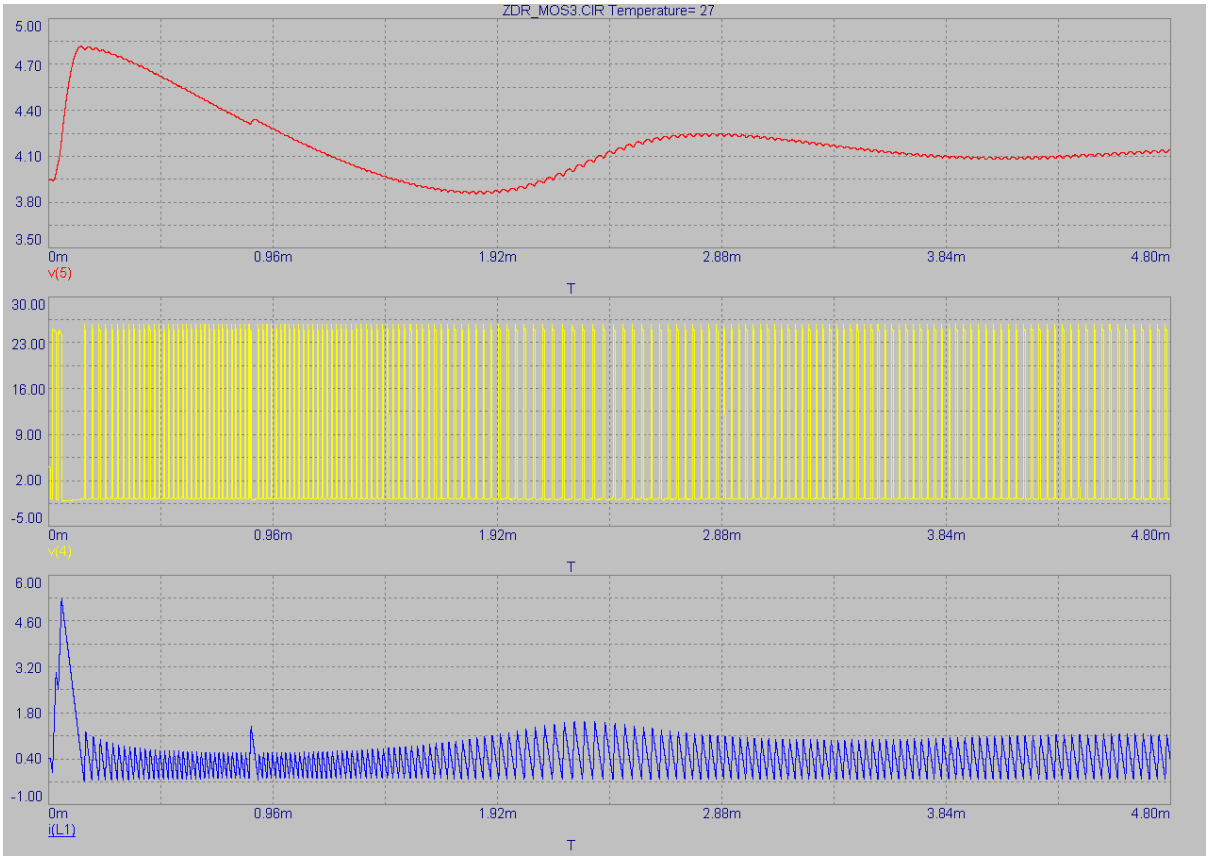
Obrázok 216, Simulačné výsledky modelu impulzového meniča



Obrázok 217, Reálne zapojenie impulzového meniča s reguláciou výstupného napätia



Obrázok 218, Simulačné výsledky reálneho zapojenia impulzového meniča



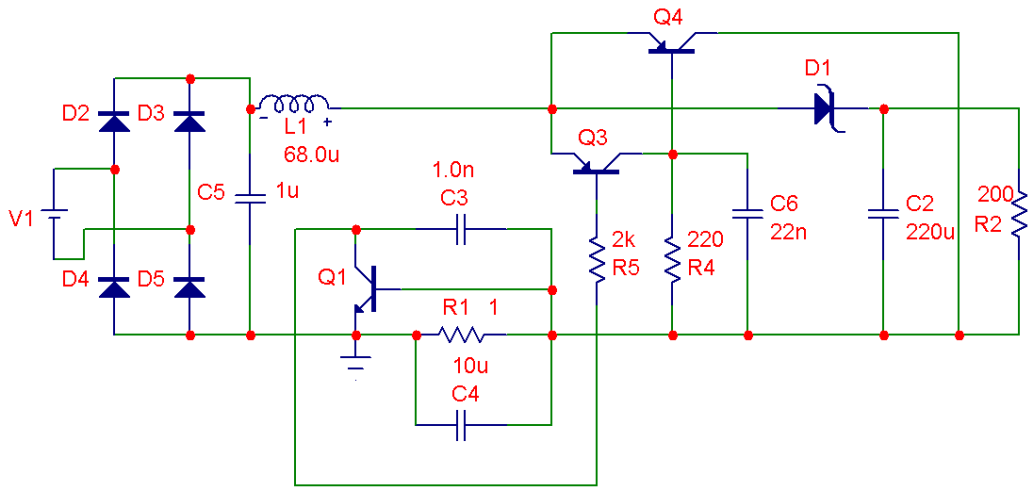
Poznámka autora: Vyššie uvedené zapojenie impulzového zdroja bolo použité ako náhrada lineárneho zdroja v čiernobielych TVP[217] a nabíjacích adaptéroch pre mobilné telefóny. Účinnosť tohto zdroja je $\eta \approx 86\%$ pri pracovnej frekvencii $f \approx 100kHz$. S výhodou je možné použiť tento typ zdroja aj pre napájanie spätného osvetlenia LCD panelov ktoré sú použité v aplikáciách s mikroprocesormi, alebo na napájanie pomocných obvodov.

[217] TVP – Televíznych prijímačov

17.13. Jednosmerný impulzový menič v zvyšovacom zapojení STEP-UP s PFC

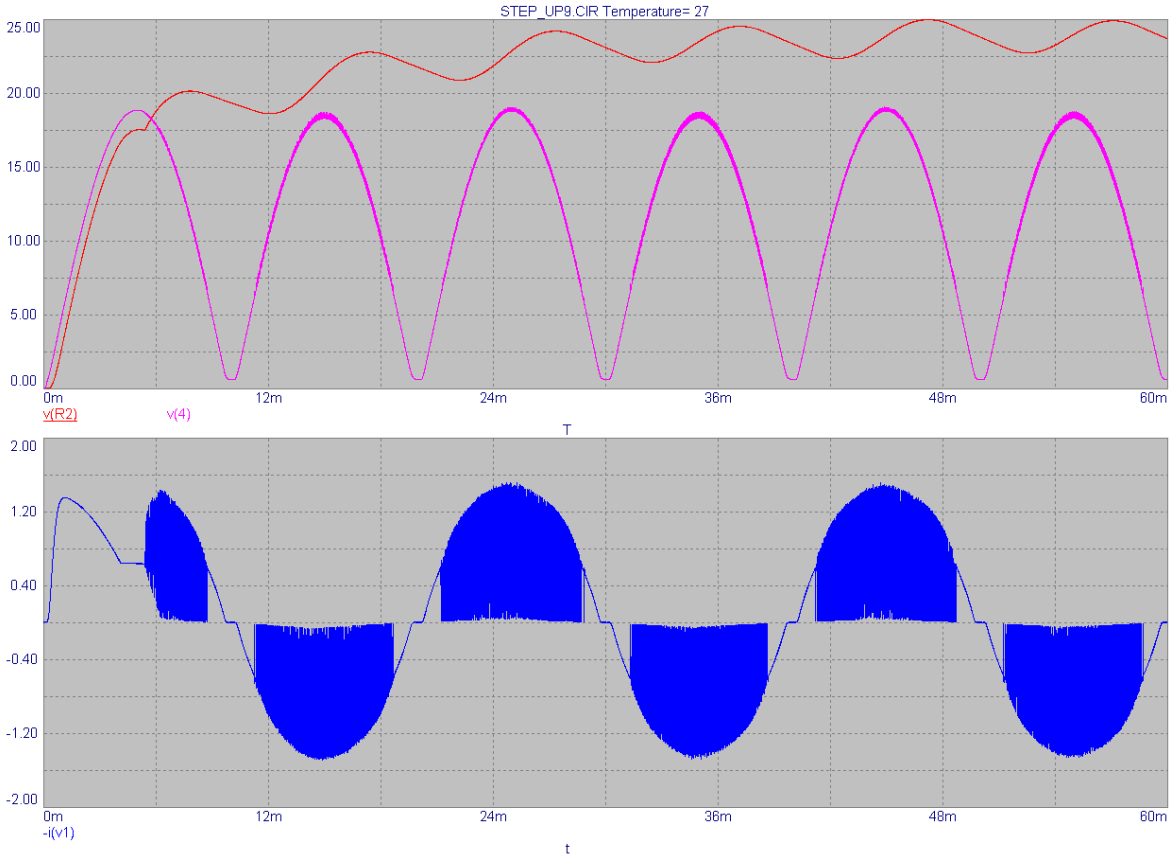
Obrázok 219 predstavuje zapojenie jednoduchého impulzového meniča s PFC[218] korekciou odoberaného vstupného prúdu meniča. V tomto zapojení je schopný menič odoberať prúd ktorého tvar po aproximácii môžeme porovnávať so sínusovým priebehom. Tento menič je používaný vo väčšine moderných prístrojov, pretože umožňuje vykonávať aj činnosť usmerňovača a aj funkciu zvyšovacieho meniča, ktorý umožňuje odoberať výkon zo striedavej siete s účinníkom $\eta \div 100\%$.

Obrázok 219, Reálne zapojenie impulzového meniča s PFC – Power Factor Correction

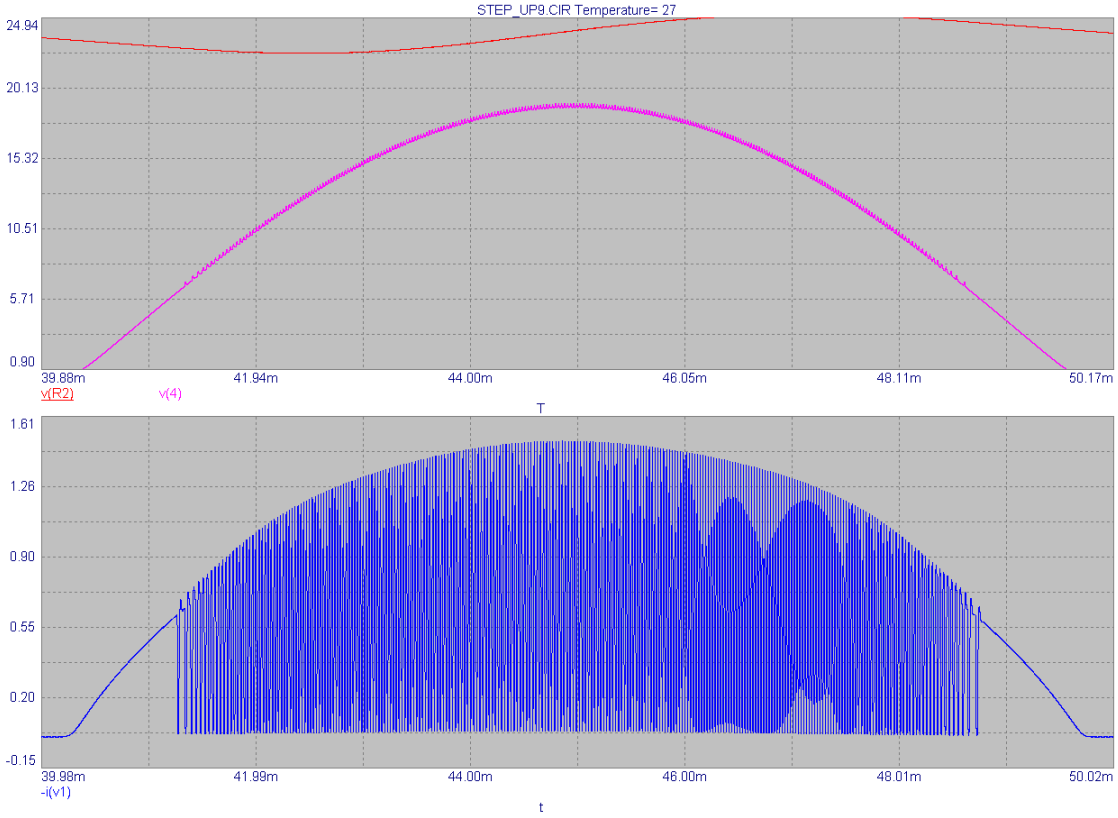


[218] Typickým predstaviteľom Boost-PFC SWPS ZCS integrovaných obvodov od NXP Semiconductor je TDA4863-2 pre použitie v impulzových zdrojoch AC/DC v rozsahu vstupného napätia 85V až 265V/50Hz a výstupné napätie 400V/DC s výstupným výkonom 160W, a výstupným prúdom I=0,4A a účinnosťou až η=85%.

Obrázok 220, Simulačné výsledky reálneho zapojenia impulzového meniča s PFC



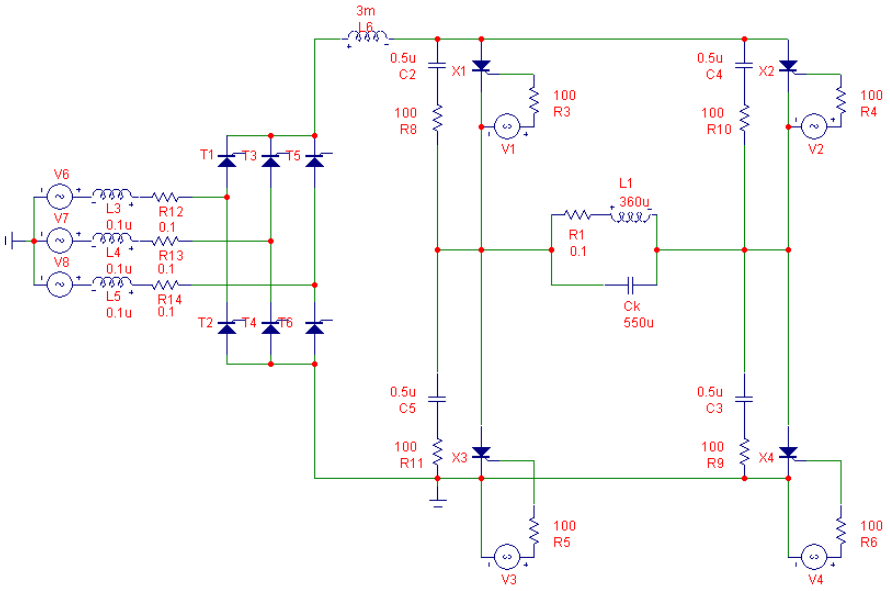
Obrázok 221, Detailné simulačné výsledky impulzového meniča s PFC



17.14. Diskrétne riadenie 3-fázového usmerňovača s tyristormi

Príklad: S využitím programovacieho jazyka C51 a RTX51 vytvorte diskrétny regulátor pre mostový usmerňovač s tyristormi pre elektrické pohony s jednosmernými motormi, alebo indukčné pece. Daný regulátor musí byť schopný samostatne generovať všetky zapínacie impulzy pre jednotlivé polovodičové prvky bez prídavných obvodov. Regulačný algoritmus PSD použite z predchádzajúcej úlohy. Využite záchytné (CTH0..2, CTL0..2) a porovnávacie (CMH0..2, CML0..2) registre mikroprocesora 80C552 pre meranie dĺžky periódy siete a pre generovanie zapínacích impulzov s fázovým posuvom α v rozsahu **0° až 150° elektrických**. Na výstupných svorkách portu P4.0 až P4.2 budú generované primárne zapínacie impulzy I_1, I_2, I_3, pre tyristory usmerňovača. Sekundárne impulzy budú odvodené od primárnych impulzov a smeru otáčania jednotlivých fázových napätí siete. Zapínací impulz[219] pre tyristory bude mať dĺžku maximálne $t_{ON} = 150\mu s$. Signál smeru otáčania bude dostupný na P4.3 a bude vstupovať do PLD obvodu s označením GAL16V8B[220], ktorý bude generovať primárne a sekundárne impulzy pre jednotlivé tyristory v kladnej a zápornej skupine.

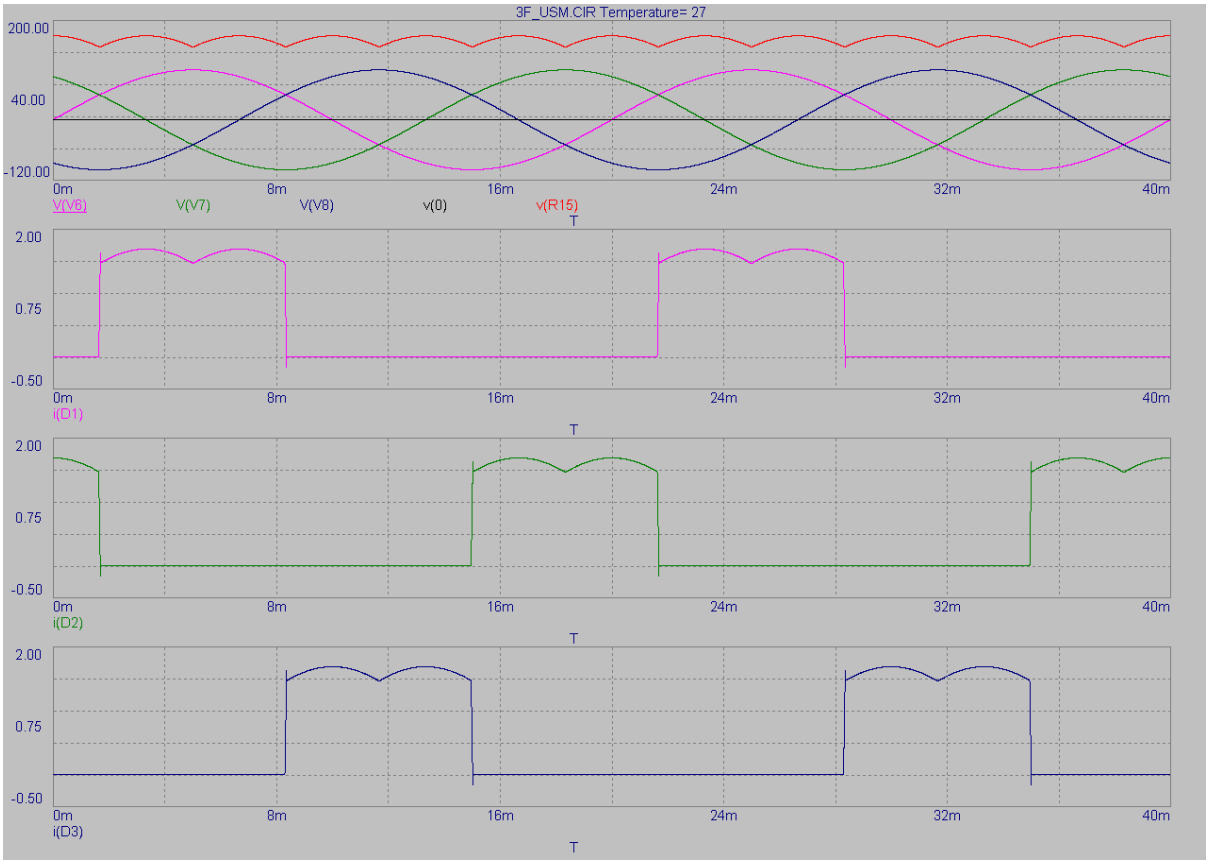
Obrázok 222, Mostový 3-fázový usmerňovač s napäťovým striedačom a rez. záťažou



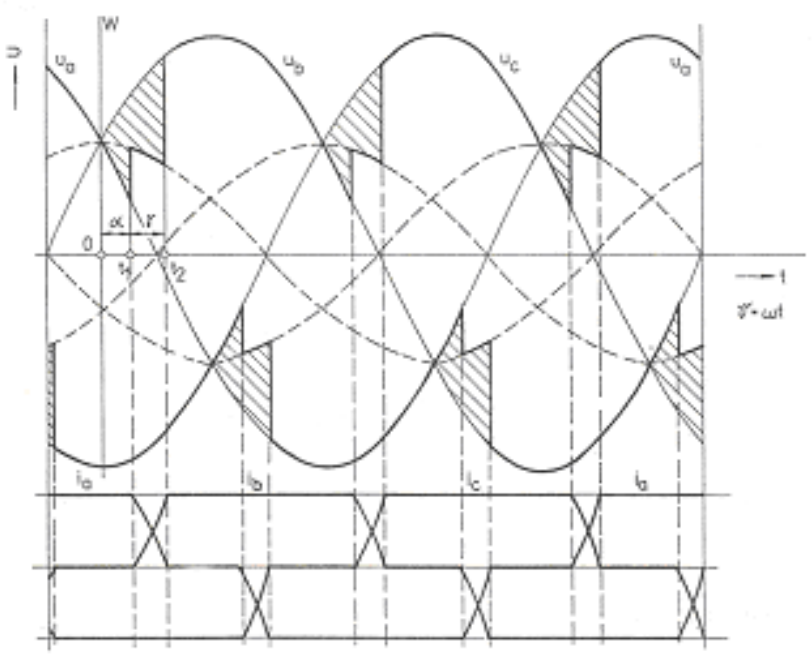
[219] Dĺžka zapínacieho impulzu privedená na riadiacu elektródu pre tyristor usmerňovača TV960/12/400 musí byť minimálne 85µs. Je to potrebné pre impulzový transformátor, aby vyrobil dostatočne „sýty“ prúdový zapínací impulz.

[220] Vyhovie v podstate akýkoľvek programovateľný obvod PAL, alebo GAL obvod s potrebným počtom I/O vývodov. Napr. GAL20V10-Q25 má oproti iným typom len 25% spotrebu energie a dopravné oneskorenie logickej hodnoty z vstupu na výstup max. 25ns.

Obrázok 223, Priebehy napätí a prúdov 3-fázového usmerňovača pri α=0°



Obrázok 224, Časové priebehy napätí a prúdov v oblasti normálnych zaťažení



Rozbor úlohy: Hlavnou úlohou tohto regulátora je generovať zapínacie impulzy pre tyristory s fázovým posuvom teoreticky 0° až 180° elektrických. Keďže ale pri tyristoroch musí dochádzať ku vzájomnej komutácii, musíme znížiť riadiaci uhol o cca. 30° elektrických, aby bola zabezpečená spoľahlivá vzájomná komutácia tyristorov. Zapínací impulz pre príslušný prvok je odvodený od príslušnej fázy napájacieho napätia označenej vo obrázku ako SL1(**V6**), SL2(**V8**) a SL3(**V7**). Na výstupe usmerňovača je napätie označené červenou farbou a prúdy príslušných tyristorov sú označené farebne rovnako ako pri napätiach. Zapínací impulz musí byť súčasne privedený na tyristor nachádzajúci sa v kladnej skupine a zároveň na niektorý tyristor v zápornej skupine. Generovanie zapínacích impulzov môže vykonávať mikroprocesor, alebo programovateľný logický obvod PLD. Obvod PLD je výhodné použiť všade tam, kde je potrebné podmieňovať generovanie impulzov, prípadne meniť dĺžku a frekvenciu impulzov napr. zámena jediného impulzu za sériu impulzov.

Závislosť výstupného napätia usmerňovača na riadiacom uhle α:

$$U_d = \frac{3}{\pi}\int_{\frac{\pi}{3}+\alpha}^{\frac{2\pi}{3}+\alpha}\sqrt{6}\cdot U_f\cdot\sin(\upsilon)d\upsilon = \frac{3\cdot\sqrt{6}}{\pi}\cdot U_f\cdot\cos(\alpha) \qquad (77.)$$

$$U_d = \frac{3\cdot\sqrt{6}}{\pi}\cdot U_f\cdot\cos(\alpha) \qquad (78.)$$

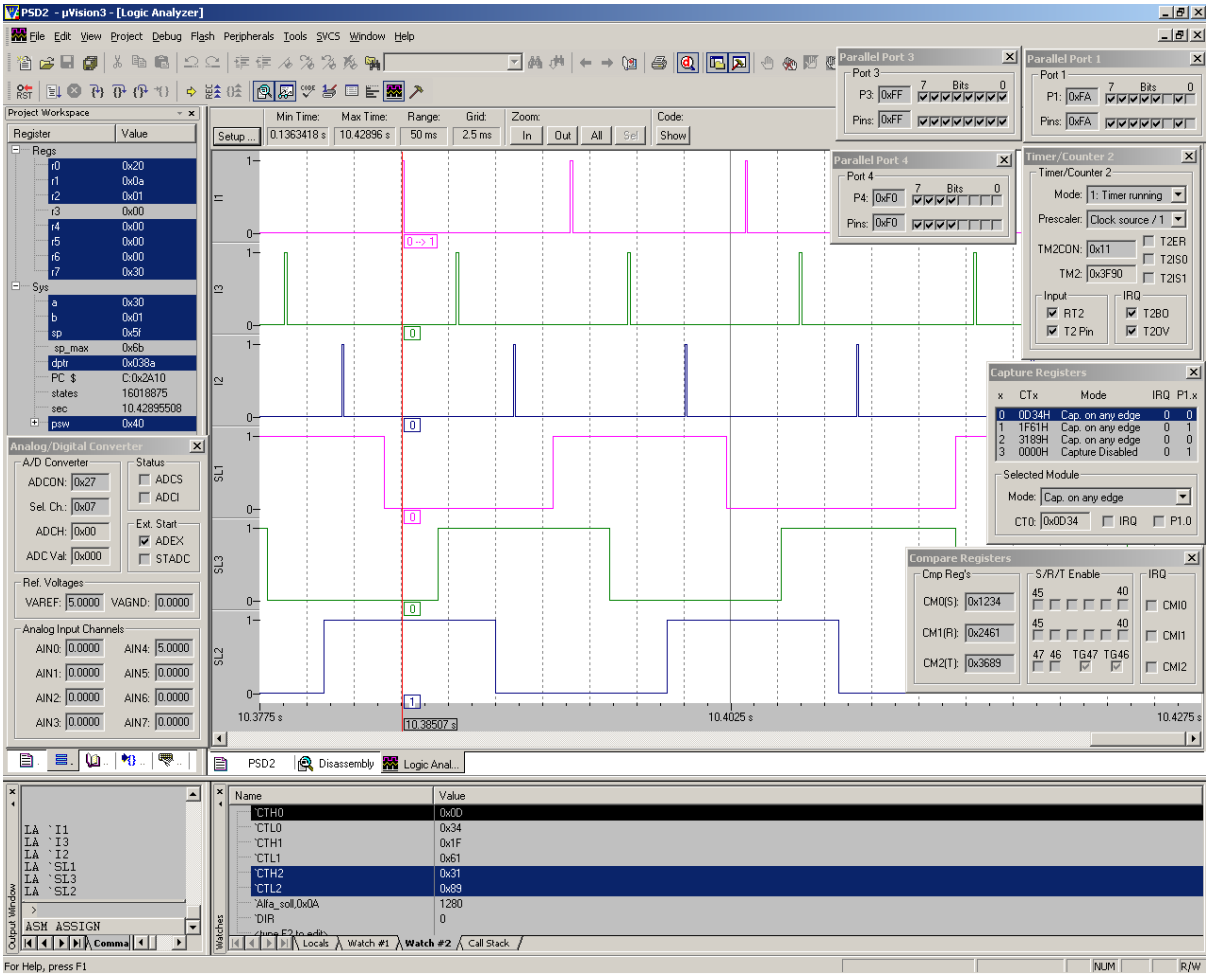
kde U_f je efektívna hodnota fázového napätia

Logická závislosť generovania zapínacích impulzov:

```
I1P=((I1 &  SL1)|(I3 & /SL3 & DIR)|(I2 & /SL2 & /DIR));
I1N=((I1 & /SL1)|(I3 &  SL3 & DIR)|(I2 &  SL2 & /DIR));
I3P=((I3 &  SL3)|(I2 & /SL2 & DIR)|(I1 & /SL1 & /DIR));
I3N=((I3 & /SL3)|(I2 &  SL2 & DIR)|(I1 &  SL1 & /DIR));
I2P=((I2 &  SL2)|(I1 & /SL1 & DIR)|(I3 & /SL3 & /DIR));
I2N=((I2 & /SL2)|(I1 &  SL1 & DIR)|(I3 &  SL3 & /DIR));
```

I1P až I3P	impulzy pre tyristory v kladnej skupine usmerňovača
I1N až I3N	impulzy pre tyristory v zápornej skupine usmerňovača
SL1 až SL3	tvarované signály jednotlivých fázových napätí napájacej siete
I1 až I3	primárne zapínacie impulzy
DIR	signál smeru otáčania sa fázových napätí

Obrázok 225, Principiálne generovanie primárnych zapínacích impulzov pre tyristory

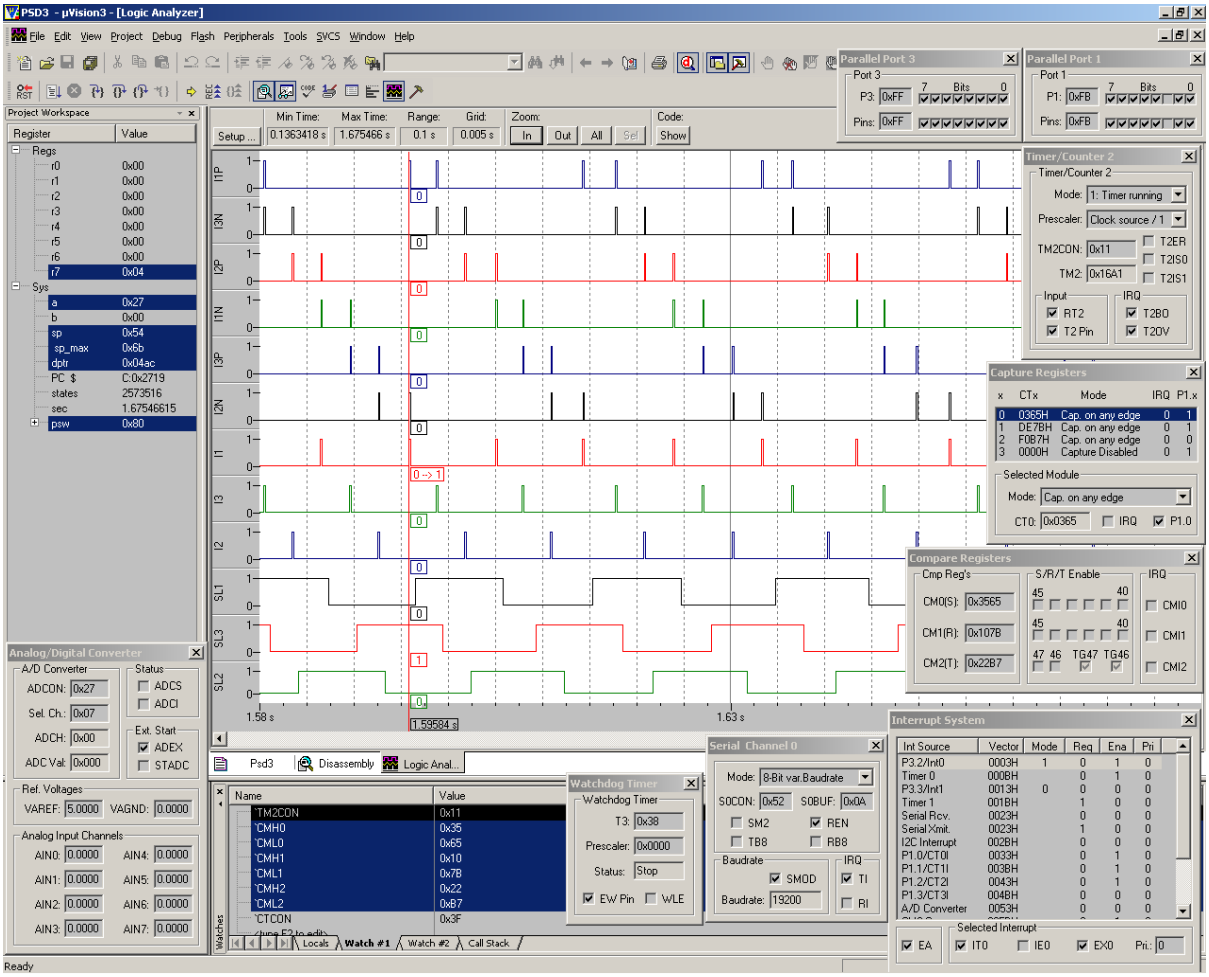


Poznámka autora: Celkovo boli zostrojené dva nezávislé regulátory s rovnakou funkciou. Jeden obsahoval analógové obvody (generátory píl, komparátory ....) ktoré spolupracovali s mikroprocesorovou jednotkou a druhá bola zostrojená len v diskrétnom tvare.[221]

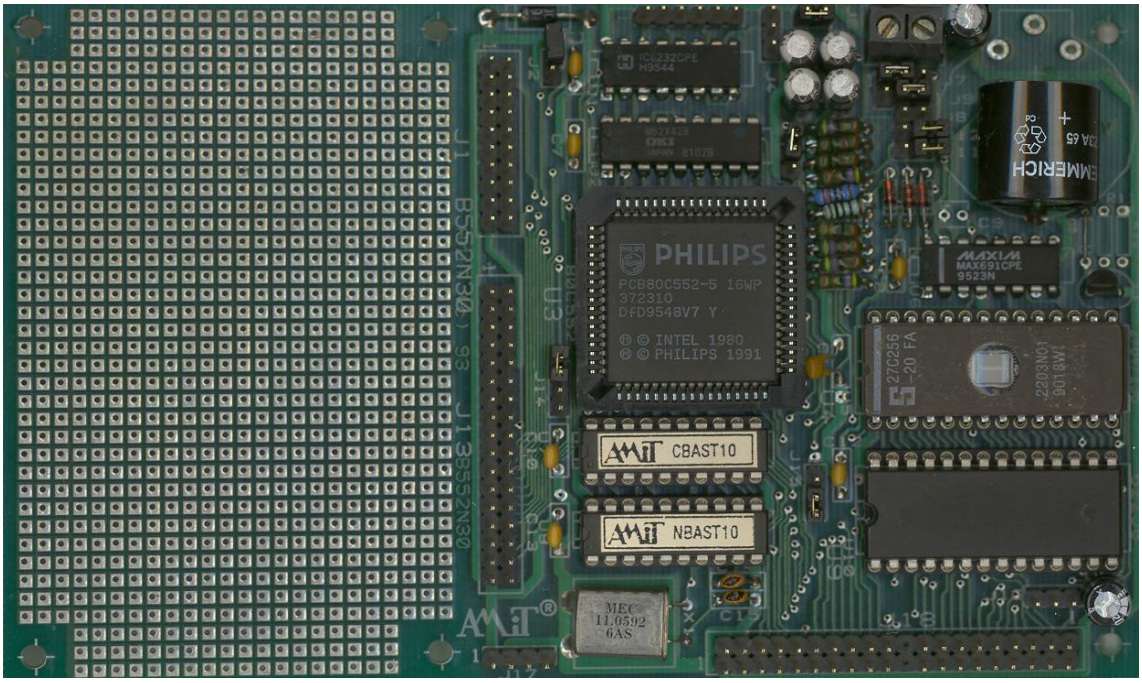
Všetky analógové obvody boli nahradené diskrétnymi obvodmi (generátory pílového napätia – časovačom T2, komparátory – číslicovými komparačnými jednotkami ...). Meranie periódy sieťového napätia sa v diskrétnom regulátore vykonávajú záchytnou jednotkou CAPTURE tvorenou registrami CTHx a CTLx. Generovanie primárnych a sekundárnych impulzov s fázovým posuvom riadiaceho uhla α je vytvorené pomocou komparačnej jednotky CAPCOM tvorenou registrami CMHx a CMLx. Diskrétny regulátor bol neskôr doplnený aj o automatickú korekciu riadiaceho uhla α pri zmene frekvencie t.j. periódy sieťového napätia v rozsahu frekvencií 40 až 70Hz.

[221] Hlavnými dôvodmi prečo sa analógový regulátor nahradil diskrétnym je niekoľko, ale najväčším problémom bola vysoká teplota (približne 75°C) okolitého prostredia v blízkosti taviacich pecí, ktorá spôsobovala častú poruchovosť regulátora taviacich pecí.

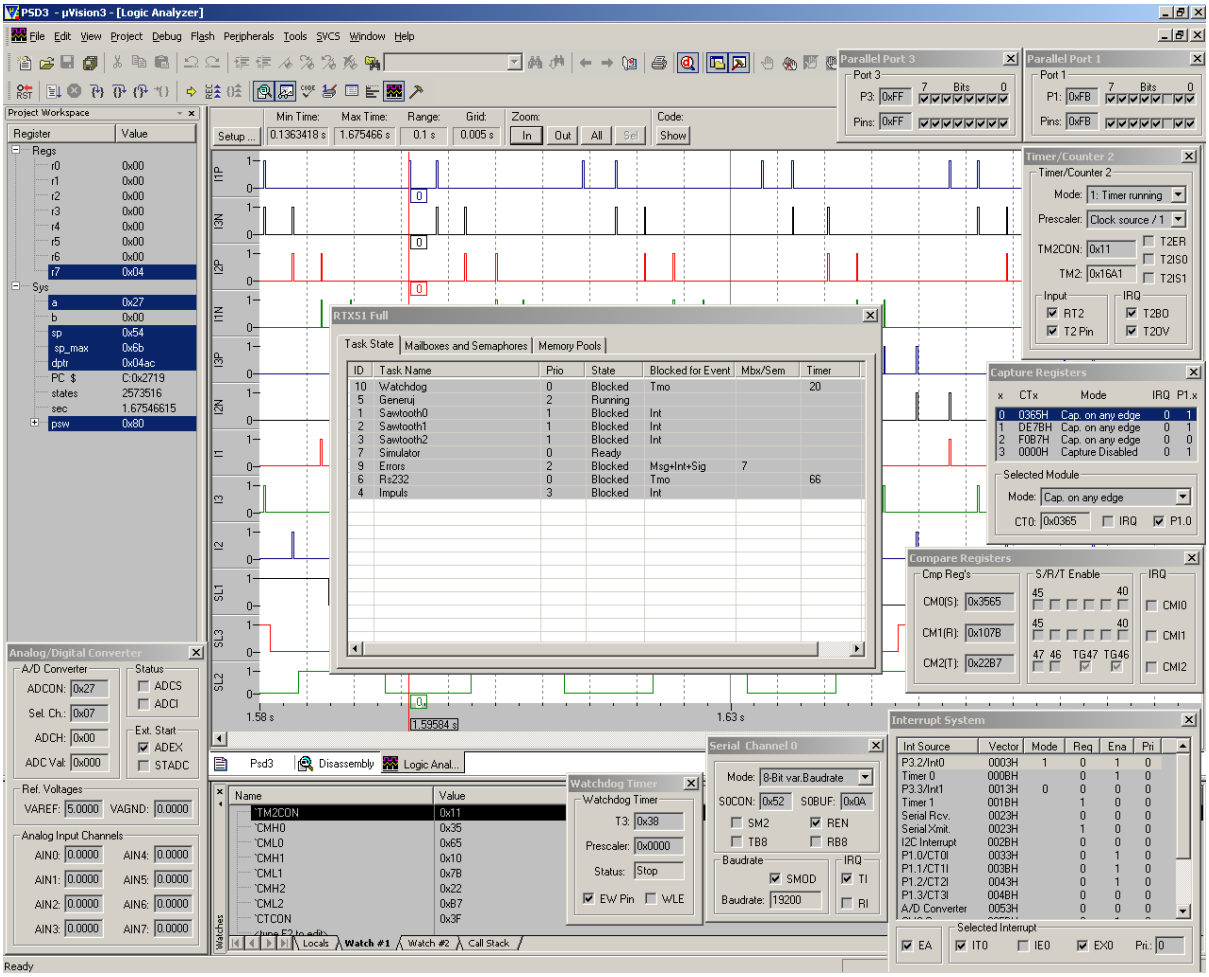
Obrázok 226, Generovanie impulzov v mikroprocesore 87C552



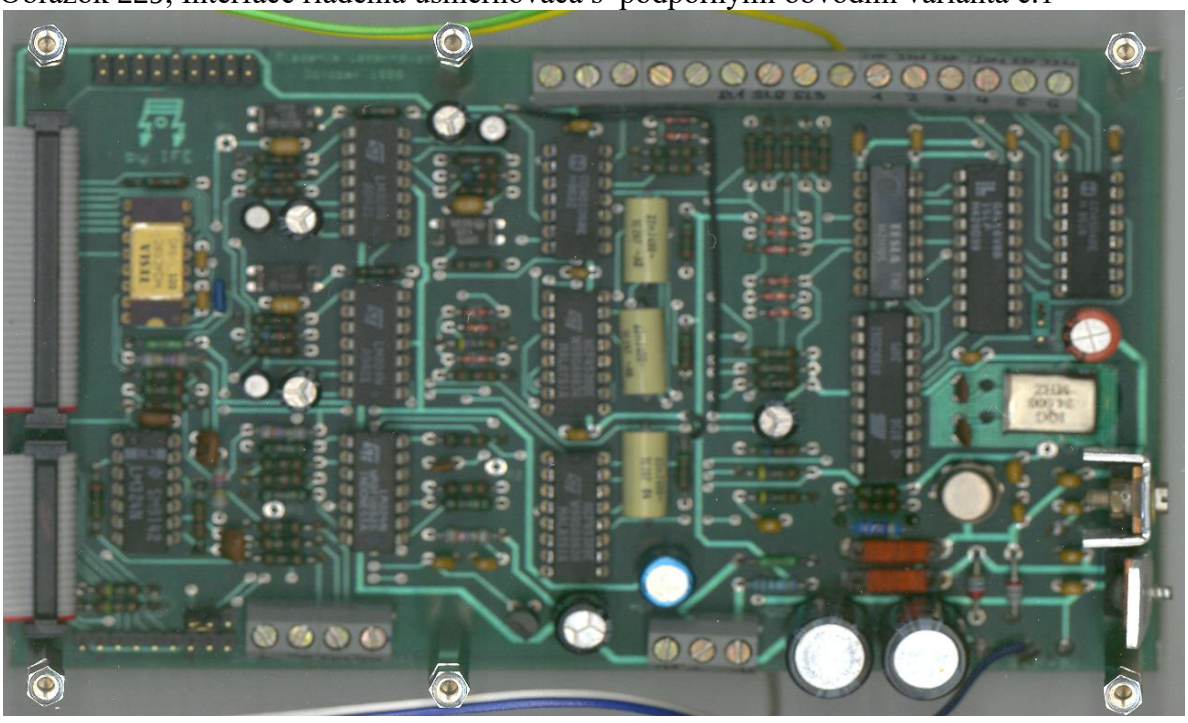
Obrázok 227, Vývojová doska diskrétneho riadenia usmerňovača s 80C552



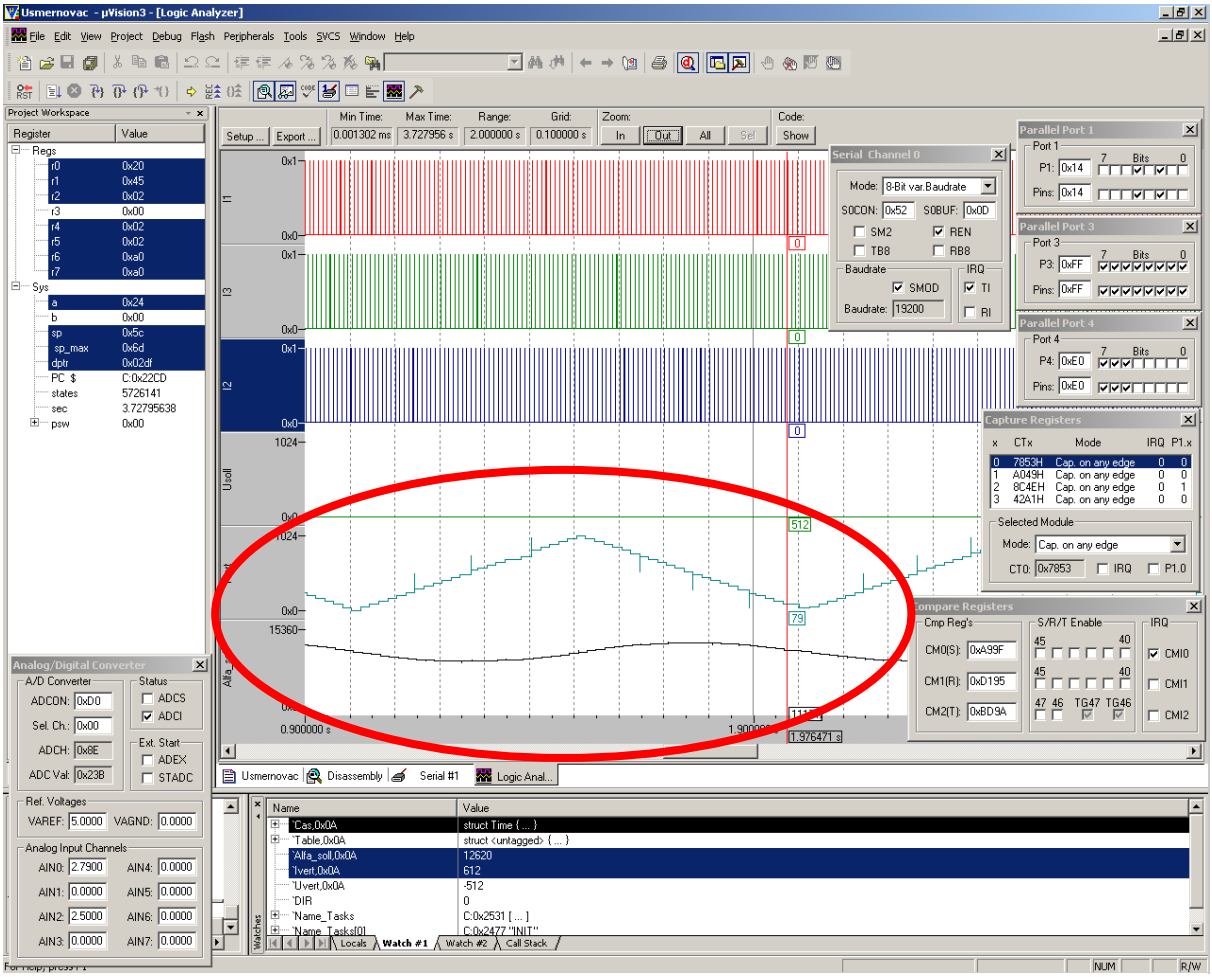
Obrázok 228, Zoznam úloh diskrétneho regulátora usmerňovača pre tyristory



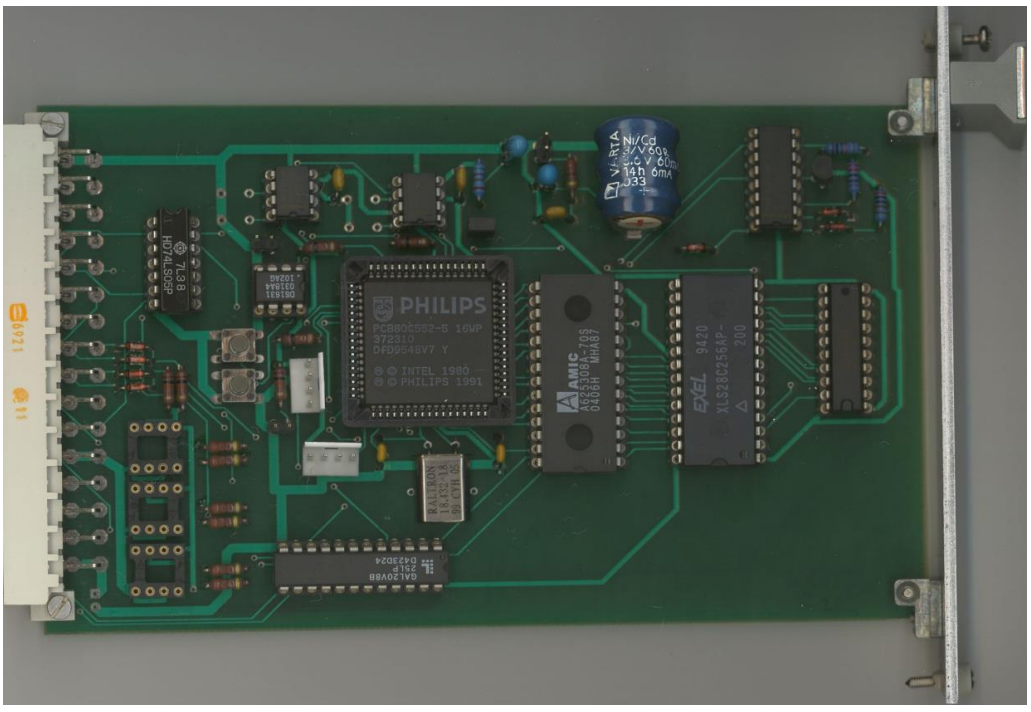
Obrázok 229, Interface riadenia usmerňovača s podpornými obvodmi varianta č.1



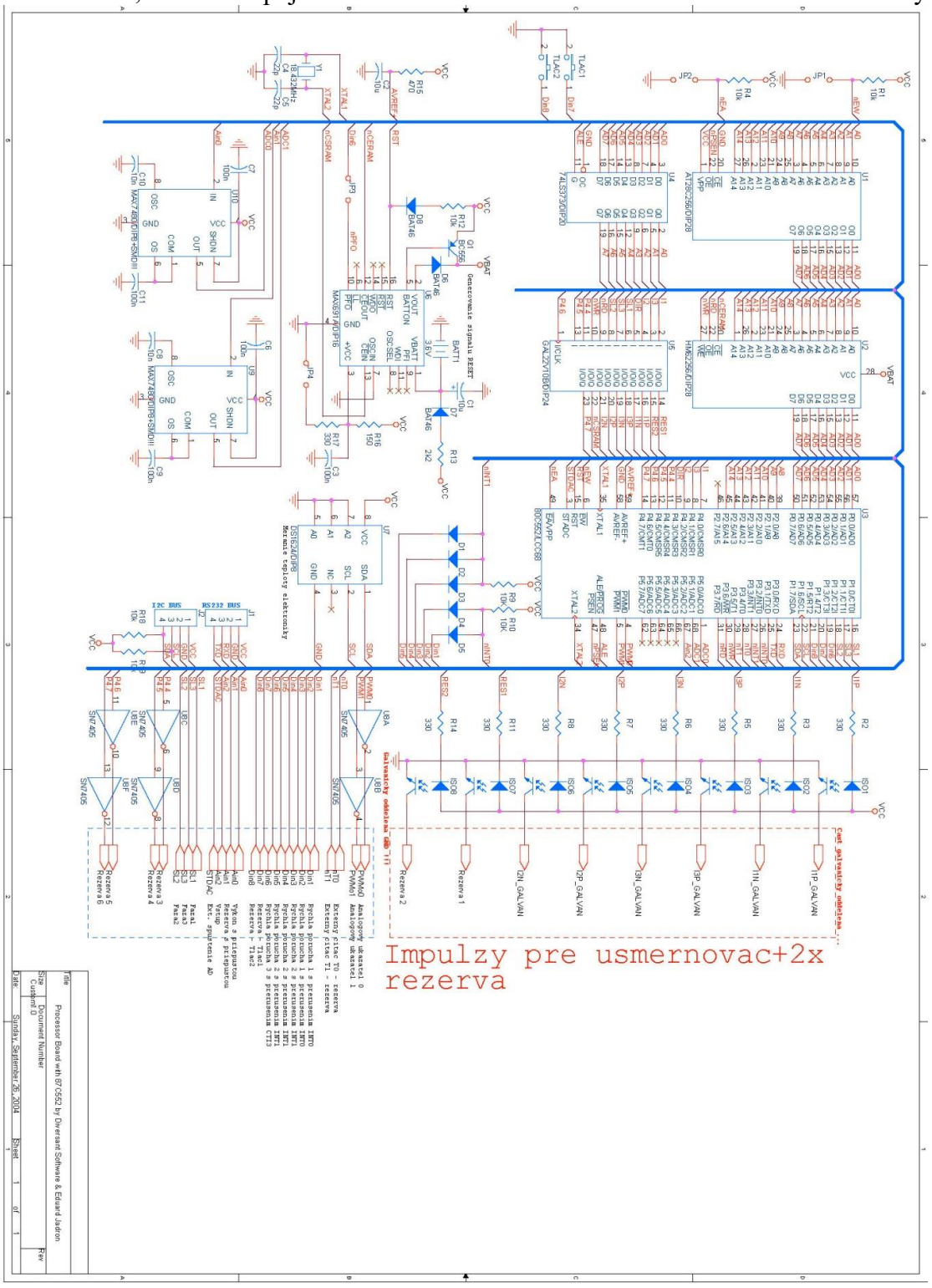
Obrázok 230, Simulácia regulátora diskrétneho riadenia s 80C552 varianty č.2



Obrázok 231, Doska diskrétneho riadenia usmerňovača s 80C552 varianty č.2



Obrázok 232, Schéma zapojenia diskrétneho riadenia[222] usmerňovača 80C552 varianty č.2



[222] V tejto variante bol použitý PSD prírastkový algoritmus s reálnou aritmetikou. Ďalšie varianty už obsahovali PSD regulátor s celočíselnou aritmetikou z dôvodu vylepšenia stability regulačnej slučky regulátora.

Zápis časti programu pre regulátor tyristorového usmerňovača.

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
157  #define  TIME  3                                  //Potrebne pre os_wait v Sawtooth0-2, idealna
     hodnota je 2 az 4 ...
158
159  void  Sawtooth0 (void) _task_  SAWTOOTH0  _priority_  1  //Priorita pre SAWTOOTHx, mala by byt
     vacsia ako 0 t.j 1 az 2, inac presne negeneruje cislicove pily !!!
160  {
161   int  Temp;
162   TM2CON=0x01;                  //T2 s frekvenciou fosc/1 pretecie za 1*40.27ms
163   CTCON|=0x02+0x01;                 //Max=(150/180)*Pocet  12800, Min=(30/180)*Pocet  2560
164   os_attach_interrupt (6);
165   while (1)
166    {
167     Watchdog;
168     switch (os_wait (K_INT+K_TMO,K*TIME,NULL))    //Cakam na fazu A a zachytavam obsahy zachytnych
     registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
169       {
170        case  INT_EVENT  : CTI0=0; BUSY=1; Temp=Alfa_soll+(int)(256*CTH0+CTL0); CMH0=high(Temp); CML0
     =low(Temp); BUSY=0; break;     //Prisla hrana fazy, tak vykonam toto ...
171     case  TMO_EVENT  : os_send_message (ERROR_MAIL,SAWTOOTH0,TIME); break;
     //Ak sa tu dostanem neprisla hrana fazy do TIME tak vyvolam chybu ...
172     }
173    }
174  }
175
176  void  Sawtooth1 (void) _task_  SAWTOOTH1  _priority_  1  //Tento task sa vyvola do cca. 400us z
     tasku s nizsou prioritou
177  {
178   int  Temp;
179   CTCON|=0x08+0x04;
180   os_attach_interrupt (7);
181   while (1)
182    {
183     Watchdog;
184     switch (os_wait (K_INT+K_TMO,K*TIME,NULL))    //Cakam na fazu C a zachytavam obsahy zachytnych
     registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
185       {
186     case  INT_EVENT  : CTI1=0; BUSY=1; Temp=Alfa_soll+(int)(256*CTH1+CTL1); CMH1=high(Temp); CML1=
     low(Temp); BUSY=0; break;     //Prisla hrana fazy, tak vykonam toto ...
187     case  TMO_EVENT  : os_send_message (ERROR_MAIL,SAWTOOTH1,TIME); break;
     //Ak sa tu dostanem neprisla hrana fazy do TIME tak vyvolam chybu ...
188    }
189    }
190  }
191
192  void  Sawtooth2 (void) _task_  SAWTOOTH2  _priority_  1
193  {
194   int  Temp;
195   CTCON|=0x20+0x10;
196   os_attach_interrupt (8);
197   while (1)
198    {
199     Watchdog;
200     switch (os_wait (K_INT+K_TMO,K*TIME,NULL))    //Cakam na fazu B a zachytavam obsahy zachytnych
     registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
201       {
202        case  INT_EVENT  : CTI2=0; BUSY=1; Temp=Alfa_soll+(int)(256*CTH2+CTL2); CMH2=high(Temp); CML2
     =low(Temp); BUSY=0; break;     //Prisla hrana fazy, tak vykonam toto ...
203     case  TMO_EVENT  : os_send_message (ERROR_MAIL,SAWTOOTH2,TIME); break;
     //Ak sa tu dostanem neprisla hrana fazy do TIME tak vyvolam chybu ...
204    }
205    }
206  }
207
208  sbit  DIR  =  P4^3;
209
210  void  OrderPhase (void) _task_  ORDERPHASE  _priority_  0
211  {
212   CTCON|=0x20;
213   os_attach_interrupt (8);                        //Citlivost nastavim na log. "0"
214   while (1)
215    {
216     Watchdog;
217     switch (os_wait (K_INT+K_TMO,K*25,NULL))
218      {
```

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
219         case    INT_EVENT    :    CTI2 =0; if (( P1 &0x07 )!= 1)  DIR =RIGHT ; else  DIR =LEFT ; os_send_signal (
    INIT ); break ;
220         case    TMO_EVENT    :    os_send_message (ERROR_MAIL ,ORDERPHASE ,TIME ); break ;
                        //Neprisli mi fazy do 500ms, tak vyhlasim ERROR ...
221        }
222       CTCON &=~ 0x20 ;
223       os_detach_interrupt (8);
224       os_delete_task (ORDERPHASE );
225      }
226    }
227
228    void  Delay (unsigned  char  Dlzka ) //reentrant
229    {
230     while ((-- Dlzka )!= 0x00 );
231    }
232
233    #pragma  REGISTERBANK  (3)
234
235    sbit  I1  =  P4 ^0;
236    sbit  I3  =  P4 ^1;
237    sbit  I2  =  P4 ^2;
238
239    void  Impuls (void ) _task_  IMPULS  _priority_  3
240    {
241     #define  Length  75                                                  //60*1.366=85us pre dlzku
    impulzu !!! ak nemam funkciu Delay reentrantnu
242     #define  Dlzka  (Length /1.366 )                                     //Aby som nemusel stale
    prepocitavat na skutocnu dlzku impulzu ...
243     os_attach_interrupt (11 );
244     os_attach_interrupt (12 );
245     os_attach_interrupt (13 );
246     while (1)
247      {
248       Watchdog ;
249       switch (os_wait (K_TMO +K_INT ,K*100 ,NULL ))
250        {
251         case  INT_EVENT  : switch (TM2IR &0x70 )                        //RTX51 HANDLER prepne do cca.
    110us na tento task zovsadial !!!
252                          {
253                        case   0x10   : CMI0 =0; I1 =1; Delay (Dlzka ); I1 =0; break ;  //Generujem 100us
    impulz I1 pre SL1 pre podmienku CMH0+CML0=TMH2+TML2
254                        case   0x20   : CMI1 =0; I3 =1; Delay (Dlzka ); I3 =0; break ;  //Generujem 100us
    impulz I3 pre SL3 pre podmienku CMH1+CML1=TMH2+TML2
255                        case   0x40   : CMI2 =0; I2 =1; Delay (Dlzka ); I2 =0; break ;  //Generujem 100us
    impulz I2 pre SL2 pre podmienku CMH2+CML2=TMH2+TML2
256                        default    : TM2IR &=~ 0x70 ; break ;                //Ak som tu tak z
    nevysvetlitelnych dovodov prisli niektore, alebo vsetky dve-tri prerusenia od komparatora !!!
    Stane sa to pri poruchach na silovych fazach a vtedy nesmie generovat impulz !!! ...
257                         }
258                  break ;
259      case  TMO_EVENT  : os_send_message (ERROR_MAIL ,IMPULS ,TIME ); break ;
260      }
261      }
262    }
```

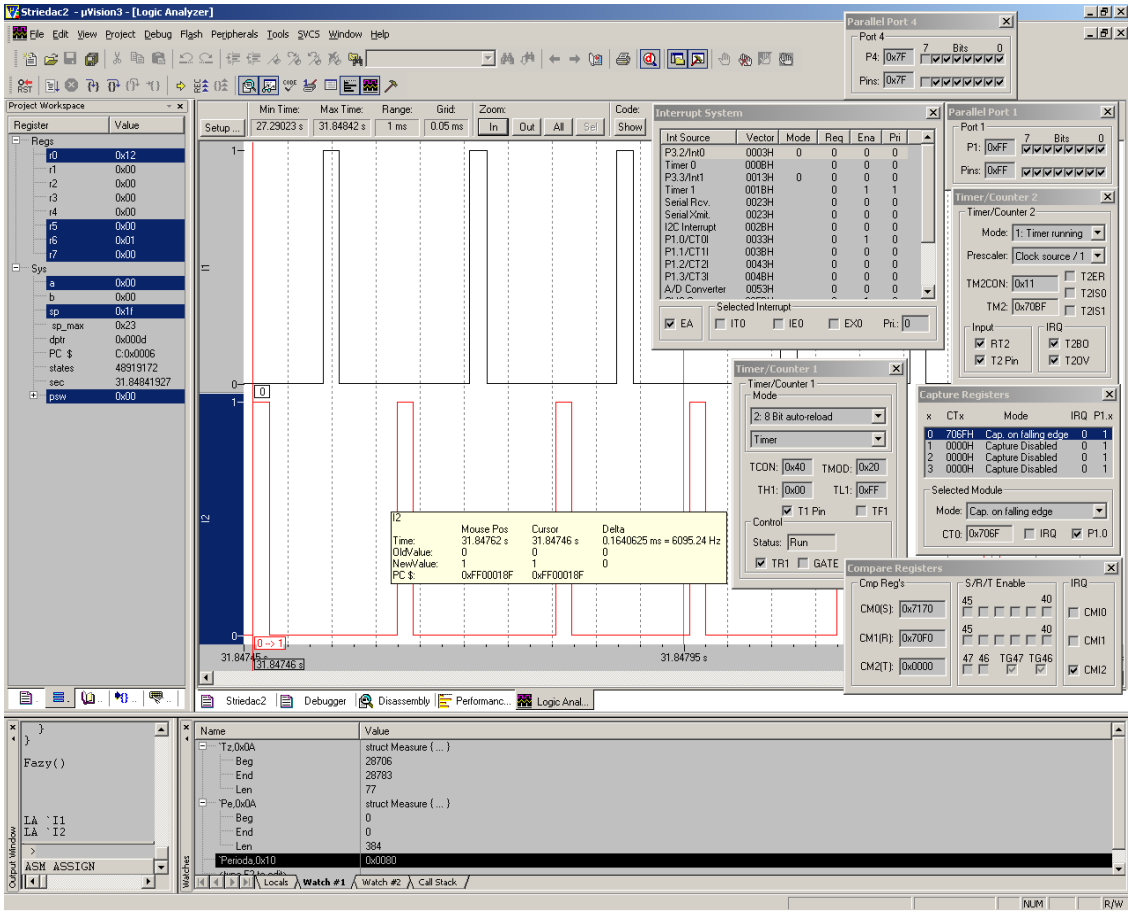
17.15. Diskrétne riadenie pre tyristorové striedače s rezonančným obvodom

Hlavnou úlohou tohto regulačného obvodu bolo riadenie chodu striedača s rezonančným obvodom a s nepretržitou kontrolou minimálnej doby zatvorenia t_z, ktorá musí byť pre správnu funkciu minimálne $t_z \geq 3 \cdot t_q$ ($\mu s; \mu s$). Čas t_q je katalógová hodnota, ktorá udáva vypínací čas tyristora. Vzájomný pomer periódy výstupného napätia T a doby zatvorenia tyristora t_z sa musia nachádzať pre optimálnu prevádzku striedača približne v rozsahu empiricky zisteného a udávaného δ.

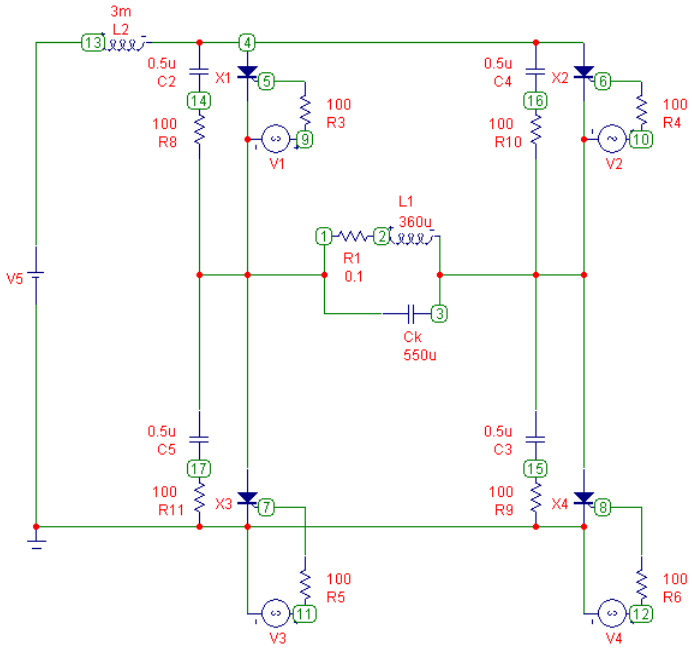
$$\delta \approx \frac{T}{t_z} \approx 6.3 \qquad (1; s,s) \qquad (79.)$$

Dĺžka generovaného zapínacieho impulzu pre tyristor striedača bude minimálne $t_{on\min} = 18\mu s$ s možnosťou nastavenia až na $t_{on\max} = 100\mu s$. Diskrétny regulátor striedača musí byť schopný generovať zapínacie impulzy až do frekvencie $f_{\max} = 6000Hz$.

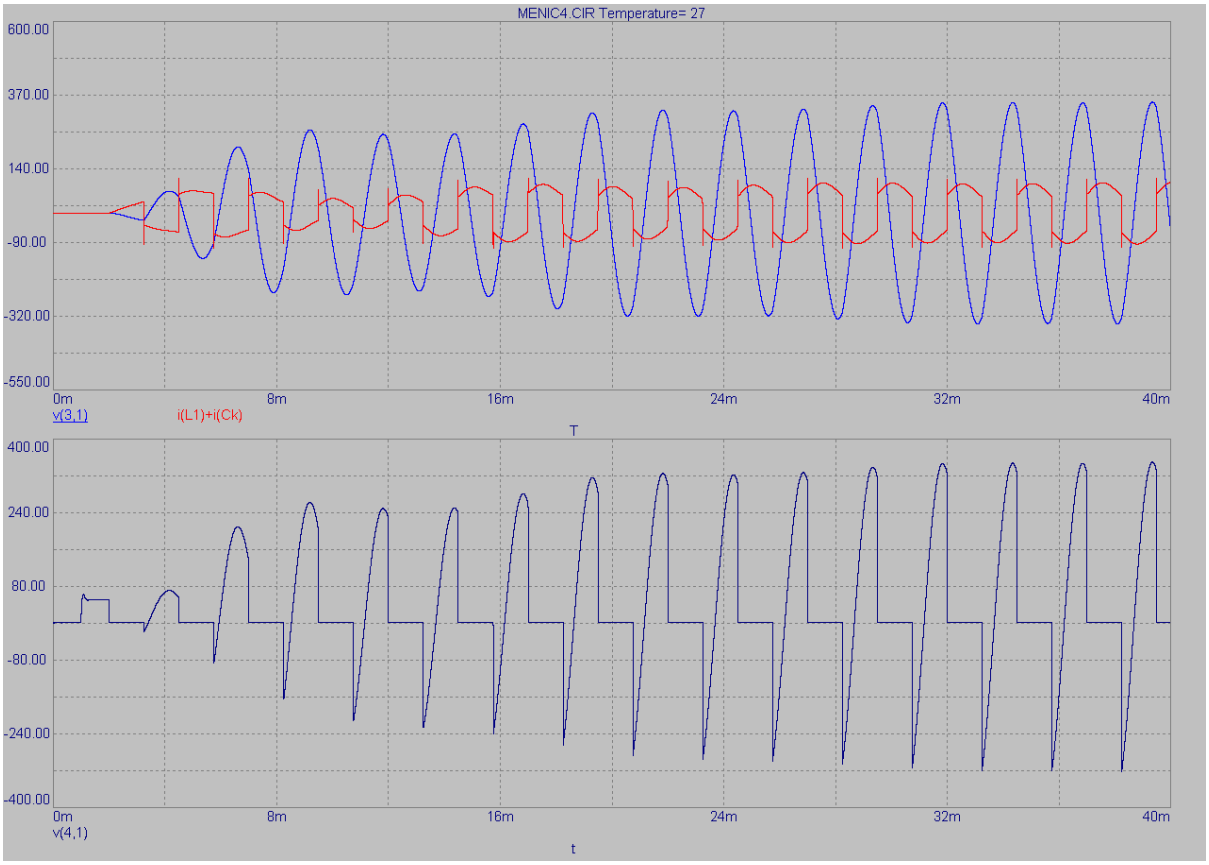
Obrázok 233, Generovanie zapínacích impulzov pre striedač v mikroprocesore 87C552



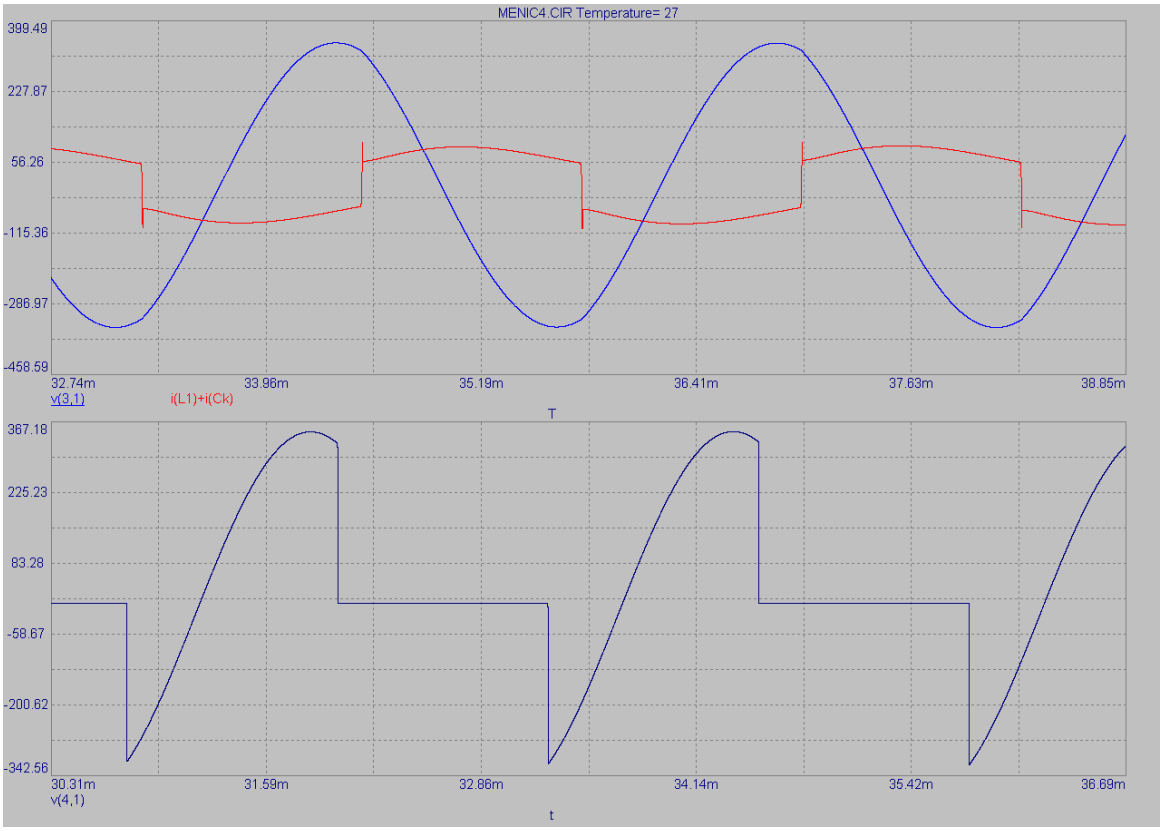
Obrázok 234, Zjednodušený model tyristorového striedača s rezonančným obvodom



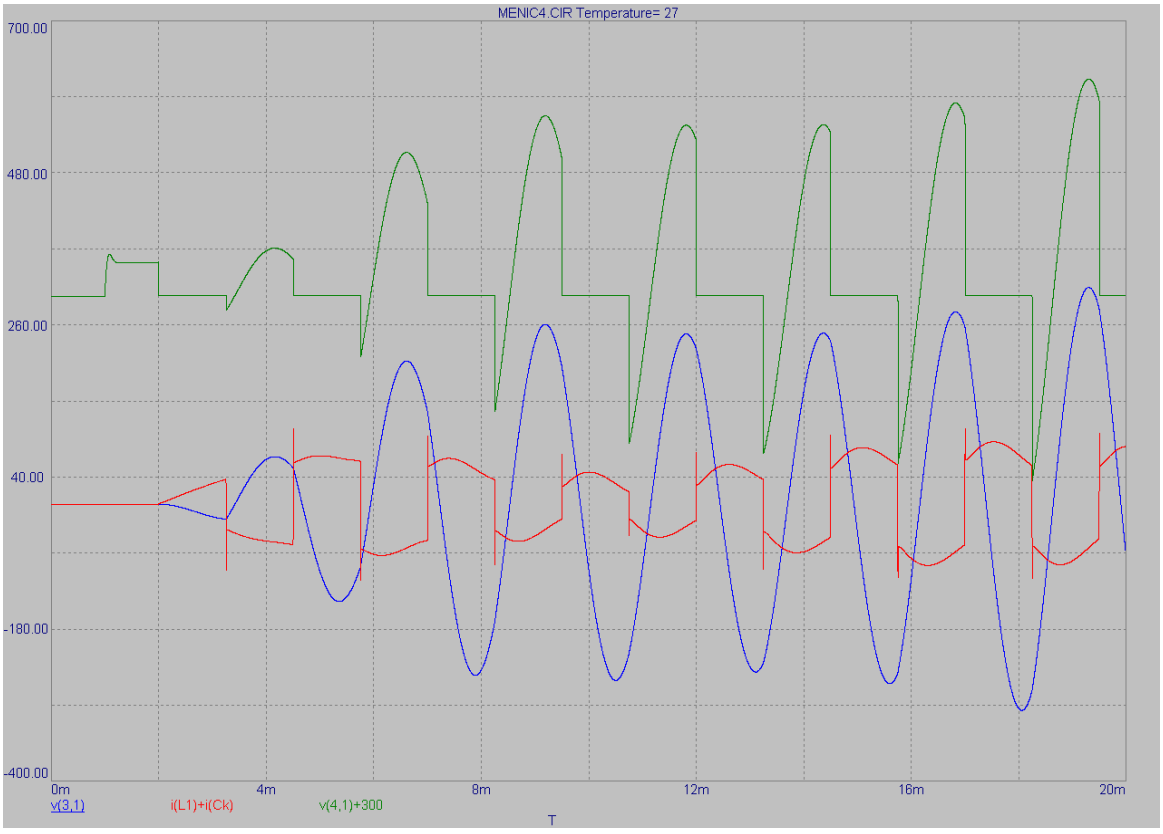
Obrázok 235, Grafické priebehy napätí v tyristorovom striedači s rezonančným obvodom



Obrázok 236, Grafické priebehy napätí v tyristorovom striedači s rezonančným obvodom



Obrázok 237, Detailné priebehy napätí v tyristorovom striedači s rezonančným obvodom



17.16. Spracovanie výsledkov meraní na striedači.

Po zostrojení striedača s diskrétnym regulátorom boli namerané tieto veličiny na tyristoroch TR961-500-12-NML s časom komutácie $t_q = 170\mu s$, pracovná frekvencia $f_{rez} = 413Hz$. Regulátor bol navrhnutý tak, aby komutačný uhol tyristora striedača bol maximálne $\beta_{el} = 26°$ elektrických, čo je v súlade s nižšie uvedeným vzťahom .

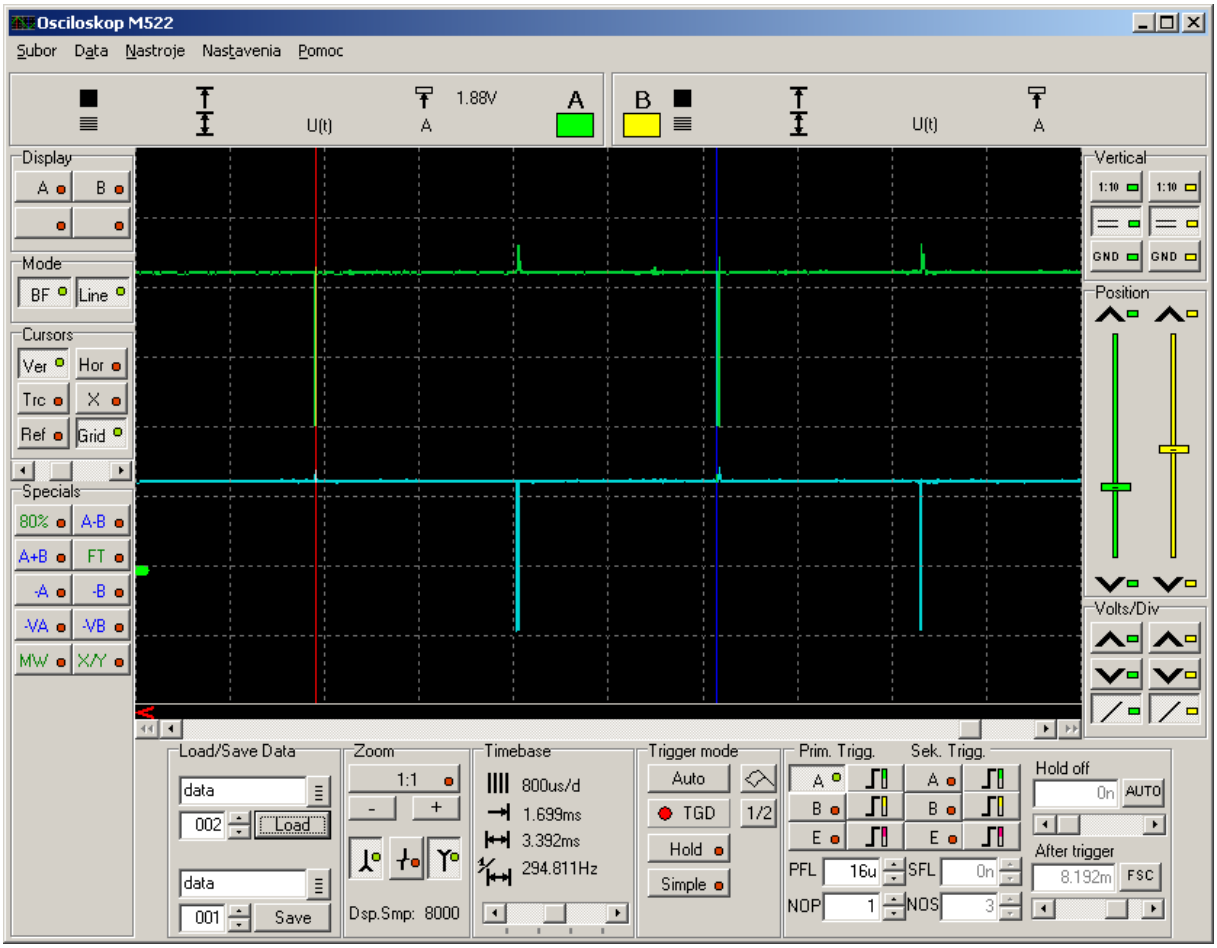
Po dosadení vyššie uvedených veličín do vzťahu vidíme, že regulátor pracuje správne

$$\beta_{el} = 2 \cdot \pi \cdot f_{rez} \cdot t_q \qquad (rad \cdot s^{-1}; \text{Hz, s}) \qquad (80.)$$

Po prevedení na stupne vynásobením $\beta_{el} \cdot \frac{180}{\pi}$

$$\beta_{el} = 360 \cdot f_{rez} \cdot t_q = 360 \cdot 413 \cdot 170 \cdot 10^{-6} = 25{,}27° \qquad (°; \text{Hz, s}) \qquad (81.)$$

Obrázok 238, Zapínacie impulzy pre jednotlivé diagonály tyristorov



Program v jazyku C51 pre tyristorový striedač

C:\Omega\data\Dropbox\Záloha\C51\termel\Striedac2\Striedac2.c

```
1   #include <reg552.h>
2   #include <intrins.h>
3   #include <absacc.h>
4   #include <stdlib.h>
5   #include <stdio.h>
6   #include <rtx51tny.h>
7   #include <dsw.h>
8
9   #define SIMULx
10
11  #define REGULATOR 0
12  #define ERRORS    1
13
14  //-------------------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------------------
15  #define XTAL 18.432E6
16  #define Frekvencia_horna 700              //478
17  #define Frekvencia_dolna 300
18  #define Cislo (XTAL/(12*2*Frekvencia_horna))
19  #define Tzmin ((XTAL*100E-6)/12)            //Tzmin=100us
20  #define Hranica_Dolna  (XTAL/(12*2*(Frekvencia_dolna))) // Dolna frekvencia (zaraska dolnej
    frekvencie)
21  #define Hranica_Horna  (XTAL/(12*2*(Frekvencia_horna)))
22  //-------------------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------------------
23  #define WLE 0x10
24  #define INTERVAL 200                      //Watchdog cca. 200ms
25  #define Watchdog ((PCON|=WLE),(T3=0x100-INTERVAL))
26  #define DCapture (IEN1&=~0x01)              //Zakazem akceptovanie pereusenia od Capture
    jednotky, kde je Tz
27  #define ECapture (IEN1|=0x01)             //Povolim akceptovanie pereusenia od Capture
    jednotky, kde je Tz
28  #define DCompare //(IEN1&=~0x10)             //Pozastavim pocas vypoctu Perioda++ a Perioda--
    generovanie impulzu v Compare jednotke
29  #define ECompare //(IEN1|=0x10)             //Povolim Compare jednotku
30  //-------------------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------------------
31
32  sbit I1 = P4^0;
33
34  int Per = 0x0000, Perioda = Cislo;
35
36  void Impuls1(void) interrupt 11 using 2
37  {
38   #define Dlzka 4      //Dlzka = 4 cca.20us s ostatnym kodom. Tu sa meni scasti dlzka impulzu !!!
39   unsigned char Temp=Dlzka;
40   I1=1;
41   Per+=Perioda;
42   CMH0=high(Per);
43   CML0=low(Per);       //Generujem pre podmienku CMH0+CML0=TMH2+TML2
44   Watchdog;
45   while(--Temp!=0x00);
46   I1=0;
47   CMI0=0;
48  }
49
50  struct Measure    {int Beg,End,Len;};          //Definujem strukturu pre meranie Tz a Pe
51  struct Measure Tz = {0x0000,0x0000,Tzmin};          //Uchovavam Tz
52  struct Measure Pe = {0x0000,0x0000,(int)(Tzmin*6.3)};   //Uchovavam dlzku periody Pe
53
54  void Capture(void) interrupt 6 using 3
55  {
56   #define CTP0 0x01
57   #define CTN0 0x02
58   switch(CTCON & (CTP0+CTN0))
59   {
60    case CTP0   : Tz.End=Pe.Beg=(256*CTH0+CTL0); Tz.Len=(Tz.End-Tz.Beg); break;
61    case CTN0   : Tz.Beg=Pe.End=(256*CTH0+CTL0); Pe.Len=(Pe.End-Pe.Beg); break;
62   }
63   CTCON^=(CTP0+CTN0);
64   CTI0=0;
65  }
66
67  void Timer(void) interrupt 3  //Vsetko napisem v assembleri ???
```

Page1

C:\Omega\data\Dropbox\Záloha\C51\termel\Striedac2\Striedac2.c

```
68  {
69   P1^=0x01;
70  }
71
72  void Initialize(void)
73  {
74   CMH0=high(Per);
75   CML0=low(Per);
76   IEN1|=0x10;       //Povolim komparacnu jednotku
77   IEN1|=0x01;       //Povolujem Capture jednotku
78   IP1|=0x10;        //Komparacna jednotka s vacsou prioritou ...
79   TM2CON=0x01;      //T2 s frekvenciou fosc/1 pretecie za 1*40.27ms
80   CTCON|=CTN0;      //Citlivost na prechod "0" CAPTURE 1
81  }
82
83  sbit ZAVES_SA      = P3^5;                          //Od tohoto okamihu znizujem
84  sbit SOM_ZAVESENY  = P3^4;
85
86  //-------------------------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------------------------
87  #define Kp 2.800                              //Proporcionalna konstanta Kp
88  #define dT 0.005                              //Interval vzorkovania pe regulator dT=20ms
89  #define Ti 0.041                              //Integracna konstanta Ti, je potrebne ju
    prehodnotit pri zmene menica ...
90  #define Td 0.001                              //Derivacna konstanta Td, ovplyvnuje strmost
    narastu pri zmene reg. odchylky, vsima si totiz strmost narastu odchylky !!!
91  //-------------------------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------------------------
92  float E0 = 0.000;                                 //Regulacna odchylka E0 nova
93  float E1 = 0.000;                                 //Regulacna odchylka E1 stara
94  float E2 = 0.000;                                 //Regulacna odchylka E2 moc stara
95  float Y0 = 0.000;                                   //Akcna velicina Y0  nova pre regulator
96  float Y1 = (float)Hranica_Horna;                            //Akcna velicina Y1 stara pre regulator
97  float KA = (Kp*(1.0+dT/(2.0*Ti)+(Td/dT)));                    //Konstanta KA prirastkoveho
    regulatora
98  float KB = (Kp*(1.0-dT/(2.0*Ti)+(2.0*Td/dT)));                  //Konstanta KB prirastkoveho
    regulatora
99  float KC = (Kp*(Td/dT));                              //Konstanta KC prirastkoveho regulatora
100 //-------------------------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------------------------
101
102 #define Pomer 4.500                                 //4.9 bolo OK ???
103
104 void Regulator(void) _task_ REGULATOR
105 {
106  unsigned char Counter0,Counter1;
107  float Pomer_vert;                             //Nastavenie Pomer_vert pri simulacii signalov
    programu START
108  bit R=0;
109  Initialize();
110  #ifdef SIMUL
111  ET1=1;
112  TR1=1;
113  #endif
114  while(1)
115   {
116    os_wait(K_TMO,5,NULL);                        //Cakam pokial nepride povel na uplynutie casu
117    if(ZAVES_SA==1)
118     {
119      Watchdog;
120    DCapture;                                 //Zakazem na chvilu prerusenie od CAPTURE
121    Pomer_vert=(float)Pe.Len/(float)Tz.Len;              //Vypocet trva cca. 800us  pri 18.432MHz
122    ECapture;
123    E0=Pomer-Pomer_vert;                                  //Vypocet trva cca. 131us  pri
    18.432MHz
124       Y0=Y1+(KA*E0)-(KB*E1)+(KC*E2);                                  //Vypocet trva cca.
    1064us pri 18.432MHz
125    if(Y0>=Hranica_Dolna)  Y0=Hranica_Dolna;
126    if(Y0<=Hranica_Horna)  Y0=Hranica_Horna;
127    DCompare;
128    Perioda=(int)Y0;
129    ECompare;
130      Y1=Y0;
131    E2=E1;
```

C:\Omega\data\Dropbox\Záloha\C51\termel\Striedac2\Striedac2.c

```
132         E1=E0;
133       if(Pomer_vert>Pomer)        R=1;
134       if(Pomer_vert<(Pomer-0.9))  R=0;
135         if(R==1) {Counter0++; Counter1=0x00;}
136       if(R==0) {Counter1++; Counter0=0x00;}
137       if(Counter0>=5) {SOM_ZAVESENY=1; Counter0=0x00; os_wait(K_TMO,20,NULL);}
138       if(Counter1>=20){SOM_ZAVESENY=0; Counter1=0x00; os_wait(K_TMO,20,NULL);}
139        }
140       else
141      {
142         Watchdog;
143       os_wait(K_TMO,200,NULL);
144       Y0=Y1=(float)Cislo;
145       Perioda=Cislo;
146       SOM_ZAVESENY=0;
147       Counter0=0;
148       Counter1=0;
149      }
150      }
151    }
```

17.17. Skalárne riadenie asynchrónneho motora pomocou modulácie SWPWM

Príklad: Pomocou programovacieho jazyka A51, C51, alebo RTX51 vytvorte program, ktorý bude riadiť frekvenčný menič ktorý napája jednosmerný motor s cudzím budením, jednofázový komutátorový motor a trojfázový asynchrónny motor ďalej len ASM s klietkovou kotvou. Daný menič musí byť schopný napájať jednosmerným a striedavým napätím s premenlivou veľkosťou napätia a frekvencie daný pohon. Napájacie napätie pre motor bude v rozsahu 0V po 400V_{AC} s frekvenciou 0 až 150Hz. Výstupy portu mikroprocesora C504 budú s pomocou 20kHz PWM ovládať trojfázové mostové zapojenie s IGBT, ktorý plní funkciu napäťového striedača. Striedač bude ovládaný pomocou dvojice tlačidiel. Dve tlačidlá budú slúžiť na voľbu otáčok motora a zároveň na zmenu smeru otáčania. Zároveň program doplňte o ovládanie pomocou sériového rozhrania RS232 s nasledujúcimi parametrami: 2400b.s^{-1}, 8 dátových bitov, 1 štart a stop bit, bez parity. Výstupné napätie bude modulované pomocou SWPWM (Sinusoidal Weight Pulse Width Modulation).

Obrázok 246 bližšie zobrazuje vytvorenie rotujúceho magnetického poľa statora asynchrónneho motora (ASM). Obrázok 247 až Obrázok 251 podrobnejšie znázorňujú problematiku riadenia ASM za pomoci mikroprocesora C504 s riadiacimi a regulačnými obvodmi.

Rozbor úlohy:

Pre otáčky asynchrónneho motora bude platiť vzťah:

$$\varpi_1 = \frac{2 \cdot \pi \cdot f}{p} \qquad \left(rad \cdot s^{-1}; Hz, 1\right) \qquad (82.)$$

alebo

$$n_1 = \frac{60 \cdot f}{p} \qquad \left(ot \cdot \min^{-1}; Hz, 1\right) \qquad (83.)$$

Pre magnetický tok v motore musí platiť vzťah:

$$\Phi = \frac{U_1}{4{,}44 \cdot f_1 \cdot N_V \cdot k_V} = k \frac{U_1}{f_1} \qquad \left(Wb; V, Hz, 1{,}1\right) \qquad (84.)$$

Ak chceme zachovať konštantný magnetický tok stroja musí platiť:

$$\frac{U_1}{f_1} = const \qquad (V, Hz; 1) \qquad (85.)$$

Pre maximálny moment stroja bude teda platiť vzťah:

$$M_M = \frac{3 \cdot U_1^2}{2 \cdot \varpi_1 \cdot X_K} = K_1 \frac{U_1^2}{f_1^2} \qquad \left(N \cdot m; V, rad \cdot s^{-1}, \Omega\right) \qquad (86.)$$

Riadiaci algoritmus mikroprocesora musí ovládať striedač tak, aby bolo možné na jeho výstupných svorkách vygenerovať striedavé napätie ktorým je pohon napájaný. Pomocou PWM modulácie je nutné vyrobiť napätie sínusového priebehu. Z externej pamäti RAM mikroprocesora musí byť výber vzorky napätia každých $\Delta T = 50\mu s$ čo je hodnota napäťovej vzorky PWM modulátora tak, aby bolo možné vyrobiť napätia s požadovanou amplitúdou a frekvenciou. Použitie operačného systému RTX51 si musíme veľmi dobre uvážiť, pretože musíme zabezpečiť periodický výber hodnôt z pamäti RAM každých $50\mu s$ na každú fázu, čo je časovo veľmi náročné.

Východiskom z tohto problému by bolo použitie internej pamäti programu mikroprocesora, kde by sa nachádzali už vopred vypočítané vzorky napätia pre jednotlivé frekvencie asynchrónneho motora. Takto by sme zabezpečili uloženie vzoriek v rozsahu pracovných frekvencií 0 až 50Hz. Množstvo potrebnej pamäti programu potrebnej pre uloženie vzoriek by sme mohli vypočítať pomocou nižšie uvedeného vzťahu s tým, že budeme do pamäti ukladať len ½ sínusového priebehu, pretože platí vzájomná symetria kladnej a zápornej polvlny.

Vzťah na výpočet množstva pamäti pre uloženie vzoriek PWM:

$$m = \sum_{i=1}^{n} \frac{1}{2.048 \cdot i} \cdot f_{PWM} \qquad (kB; 1, Hz, Hz) \qquad (87.)$$

n	počet pracovných frekvencií ASM motora	(-)
i	***i***-tá pracovná frekvencia ASM motora	(Hz)
f_{PWM}	pracovná frekvencia PWM modulátora	(Hz)
m	množstvo pamäti programu pre vzorky	(kB)

Pre $n = 50$ a $f_{PWM} = 20kHz$ bude potrebných 43.94kB pamäti programu.

Ak by sme ale chceli zvýšiť počet pracovných frekvencií ($n = 100$ a pri zachovaní f_{PWM}) ASM pohonu, vďaka symetrii sínusového priebehu by stačilo do pamäti ukladať len ¼ periódy napätia pre ASM motor. Takto by postačovalo využiť pre vzorky len $m = \sum_{i=1}^{n} \frac{1}{2 \cdot 2.048 \cdot i} \cdot f_{PWM} = 25.33 kB$ pamäti programu. Zvyšná pamäť by mohla byť využitá pre riadiaci program pohonu.

Najväčším problémom je množstvo vzoriek, ktoré sú ukladané do pamäti programu mikroprocesora C504. Ak chceme zachovať konštantný magnetický tok je výhodnejšie a rýchlejšie nahradiť operáciu násobenia operáciou sčítania podľa nasledujúceho vzťahu. Tento príklad je zjednodušený variant predchádzajúceho riešenia.

$$U = A \cdot \sin B = \cos(\arccos A) \cdot \sin B \tag{88.}$$

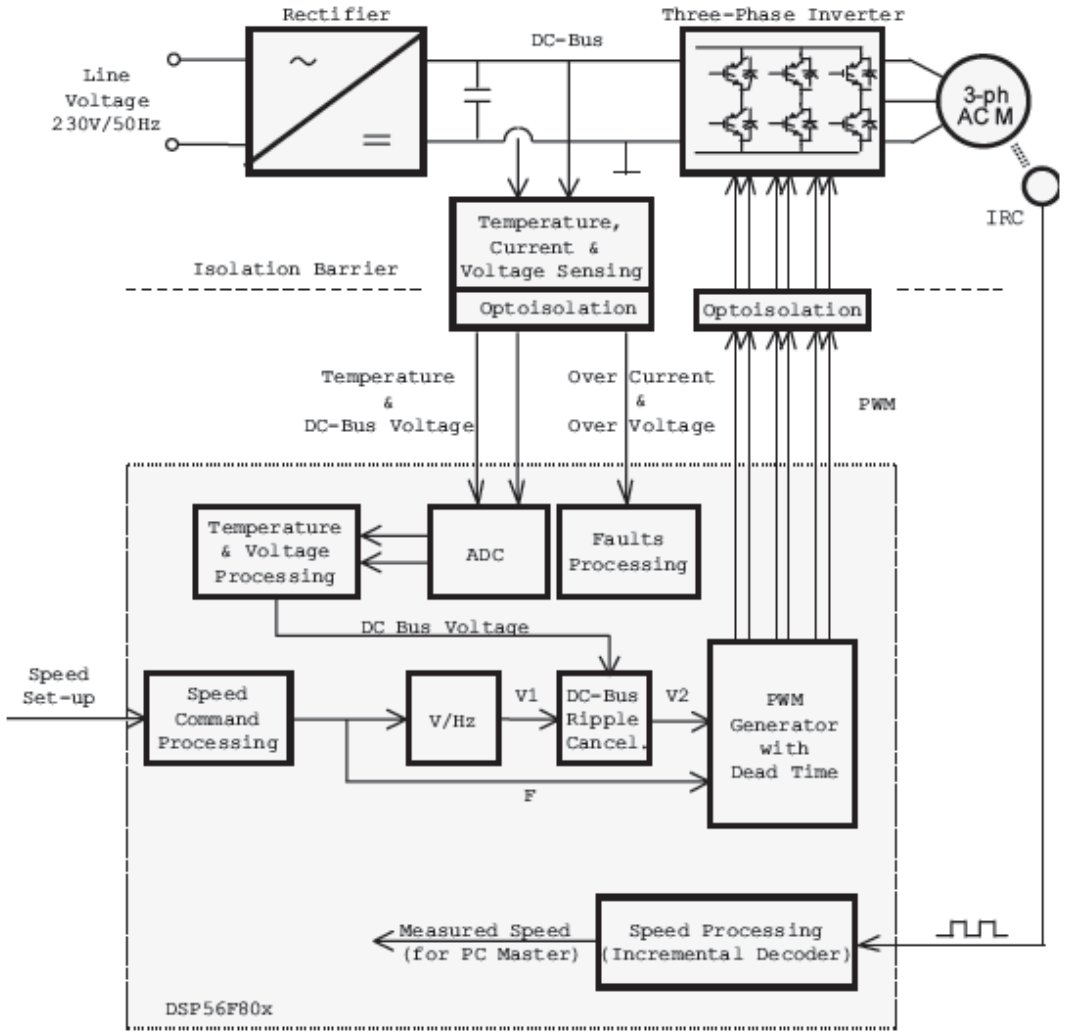
Ak bude $A' = \arccos A$, potom sa nahradí násobenie tzv. sčítacím teorémom

Po rozklade dostaneme vzťah:

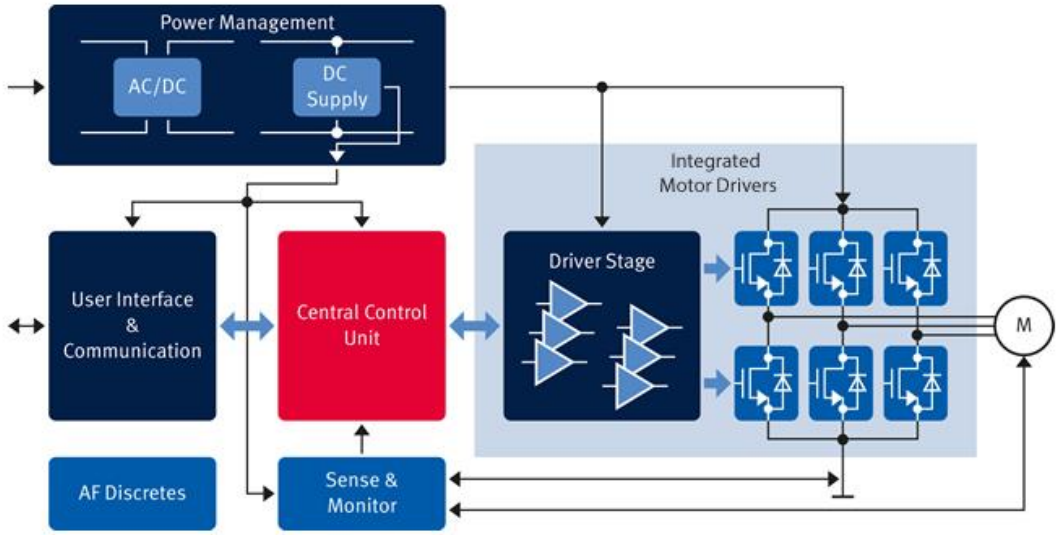
$$U = \cos A' \cdot \sin B = \frac{1}{2} \cdot \left[\sin\left(B - A'\right) + \sin\left(B + A'\right) \right] \tag{89.}$$

Kde B predstavuje vzorku napätia ktorá je uložená v tabuľke a je upravená o hodnotu A', ktorá zabezpečí zmenu napätia na pohone tak, aby platil konštantný pomer $\frac{U}{f}$.

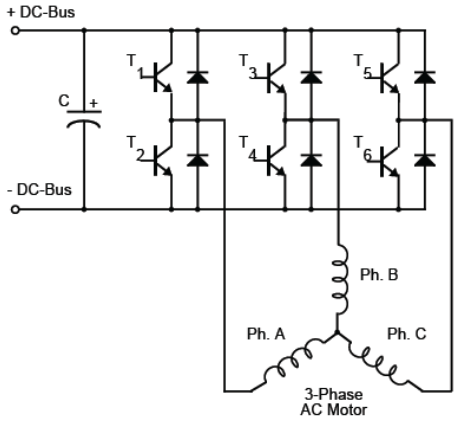
Obrázok 239, Schéma zapojenia skalárneho riadenia 3fázového ASM motora



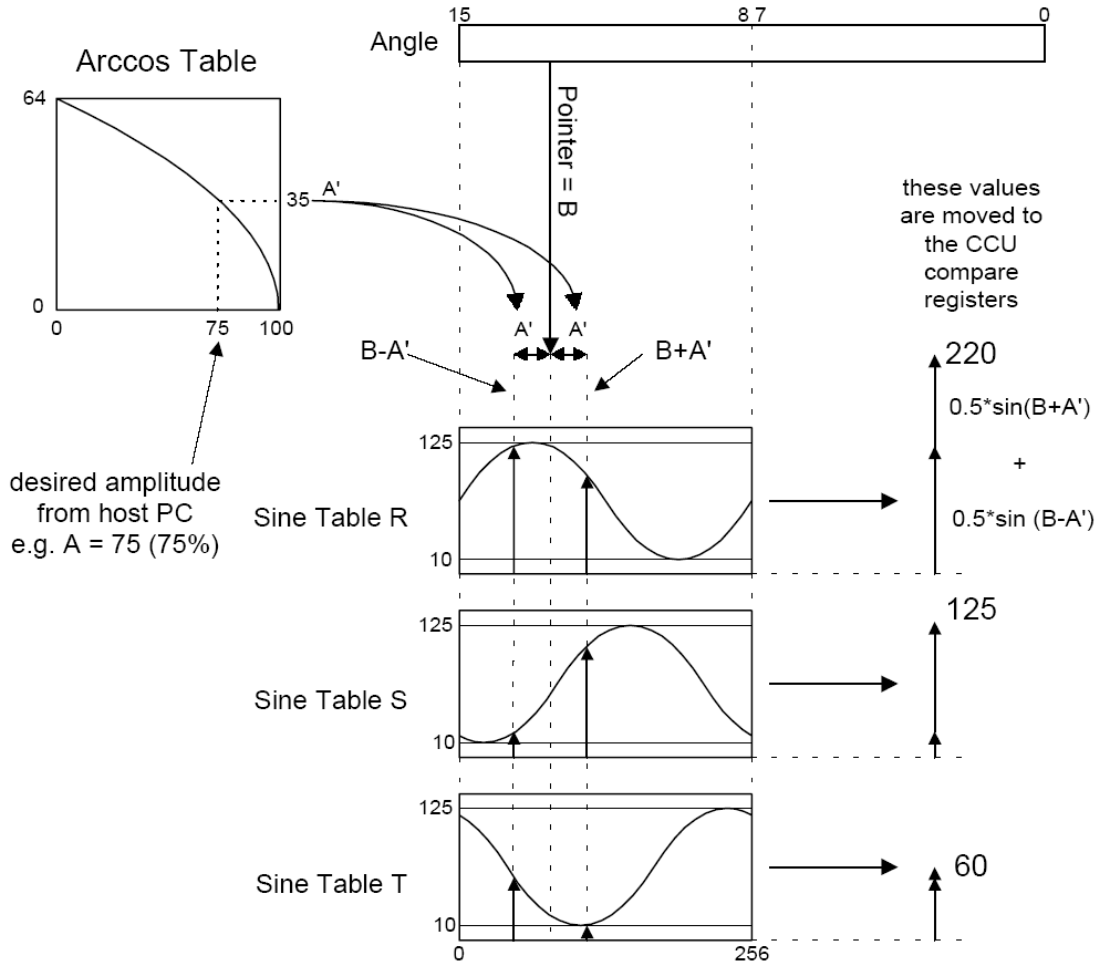
Obrázok 240, Bloková schéma skalárneho ASM pohonu s procesorom Infineon XC866



Obrázok 241, Výkonová časť 3-fázového striedača ASM

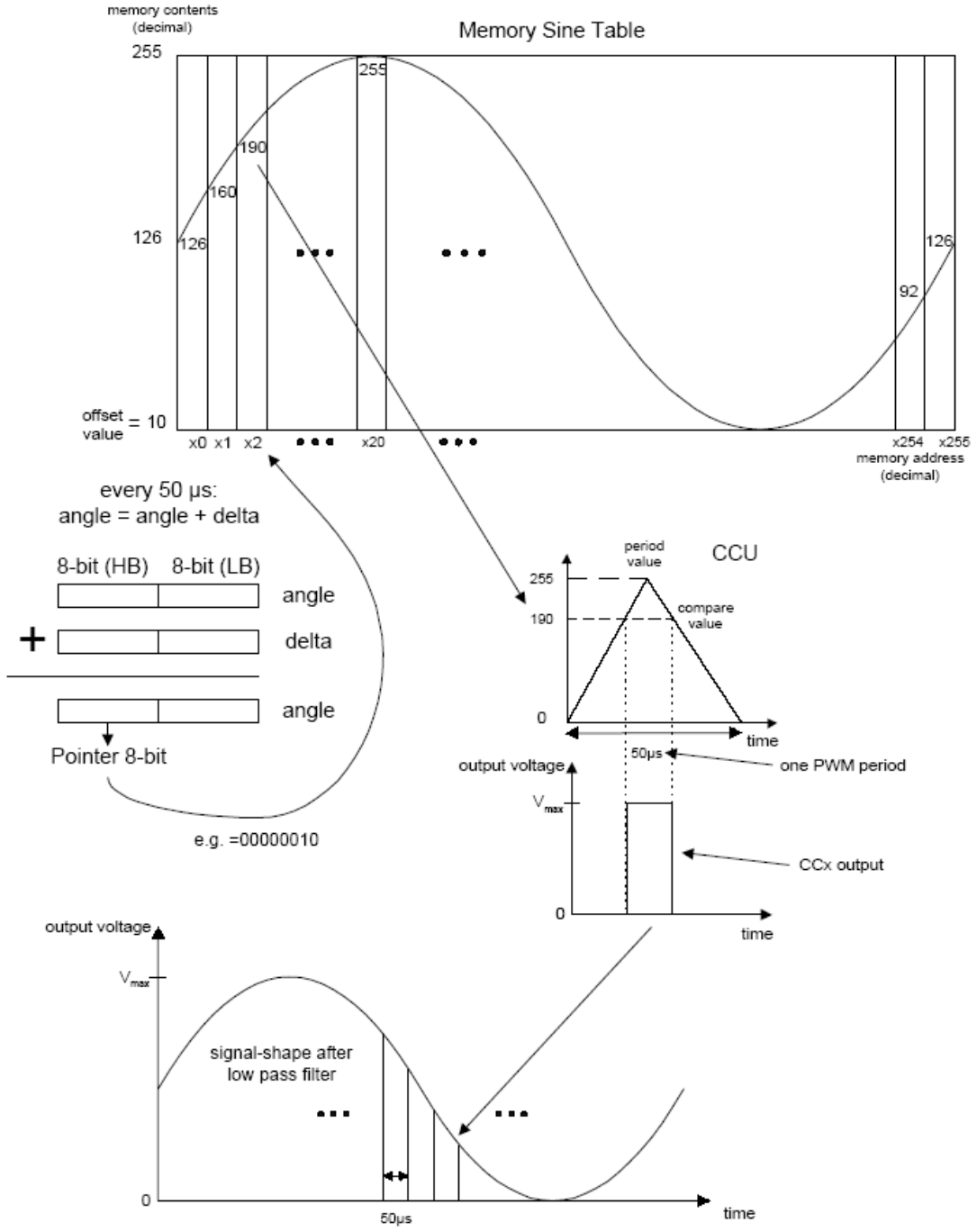


Obrázok 242, Princíp zmeny napätia na asynchrónnom pohone pomocou tabuľky



Vyššie uvedený spôsob umožňuje značne urýchliť výpočet napätia pre jednotlivé fázy tým, že mikroprocesor už má vopred vypočítané hodnoty sínusových vzoriek a časovo náročné násobenie je nahradené jednoduchým súčtom dvoch 8 bitových čísel pomocou ďalšej tabuľky.

Obrázok 243, Princíp tvorby výstupného sínusového napätia s tabuľkou v pamäti C504



Príklad výpočtu pohonu:

Frekvencia PWM je v tomto prípade:

$$f_{PWM} = 20 \cdot 10^3 Hz$$

Časovač T1 musí teda byť schopný generovať prerušenie každých:

$$T_{PWM} = \frac{1}{f_{PWM}} = \frac{1}{20 \cdot 10^3} = 50 \cdot 10^{-6} = 50 \mu s \qquad \text{(s; Hz)} \qquad (90.)$$

$$PERIODE = \frac{f_{OSC}}{prescaler \cdot f_{PWM} \cdot 2} = \frac{40 \cdot 10^6}{4 \cdot 20 \cdot 10^3 \cdot 2} = 250 \qquad \text{(1; Hz, Hz)} \qquad (91.)$$

Pre zabezpečenie tzv. „dead time“ je potrebné dodržať minimálne $t_{dead} = 1 \mu s$

$$OFFSET = \frac{t_{dead} \cdot f_{OSC}}{prescaler} = \frac{1 \cdot 10^{-6} \cdot 40 \cdot 10^{-6}}{4} = 10 \qquad \text{(1; s, Hz, 1)} \qquad (92.)$$

Minimálnu a maximálnu frekvenciu je možné vyjadriť nasledujúcim vzťahom:

$$f_{MIN} = \frac{1}{65536 \cdot T_{PWM}} = \frac{1}{65536 \cdot 50 \cdot 10^{-6}} = 0.305 Hz \qquad \text{(Hz; s)} \qquad (93.)$$

$$f_{MAX} = \frac{1}{256 \cdot T_{PWM}} = \frac{1}{256 \cdot 50 \cdot 10^{-6}} = 78.125 Hz \qquad \text{(Hz; s)} \qquad (94.)$$

Vzťah potrebný pre zadanie požadovanej frekvencie asynchrónneho pohonu vyjadríme ako:

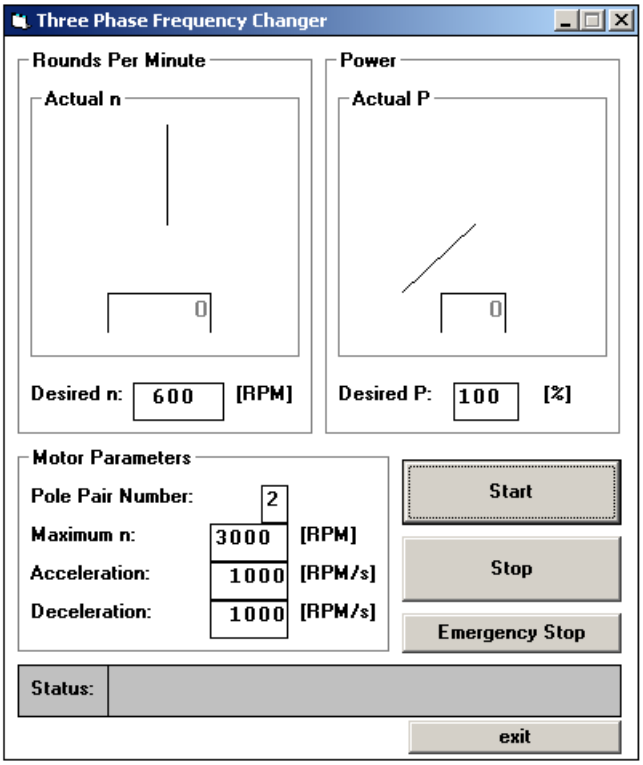
$$angle = angle + delta \qquad \text{(1; 1, 1)} \qquad (95.)$$

Kde $angle$ je 16 bitová premenná a vyššia časť premennej obsahuje ukazovateľ do tabuľky vypočítaných hodnôt sínusových vzoriek a nižšia časť premennej $delta$ znamená veľkosť- „rýchlosť“ prírastku pre výber hodnoty sínusu z tabuľky. Hodnota $delta \in (1 \div 256)$ zvyšuje obsah premennej $angle$ o hodnotu $delta$ každých $50 \mu s$.

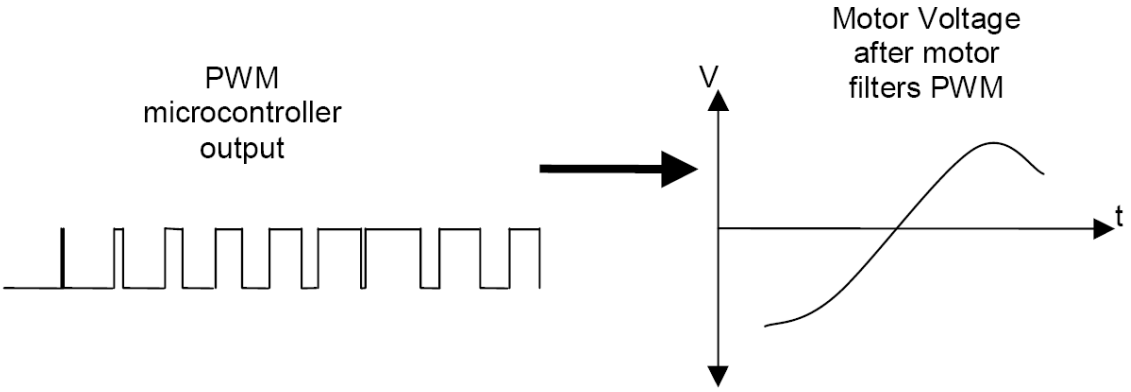
$$delta_{MAX\_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 65536 = 256 \qquad \text{(1; Hz, Hz)} \qquad (96.)$$

$$delta_{MIN\_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 256 = 1 \qquad \text{(1; Hz, Hz)} \qquad (97.)$$

Obrázok 244, Ovládací software pre pohon asynchrónneho motora

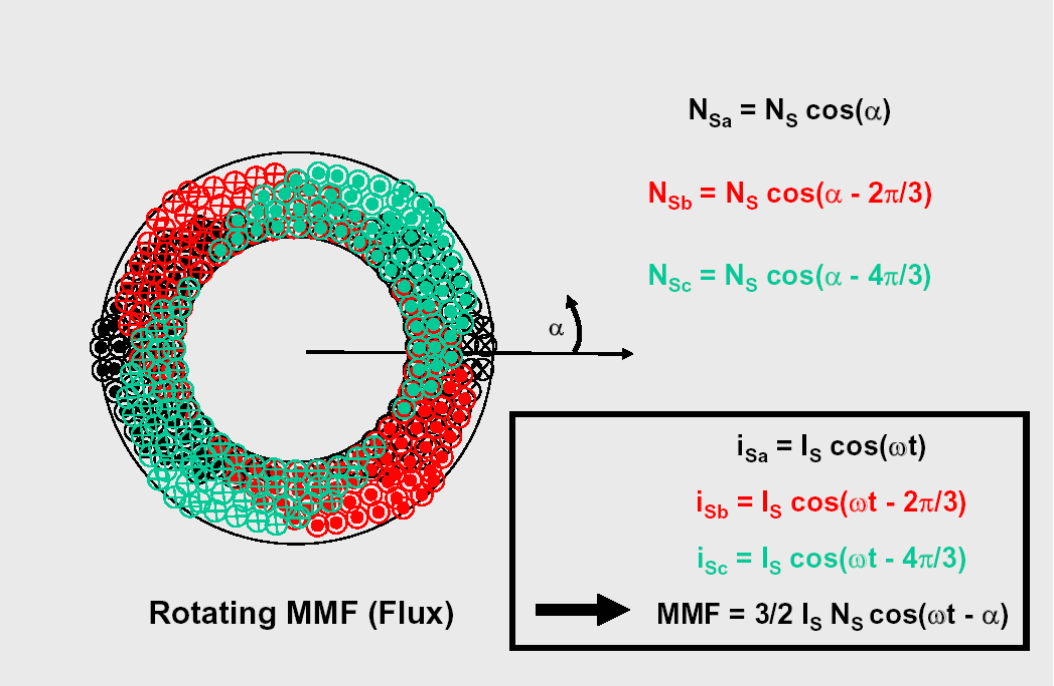


Obrázok 245, Napätie generované metódou SWPWM[223].

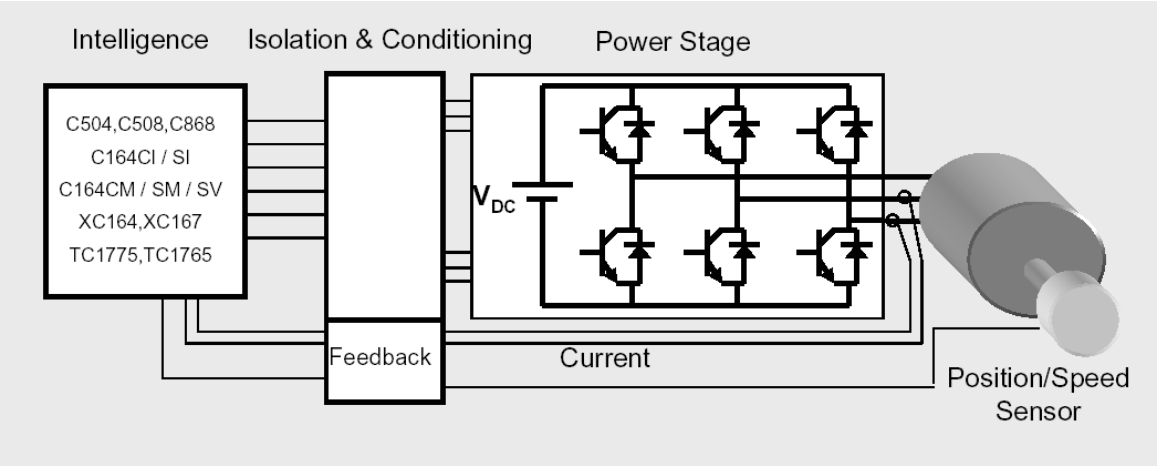


[223] SWPWM – Sinusoidal Weight Pulse Width Modulation

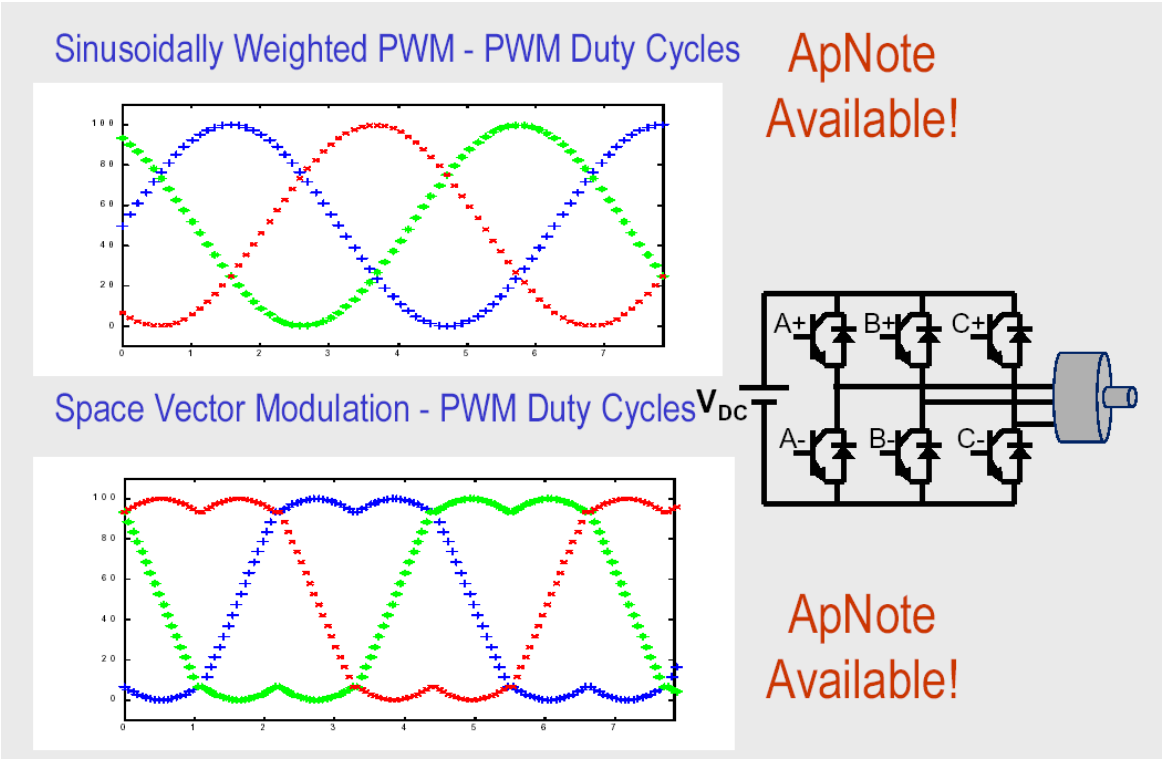
Obrázok 246, Vytvorenie rotujúceho magnetického poľa statora ASM



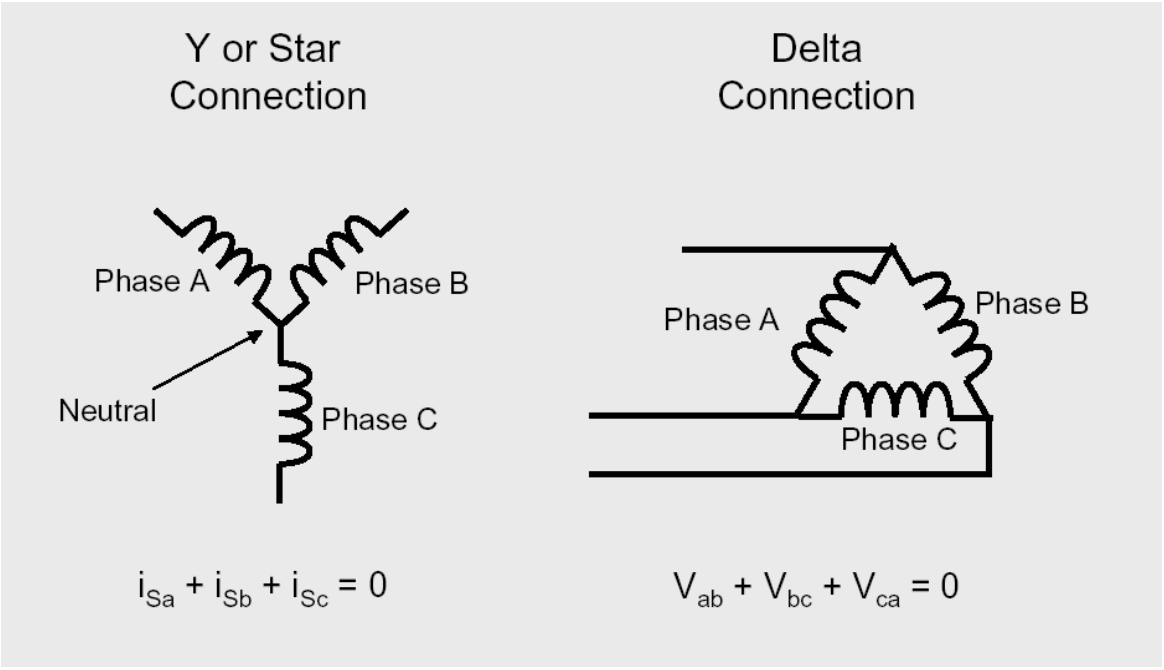
Obrázok 247, Schéma riadenia asynchrónneho motora s uzavretou regulačnou slučkou



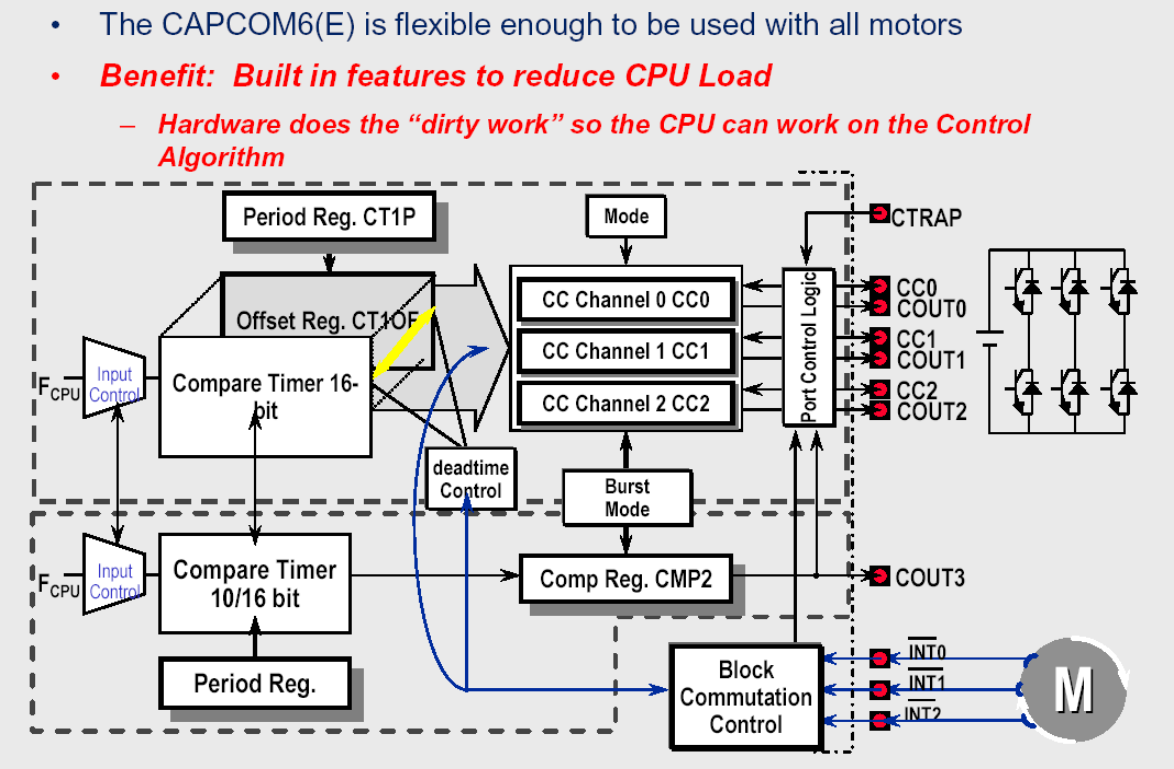
Obrázok 248, Principiálne znázornenie tvorby viacfázovej sústavy



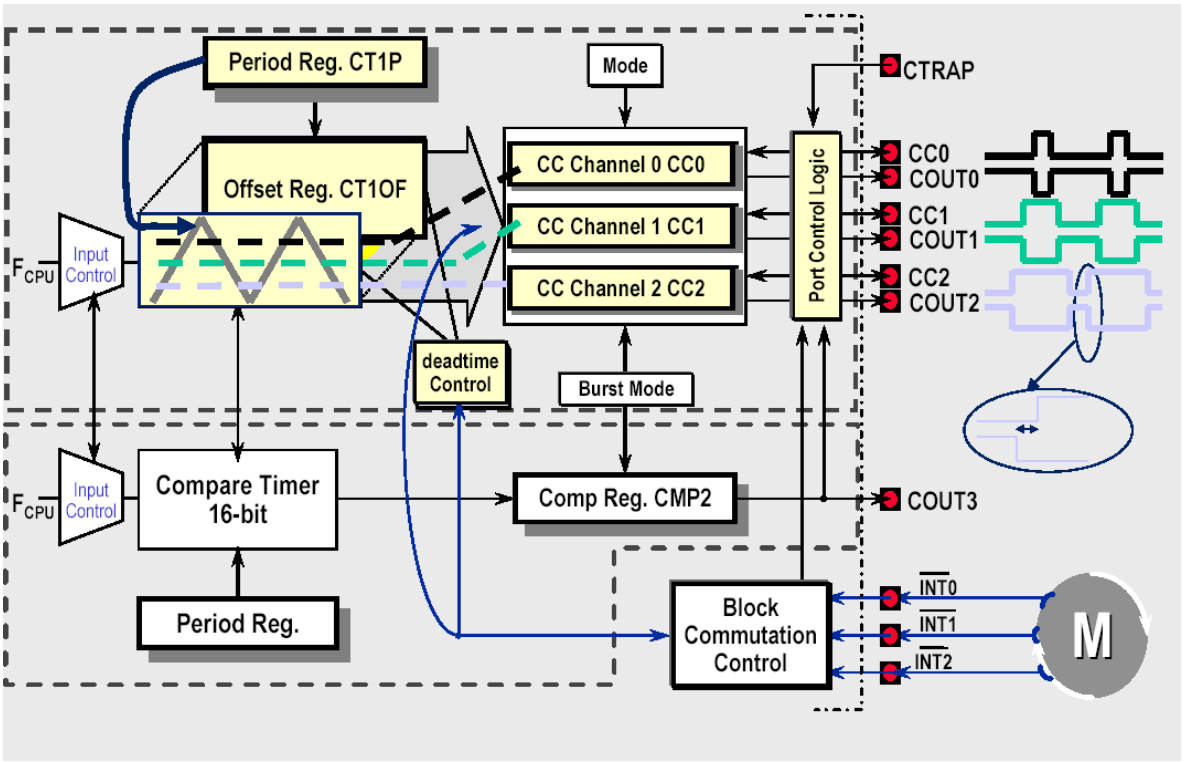
Obrázok 249, Základné zapojenie statora asynchrónneho motora



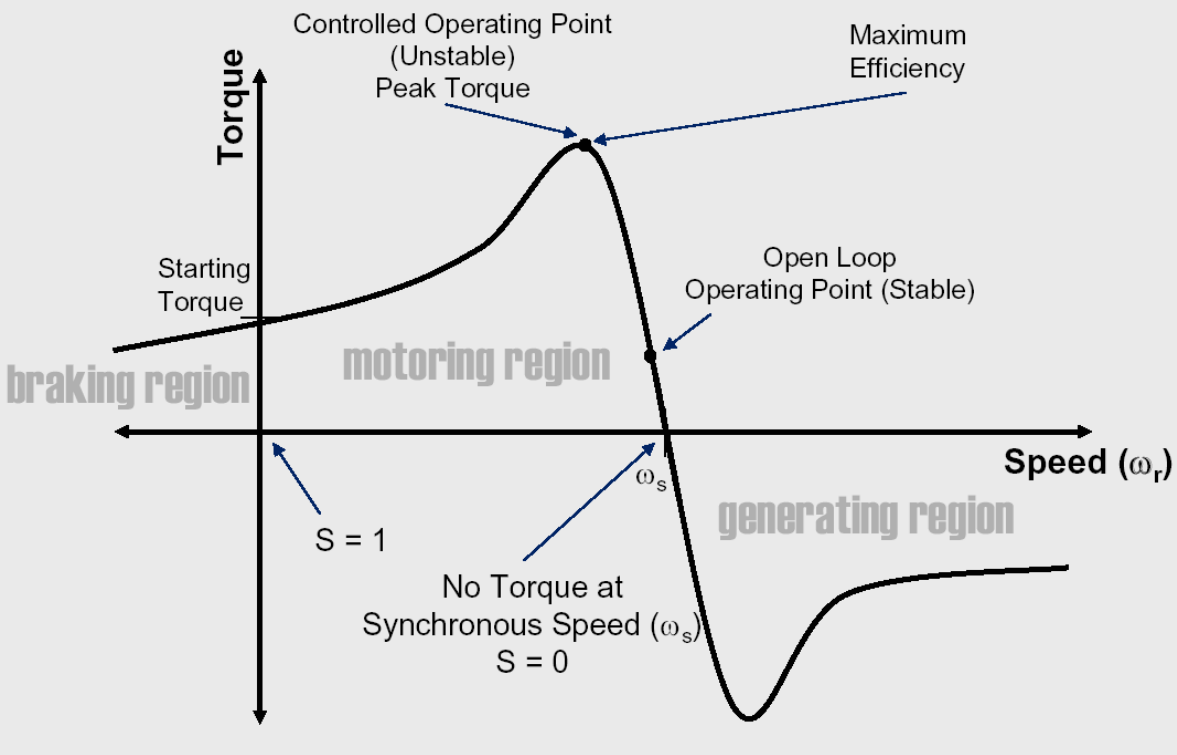
Obrázok 250, Riadenie asynchrónneho motora s SWPWM (Sinusoidal Weight PWM)



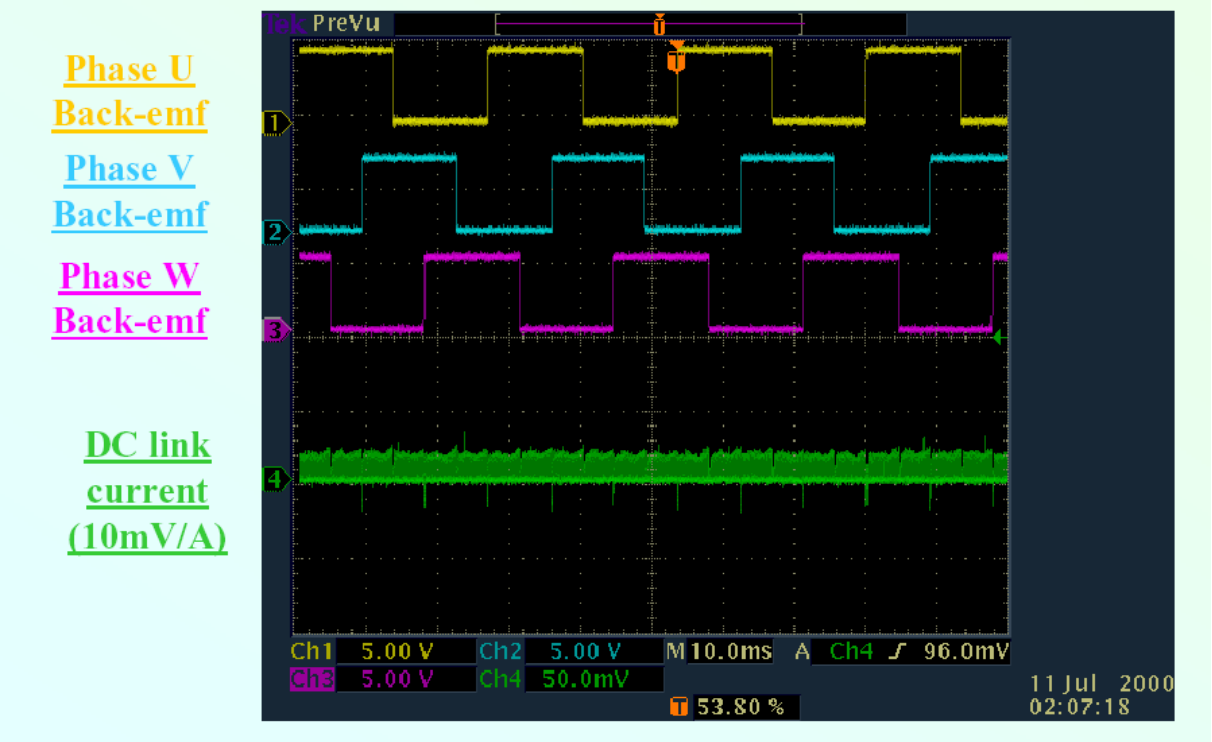
Obrázok 251, Riadenie asynchrónneho motora pomocou SVM (Space Vector Modulation)



Obrázok 252, Momentová charakteristika asynchrónneho motora



Obrázok 253, Namerané priebehy napätia na porte mikroprocesora C504



Zápis programu:

C:\Omega\data\Dropbox\Záloha\C51\ASM motor\ASM Motor.c

```
1   #include <c:\keil\c51\inc\infineon\reg504.h>
2   #include <c:\keil\c51\inc\stdio.h>          /* standard I/O .h-file               */
3   #include <c:\keil\c51\inc\ctype.h>          /* character functions                */
4   #include <c:\keil\c51\inc\string.h>         /* string and memory functions        */
5   #include <c:\keil\c51\inc\stdlib.h>
6   #include <c:\keil\c51\inc\rtx51tny.h>
7   #include <c:\keil\c51\inc\absacc.h>
8   #include <c:\keil\c51\inc\math.h>
9
10  #define  Fastx
11
12  #define  PORT1  P1
13  #define  PORT2  P2
14
15  enum  Switch  {ON=0, OFF=1};
16
17  sbit  Tlac  = P3^0;
18
19  void  Pwm(void)
20  {
21  }
22
23  void  Init(void)
24  {
25   #define  XTAL  40E6
26   TH1=TL1=256-(unsigned  char)((long  int)XTAL/(long  int)240000);   //Generujem prerusenie kazdych
    50us
27   TMOD|=0x20;
28   ET1=1;
29   PT1=1;
30   EA=1;
31  }
32
33  #define  frek_pwm  20000
34  #define  Krok  50.0E-6
35  xdata  int  Uac,Umax=255;
36  xdata  float  t,w,u_float,n;
37  xdata  unsigned  char   f,p,U;
38  unsigned  int  i,Pocet_Vzoriek;
39  #ifdef  Fast
40  xdata  unsigned  char  x[10000];
41  #else
42  #include <c:\keil\c51\examples\asm motor\init_mem.c>
43  void  *ptr;
44  #endif
45
46  void  Calcul(void)
47  {
48   #define  MEM_BEGIN   0x1000
49   #define  MEM_END   0x8000
50   i=0;
51   t=0;
52   w=2*3.14*f;
53   Uac=(Umax/50.0)*f;
54   n=60.*f/2;
55   if(Uac>=Umax)  Uac=Umax;
56   Pocet_Vzoriek=frek_pwm/(2*f);
57   init_mempool(&XBYTE[MEM_BEGIN],MEM_END-MEM_BEGIN);  //Inicializujem dostupnu pamat pre vzorky
    sinusovky deklarovane v subore INIT_MEM.C
58   #ifndef  Fast
59   if(ptr!=NULL)  free(ptr);
60   ptr=malloc(Pocet_Vzoriek);
61   #endif
62   for(i=0x00;  i<Pocet_Vzoriek;  i++)
63    {
64     u_float=Uac*sin(w*t);
65     U=(unsigned  char)u_float;
66     t=t+Krok;
67     #ifdef  Fast
68     x[i]=U;
69     #else
70     ((unsigned  char  *)ptr)[i]=U;
71     #endif
72    }
```

Page1

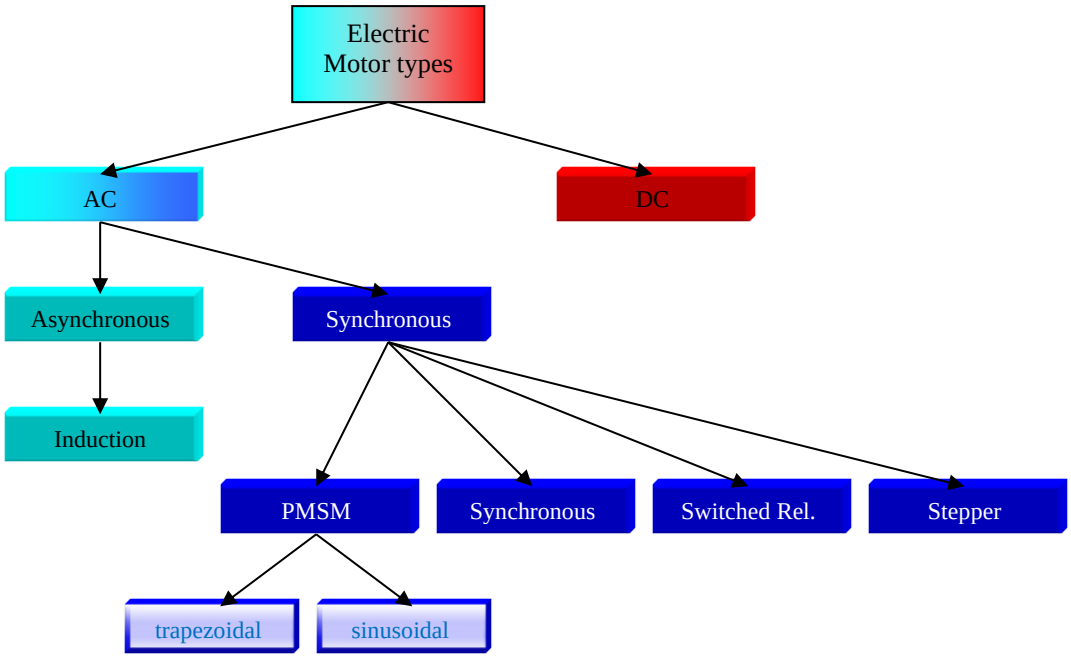
C:\Omega\data\Dropbox\Záloha\C51\ASM motor\ASM Motor.c

```
 73     i=0;
 74    }
 75
 76    bit  DIAG;
 77
 78    void  Int_T1 (void) interrupt  3 using  2
 79    {
 80     #ifdef  Fast
 81     if (DIAG == ON)  PORT1 =x[i++];  else  PORT2 =x[i++];
 82     #else
 83     if (DIAG == ON)  PORT1 =((unsigned  char  *) ptr)[i++];  else  PORT2 =((unsigned  char  *) ptr)[i++];
 84     #endif
 85     if (i== Pocet_Vzoriek)
 86      {
 87       i=0x00;
 88       DIAG =~ DIAG;
 89       PORT1 =PORT2 =0x00;
 90      }
 91    }
 92
 93    void  main (void)
 94    {
 95     Init ();
 96     Pwm ();
 97     f=50.0;
 98     Uac =255;
 99     Calcul ();
100     TR1 =1;
101     while (1)
102      {
103       if (Tlac == 0)
104        {
105       TR1 =0;
106       PORT1 =PORT2 =0x00;
107          Calcul ();
108       TR1 =1;
109      }
110      }
111    }
112
113
```

17.18. Vektorové riadenie asynchrónneho motora

V automobilovom priemysle sú viacfázovými motormi vo väčšej miere nahrádzané komutátorové motory. Ako príklad môžeme uviesť synchrónny motor s permanentnými magnetmi (PMSM – Permanent Magnetic Synchronous Motors), ktoré sú nazývané ako bezkomutátorové synchrónne motory nazývané ako Brushless DC motors. Pre výborné vlastnosti ako vysoký požadovaný výkon a konštantný moment motora je predurčený do výkonovej elektroniky a moderných elektrických pohonov. Pre optimálne využitie týchto vlastností je používané vektorové riadenie – modulácia. Bližšie informácie (Vittek, 2004).

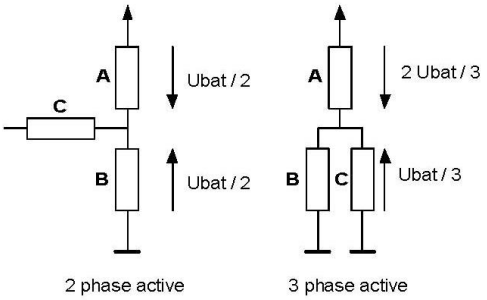
Obrázok 254, Rozdelenie elektrických pohonov



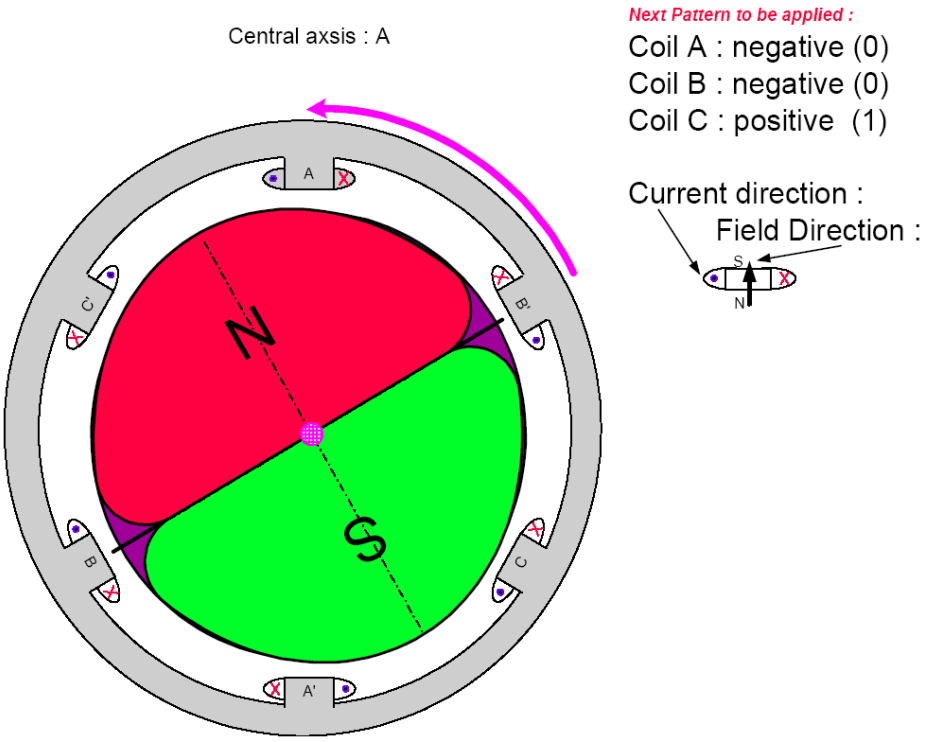
17.18.1. Výhody vektorového riadenia SVM (Space Vector Modulation)

- ✓ Vyššia dynamika riadenia
- ✓ Vysoká účinnosť dosahujúca až 86% oproti 75% pri klasickej SWPWM
- ✓ nižšie rýchlosti otáčania pri konštantnom momente
- ✓ konštantný moment pohonu
- ✓ lepší rozbeh pohonu s riadením momentu zvratu
- ✓ výborná dynamika pohonu s SVM

Obrázok 255, Základný rozdiel medzi SWPWM a SVM



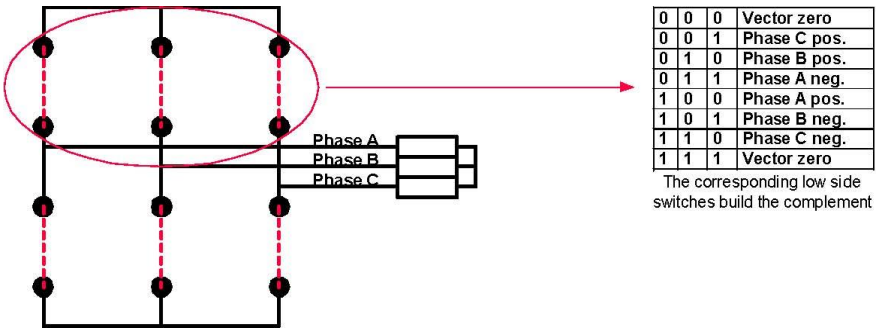
Obrázok 256, Detailné znázornenie magnetického obvodu PMSM s jedným pólom



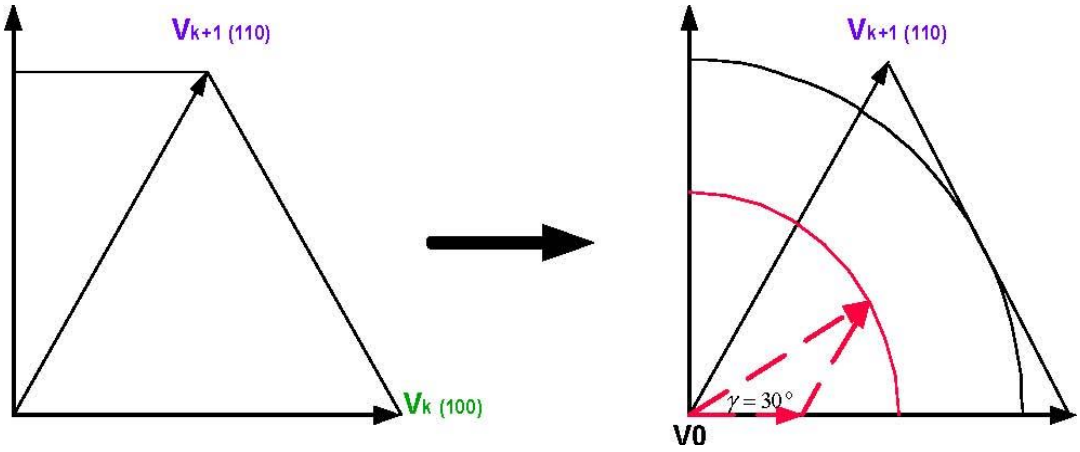
17.18.2. Principiálny opis SVM

Jednou z bežných metód reprezentácie SVM je model trojfázového modelu napájacieho zdroja, ktorý je schopný dodávať na výstupných svorkách kladné a záporné napätie. Obrázok 257 vysvetľuje, že môže nastať 8 možných stavov na spínacích prvkoch výkonového meniča. Každý stav z nich je reprezentovaný ako vektor. Tento spôsob umožňuje vytvoriť v rotujúci vektor napätia vo výkonovom meniči a môže sa nachádzať v niektorom zo 6 stavov. Zostávajúce 2 stavy sú nazývané ako nulové vektory (111, 000) a sú pripojené na GND, alebo Vcc. Nulové vektory sa nachádzajú vo vnútri geometrického útvaru šesťuholníka (hexagón) ako zobrazuje nasledujúci obrázok. Uhol medzi dvomi susednými sektormi je γ=60° elektrických.

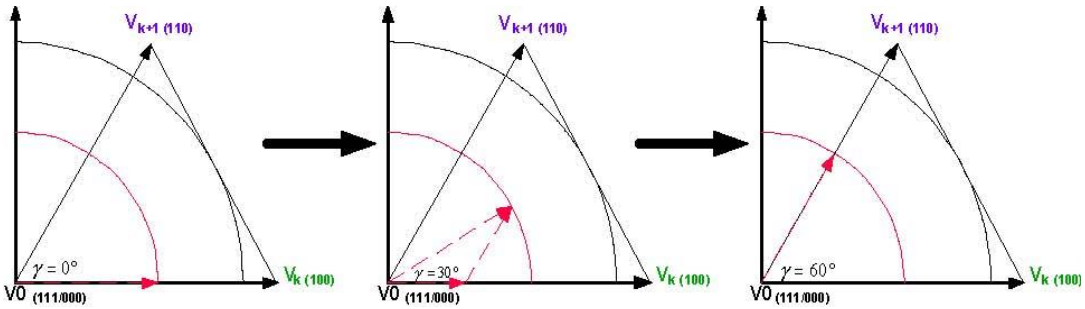
Obrázok 257, Spínací plán výkonového meniča s riadením SVM



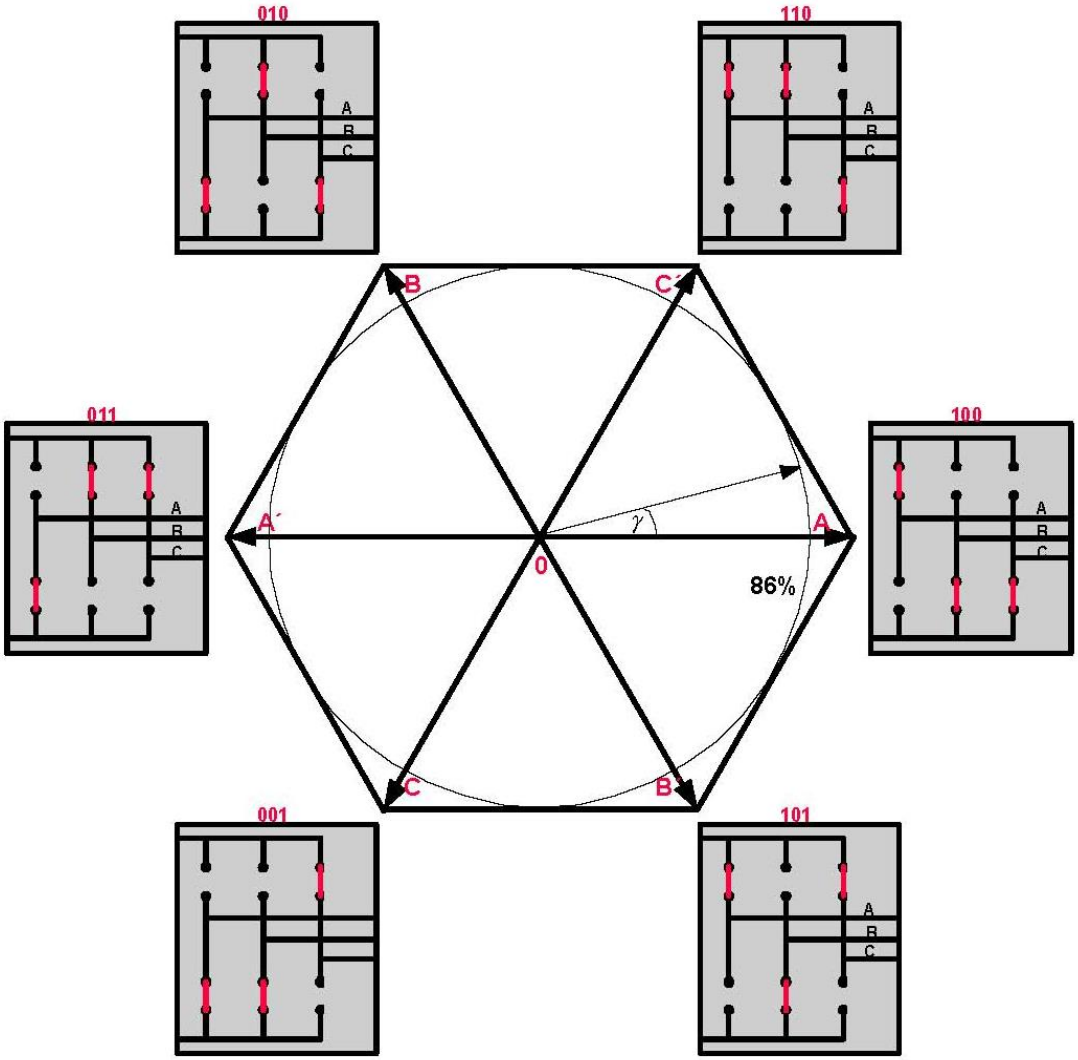
Obrázok 258, Znázornenie rotujúceho vektora magnetického poľa pohonu s SVM



Obrázok 259, SVM modulácia pre γ (0°,30°, 60°)



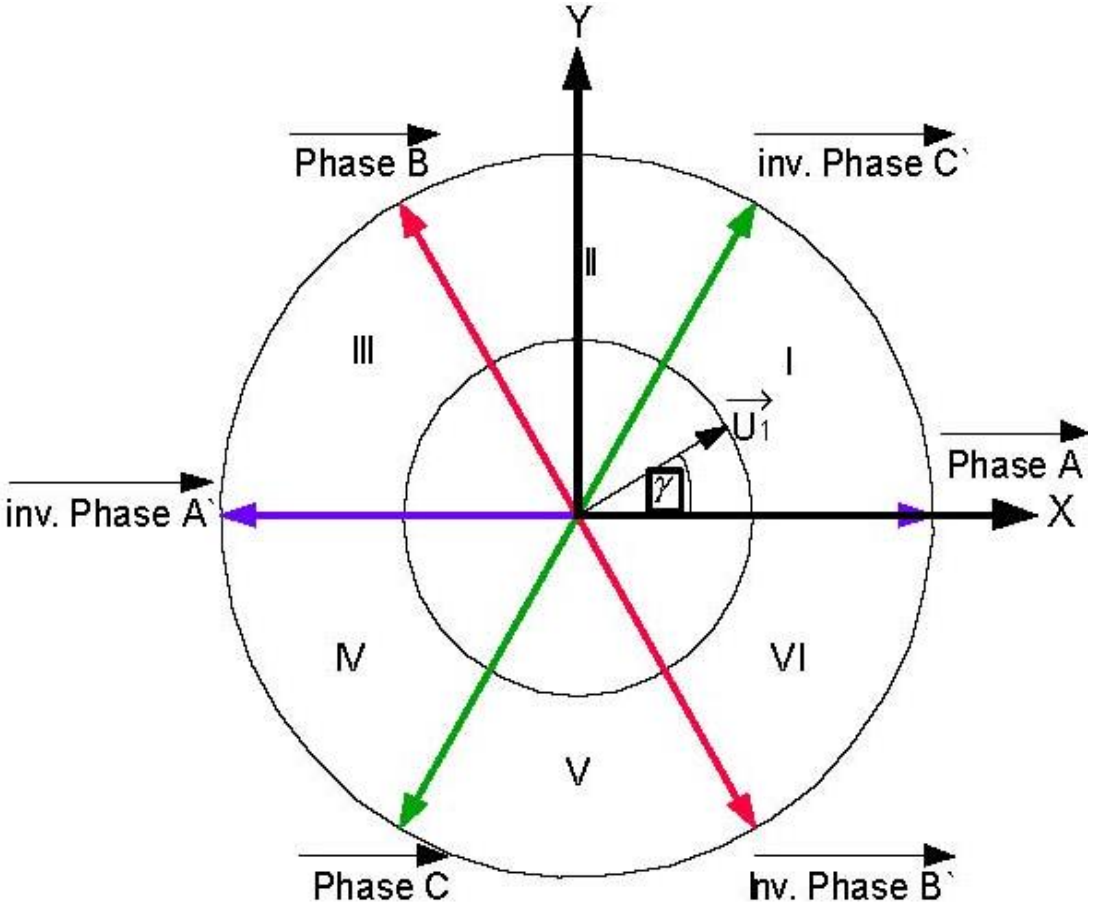
Obrázok 260, Vytvorenie rotujúceho vektora elektrického napätia pomocou SVM



17.18.3. Generovanie výstupného napätia pomocou SVM

SVM modulácia je založená na princípe, ktorý umiestňuje vektor napätia spínaním výkonových prvkov vo výkonovom meniči – invertore medzi 6 sektormi. Vektor napätia je teda reprezentovaný veľkosťou strednej hodnoty napätia dvoch susedných aktívnych veľkostí vektorov v priebehu jednej periódy PWM.

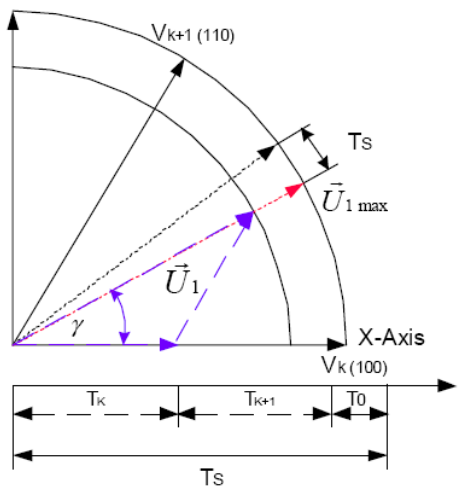
Obrázok 261, Sektory SVM znázorňujúce priebeh rotácie vektora



17.18.4. Výpočet vzoriek PWM pri SVM

K vytvoreniu rotujúceho poľa vo vnútri kružnice šesťuholníka je použitá moderná technika ktorú nazývame SVM. Orientácia ľubovoľného úseku poľa je jednoznačne daná uhlom **γ** a veľkosťou **U,** ktorá je daná dĺžkou vektora. Každá pozícia vo vnútri sektorového trojuholníka je dosiahnutá kombináciou vektora $V_{K,}$ V_{K+1} a nulového vektora T_0. Všetky priestorové vektory sú realizované pomocou časového multiplexu všetkých troch vektorov V_K, V_{K+1}, T_0 počas vzorkovacieho intervalu T_S. Každá nasledujúca PWM sekvencia skladajúca sa z V_K, V_{K+1}, T_0 je uložená po uplynutí vzorkovacieho intervalu T_S, do CAPCOM6 registrov mikroprocesora. Vyššie uvedené znamená, že výber, alebo výpočet hodnôt musí byť ukončený za pred začatím ďalšieho vzorkovacieho intervalu T_S.

Obrázok 262, Napäťový vektor pre SVM



$$\vec{U}_1 = \frac{1}{T_S} \times \left(T_{K+1} \times V_{K+1} + T_K \times V_K\right) \quad (98.)$$

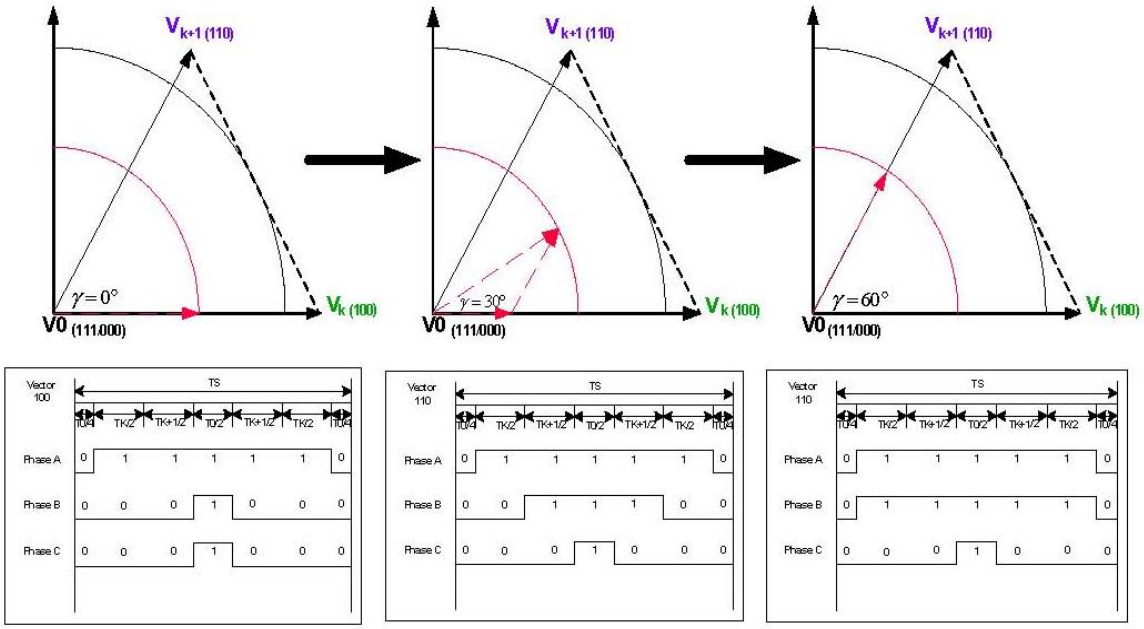
$$T_K = T_S \times \frac{\left|\vec{U}_1\right|}{U_{BAT}} \times \sqrt{3} \times \sin\left(\frac{\pi}{3} - \gamma\right) \quad (99.)$$

$$T_{K+1} = T_S \times \frac{\left|\vec{U}_1\right|}{U_{BAT}} \times \sqrt{3} \times \sin(\gamma) \quad (100.)$$

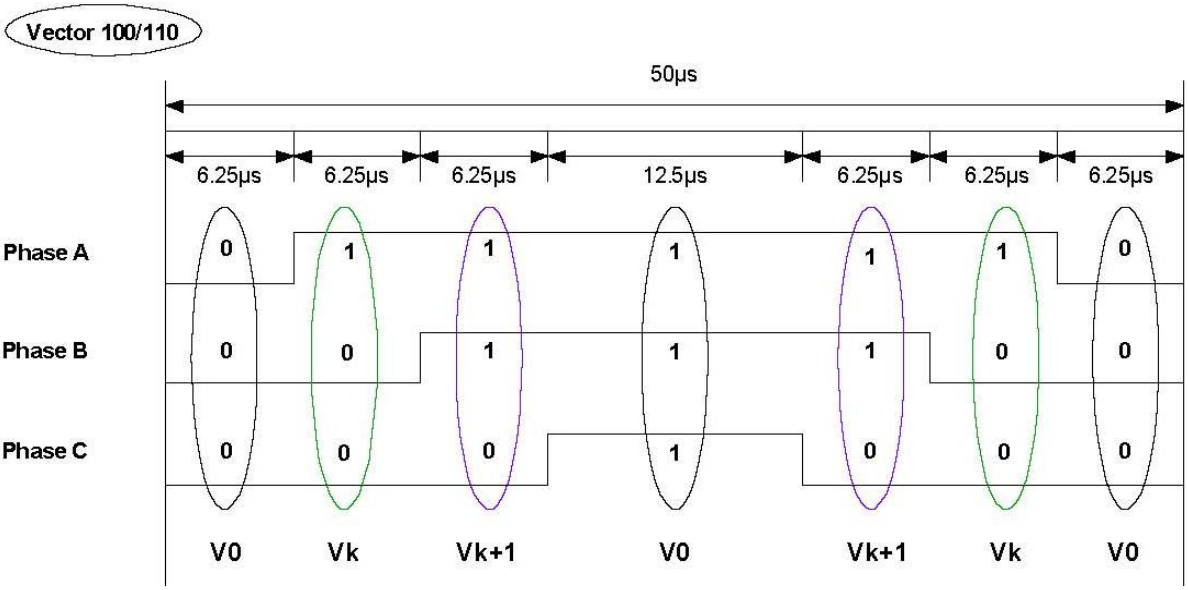
$$T_0 = T_S\left(1 - T_{K+1} - T_K\right) \quad (101.)$$

K získaniu takých vlastností SVM, akou je optimálny výkon a minimálna spínacia frekvencia prvkov výkonového meniča pre každé zariadenie, musí byť spínanie jednotlivých prvkov vykonávané tak, aby v ľubovoľnom okamihu bol len jeden v režime „invertor“. Tento stav nastane ak sekvencia spínania prvkov začína z nulového stavu [000] a invertorické spínacie prvky sú nastavené pokiaľ nie je dosiahnutý stav [111]. Obrázok 262 zobrazuje kompletný cyklus a jeho ukončenie, keď sa sekvencia vráti späť do nulového stavu.

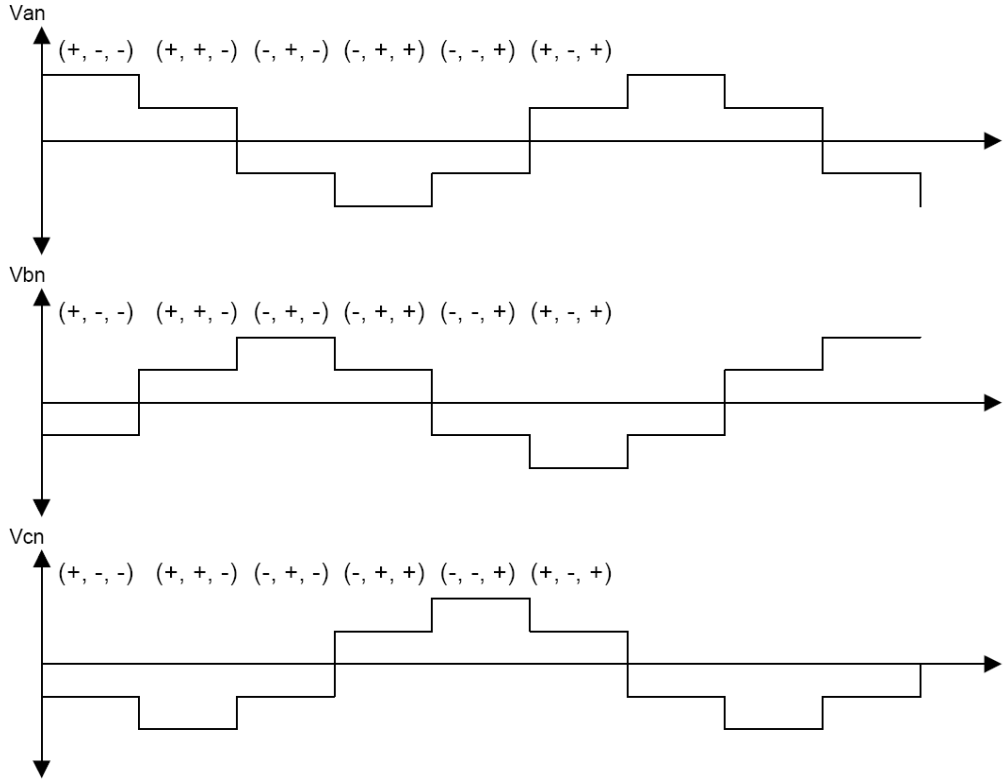
Obrázok 263, Priebehy napätí na spínacích prvkoch pri γ=0°,30°,60°.



Obrázok 264, Priebehy napätí prvkoch pri γ=30° a vector length=50%, f_{PWM}=20kHz



Obrázok 265, Priebehy jednotlivých fázových napätí pri SVM modulácii.



Súčet všetkých fázových napätí V_{AN}, V_{BN}, V_{CN} v ľubovoľnom časovom okamihu je rovný nule a je možné ho jednoducho transformovať pomocou nasledujúceho vzťahu na vektor $\underline{u}_S$:

$$\underline{u}_S = V_{AN}(t)\cdot e^{j0} + V_{BN}(t)\cdot e^{j\frac{2}{3}\pi} + V_{CN}(t)\cdot e^{-j\frac{2}{3}\pi}$$

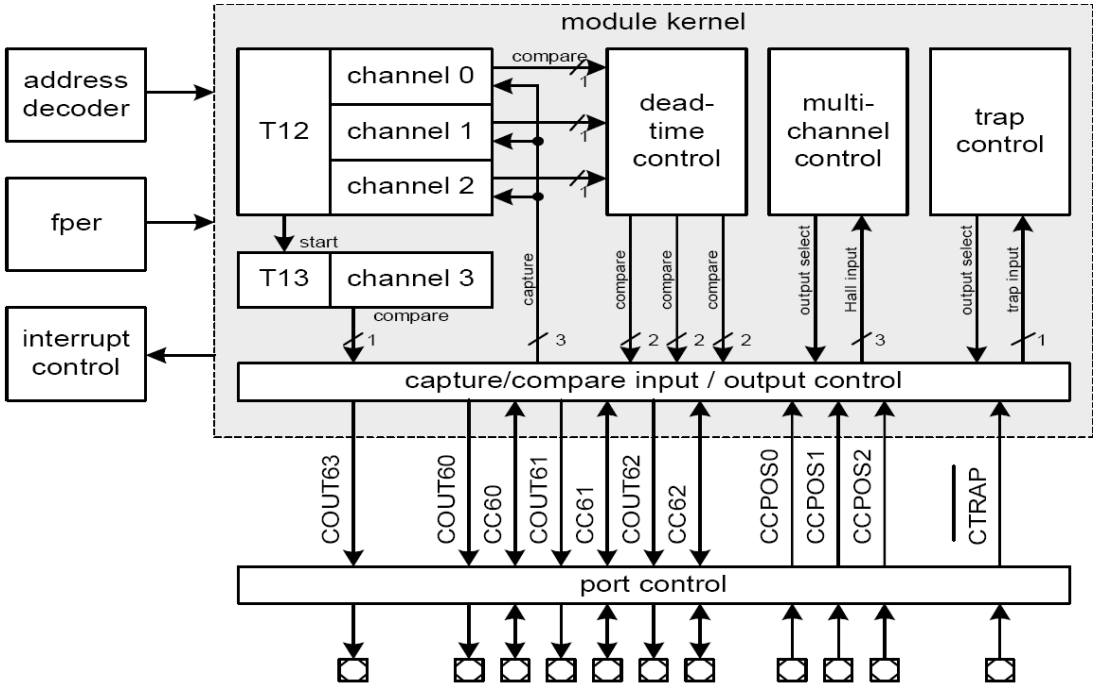
(102.)

Tabuľka 26, Vzájomná závislosť medzi uhlom, T_K, T_{K+1}, T_0, α, β.

Degree(°)	TS(µs)	TK(µs)	TK+1(µs)	T0(µs)	PWM(%)	Alpha	Beta
0	50	43.30	0	6.69	100	0.866	0
10	50	38.30	8.68	3.02	100	0.766	0.174
20	50	32.14	17.10	0.76	100	0.643	0.342
30	50	25.00	25.00	0	100	0.5	0.5
40	50	17.10	32.14	0.76	100	0.342	0.643
50	50	8.68	38.30	3.02	100	0.174	0.766
60	50	0	43.30	6.69	100	0	0.866

Degree(°)	TS(µs)	TK(µs)	TK+1(µs)	T0(µs)	PWM(%)	Alpha	Beta
0	50	21.65	0	28.35	50	0.866	0
10	50	19.15	4.34	26.5	50	0.766	0.174
20	50	16.07	8.55	25.38	50	0.643	0.342
30	50	12.5	12.5	25.0	50	0.5	0.5
40	50	8.55	16.07	25.38	50	0.342	0.643
50	50	4.34	19.15	26.5	50	0.174	0.766
60	50	0	21.65	28.35	50	0	0.866

Obrázok 266, Detailné zobrazenie CAPCOM6 jednotky mikroprocesora C164 a C508.



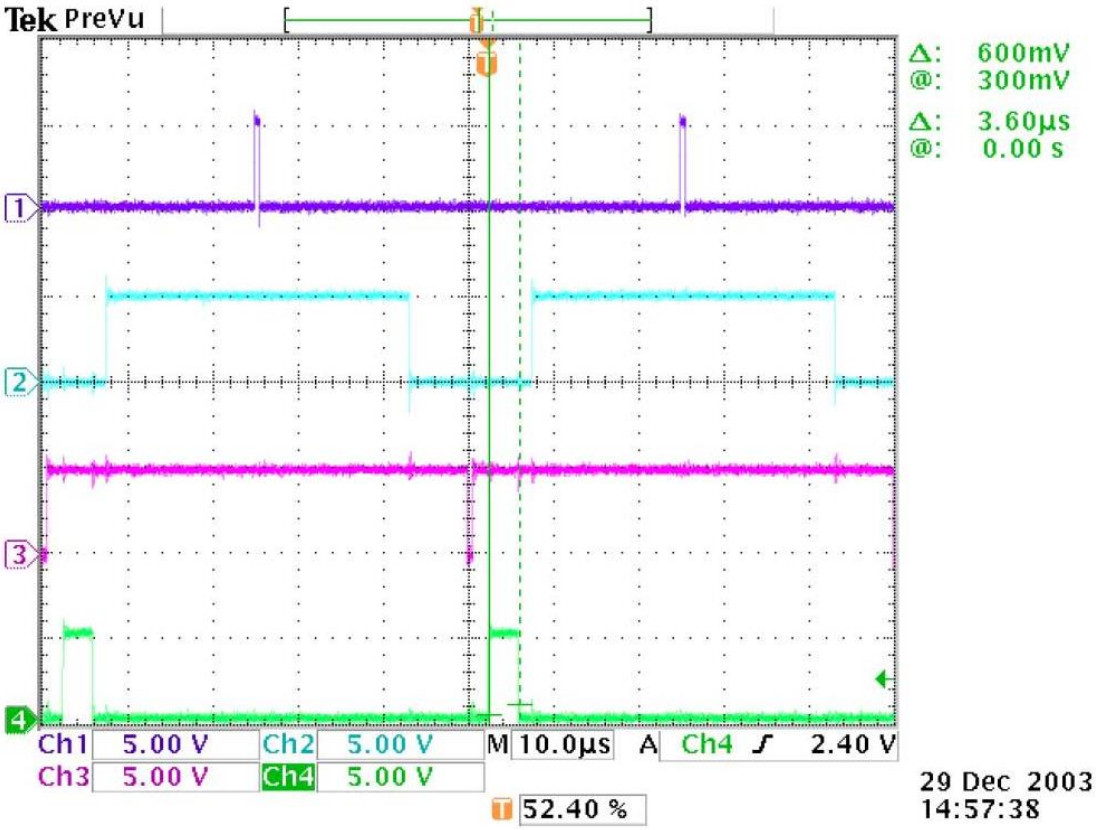
Program a príslušnú dokumentáciu[224] je možné nájsť na stránkach www.infineon.com.

[224] http://www.infineon.com/dgdl/Ap0802211_3-Phase-Currents.pdf?folderId=db3a304412b407950112b41a69152a86&fileId=db3a304412b407950112b41a69672a87

17.18.5. Analýza vyťaženia mikroprocesora C508 pri modulácii SVM

Obrázok 267 ukazuje koľko času potrebuje mikroprocesor C508 pre výpočet SVM. Je to približne 3,6µs pri frekvencii CPU 40MHz na uloženie nových obsahov CAP/COM registrov jednotky CAPCOM6 mikroprocesora.

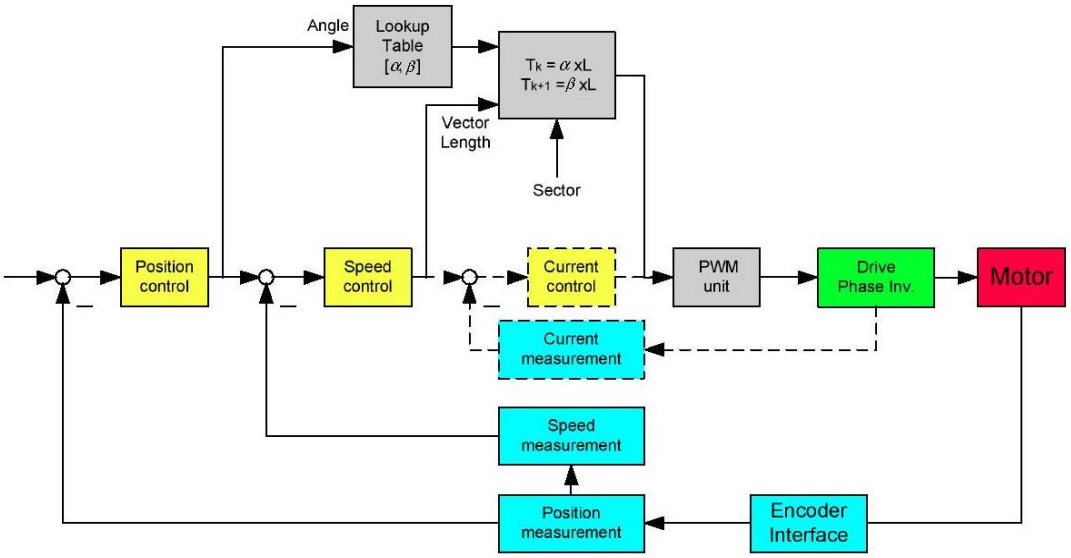
Obrázok 267, Vyťaženie mikroprocesora (CPU Load) pri SVM



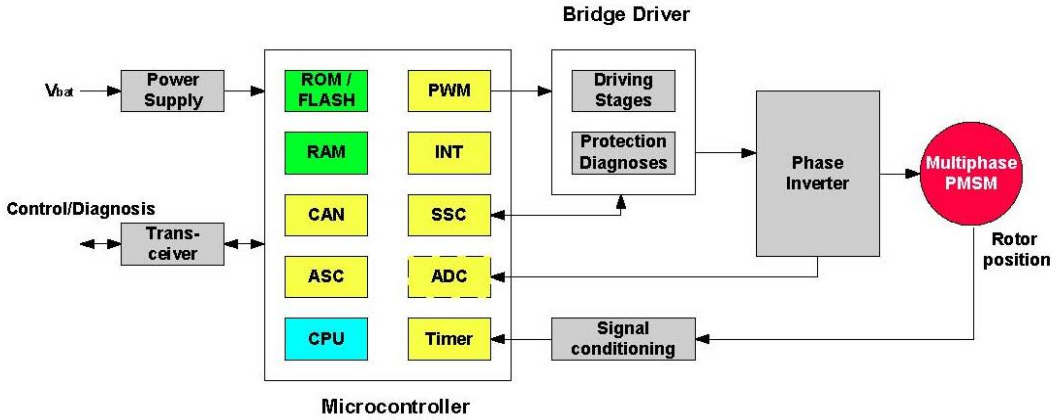
Poznámka: bližšie informácie[225] je možné získať na www.infineon.com.

225 http://www.infineon.com/dgdl/ap0803620_Space_Vector_Modulation.pdf?folderId=db3a304412b407950112b40c497b0af6&fileId=db3a304412b407950112b4199b7b28b3&sId=db3a30433b47825b013b5c7695e449ee

Obrázok 268, Kaskádne riadenie viacfázového pohonu s SVM



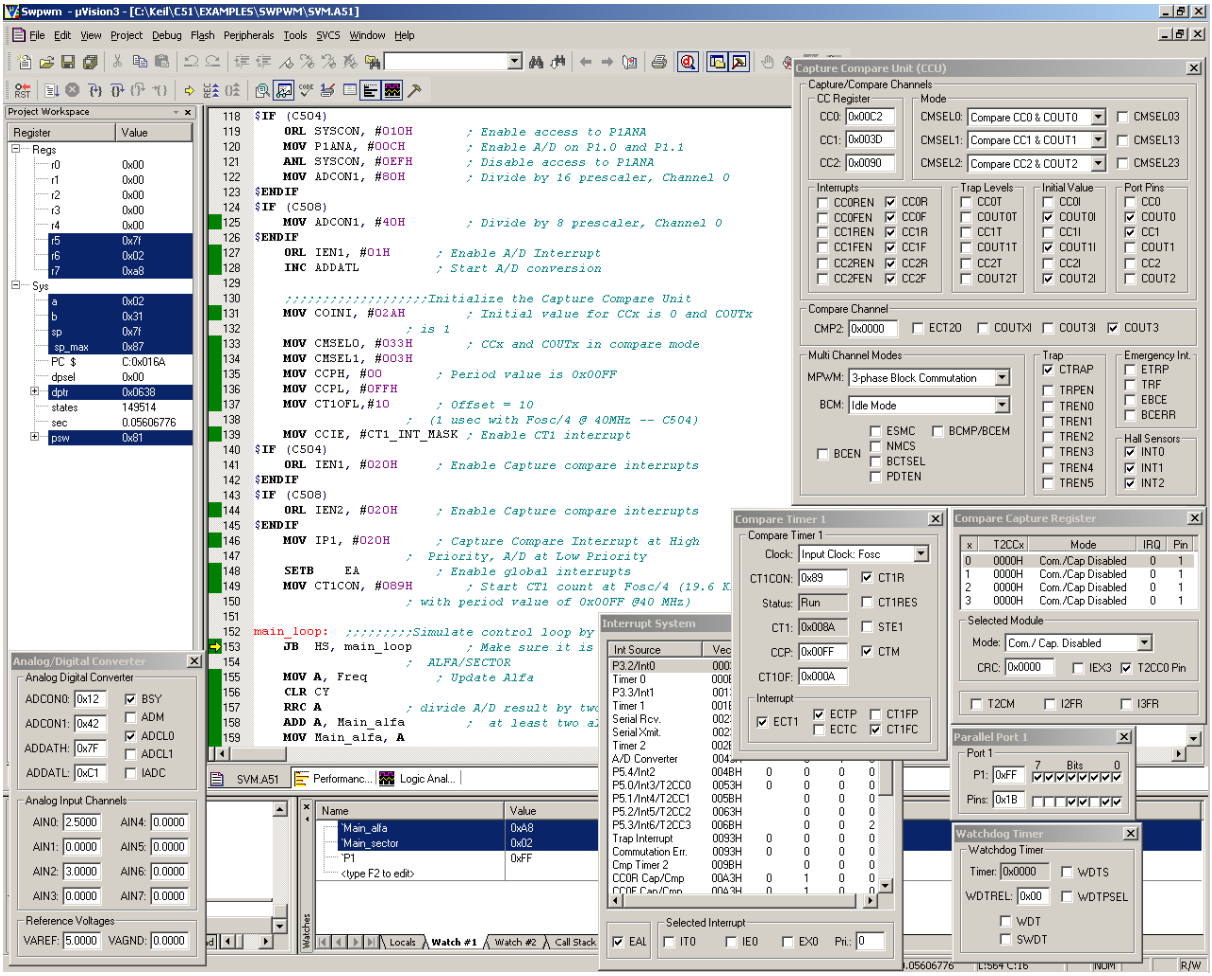
Obrázok 269, Principiálne riadenie viacfázového pohonu PMSM s SVM



Poznámka: bližšie informácie[226] je možné získať na www.infineon.com.

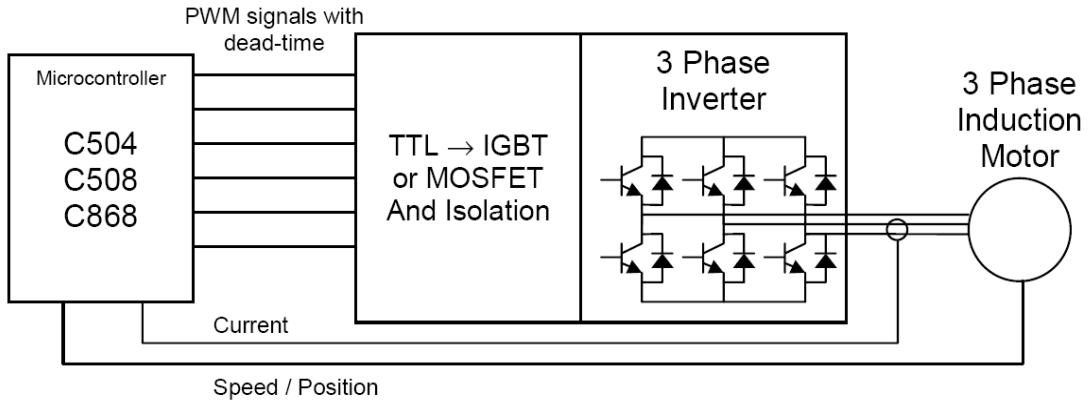
226 http://www.infineon.com/dgdl/p1605710_Space_Vector.pdf?folderId=db3a304412b407950112b409d4b00386&fileId=db3a304412b407950112b40a12e7043b&sId=db3a30443b360d0e013b594aa5983452

Obrázok 270, Ladenie a simulácia programu v prostredí µVision3

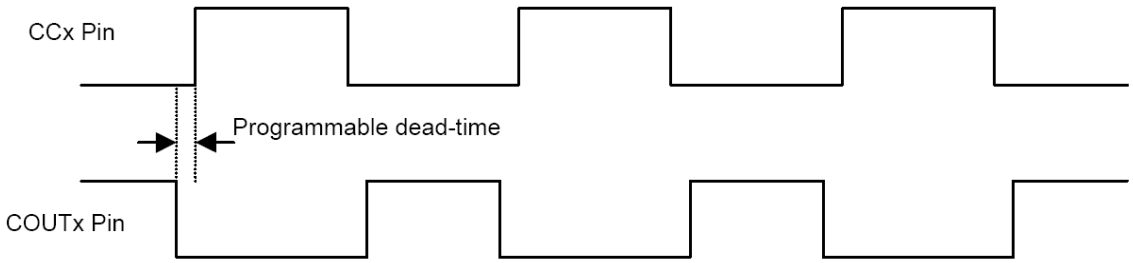


Príklad: S pomocou programovacieho jazyka A51, C51, vytvorte program, ktorý bude riadiť frekvenčný menič s IGBT ktorý napája jednosmerný motor s cudzím budením, jednofázový komutátorový motor a trojfázový asynchrónny motor (**ASM**) s klietkovou kotvou. Pracovná frekvencia je 20kHz. Daný menič musí byť schopný napájať jednosmerným a striedavým napätím s premenlivou veľkosťou napätia a frekvencie daný pohon. Napájacie napätie pre motor bude v rozsahu 0V po $400V_{AC}$ s frekvenciou 0 až 150Hz. Výstupy portu mikroprocesora C508 budú s pomocou 20kHz PWM ovládať trojfázové mostové zapojenie s IGBT, ktorý plní funkciu napäťového striedača. Striedač bude ovládaný pomocou dvojice tlačidiel. Dve tlačidlá budú slúžiť na voľbu otáčok motora a zároveň na zmenu smeru otáčania. Zároveň program doplňte o ovládanie pomocou sériového rozhrania RS232 s nasledujúcimi parametrami: $2400b.s^{-1}$, 8 dátových bitov, 1 štart a stop bit, bez parity. Výstupné napätie bude modulované pomocou SVM (Space Vector Modulation).

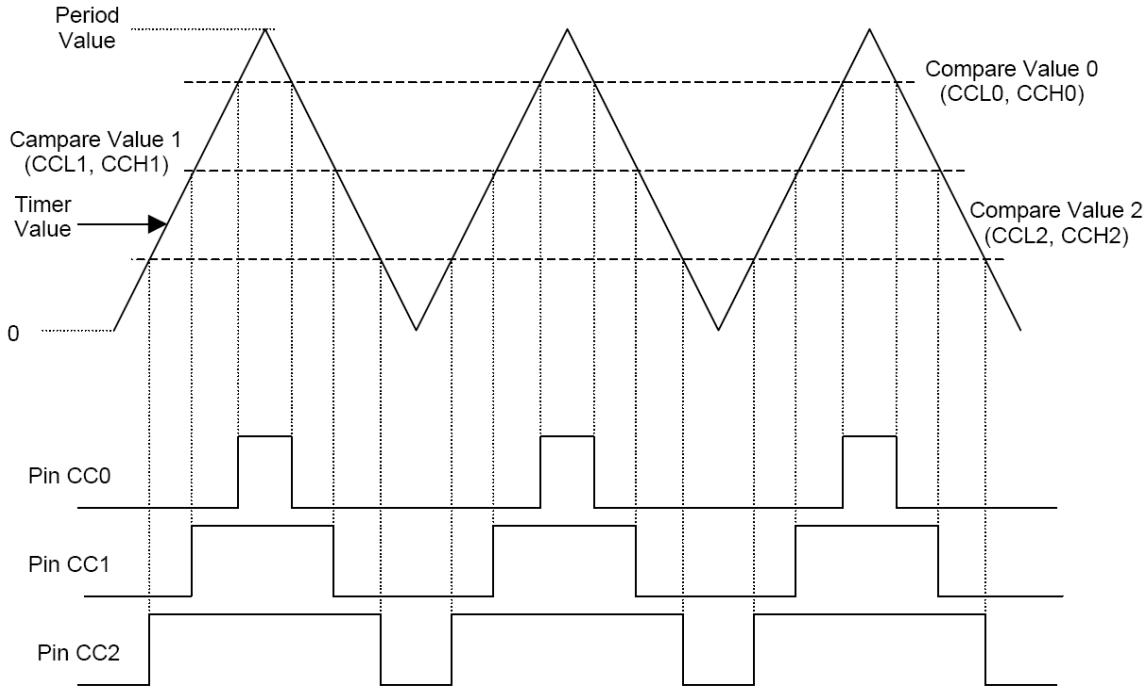
Obrázok 271, Schéma zapojenia pohonu s SVM (Space Vector Modulation)



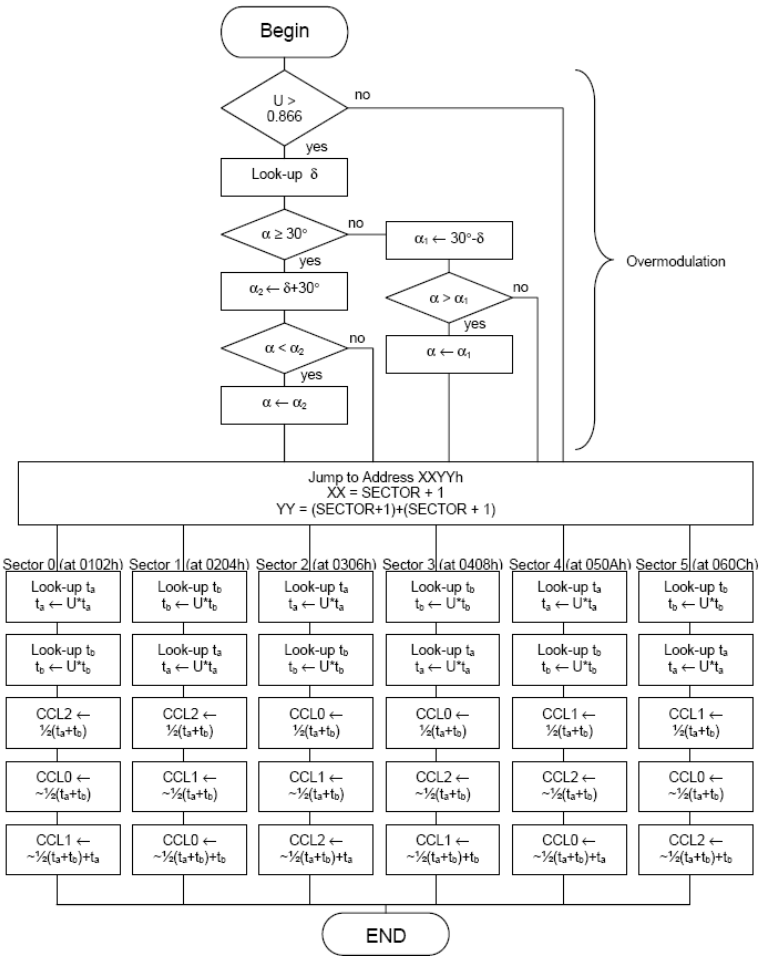
Obrázok 272, Výstup mikroprocesora CCx a COUTx s programovateľným dead-time



Obrázok 273, Generovanie napätia s CAPCOM6 v móde 1 pre SVM (Center PWM).



Obrázok 274, Vývojový diagram riadenia pohonu s SVM (Space Vector Modulation)



Obvyklý spôsob modulácie napätia SWPWM pre pohony spočíva vo vytvorení sústavy troch sínusových napätí s rovnakou amplitúdou a vzájomným fázovým posunom 120° elektrických. Ak generujeme napätie pomocou SVM, tak periodicky zyšujeme uhol vektora pomocou programu v mikroprocesore, čo po dosadení do vzťahov uvedených nižšie rotáciu vektora napätia okolo vlastnej osi. Ak aproximujeme $\underline{u}_S$ použitím $\underline{u}_1$ a $\underline{u}_2$ vytvoreného pomocou t_a a t_b vytvoríme tak vektor napätia, ktorý môžeme umiestniť kdekoľvek v priestore. Obrázok 278 to názornejšie vysvetľuje.

Pri transformácii uvažujeme napätie priestorového vektora vyjadreného ako

$$\underline{u}_S = t_a \cdot \underline{u}_1 + t_b \cdot \underline{u}_2$$

(103.)

$$t_a = \frac{2 \cdot U}{\sqrt{3}} \cdot \sin\alpha$$

(104.)

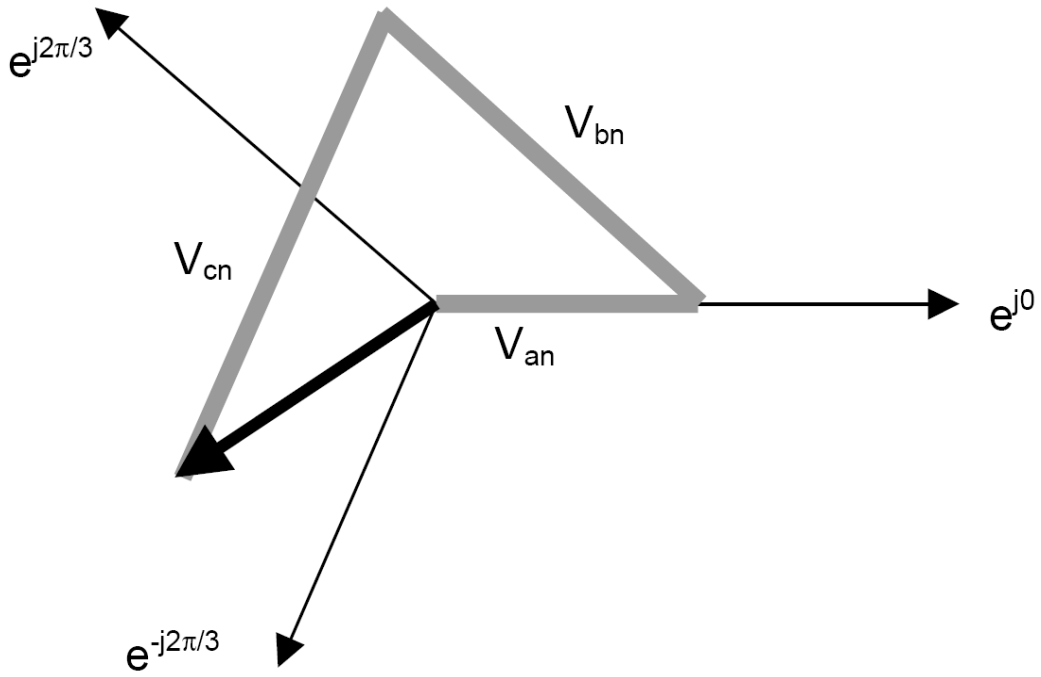
$$t_b = U \cdot \left[\cos(\alpha) - \frac{\sin(\alpha)}{\sqrt{3}} \right]$$

(105.)

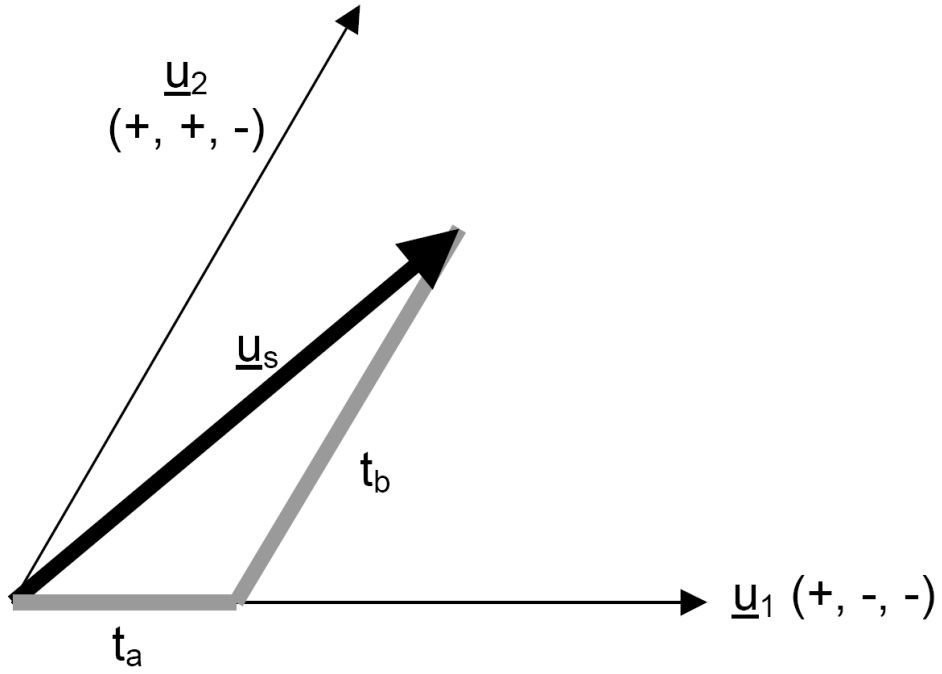
kde $U = |\underline{u}_S|$ je tzv. Modulačný index, $\alpha = \angle \underline{u}_S$ je uhol vektora napätia

(106.)

Obrázok 275, Transformácia napätí do priestorového vektora SSV (Single Space Vector)


e^j2π/3
V_bn
V_cn
e^j0
V_an
e^-j2π/3

Obrázok 276, Grafické znázornenie aproximácie vektora $\underline{u}_S$ pomocou $\underline{u}_1$ a $\underline{u}_2$.



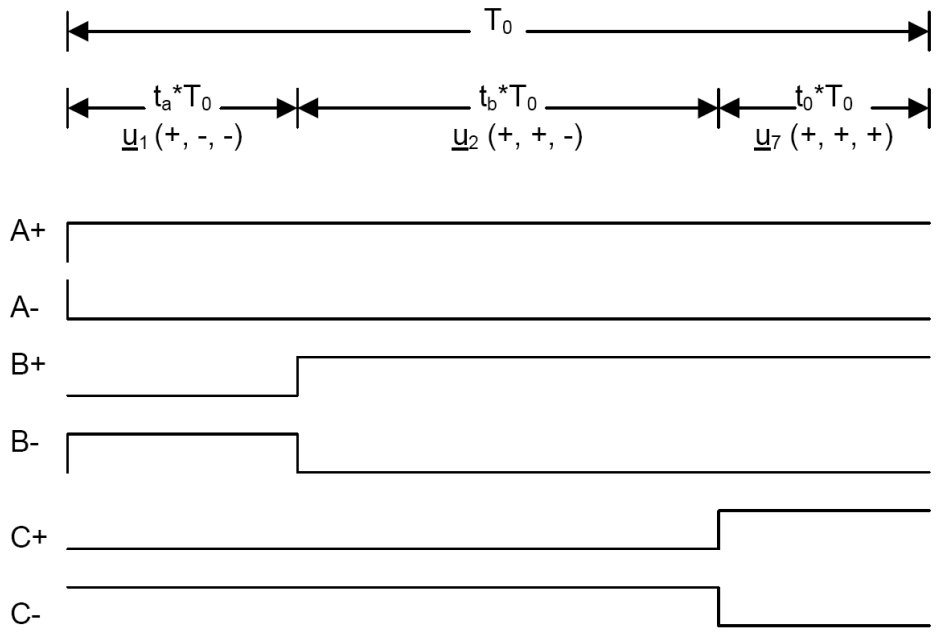
Ak modulačný index je malý, tak $(t_a + t_b) < 1$ a následne vzťah $t_a \cdot \underline{u}_1 + t_b \cdot \underline{u}_2$ je menší ako T_0.

Potom môžeme vyjadriť t_0 ako

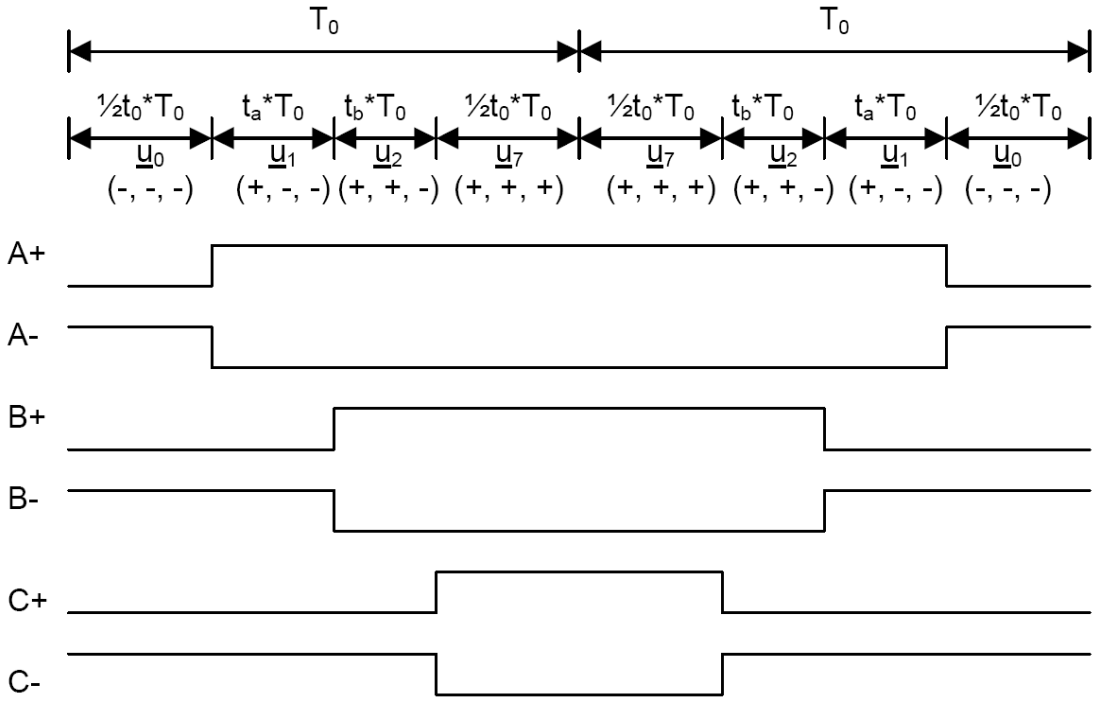
$$t_0 = T_0 \cdot (1 - t_a - t_b)$$

(107.)

Obrázok 277, Generovanie napätia pomocou metódy aproximácie vektora $\underline{u}_S$.



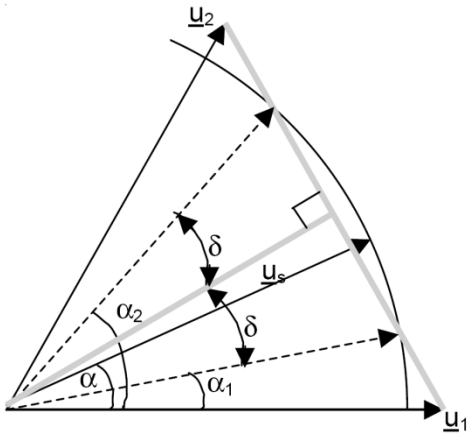
Obrázok 278, Generovanie napätia s modif. SSVM (Symetric Space Vector Modulation).



17.18.6. Zvyšovanie napätia SVM

Overmodulation je moderná technika, ktorá umožňuje SVM zvýšiť vyššiu amplitúdu napätia oproti SWPWM. Modulačný index U je obmedzený na hodnotu ~ 0.866. Táto technika je používaná na rozšírenie rozsahu modulačného indexu. Overmodulation nastane, ak veľkosť vektora (šípka vektora) $\underline{u}_S$ sa nachádza mimo kružnice vpísanej do šesťuholníka. Obrázok 279 predstavuje jednoduchú metódu založenú na uhle α referenčného vektora $\underline{u}_S$. Ak uhol $\alpha < \alpha_1$ tak SVM je vykonávané s obvyklým U. Ak sa vektor $\underline{u}_S$ nachádza vo vnútri šesťuholníka t_0 je väčšie ako nula. Ak ale α sa nachádza medzi α_1 a $\frac{\pi}{6}$, hodnota t_0 môže byť menšie ako nula. K správnej činnosti je potrebné použiť pri SVM nový uhol α_1 a U. Ak ale α sa nachádza medzi $\frac{\pi}{6}$ a α_2 je potrebné použiť nový uhol α_2 a U, pretože t_0 je menšie ako nula. Použitie tohto spôsobu riadenia SVM sa zvýši veľkosť množstva harmonických zložiek a skreslenie výstupného napätia na výstupných svorkách výkonového meniča. Výstupné napätie sa týmto zvýši o hodnotu približne 12% oproti SVM bez overmodulation.

Obrázok 279, Grafické znázornenie priebehu vektora s modulačným indexom $U > 0.866$



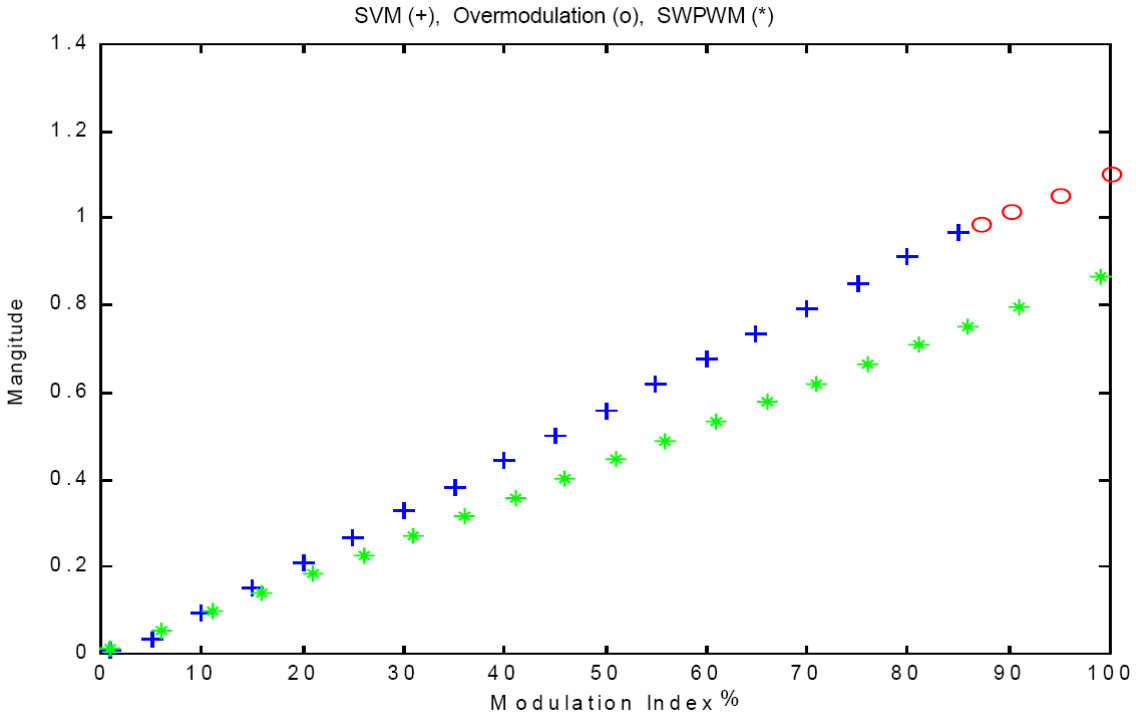
$$\alpha_1 = \frac{\pi}{6} - \delta \quad (108.)$$

$$\alpha_1 = \frac{\pi}{6} + \delta \quad (109.)$$

$$\delta = \arccos\left(\frac{\sqrt{3}}{2 \cdot U}\right) \quad (110.)$$

Angle of $\underline{u}_s$ (α)	Modulation Index	Angle
$\alpha_1 < \alpha < \alpha_2$	U	α
$\alpha_1 < \alpha < \pi/6$	U	α_1
$\pi/6 < \alpha < \alpha_2$	U	α_2

Obrázok 280, Veľkosti napätí pri použití metód SWPWM, SVM a Overmodulation



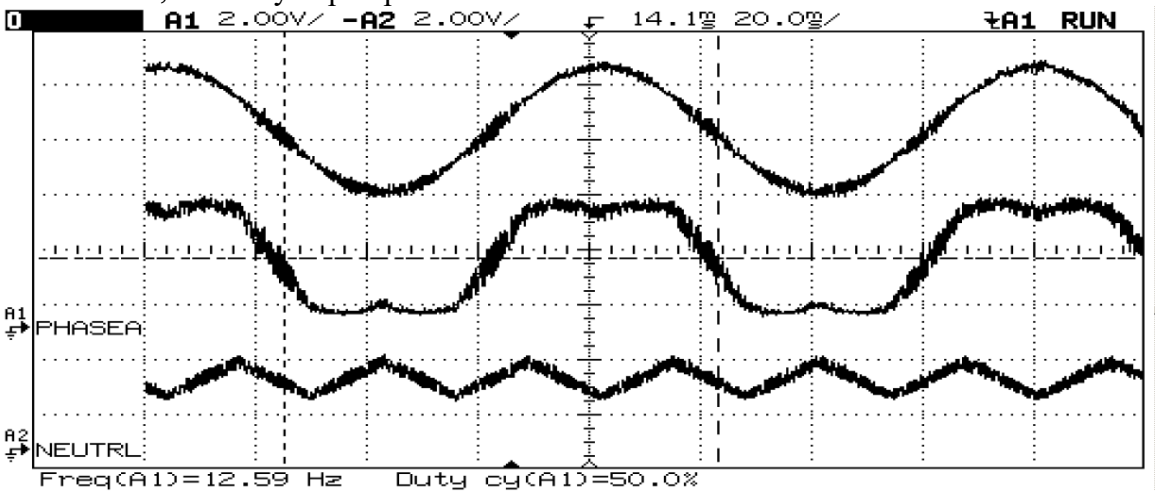
Tabuľka 27, Zoznam funkcií a použitých premenných pre SVM

Variable	Digital Representation	Engineering Unit Resolution	Comments
U ($\lvert\underline{u}_s\rvert$)	8-bit value (0x00 – 0xFF)	1/255 (~0.00392)	Input Variable
α ($\angle\underline{u}_s$)	11-bit value (0x0000 – 0x05FF)	π/765 radians (~0.235 degrees)	Bits 0-7 contain α, Bits 8 – 10 contain sector (input variable)
T_0	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	~19.6 kHz Carrier Frequency (fixed)
t_b	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	Stored in a table Table contains t_b for U = 1.0 (0xFF)
t_a	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	Stored in a table t_b table is used in reverse
t_0	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	Calculated $t_0 = \sim(U*t_a+U*t_b)/256$
δ	7-bit value (0x00 – 0x7F)	π/765 radians (~0.235 degrees)	Stored in a table
α_1 and α_2	8-bit values (0x00 – 0xFF)	π/765 radians (~0.235 degrees)	Calculated as needed from δ

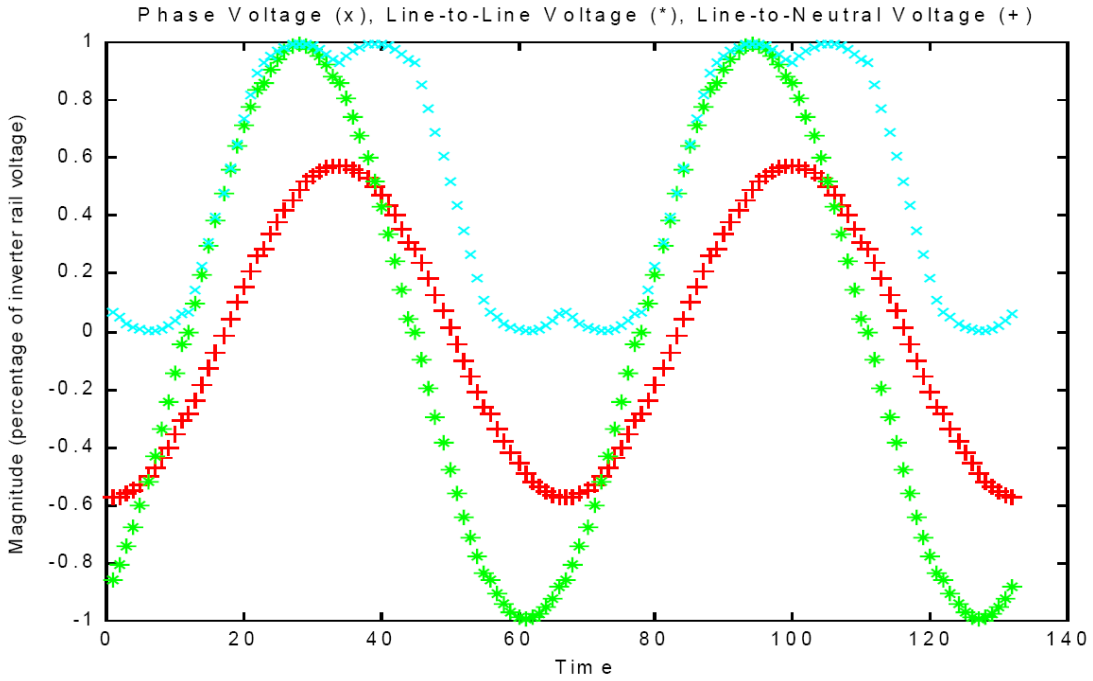
Tabuľka 28, Zoznam funkcií a premenných pre jednotlivé kvadranty SVM

Sector	Phase A Compare Value	Phase B Compare Value	Phase C Compare Value
0	½ t_0	½ t_0 + t_a	T_0 – ½ t_0
1	½ t_0 + t_b	½ t_0	T_0 – ½ t_0
2	T_0 – ½ t_0	½ t_0	½ t_0 + t_a
3	T_0 – ½ t_0	½ t_0 + t_b	½ t_0
4	½ t_0 + t_a	T_0 – ½ t_0	½ t_0
5	½ t_0	T_0 – ½ t_0	½ t_0 + t_a

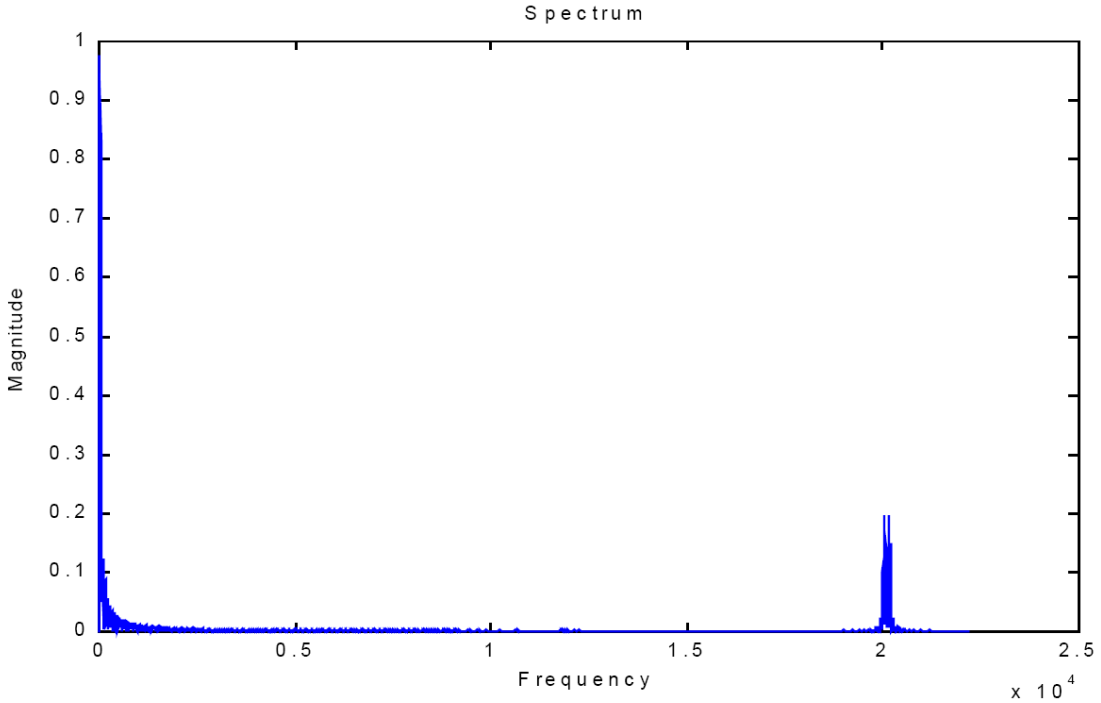
Obrázok 281, Priebehy napätí pri modul. indexe $U < 0.866$ voči neutrálnemu bodu



Obrázok 282, Priebeh napätia modulovaného pomocou SVM (Space Vector Modulation)



Obrázok 283, Obsah vyšších harmonických zložiek s SVM pri modul. indexe $U \sim 0.866$



Poznámka: bližšie informácie[227] je možné získať na www.infineon.com.

227 http://www.infineon.com/dgdl/ap0803620_Space_Vector_Modulation.pdf?folderId=db3a304412b407950112b40c497b0af6&fileId=db3a304412b407950112b4199b7b28b3&sId=db3a30433b47825b013b5c7695e449ee

17.19. Zdieľanie sériového kanála viacerými úlohami

Príklad: Napíšte program, ktorý bude zdieľať sériové rozhranie RS232 súčasne viacerými úlohami a v periodických intervaloch bude vysielať údaje o momentálne spustenej úlohe na sériovú linku.

Príklad programu:

C:\Keil\C51\Examples\termel\Serials\PfSema\main.c

```
#include <rtx51.h>
#include <reg51.h>
#include <stdio.h>

// Task Definitions
#define STARTUP_TASK     3
#define TASK1            1
#define TASK2            2

// Semaphore Definitions
#define SEM_PRINTF       8

#define USE_SEMAPHORE    1

/***
make sure to link with...
ol(* ! ?PR?PRINTF?PRINTF)
***/

void setup_serial_io (void)
{
 SCON  = 0x50;                    /* SCON: mode 1, 8-bit UART, enable rcvr */
 TMOD |= 0x20;                    /* TMOD: timer 1, mode 2, 8-bit reload */
 TH1   = 0xF0;                    /* TH1:  reload value */
 TR1   = 1;                       /* TR1:  timer 1 run */
 TI    = 1;                       /* TI:   set TI to send first char of UART */
}

void task_1 (void) _task_ TASK1 _priority_ 0
{
while (1)
  {
  os_wait (K_IVL, 150, 0);                                    // Delay for 250 ticks

#if USE_SEMAPHORE
  os_wait (K_MBX + SEM_PRINTF, 255, 0);                       // Wait for printf
  printf ("Task # %d\n", (int) os_running_task_id ());
  os_send_token (SEM_PRINTF);                                 // Done with printf
#else
  printf ("Task # %d\n", (int) os_running_task_id ());
#endif
  }
}

```

C:\Keil\C51\Examples\termel\Serials\PfSema\main.c

```
45  void task_2 (void) _task_ TASK2 _priority_ 1
46  {
47  while (1)
48    {
49    os_wait (K_IVL, 100, 0);                                    // Delay for 200 ticks
50
51  #if USE_SEMAPHORE
52    os_wait (K_MBX + SEM_PRINTF, 255, 0);                       // Wait for printf
53    printf ("Task # %d\n", (int) os_running_task_id ());
54    os_send_token (SEM_PRINTF);                                 // Done with printf
55  #else
56    printf ("Task # %d\n", (int) os_running_task_id ());
57  #endif
58    }
59  }
60
61  void startup_task (void) _task_ STARTUP_TASK _priority_ 2
62  {
63  os_set_slice (1000);
64
65  os_create_task (TASK1);
66  os_create_task (TASK2);
67
68  #if USE_SEMAPHORE
69    os_send_token (SEM_PRINTF);                                 // Setup printf semaphore
70  #endif
71
72  os_delete_task (os_running_task_id ());
73
74  while (1)
75    {
76    /*** This should never happen ***/
77    }
78  }
79
80  void main (void)
81  {
82  setup_serial_io ();
83  printf ("Getting ready to start\n\n");
84
85  os_start_system (STARTUP_TASK);
86
87  while (1)
88    {
89    /*** This should never happen ***/
90    }
91  }
```

C:\Omega\Data\Dropbox\Zaloha\C51\RTX51 Token\RTX51 Token.c

```
#include <reg552.h>
#include <Rtx51.h>
#include <stdio.h>
#include <stdlib.h>

#define  RTX_DONE       0
#define  WAIT_FOREVER   0xFF
#define  NO_MESSAGE     0

#define  TASK_ONE         1  // Task number for the first task acting on the
                             // shared variable.
#define  TASK_TWO         2  // Task number for the second task acting on the
                             // shared variable.

#define  TASK_PRIORITY  0

#define  MUTEX_SEMAPHORE  10

int SharedVariable=0;

void One(void) _task_ TASK_ONE _priority_ TASK_PRIORITY
{
   signed char RtxCompletion;
   TI=1;
   RtxCompletion=os_send_token(MUTEX_SEMAPHORE);
   RtxCompletion=os_create_task(TASK_TWO);
   for(;;)
   {
      switch(os_wait(K_MBX+MUTEX_SEMAPHORE,WAIT_FOREVER,NO_MESSAGE))
       {
        case  SEM_EVENT : SharedVariable*=3;
                          SharedVariable/=3;
                          SharedVariable++;
                          printf("Task%2bd\r",os_running_task_id());
                          RtxCompletion=os_wait(K_TMO,100,NO_MESSAGE);
                          RtxCompletion=os_send_token(MUTEX_SEMAPHORE);
                          break;
        default         : break;
       }
   }
} // One

void Two(void) _task_ TASK_TWO _priority_ TASK_PRIORITY
{
   signed char RtxCompletion;
   for(;;)
   {
      switch(os_wait(K_MBX+MUTEX_SEMAPHORE,WAIT_FOREVER,NO_MESSAGE))
       {
        case  SEM_EVENT : SharedVariable*=3;
                          SharedVariable/=3;
                          SharedVariable++;
                          printf("Task%2bd\r",os_running_task_id());
                          RtxCompletion=os_wait(K_TMO,100,NO_MESSAGE);
                          RtxCompletion=os_send_token(MUTEX_SEMAPHORE);
                          break;
        default         : break;
       }
   }
} // Two

void main(void)
{
   signed char RtxCompletion;   // RTX completion code
   RtxCompletion=os_start_system(TASK_ONE);
   for(;;);
} // main
```

17.20. Meranie amplitúdy jednosmerného napätia

Príklad: Napíšte program s pomocou RTX51, alebo ARTX166, ktorý bude vykonávať meranie napätia pomocou integrovaného 10 bitového ADC analógového prevodníka a v periodických intervaloch bude vysielať tieto údaje na sériové rozhranie RS232.

Príklad programu pre obsluhu AD prevodníka v assembleri:

c:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\ADC_Converter.asm

```
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Repeat              macro    Num
                        mov  a,ADCH
                        anl  a,#Num
                        rl   a
                        rl   a
                    ENDM
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;---------------------------------------------------------------------
public              _ADC_CONVERSION         ;ADC_Conversion(unsigned char)
public              CONVERSION              ;Conversion() with ADCON+ADCON
;-------------------------------------------------------------------------------------
ADCON               data  0C5h
ADCH                data  0C6h
;-------------------------------------------------------------------------------------
ADCS                equ  00001000B          ;Spustim prevod AD prevodnika
ADCI                bit  acc.4              ;Kontrolujem ci je prevod ukonceny ADCON.4...
MASKA               equ  11000000B
;-------------------------------------------------------------------------------------
Prog_ADC_Converter  segment  code
                    rseg  Prog_ADC_Converter
;-------------------------------------------------------------------------------------
_ADC_CONVERSION :   mov  ADCON , r7         ;Vstupny parameter je cislo kanala pre meranie v R7
                    orl  ADCON , #ADCS
?PR?ADC_END :       mov  a,ADCON
                    jnb  ADCI , ?PR?ADC_END ;Je prevod ukonceny ???
                    clr  ADCI               ;bit ADCI nastavujem na 0
                    mov  ADCON , a
CONVERSION :        anl  a,#MASKA           ;LSB vysledku je v D7,D6 v ADCON
                    mov  r7 , a
                    Repeat  MASKA
                    mov  r6 , a             ;MSB vysledku je v R6.1 a R6.0
                    xch  a,r7               ;LSB vysledku je v R7.1 a R7.0
                    Repeat  ~MASKA          ;Vymaskujem D7 a D6 aby som mal D0 az D5 k spracovaniu
                    orl  a,r7               ;LSB R7.1 a R7.0 skombinujem s ADCH0..ADCH5
                    xch  a,r7               ;Vysledok LSB je v R7
                    ret
;-------------------------------------------------------------------------------------
;                   cseg at 0053h           ;os_attach_interrupt(10)
;                   push acc
;                   pop acc
;                   reti
;-------------------------------------------------------------------------------------
                    end
```

Príklad programu pre obsluhu AD prevodníka v jazyku C:

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
46   int ADC (unsigned char Channel )
47   {
48    #define ADCS  0x08
49    #define ADCI  0x10
50    #define ADEX  0x20
51    ADCON =Channel ;                          //Vyber kanalu pre prevod 0..7
52    ADCON |=ADCS ;                            //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
53    while ((ADCON &ADCI )== 0x00 );           //Cakam na ukoncenie prevodu cca. 80us
54    ADCON ^=ADCI ;                            //Nulujem priznak AD prevodnika
55    return (256 *ADCH +(ADCON &0xC0 )>> 6);
56   }
```

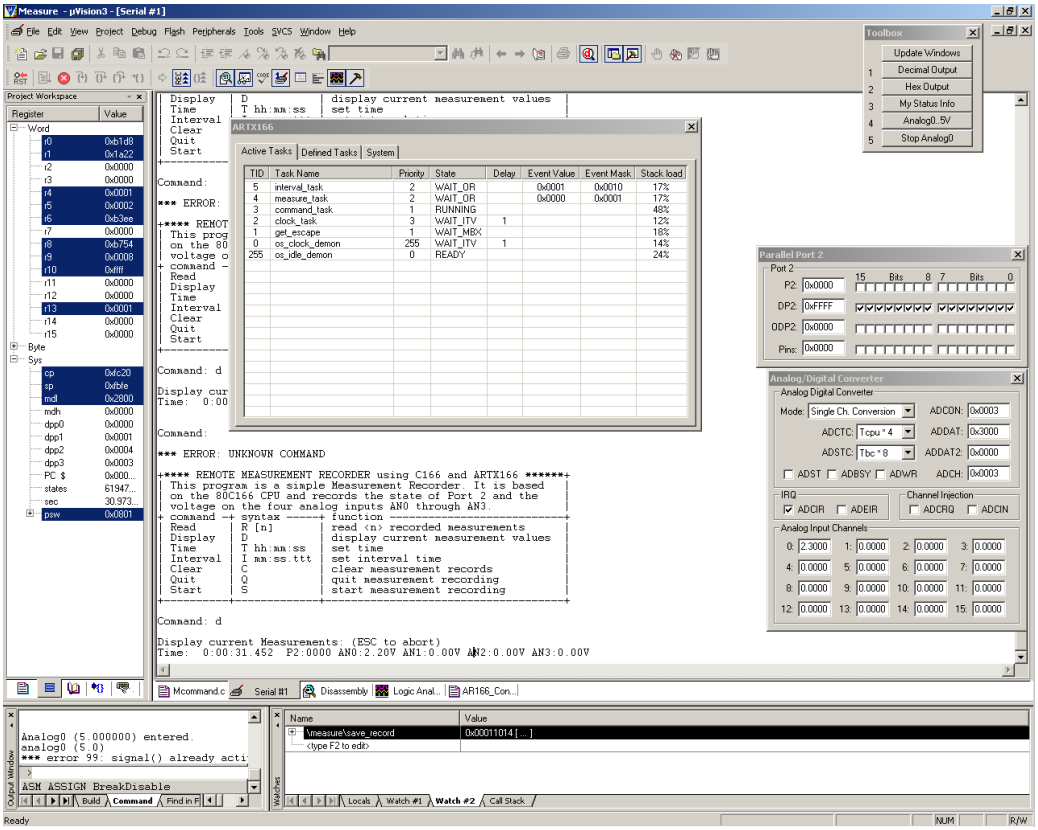
Poznámka: program je napísaný ako funkcia vracajúca **int**, kde parametrom je číslo kanálu pre prevod analógovej veličiny.

Príklad programu pre obsluhu AD prevodníka pomocou RTX51:

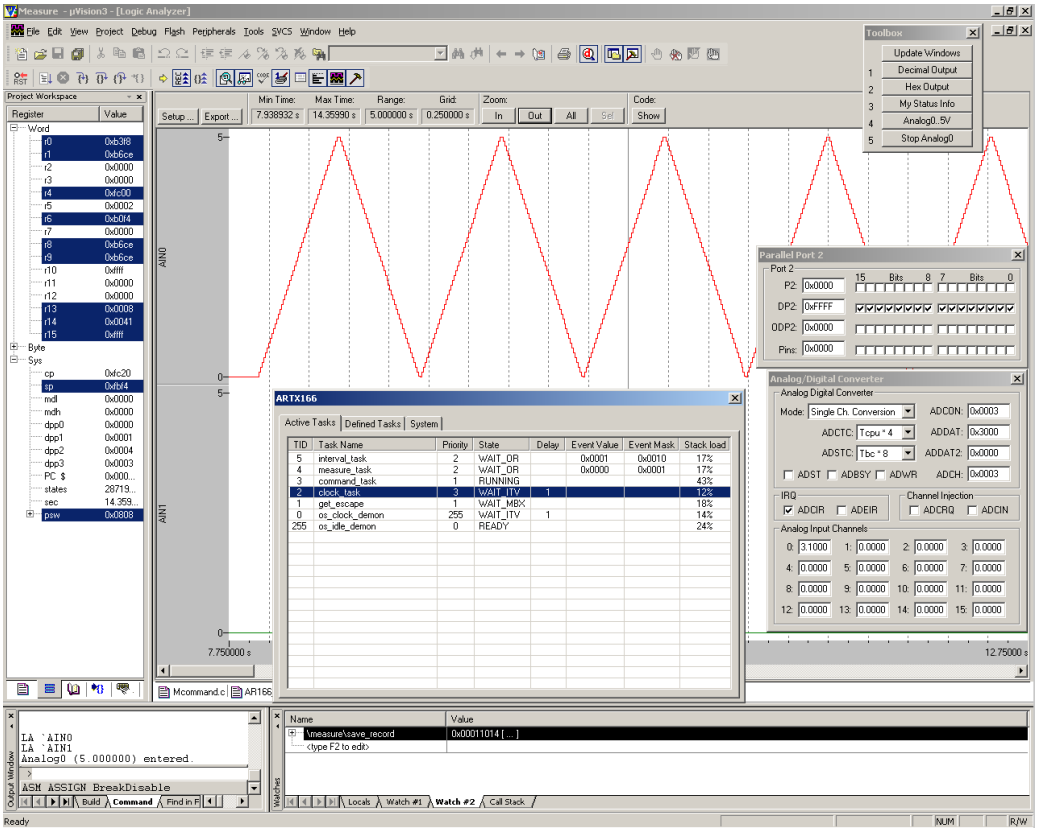
C:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\I2c_DSW.c

```
46   #define ADCI   0x10
47   #define ADCS   0x08
48
49   unsigned int Voltage ;
50
51   #ifdef ADCINT
52
53   #pragma REGISTERBANK (3)
54
55   void Adc (void ) _task_ ADC _priority_ 3
56   {
57    signed char RtxCompletion ;
58    os_attach_interrupt (10 );
59    while (1)
60     {
61      Watchdog ;
62      switch (RtxCompletion =os_wait (K_TMO +K_INT +K_SIG ,10 ,NULL ))
63       {
64        case SIG_EVENT  : //Voltage=ADC_Conversion(2); break;
65        case TMO_EVENT  : ADCON |=(ADCS +2);
66        case INT_EVENT  : ADCON ^=ADCI ; Voltage =Conversion (); break ;
67       }
68     }
69   }
70
71   #pragma REGISTERBANK (0)
72
73   #else
74
75   #pragma REGISTERBANK (0)
76
77   void Adc (void ) _task_ ADC _priority_ 0
78   {
79    signed char RtxCompletion ;
80    while (1)
81     {
82      Watchdog ;
83      Voltage =ADC_Conversion (2);
84      RtxCompletion =os_wait (K_TMO ,10 ,NULL );
85     }
86   }
87
88   #endif
```

Obrázok 284, Ukážka grafického výstupu programu na meranie jednosmerného napätia



Obrázok 285, Ukážka priebehu vstupného napätia pre AD provodník mikroprocesora



17.21. Harmonická analýza analógových veličín s C51 a RTX51

Rozbor príkladu: Harmonickou analýzou analógového priebehu je možné zistiť veľkosti amplitúd jednotlivých harmonických napätí obsiahnutých v neharmonickom napäťovom priebehu. Vytvorte program, ktorý bude v pravidelných periodických intervaloch ΔT=1,0 ms. vzorkovať napätie na vstupe analógového prevodníka P5.0 mikroprocesora 80C552, zároveň bude vypočítavať Fourier-ove[228] koeficienty[229] daného analógového priebehu a zobrazovať ich cez sériové rozhranie RS232 na LCD zobrazovači.

Pre neharmonický prúd záťaže môžeme napísať :

$$i_{ZAT}=\frac{a_o}{2}+\sum_{n=1}^{\infty}(a_n\cdot\cos(\omega_n\cdot t))+\sum_{n=1}^{\infty}(b_n\cdot\sin(\omega_n\cdot t))$$

(111.)

kde a_n a b_n sú koeficienty Fourier-ovho radu a a_0 je jednosmerná zložka analyzovaného priebehu. Amplitúda a fázový posun prvej harmonickej záťažového prúdu sú:

$$I_{1ZAT}=\sqrt{a_1^2+b_1^2};\ \varphi_1=\arctan\left(\frac{b_1}{a_1}\right)$$

(112.)

Amplitúda sieťového prúdu s nulovým fázovým posunom oproti napájaciemu napätiu je daná ako:

$$I_{1SIET}=I_{1ZAT}\cdot\cos\varphi_1$$

(113.)

Pre určenie prvej harmonickej záťažového prúdu a jeho fázového posunu využijeme Park-Clarke-ovú transformáciu. Potom:

$$i_{ZAT}(t)=i_\alpha+j\cdot i_\beta$$

(114.)

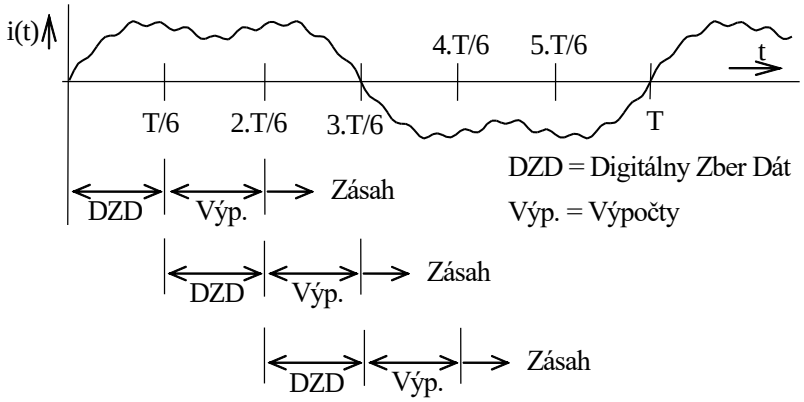
Dá sa dokázať, že pre **symetrickú trojfázovú záťaž** je možné súradniciach α,β vypočítať komplexné koeficienty Fourier-oveho radu vzoriek získaných v $\frac{1}{6}$ periódy záťažového prúdu.

[228] http://matematika.cuni.cz/dl/analyza/37-fou/lekce37-fou-pmax.pdf

[229] Zvýšenie rýchlosti výpočtu jednotlivých koeficientov je možné nahradením klasického mikroprocesora 80C552 moderným 32-bitovým DSP mikroprocesorom STM32F407VGTx ktorý obsahuje aj numerický procesor (Floating Point Hardware) pre urýchlenie výpočtov. V tomto prípade je potrebné urobiť relatívne malé zmeny v zápise programu, pretože sa jedná o iný typ mikroprocesora.

Tým sa výrazne zvýši dynamika celého aktívneho filtra, lebo ak výpočet koeficienta pre prvú harmonickú prúdu záťaže (***C₁***) stihne riadiaci systém za ďalšiu šestinu, tak v nasledujúcej šestine môže byť vykonaný zásah do sústavy. Nasledujúci obrázok ukazuje zreťazenie jednotlivých celkov riadiaceho algoritmu.

Obrázok 286, Sekvencie riadiaceho algoritmu aktívneho filtra



atď.

Pre výpočet komplexných koeficientov Fourier-oveho radu platí nasledujúci vzťah:

$$C_\upsilon = \frac{3}{\pi}\int_0^{\pi/3} RE\{f(\omega t)\}\cdot e^{-j\upsilon\omega t}\cdot d\omega t + j\frac{3}{\pi}\int_0^{\pi/3} IM\{f(\omega t)\}\cdot e^{-j\upsilon\omega t}\cdot d\omega t$$

(115.)

kde $RE = i_\alpha$ a $IM = i_\beta$ ďalej:

$$C_1 = \frac{3}{\pi}\int_0^{\pi/3}\left[\left(RE\cdot\cos(\omega t)+IM\cdot\sin(\omega t)\right)+j\cdot\left(IM\cdot\cos(\omega t)-RE\cdot\sin(\omega t)\right)\right]d\omega t \qquad (116.)$$

a po prechode na diskrétny tvar dostaneme:

$$\begin{aligned}C_1 = {} & \frac{1}{N}\left\{\sum_{i=0}^{N-1} RE(i)\cdot\cos\left(\frac{i\cdot\pi}{3\cdot N}\right)+\sum_{i=0}^{N-1} IM(i)\cdot\sin\left(\frac{i\cdot\pi}{3\cdot N}\right)+\right.\\ & +\frac{RE(N)\cdot\cos(\pi/3)-RE(0)\cdot\cos(0)}{2}+\\ & \left.+\frac{IM(N)\cdot\sin(\pi/3)-IM(0)\cdot\sin(0)}{2}\right\}+\\ & +j\frac{1}{N}\left\{\sum_{i=0}^{N-1} IM(i)\cdot\cos\left(\frac{i\cdot\pi}{3\cdot N}\right)-\sum_{i=0}^{N-1} RE(i)\cdot\sin\left(\frac{i\cdot\pi}{3\cdot N}\right)+\right.\\ & +\frac{IM(N)\cdot\cos(\pi/3)-IM(0)\cdot\cos(0)}{2}-\\ & \left.-\frac{RE(N)\cdot\sin(\pi/3)+RE(0)\cdot\sin(0)}{2}\right\}\end{aligned}$$

(117.)

Túto rovnicu je nutné vyriešiť pomocou FFT[230] v reálnom čase, pričom je jasné, že čím viac vzoriek v danom intervale nasnímame, tým dlhšie bude trvať výpočet[231]. Pre konkretizáciu ešte uvedieme, že výsledkom je komplexné číslo tvaru:

$$C_1 = a_1 + j \cdot b_1$$

(118.)

a dosadením do vzťahov[232] dostaneme hodnotu prvej harmonickej sieťového prúdu. Ďalej analogicky dostávame jednotlivé hodnoty referenčného prúdu pre aktívny filter.

[230] FFT (Fast Fourier Transform) je implementovaná v najrozšírenejších matematických programoch ak sú napríklad Matlab, MathCad, Mathematica a iné.

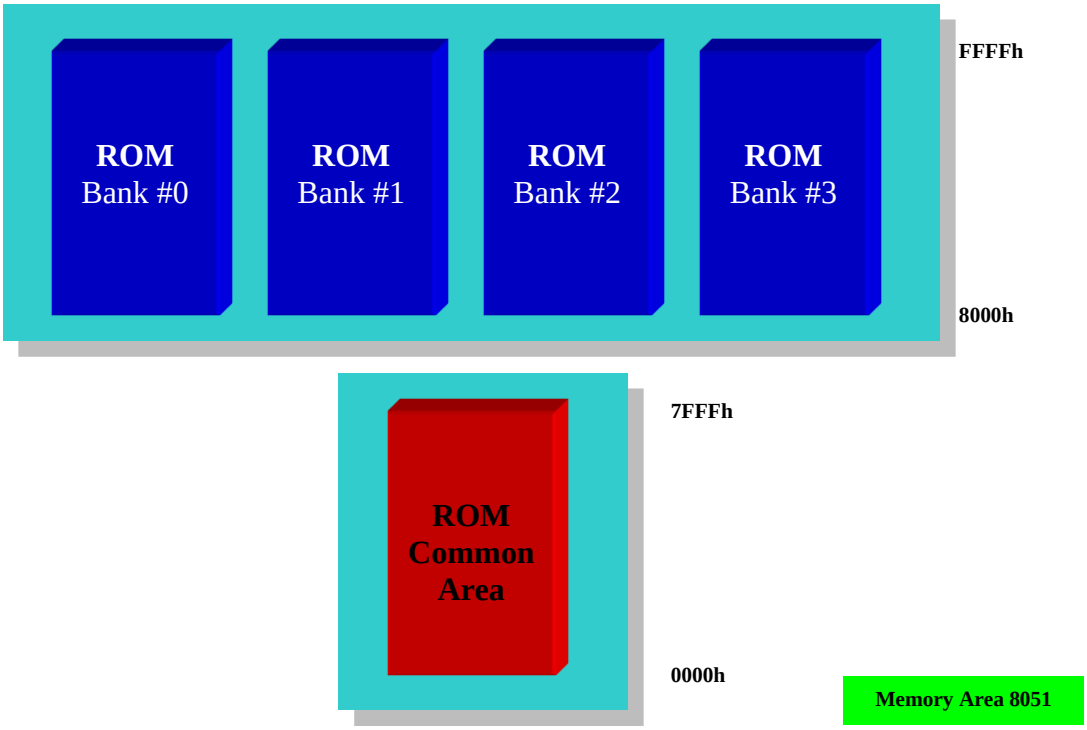
[231] Pre výpočty je vhodné použiť procesory s integrovanými numerickými procesormi, ktoré umožnia rapídne zvýšiť rýchlosť výpočtu zložitých matematických funkcií.

[232] Rýchlosť výpočtu matematických vzťahov v tomto prípade má absolútnu prioritu, preto je na uvážení programátora či použije pomerne neefektívnu architektúru mikroprocesora 8051 bez operačného systému RTX51 a nesiahne radšej po modernejšej 32 bitovej ARM architektúre s podporou operačného systému RLARM verzie minimálne 4.12 alebo vyššej.

17.22. Práca s externým pamäťovým priestorom xBANKING

Príklad: Napíšte program ktorý bude pracovať so sériovým kanálom a ktorého funkcie budú uložené v dvoch fyzicky oddelených externých pamätiach programu nazývaných BANK.

Rozbor príkladu: Tvorba programov s použitím technológie **xBANKING** pri jednočipových mikroprocesoroch 8051 umožňuje používať externé pamäťové priestory presahujúce maximálnu hranicu pre 8051 t.j. 64kB programu a dát. K adresácii tohto pamäťového priestoru sa používajú prídavné adresné vodiče portu P1. V závislosti na type procesora je potom možné pracovať[233] až s 256kB pamäti externej pamäti programu a dát a maximálne 256 bánk pamäti.
Obrázok 287, Pamäťový model externej pamäti programu - xBANKING



Vyššie uvedený príklad ukazuje možnosť pripojiť k mikroprocesoru 8051 navyše ešte 4 banky pamäti[234] programu ROM a 4 banky pamäti dát RAM. Pomocou linkera L51, alebo LX51 je možné vygenerovať s výsledný program pre mikroprocesor 8051 vo formáte INTEL HEX, alebo HEX-386[235], ktorý môžeme zapísať do pamätí s veľkosťou presahujúcou 64kB[236]. Týmto

[233] Bližšie informácie http://www.keil.com/support/man/docs/bl51/bl51_bk_example4.htm
[234] Mikroprocesory DS80C390, DS80C400, DS80C410 a DS80C411 majú možnosť lineárneho adresovania oproti štandardnej 16 bitovej adresy 8051 pomocou úplnej 24 bitovej adresy t.j. až 16MB pamäti ROM, alebo RAM.
[235] Len s LX51
[236] https://developer.arm.com/documentation/ka004583/latest

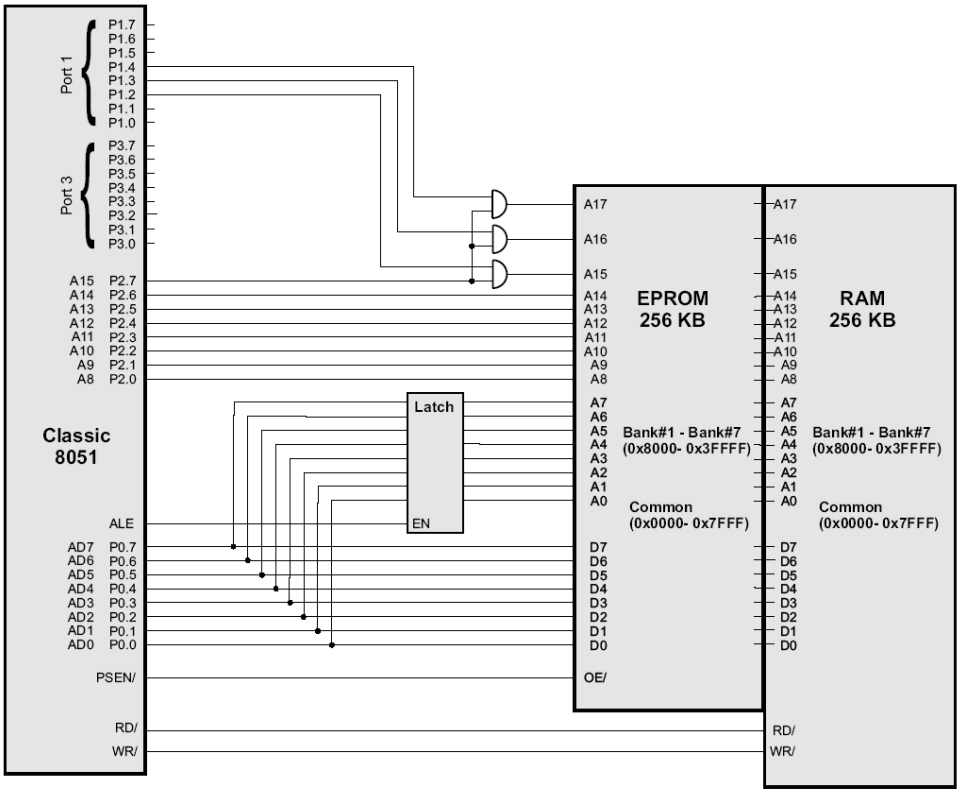
je možné vytvoriť pamäť pozostávajúcu s niekoľkých fyzických segmentov do jednej. Obrázok 287 to názorne zobrazuje.

Obmedzenia použitia pre Common Code Area Bank (CCAB):

- Inicializácia po RESET-e a adresy vektorov prerušení musia byť umiestnené v CCAB, pretože na rozdiel od mikroprocesora 8051 linker LX51 nedokáže vopred zistiť či prišiel signál RESET, alebo prerušenie. Z tohto dôvodu linker LX51 umiestňuje vektory prerušenia a inicializačnú sekvenciu do CCAB.
- Konštanty programu (reťazce, znaky, číselné hodnoty, tabuľky ...) musia byť umiestnené v CCAB, pretože musí mať k nim 8051 prístup z ľubovoľnej časti programu, alebo banky programu. Dodatočne je možné konštanty relokovať z CCAB do jednotlivých bánk, ale môže dochádzať k nekonzistentnosti údajov navzájom.
- Prerušenia musia byť umiestnené vždy v CCAB, pretože prerušenia môžu byť vyvolané z ľubovoľnej časti programu, banky. Kompilátor C51 hlási varovanie ak obslužná rutina prerušenia je deklarovaná v niektorej banke programu.
- Bank Switch Code – obslužný program riadenia xBANKING je kompilátorom umiestňovaný v CCAB a prepínanie je uskutočňované pomocou špeciálnej tabuľky (Switching Code Table) skokov ku ktorej majú prístup všetky banky. Nie je možné pristupovať do tejto tabuľky z jednotlivých bánk. Prístup do tabuľky je umožnený len z CCAB.
- Library Functions – knižnice podprogramov, funkcií použité kompilátorom Cx51, alebo PL/M-51 musí byť umiestnené v CCAB. Pri odovzdávaní hodnôt medzi CCAB a niektorou bankou, alebo vzájomnom prenose návratových hodnôt funkcií sú používané registre mikroprocesora 8051. Linker Lx51 umiestňuje tento kód len do CCAB a nenachádza sa v žiadnej banke! Z tohto dôvodu je veľmi obtiažné určiť veľkosť výsledného programového kódu, pretože jeho veľkosť závisí na použitom hardware 8051 a množstva použitých funkcií. Linker Lx51 niekedy nedokáže účinne odstrániť duplicitné časti programu z niektorých bánk, čo v konečnom dôsledku môže spôsobiť nedostatok systémových prostriedkov pamäte mikroprocesora 8051.
- Prepnutie medzi bankami vyžaduje približne 50 cyklov 8051 a spotrebuje 2 byte na vrchole zásobníka. Z tohto dôvodu sa odporúča, aby často vykonávané procedúry a funkcie boli umiestňované v CCAB a nie v niektorej banke.

Bližšie informácie o výhodách a nevýhodách xBANKINGu je možné nájsť na www.keil.com.

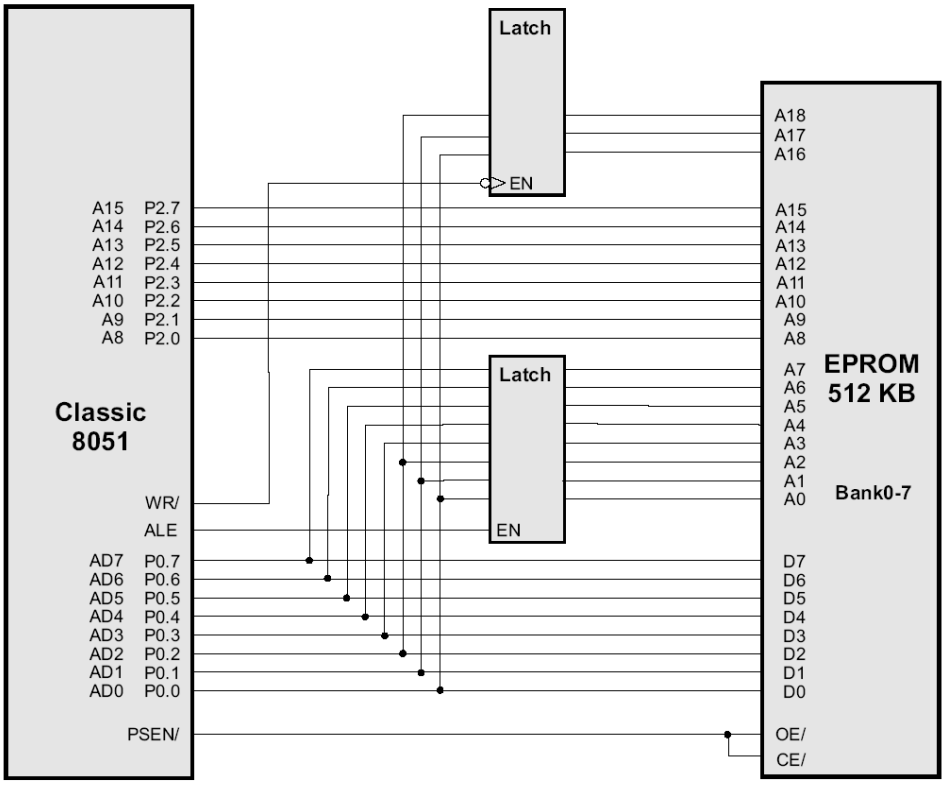
Obrázok 288, Zapojenie pamäti 256kB RAM a EPROM k 8051, 8x32kB.



Vyššie uvedený obrázok ukazuje možnosť využitia prídavnej pamäti RAM a ROM celkovo s veľkosťou 256kB. Adresovanie je vytvorené pomocou prídavných adresných vodičov A16, A17, A18 ktoré vznikli logickým súčinom A15 a príslušnými P1.2, P1.3, P1.4.

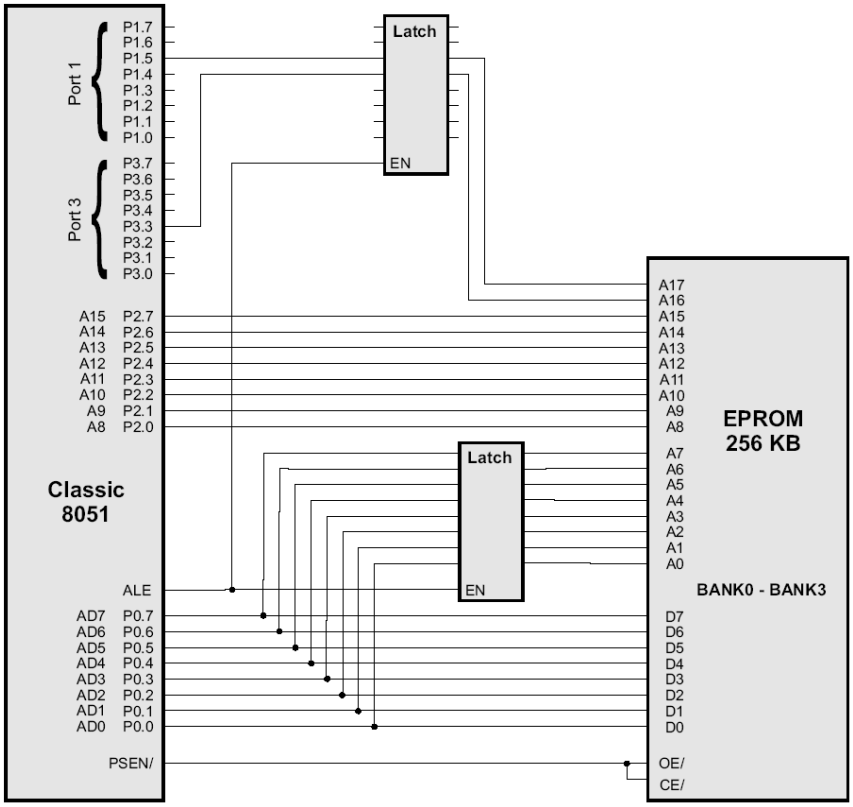
Tento spôsob má oproti iným zapojeniam veľkú výhodu, pretože okrem pamäti programu zväčšuje aj pamäť dát. Takto môže programátor mať k dispozícii pre každú banku programu aj vlastnú banku dát, ktorá je fyzicky oddelená. Určitou nevýhodou je, že maximálne množstvo pamäti ktoré môžeme týmto spôsobom využívať je 256kB v adresnom priestore 0000h až 7FFFh pre *Common Code Area Bank* a od 8000h do 3FFFFh pre program a dáta Bank0 až Bank7. Takže pre programátora je k dispozícii priestor v rozsahu 8000h až FFFFh t.j. 32kB pre Bank0 a 18000h až 1FFFFh pre Bank1 t.j. ďalších 32kB.

Obrázok 289, Zapojenie 512kB pamäti programu k 8051 pomocou adresovania XDATA



Tento spôsob adresovania umožňuje adresovať maximálne 512kB pamäti programu. Pre adresné vodiče A16 až A18 je použitý záchytný obvod LATCH, ktorý využíva ako strobovací (potvrdzovací) vodič zápisový signál $\overline{WR}$ pamäti dát mikroprocesora 8051. Signál *ALE* je použitý pre zachytenie 8 bitovej adresy ktorá je multiplexovaná na porte P0. Výberový signál $\overline{PSEN}$ je použitý pre pamäť programu. Programátor má k dispozícii pamäťový priestor pozostávajúci z ôsmich 64kB bánk programu v adresnom priestore 0000h až 80000h t.j. Bank0 až Bank7. Bank0 má k dispozícii pamäťový priestor v rozsahu adries 0000h až FFFFh, Bank1 10000h až 1FFFFh atď.

Obrázok 290, Zapojenie 256kB pamäti programu k 8051 pomocou štyroch 64kB bánk



Tento spôsob adresovania umožňuje adresovať maximálne 256kB pamäti programu. Signál *ALE* je použitý pre zachytenie 8 bitovej adresy ktorá je multiplexovaná na porte P0 a zároveň aj pre adresné vodiče A16 a A17 ktoré sú použité spolu s A15 pre vytvorenie 19 bitovej adresy pre rozšírený pamäťový priestor 8051. Výberový signál $\overline{PSEN}$ je použitý pre pamäť programu. Programátor má k dispozícii pamäťový priestor pozostávajúci zo štyroch 64kB bánk programu v adresnom priestore 0000h až 40000h t.j. Bank0 až Bank3. Pamäťový priestor pre Bank0 je v rozsahu 0000h až FFFFh, pre Bank1 10000h až 1FFFFh atď.

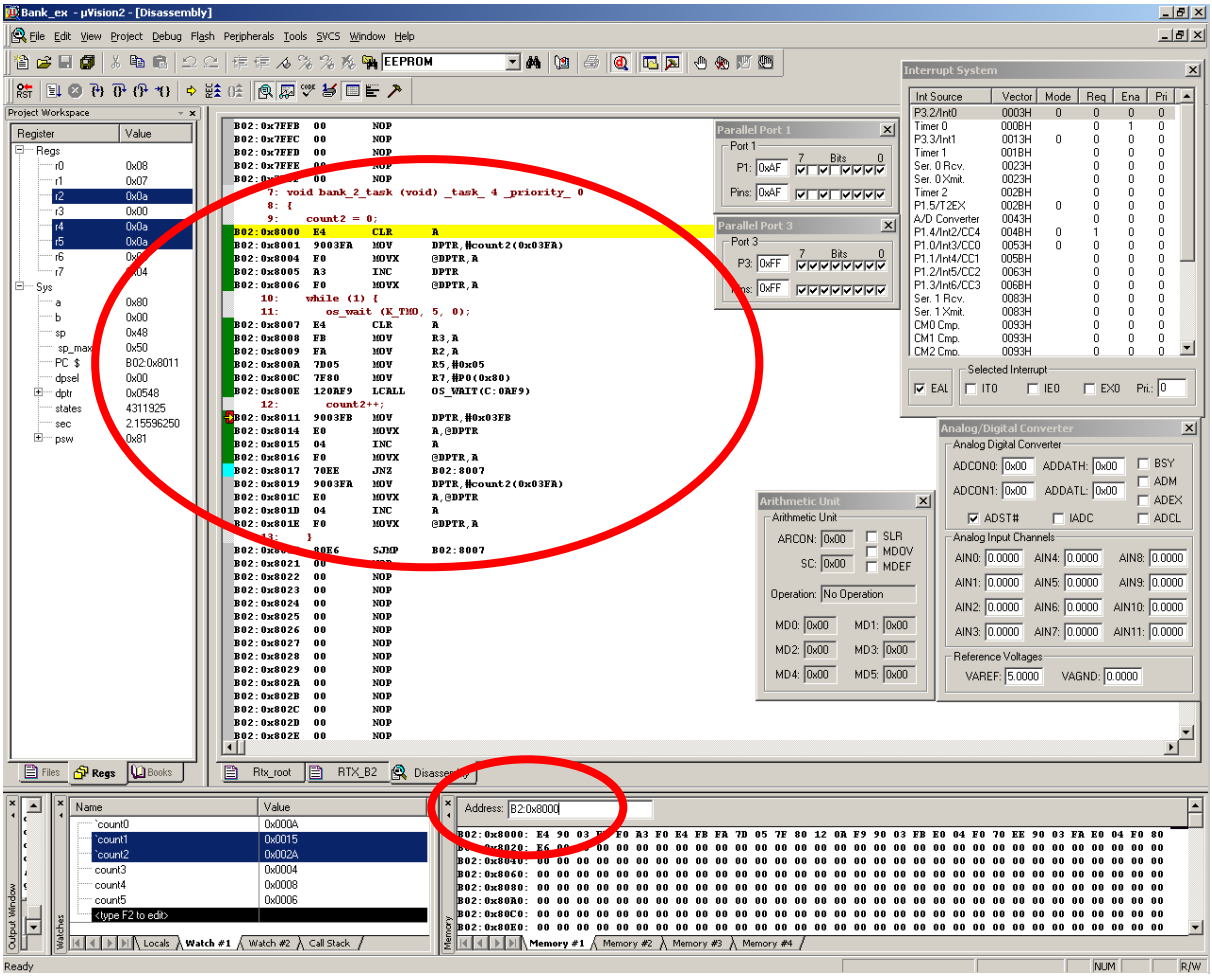
Obrázok 291, Popis premenných (Environments) pre xBANKING 8051

Name	Description
?B_NBANKS	number of banks to be supported. The following values are allowed: 2, 4, 8, 16, and 32. Two banks require one additional address (or I/O Port) line; four banks require two lines; eight banks require three lines; sixteen banks require four lines, and thirty-two banks require five address lines.
?B_MODE	indicates the way how the address extension is done. 0 for using an standard 8051 I/O Port, 1 for using an XDATA port; 2 for using 80C51MX address line; and 4 for user provide bank switch code.
?B_RTX	specifies if the application uses RTX-51 Full. Only ?B_MODE 0 and 1 are supported by RTX51 Full.
?B_VARBANKING	Enables variable banking in XDATA and CODE memory. Variable banking requires the **LX51** linker/locater. It is not supported by **BL51**. Refer to "Banking With Common Area" on page 303 for an example on how to setup the **LX51** linker/locater.
?B_RST_BANK	Specifies the default bank that is selected after CPU reset. This setting is used by the **LX51** linker/locater to reduce the entries in the INTERBANK CALL TABLE. The value 0xFF disables this optimization. This value is not used by **BL51**.
For ?B_MODE = 0 (bank switching via 8051 I/O Port) define the following:	
?B_PORT	used to specify the address of the internal data port. The SFR address of an internal data port must be specified. (For example: P1 as for port 1).
?B_FIRSTBIT	indicates which bit of the 8051 I/O port is to be assigned first. The value 3 indicates that port bit 3 is used as first port line for the address extension. If, for example, two address lines are used, P1.3 and P1.4 are allocated in this case. The remaining lines of the 8051 I/O port can be used for other purposes.

Obrázok 292, Zoznam argumentov LX51 a BL51 pre xBANKING 8051

Argument	Description
address	A value representing a memory location. For BL51, L251 and LX51 in Philips 80C51MX mode plain numbers are used to represent an address. LX51 uses for classic 8051 devices a memory prefix in the address specification. For example: D:0x55 refers to DATA memory address 0x55 C:0x8000 refers to CODE memory address 0x8000 B4:0x4000 refers to CODE memory address 0x4000 in code bank 4.
classname	A name of a memory class. The ***x51*** tools allows basic classes and user defined classes. Refer to "Memory Classes and Memory Layout" the page 27 for more information about memory classes.
filename	A file name that corresponds to the Windows file name conventions.
modname	A module name. Can be up to 40 characters long and must start with: **A – Z**, **?**, **_**, or **@**; following characters can be: **0 – 9**, **A – Z**, **?**, **_**, or **@.**
range	An address range in the format: ***startaddress*** [**– *endaddress***] The ***startaddress*** is the first address specified by the range. The ***endaddress*** is optional and specifies the last address which is included in the address range.
segname	A segment name. Can be up to 40 characters long and must start with: **A – Z**, **?**, **_**, or **@**; following characters can be: **0 – 9**, **A – Z**, **?**, **_**, or **@.**
sfname	A segment or function name.
value	A number, for example, 1011B, 2048D, 0x1000, or 0D5FFh.

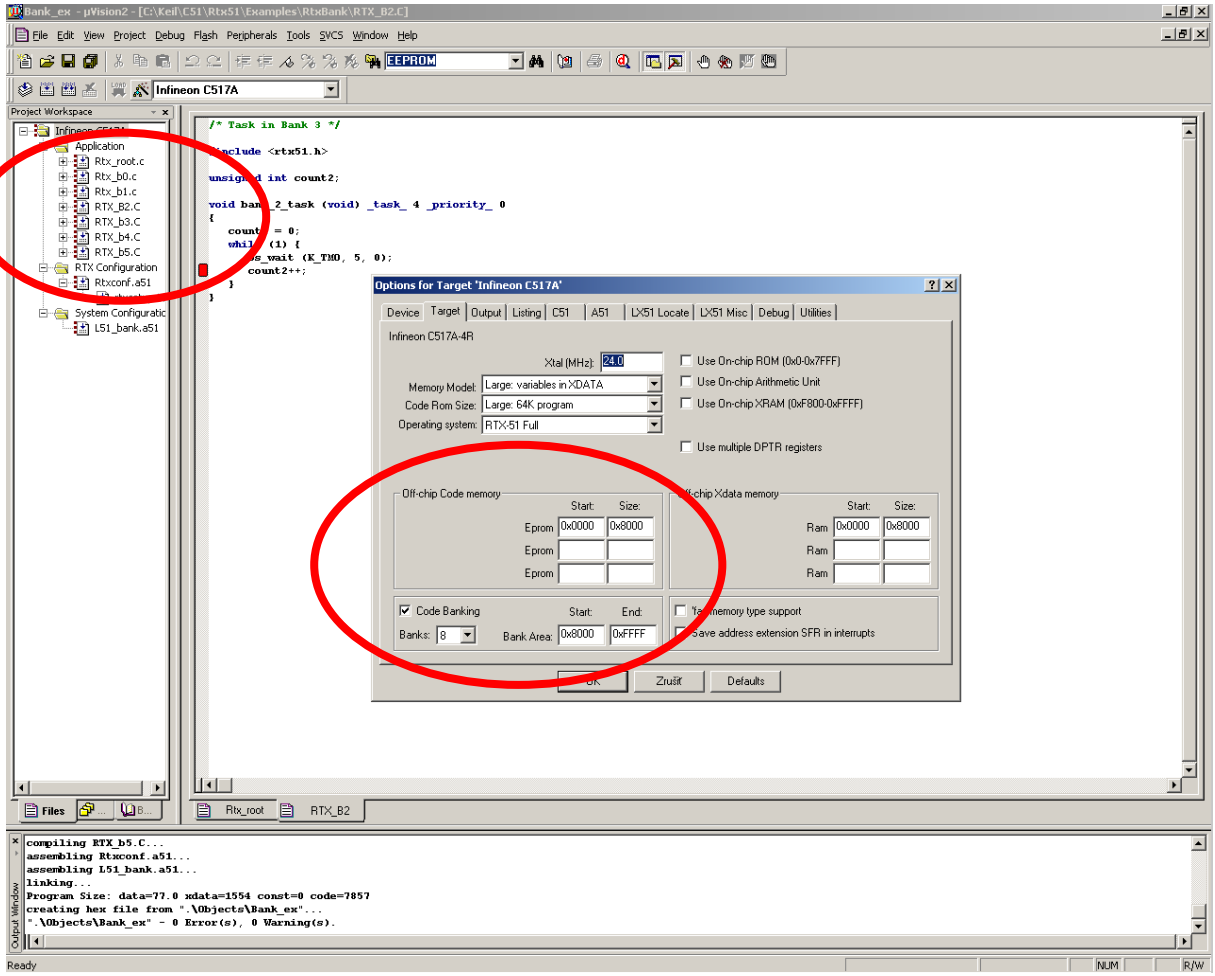
Obrázok 293, Ukážka odlaďovaného programu s RTX51 Full a xBANKING (6 bánk)



Vyššie uvedená ukážka zobrazuje princíp zdieľania a samotnej existencie pamäťového priestoru pre 8051. Principiálne tu existuje 6 bánk v ktorých je uložený program od adresy 8000h do FFFFh s periodickým offsetom 10000h. V pamäťovom priestore od 0000h až do 7FFFh s periodickým offsetom 10000h je uložený hlavný program obsahujúci hlavný kód programu a obsluhu pre xBANKING. Prepínanie pamäťového priestoru je vytvorené doplnením troch adresných vodičov A16 až A18, ktoré umožňujú adresovať pripojenú pamäť programu ROM, prípadne pamäť dát RAM.

Pri samotnom odlaďovaní programu v prostredí Keil µVision3 je potom adresa programu v rozšírenom pamäťovom priestore zobrazovaná v tvare B02:0x8000, kde prvá časť adresy nesie o informáciu o príslušnej banke programu a druhá časť jej adresu vo fyzickom pamäťovom priestore. Žltým pruhom je zobrazený začiatok programu uložený v príslušnej banke B02 na adrese 8000h.

Obrázok 294, Nastavenie prostredia KEIL µVision 3 pre prácu s xBANKING



Príklad programu C_ROOT.C:

```
/*---------------------------------------------------------------------------
C_ROOT.C

Copyright 1995-1996 Keil Software, Inc.
---------------------------------------------------------------------------*/

#include <stdio.h>
#include <reg51.h>

extern void func0(void);
extern void func1(void);

void main(void)
{
/* INITIALIZE SERIAL INTERFACE TO 2400 BAUD @12MHz */
  SCON = 0x52;    /* SCON */
  TMOD = 0x20;    /* TMOD */
  TCON = 0x69;    /* TCON */
  TH1 =  0xf3;    /* TH1 */
  printf("MAIN PROGRAM CALLS A FUNCTION IN BANK 0 \n");
  func0();
  printf("MAIN PROGRAM CALLS A FUNCTION IN BANK 1 \n");
  func1();
  while(1);
}
```

Príklad programu C_BANK0.C:

```
/*---------------------------------------------------------------------------
C_BANK0.C

Copyright 1995-1996 Keil Software, Inc.
---------------------------------------------------------------------------*/

#include <stdio.h>

extern void func2(void);

void func0(void)
{
  printf("FUNCTION IN BANK 0 CALLS A FUNCTION IN BANK 2 \n");
  func2();
}
```

Príklad programu C_BANK1.C:

```
/*---------------------------------------------------------------------------
C_BANK0.C

Copyright 1995-1996 Keil Software, Inc.
---------------------------------------------------------------------------*/

#include <stdio.h>

extern void func2(void);

void func0(void)
{
  printf("FUNCTION IN BANK 0 CALLS A FUNCTION IN BANK 2 \n");
  func2();
}
```

Príklad programu C_BANK2.C:

```
/*---------------------------------------------------------------------------
C_BANK2.C

Copyright 1995-1996 Keil Software, Inc.
---------------------------------------------------------------------------*/

#include <stdio.h>

void func2(void)
{
  printf("THIS IS A FUNCTION IN BANK 2! \n");
}
```

18 Inštalácia programu µVision

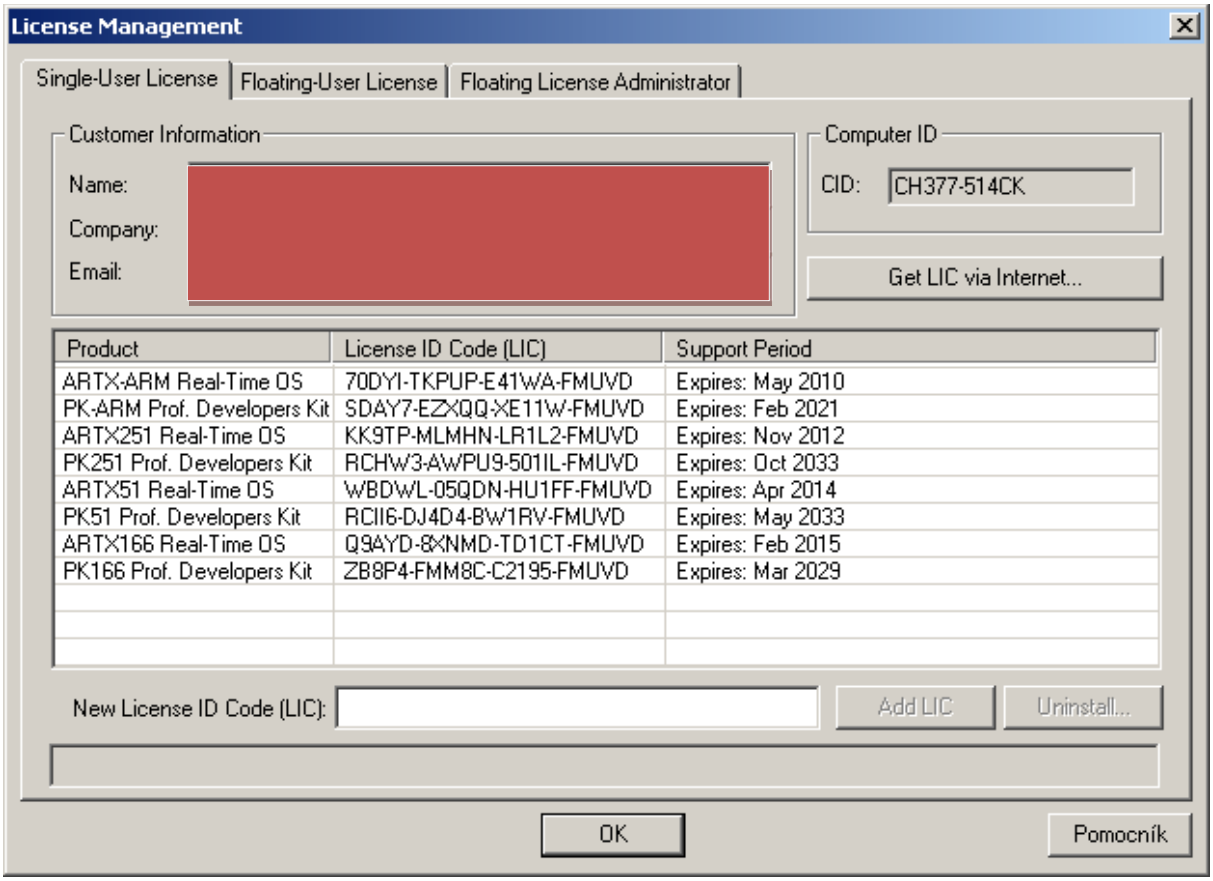
18.1. Inštalácia programu µVision2

Vyššie uvedený program je dodávaný štandardne na CDROM. Súčasťou dodávky je aj tzv. Addon disketa, ktorá obsahuje licenciu s unikátnym číslom, ktoré sa pri inštalácii odosiela pomocou internetu do ústredia firmy, ktorá inštalovaný produkt zaregistruje. Podľa typu licencie sa nainštaluje do PC zakúpený produkt. Práve táto disketa spolu s HW kľúčom „Dongle“ pripojenému k portu LPT určuje či sa bude inštalovať len A51, CA51, DK51, PK51. Inštalačné súbory nachádzajúce sa na CDROM sú pre všetky distribuované verzie. Celý inštalačný proces je automatizovaný, takže ho zvládne aj začiatočník. Ceny týchto produktov sa pohybujú od 375€ do 3500€. Ceny operačných systémov RTX od 350€ do 10000€. Ceny sú platné pre rok prvú polovicu roka 2005.

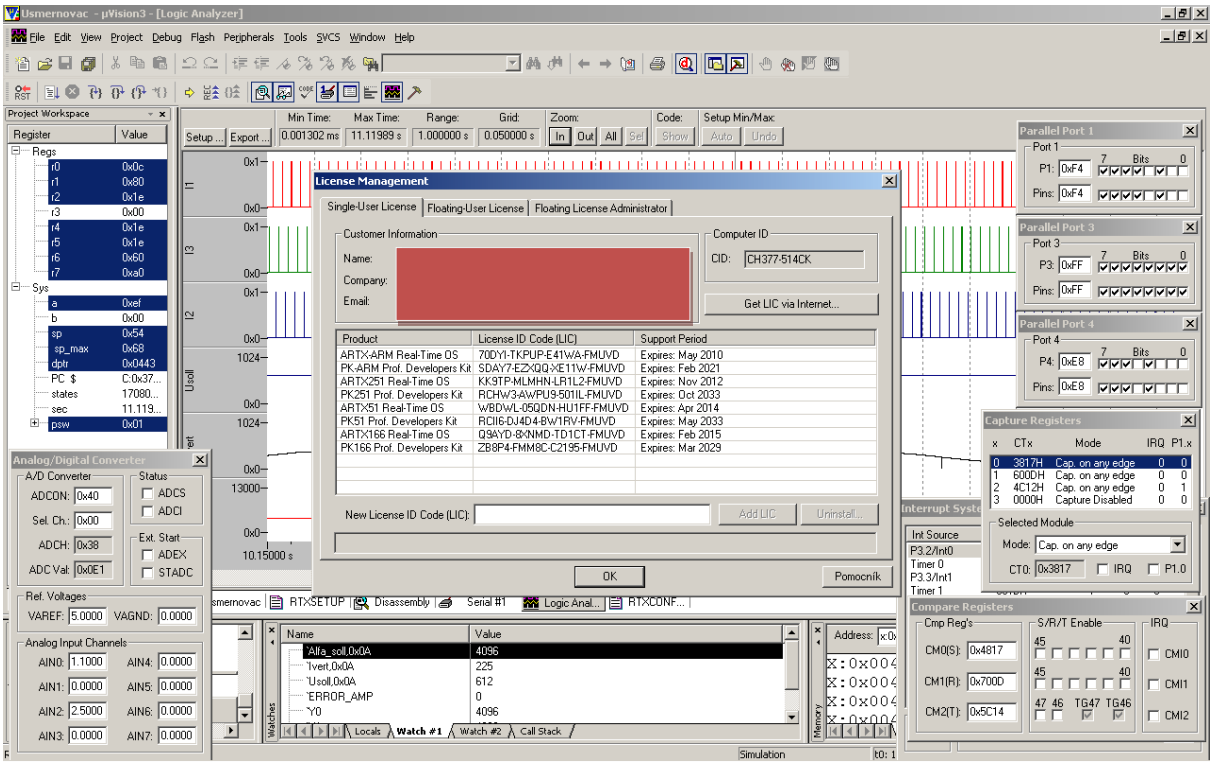
18.2. Inštalácia programu µVision3, µVision4 a vyšších verzií

Firma Keil Software inc. vzhľadom na prelomenie ochrany proti nelegálnemu kopírovaniu a neoprávnenému šíreniu svojich produktov pristúpila k zmene spôsobu nadobúdania licencií. K inštalácií produktov od firmy Keil je potrebné mať len inštalačné CDROM, HW kľúč a sériové číslo (Serial Number), ktoré je priložené v balíku priložené. Tieto produkty už nie je možné nainštalovať bez HW kľúča pripojeného k portu USB. Inštalácia je viazaná unikátnym číslom k danému PC. Toto číslo je generované podľa sériového čísla jednotlivých komponentov osadených v počítači. Pri inštalácií sa v menu License Management vygeneruje tzv. CID (Computer ID), ktoré sa odosiela do ústredia firmy Keil, ktorá po korektnom zaregistrovaní produktu odošle spätne LIC (New License ID Code), ktoré sa zapíše do okna License Management. Od okamihu zadania LIC program poskytuje užívateľovi len tie programové produkty, ktoré má legálne zakúpené.

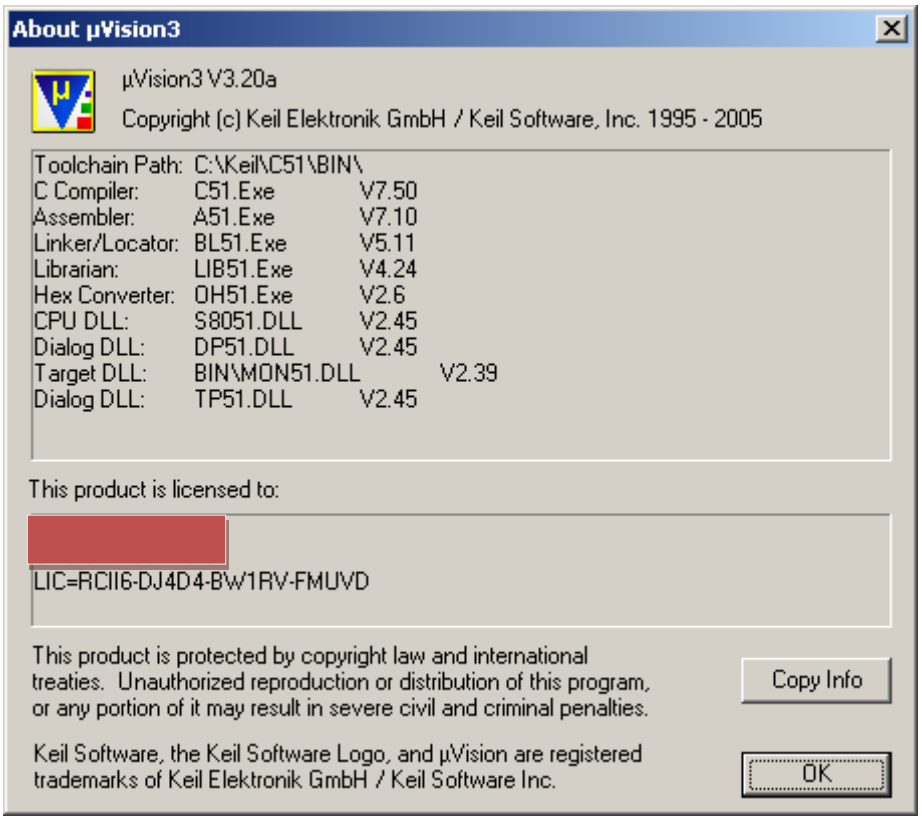
Obrázok 295, Správa a inštalácia licencií v prostredí µVision3



Obrázok 296, Zobrazenie licencií v prostredí µVision3



Obrázok 297, Informačné okno programu µVision3



Poznámka autora:

Práve v P2P sietiach bolo možné získať plne funkčnú „crack“ verziu vyššie uvedených produktov. Tie sú dostupné dodnes, ale už len v starších verziách neumožňujúcich pracovať v prostredí µVision3. Crack na µVision3 a µVision4 už existuje, ale vzhľadom na dobré a seriózne vzťahy s firmou Keil Software nie je ďalej distribuovaný. Tiež spôsob ochrany tohto produktu zatiaľ nebol doteraz zverejnený a podľa mojich vedomostí je tento „crack“ autorovým duševným vlastníctvom jedného programátora zo stredného Slovenska. Na prekonanie jednoduchšieho typu ochrany µVision2 potreboval cca. 9 mesiacov občasnej práce popri výkone vojenskej služby. Produkt ochrany produktu µVision3 už bol prekonaný za cca. 1 týždeň a µVision 4 doteraz nedoznal podstatnejších zmien.[237]

Keďže vyššie uvedený programový produkt je naozaj výnimočným profesionálnym pomocníkom, budúcemu programátorovi isto nebude ľúto v budúcnosti vynaložiť finančné prostriedky na zakúpenie tohto nástroja.

[237] V súčasnosti je už k dispozícii prostredie µVision4 vo verzii PKC51 ver. 9.50a a vyššie uvedená ochrana je s malými obmenami opäť použitá. Od verzie 9.53a je už distribuované prostredie označované ako µVision5, ktoré je už oveľa prepracovanejšie a stabilnejšie ako predchádzajúce verzie.

19 Zoznam bibliografických odkazov

Abelovský, M. 2003. *Pozorovatele stavových veličín bezsnímačových servopohonov s AM.* Bratislava : Dizertačná práca, STU v Bratislave, 2003.
ALEXÍK, Mikuláš, JURÍČEK, Jozef a MIČEK, Juraj. 1993. *Základy automatického riadenia.* Druhé vydanie. Žilina : Vysoká škola dopravy a spojov v Žiline, 1993. s. 119. ISBN 80-7100-113-9.
Allworth, S.T. *Introduction to Real-Time Software Design.* New York. : Springer-Verlag Inc.
Balátě, Jaroslav. 2003. *Automatické řízení.* Praha : BEN - technická literatura, Věšínova 5, Praha 10, 2003. ISBN 80-7300-020-2.
—. 1996. *Vybrané statě z automatického řízení.* Brno : VUT, 1996.
Bednárik, B. a kol. 1983. *Elektroenergetika v doprave.* Bratislava : Alfa, 1983.
Bobál, V, a iní. 2005. *Advanced Textbooks in Control and SignalProcessing: Algorithms,Implementation and Applications.* Prvá. London : Springer-Verlag London, 2005. s. 315. ISBN 1-85233-980-2.
Bobál, V., a iní. 2005. *Digital Self-tuning Controllers Algorithms, Implementation and Applications.* Prvá. London : Springer-Verlag London Limited 2005, 2005. s. 328. ISBN-10: 1852339802.
Boldea, I. a Nasar, S. A. 1998. *Electric Drives.* s.l. : CD Interactive Version, 1998.
Burkhard, Mann . 2003. *C pro mikrokontroléry (ANSI-C, kompilátory, linkery, příklady, nástroje, tipy a triky,..).* Prvá. Praha : BEN - technická literatura, 2003. s. 280. ISBN 80-7300-077-6.
Caha, Z. a Černý, M. 1990. *Elektrické pohony.* Praha : SNTL, 1990.
Deitel, H.M. 1990. *Operating Systems.* 2. vydanie. s.l. : Addison-Wesley Publishing Company, 1990.
Dogan, Ibrahim. 2000. *Microcontroller Projects in C for the 8051.* Second Edition. Burlington : P T R Prentice-Hall, Inc., 2000. ISBN 0-13-753815-4.
eCert. 2008. Certifikačná autorita ministerstva školstva. *www.ecert.sk.* [Online] 1. 1. 2008. [Dátum: 1. 5. 2010.] <http://www.ecert.sk/doc/>.
Gieras, J. F. 1994. *„Linear Induction Drives“.* New York : Oxford University Press Inc., 1994. ISBN 0-19-859381-3.
Goldsmith, Sylvia. *A practical guide to Real-Time Systems Development.* s.l. : Prentice Hall.
Hrabovcová, V., a iní. 2004. *Meranie a modelovanie elektrických strojov.* Žilina : EDIS - vydavateľstvo ŽU, 2004.
HRUBINA, K., JADLOVSKÁ, A a HREHOVÁ, S. 2005. *Algoritmy optimalizačných metód s využitím programových systémov.* Prvá. Košice : Edícia monografií TU Košice, Informatech Ltd., 2005. s. 226-302. EAN 97 880 88941316. ISBN 80-88941-31-8.
Hrubý, Dušan. 2009. *Programovanie v jazyku C a Keil C51.* I. Nitra : Slovenská poľnohospodárska univerzita v Nitre, 2009. s. 181. ISBN 978-80-552-0219-8.
ir. drs. E.H.W. van de Logt. 2011. *PID Controller Calculus, V3.20.pdf.* s.l. : ir. drs. E.H.W. van de Logt, 2011. PID Controller Calculus for HERMS home-brewing system.
Jadroň, Eduard. 2015. *Príručka programovania v ASM a C s RTX51 od Diversant Software s príkladmi.* 2. Martin : Diversant Software, 2015. s. 260. DSW.
Javůrek, J. 2003. *Regulace moderních elektrických pohonů.* Praha : GRADA Publishing, 2003. ISBN 80-247-0507-9.
Keil Elektronik GmbH, Mettler & Fuchs AG a Keil Software Inc. 2002. *RTX51 Real-Time Executive for the 8051 Microcontroller.* s.l. : Silicon Valley, 2002.
Keil Embedded Development. 2016. RTX51 Tiny User's Guide. *www.keil.com.* [Online] KEIL, 1. 9 2016. http://www.keil.com/support/man/docs/tr51/tr51_libref.htm.

Keringham, Brian W. a Ritchie, Dennis M. 1998. *The C Programming Language.* Enlwood Cliffs, New Jersey 07632 : PRENTICE HALL, 1998.
Kopecký, Ladislav. 2008. *Reluktančný motor a elektromobil.* [PDF dokument] Brno : s.n., 2008.
NORSONIC, Slovensko s.r.o. 2008. http://norsonic.sk. *NORSONIC.* [Online] 1. január 2008. [Dátum: 20. október 2011.] www.norsonic.sk/img/Dynamicke_vlastnosti_automobilu.pdf.
Novák, Vilém. 2000. *Základy fuzzy modelování.* 1. vyd. Praha : BEN - technická literatura, 2000. ISBN 80-7300-009-1.
Poliak, F., Fedák, V. a Zboray, L. 1987. *Elektrické pohony.* Bratislava : Alfa, 1987.
Richter a Lutz. 1985. *Betriebssysteme.* s.l. : Teubner Stuttgart, 1985.
Ripps a David. 1988. *A Guide to Real-Time Programming.* s.l. : Englewood Cliffs, N.J, Prentice Hall, 1988.
RTX Real Time Operating System. **Jadroň, Eduard, Feňo, Ivan a Špánik, Pavol. 2001.** Žilina : 4th International Scientific Conference, 2001.
Skýva, L. 1987. *Teória automatického riadenia I.* Bratislava : ALFA, 1987.
ŠVARC, IVAN. 2002. *Automatizace – Automatické řízení.* Brno : Akademické nakladatelství CERM,s.r.o., 2002.
—. 2003. *Teorie automatického řízení.* Brno : VÚT v Brně, 2003.
Trevor, Martin. 2013. *The Designer's Guide to the Cortex-M Processor Family.* Druhá. 225 Wyman Street, Waltham, MA 02451, USA : Newnes is an imprint of Elsevier, The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB, 2013. s. 318. ISBN: 978-0-08-098296-0.
Vegr, Jaromír. 2009. Elektromobily - historie a současnost. *http://www.pro-energy.cz.* [Online] 29. 9 2009. [Dátum: 24. 10 2011.] http://www.pro-energy.cz/clanky7/3.pdf.
Vittek, Ján. 2004. *Vybrané metódy riadenia elektrických pohonov v prostredí MATLAB-Simulink,.* Trenčín : Trenčianská univerzita Alexandra Dubčeka, 2004. ISBN 80-8075-039-4.
Zboray, L., Ďurkovský, F. a Tomko, J. 2000. *Regulované pohony.* Košice : Vienala Košice, 2000. ISBN 80-88922-13-5.
Zeman, K., Peroutka, Z. a Janda, M. 2004. *Automatická regulace pohonů s asynchronními motory.* Plzeň : Západočeská univerzita v Plzni, 2004. ISBN 80-7043-350-7.
Želmíra, Ferková. 2004. Bezsnímačové riadenie spínaného reluktančného motora. *AT&P journal.* AT&P journal, 2004.

20 Zoznam obrázkov

21 Zoznam videosekvencií

22 Zoznam tabuliek

23 Zoznam matematických vzťahov

$$T_{PWM} = \frac{1}{f_{PWM}} = \frac{1}{20 \cdot 10^3} = 50 \cdot 10^{-6} = 50\mu s$$ (s; Hz) (90.) 301

$$PERIODE = \frac{f_{OSC}}{prescaler \cdot f_{PWM} \cdot 2} = \frac{40 \cdot 10^6}{4 \cdot 20 \cdot 10^3 \cdot 2} = 250$$ (1; Hz, Hz) (91.) 301

$$OFFSET = \frac{t_{dead} \cdot f_{OSC}}{prescaler} = \frac{1 \cdot 10^{-6} \cdot 40 \cdot 10^{-6}}{4} = 10$$ (1; s, Hz, 1) (92.) 301

$$f_{MIN} = \frac{1}{65536 \cdot T_{PWM}} = \frac{1}{65536 \cdot 50 \cdot 10^{-6}} = 0.305Hz$$ (Hz; s) (93.) 301

$$f_{MAX} = \frac{1}{256 \cdot T_{PWM}} = \frac{1}{256 \cdot 50 \cdot 10^{-6}} = 78.125Hz$$ (Hz; s) (94.) 301

$$angle = angle + delta$$ (1; 1, 1) (95.) 301

$$delta_{MAX\_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 65536 = 256$$ (1; Hz, Hz) (96.) 301

$$delta_{MIN\_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 256 = 1$$ (1; Hz, Hz) (97.) 301

$$\vec{U_1} = \frac{1}{T_S} \times \left(T_{K+1} \times V_{K+1} + T_K \times V_K\right)$$ (98.) 314

$$T_K = T_S \times \frac{\left|\vec{U_1}\right|}{U_{BAT}} \times \sqrt{3} \times \sin\left(\frac{\pi}{3} - \gamma\right)$$ (99.) 314

$$T_{K+1} = T_S \times \frac{\left|\vec{U_1}\right|}{U_{BAT}} \times \sqrt{3} \times \sin(\gamma)$$ (100.) 314

$$T_0 = T_S\left(1 - T_{K+1} - T_K\right)$$ (101.) 314

$$\underline{u}_S = V_{AN}(t) \cdot e^{j0} + V_{BN}(t) \cdot e^{j\frac{2}{3}\pi} + V_{CN}(t) \cdot e^{-j\frac{2}{3}\pi}$$ (102.) 316

$$\underline{u}_S = t_a \cdot \underline{u}_1 + t_b \cdot \underline{u}_2$$ (103.) 323

$$t_a = \frac{2 \cdot U}{\sqrt{3}} \cdot \sin\alpha$$ (104.) 323

$$t_b = U \cdot \left[\cos(\alpha) - \frac{\sin(\alpha)}{\sqrt{3}}\right]$$ (105.) 323

kde $U = |\underline{u}_S|$ je tzv. Modulačný index, $\alpha = \angle \underline{u}_S$ je uhol vektora napätia (106.) 323

$$t_0 = T_0 \cdot \left(1 - t_a - t_b\right)$$ (107.) 325

$$\alpha_1 = \frac{\pi}{6} - \delta$$ (108.) 327

$$\alpha_1 = \frac{\pi}{6} + \delta$$ (109.) 327

$$\delta = \arccos\left(\frac{\sqrt{3}}{2 \cdot U}\right) \quad (110.) \ldots 327$$

$$i_{ZAT} = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cdot \cos(\omega_n \cdot t)) + \sum_{n=1}^{\infty}(b_n \cdot \sin(\omega_n \cdot t)) \quad (111.) \ldots 336$$

$$I_{1ZAT} = \sqrt{a_1^2 + b_1^2}\,;\ \varphi_1 = \arctan\left(\frac{b_1}{a_1}\right) \quad (112.) \ldots 336$$

$$I_{1SIET} = I_{1ZAT} \cdot \cos\varphi_1 \quad (113.) \ldots 336$$

$$i_{ZAT}(t) = i_\alpha + j \cdot i_\beta \quad (114.) \ldots 336$$

$$C_\upsilon = \frac{3}{\pi}\int_0^{\pi/3} RE\{f(\omega t)\} \cdot e^{-j\upsilon\omega t} \cdot d\omega t + j\frac{3}{\pi}\int_0^{\pi/3} IM\{f(\omega t)\} \cdot e^{-j\upsilon\omega t} \cdot d\omega t \quad (115.) \ldots 337$$

$$C_1 = \frac{3}{\pi}\int_0^{\pi/3}\left[(RE \cdot \cos(\omega t) + IM \cdot \sin(\omega t)) + j \cdot (IM \cdot \cos(\omega t) - RE \cdot \sin(\omega t))\right] d\omega t \quad (116.) \ldots 337$$

$$C_1 = \frac{1}{N}\left\{\sum_{i=0}^{N-1} RE(i) \cdot \cos\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \sum_{i=0}^{N-1} IM(i) \cdot \sin\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \right.$$
$$+ \frac{RE(N) \cdot \cos(\pi/3) - RE(0) \cdot \cos(0)}{2} +$$
$$\left. + \frac{IM(N) \cdot \sin(\pi/3) - IM(0) \cdot \sin(0)}{2}\right\} + \quad (117.) \ldots 337$$
$$+ j\frac{1}{N}\left\{\sum_{i=0}^{N-1} IM(i) \cdot \cos\left(\frac{i \cdot \pi}{3 \cdot N}\right) - \sum_{i=0}^{N-1} RE(i) \cdot \sin\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \right.$$
$$+ \frac{IM(N) \cdot \cos(\pi/3) - IM(0) \cdot \cos(0)}{2} -$$
$$\left. - \frac{RE(N) \cdot \sin(\pi/3) + RE(0) \cdot \sin(0)}{2}\right\}$$

$$C_1 = a_1 + j \cdot b_1 \quad (118.) \ldots 338$$