1 VÝVOJOVÝ MODUL ANALOG DEVICES EZ-KIT2181 LITE

1.1 ÚVOD

Vývojový modul Motorola **EZ-KIT2181 Lite** je lacným vývojovým prostriedkom, ktorý umožňuje ladenie programov pre procesor ADSP2181[1]. Doska obsahuje **16 bitový** multimediálny stereo AD/DA **kodek AD1847** od firmy Analog Devices, ktorý spolu s ADSP2181 umožňuje realizovať klasický systém ČSS. Pretože AD a DA prevodníky využívajú **sigma-delta moduláciu**, sú vstupné a výstupné analógové obvody relatívne jednoduché. Doska obsahuje aj $128K \times 8$ bitov EPROM pamäte, ktorá je využitá na zavedenie programu a tým umožňuje využívanie dosky EZ-KIT aj bez nadradeného PC počítača. S pomocou špeciálneho programu (**monitora**) a **PC** je možné ladiť programy pre DSP.

Podobné vývojové dosky existujú aj pre DSP od iných výrobcov. Počas cvičenia bude demonštrovaná činnosť vývojovej dosky Motorola EVM56002, ktorá využíva **16 bitový** multimediálny stereo AD/DA **kodek CS4215** od firmy Crystal Semiconductor. Doska naviac obsahuje[2] $32K \times 24\ bitov$ rýchlej statickej RAM pamäte (SRAM), pričom obsahuje aj päticu pre dodatočnú pamäť FLASH EPROM ($32K \times 8\ bitov$), ktorú je možné využiť na zavedenie programu. S pomocou špeciálneho **OnCE rozhrania** a **PC** je možné veľmi jednoducho[3] ladiť programy pre DSP.

1.2 VÝVOJOVÝ MODUL EZ-KIT 2181 LITE

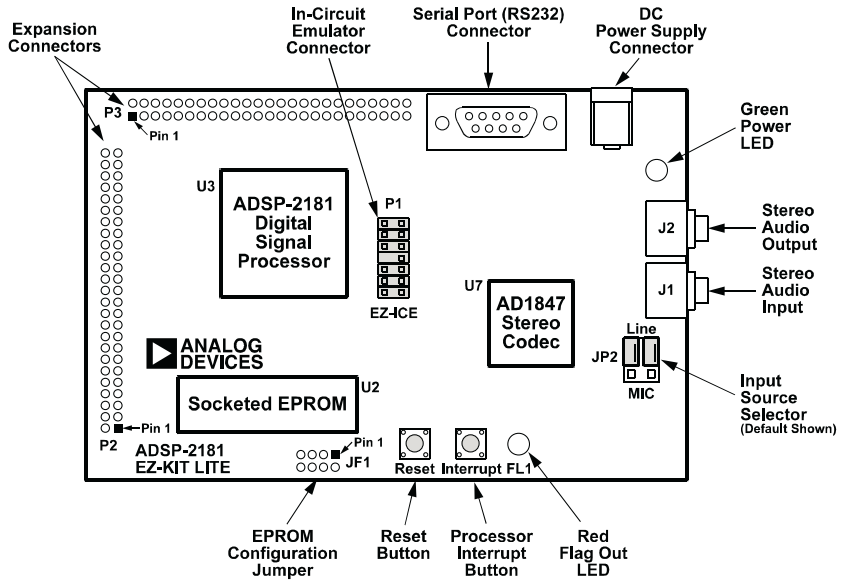
Modul je umiestnený na doske plošných spojov s rozmermi $3,5 \times 5,5$ palcov. Na doske sa nachádza okrem procesora ADSP2181 aj EPROM a kodek AD1847. Tieto súčiastky ako aj ďalšie ovládacie prvky a konektory sú zobrazené na obrázku 1.

---

[1] Dosky EZ-KIT2181 Lite, ktoré budeme využívať na cvičeniach, sú osadené procesorom ADSP2181KS-133 ktorý vykonáva 33 MIPS pri externej taktovacej frekvencii 16,667 MHz. V súčasnosti sú k dispozícii aj vývojové dosky pre novšie verzie procesorov ADSP 218x.

[2] Umiestnenie externých SRAM pamätí kompenzuje veľmi malé interné SRAM pamäte, ktoré sú umiestnené priamo v procesoroch Motorola DSP56002 ($2 \times 256 \times 24$ bitov dátových pamätí a 512×24 bitov programovej pamäte). Procesory ADSP2181 majú na čipe až 80 Kbitov ($16K \times 16$ bitov dátovej pamäte a $16K \times 24$ bitov programovej pamäte) interných pamätí, čo umožňuje realizovať aj zložitejšie systémy ČSS bez externých SRAM pamätí a tým znížiť celkovú cenu systému.

[3] Obvody OnCE umožňujú s využitím lacného externého hardvéru ladenie cieľovej aplikácie bez nutnosti spúšťania monitora v cieľovom systéme. Z tohto pohľadu je ladenie v procesoroch DSP56002 vyriešené lepšie.



Obr.1 Vývojová doska EZ-KIT2181 Lite

EZ-KIT2181 Lite je príkladom **minimálnej implementácie** systému s procesorom ADSP2181, pričom schémy zapojenia jednotlivých blokov dosky sú uvedené v prílohe.

Pamäť EPROM[4] je k procesoru ADSP2181 pripojená pomocou **portu Byte DMA**. Toto pripojenie využíva iba 8 z 24 dátových vodičov (D8-D15). Ďalších 8 dátových vodičov (D16-D23) je využitých ako dodatočné adresové vodiče, čo umožňuje adresovania až 32 Mbitovej EPROM. Procesor ADSP2181 je nakonfigurovaný tak (vývody MMAP a BMODE sú v log. 0), že po resete sú interné SRAM pamäte **incializované z EPROM pamäte**.

Kodek AD1847 je pripojený k ADSP2181 pomocou rýchleho synchrónneho portu SPORT0. Vzhľadom na využitie kodeku so sigma-delta[5] moduláciou sú vstupné analógové obvody pomerne jednoduché. Pomocou prepojky je možné nakonfigurovať AD vstup ako **linkový** (AC viazaný, rozsah 2 V RMS) alebo ako **mikrofóny** vstup (AC viazaný, rozsah 20 mV RMS). Výstup je pevný s rozsahom 1 V RMS.

Port SPORT1 je v systéme využitý na asynchrónnu komunikáciu pomocou rozhrania RS232 (U5). Vývody Flag In a Flag Out sú využité na príjem resp. vysielanie dát. Prijímané dáta sú privedené na IRQ1 a ADSP2181 tak môže detekovať (pomocou prerušenia) aktivitu na prijímacom vodiči bez nutnosti periodického testovania (tzv. pooling) pinu Flag In. DSP emuluje UART a umožňuje tak realizovať asynchrónny prenos s prenosovou rýchlosťou 9600 bitov za sekundu.

Na doske je umiestnená indikačná červená LED (D1) pripojená k vývodu FL1, ktorú je možné ovládať softvérovo pomocou inštrukcií

[4] Na doske je osadená 1 Mbitová EPROM ($128K \times 8$). Pomocou konfiguračných prepojek je možné využiť až 8 Mbitovú EPROM.

[5] Tieto typy kodekov typicky využívajú pripojenie kryštálu na vytvorenie interných hodinových signálov. Alternatívne je možné využiť externé hodinové signály generované napr. signálovým procesorom. Využitie relatívne vysokých hodinových signálov je priamym dôsledkom použitého princípu (vysokého prevzorkovania) prevodu medzi analógovým a číslicovým signálom.

```
SET FL1, RESET FL1, TOGGLE FL1
```

Pomocou týchto inštrukcií je možné LED zapnúť, vypnúť a prepnúť, čo môže byť využité napr. pri ladení programov. V prípade, že je potrebné napr. prepínať rýchlosť blikania, je možné využiť napr. nasledujúci časť kódu

```
        cntr = 80;
        do lp1 until ce;
            cntr = 50;
            do lp2 until ce;
                    cntr = 12000;
                    do lp3 until ce;
lp3:                        nop;
lp2:                 nop;
lp1:        toggle fl1;
```

Vývojovú dosku je možné konfigurovať pomocou konfiguračných prepojek a v prípade potreby je možné využiť prakticky všetky podstatné signály, ktoré sú vyvedené na konektory P2 a P3. Detailné informácie je možné nájsť v [1].

1.3 PROGRAM PRE PRÁCU S AD A DA PREVODNÍKMI

Prevodníky AD1847 sú relatívne zložité obvody. Komunikácia medzi prevodníkom a ADSP2181 prebieha pomocou paketov, ktorých detailná štruktúra je opísaná v manuále obvodu Analog Devices AD1847 [2]. V rámci cvičení nemá zmysel podrobnejšie analyzovať tento prevodník (pri inom type prevodníka bude ovládanie úplne iné) a je výhodné využiť pripravené odladené programy pre prístup k AD a DA prevodníkom, ktoré sú súčasťou balíka **ADDA_test.zip** [3].

Túto funkciu plní podprogram na incializáciu kodeka - podprogram **init_1847**, ktorý umožňuje okamžité využitie prevodníkov, ako aj zmenu ich konfigurácie. Kompletný demonštračný program, ktorý využíva prístup k AD a DA prevodníkom je v súbore **ad_da.asm**. Aj keď program je na prvý pohľad pomerne zložitý, pre využitie je dôležitá predovšetkým časť zapísaná zvýrazneným písmom. Predstavuje tabuľku vektorov prerušení a hlavný program v ktorom je možné realizovať spracovanie vzoriek prijatých z AD prevodníkov (premenné **_left_in, _right_in**) resp. zapisovaných do DA prevodníkov (premenné **_left_out, _right_out**). Pretože spracovanie dát v prerušení je realizované pomocou tieňovej banky registrov, je možné v hlavnej slučke využívať prakticky všetky registre procesora okrem registrov **i2, i3, l2, l3, m1**, ktoré sú využívané na autobufrovanie portu SPORT0.

Kompletný program ad_da.asm je uvedený v nasledujúcej časti:

```
/*************************************************************************************************************************
Nazov suboru:       AD_DA.asm

Datum modifikacie: 28-03-2002 MD

Opis:   Demonstruje inicializaciu a jednoduchu obsluhu stereo   AD/DA kodeka, ktory je na doske
        ADSP2181 EZ-KIT Lite.  Vstupne vzorky su nacitavane z prijimacieho registra RX0
        serioveho rozhrania SPORT0 a vystupne vzorky su zapisovane do vysielacieho registra TX0 portu
        SPORT0.
        Vstupne a vystupne 16-bitove vzorky su spracovavane  v preruseni. V hlavnej slucke je mozne tieto
        Vzorky  spracovat. Program bol upraveny tak, aby umoznoval aj lahke vclenenie do C prostredia
        (t.j. pre autobufrovanie kodeku boli vyuzite  registre i2,i3, a vzorky su zapisovane do premennych)
*************************************************************************************************************************/
```

```
#include    "def2181.h"

.SECTION/DM        data1;
.var    stat_flag;
/* premenne vyuzivane v hlavnej slucke */
.var        _left_in, _right_in;            // premenne pre vstupne vzorky kodeka
.var        _left_out, _right_out;          // premenne pre vystupne vzorky kodeka

/*** main program ***/
.SECTION/DM                 buf_var0;
/* tento cirkulacny bufer je vyuzivany na prijem dat z kodeku, ktore prichadzaju
   v tvare Status + L data + R data. Prerusenie sa vyvola vzdy az po prijme vetkych
   3 udajov, ktore su v preruseni zapisane do _left_in, _right_in */
.var      rx_buf[3];                        /* Status + L data + R data */

/* init_cmds[13] definuje aktualnu konfiguraciu kodeku ADSP 1847 (podrobnejsie
   informacie je mozne najst v katalogovom liste) Tieto prikazy sa vyuzivaju pocas
   inicialicacie kodeku (call init_1847) */
.SECTION/DM        buf_var1;
.var                    init_cmds[13] = 0xc002,     /*
                    Left input control reg
                    b7-6: 0=left line 1
                         1=left aux 1
                         2=left line 2
                         3=left line 1 post-mixed loopback
                    b5-4: res
                    b3-0: left input gain x 1.5 dB */
        0xc102,     /*
                    Right input control reg
                    b7-6: 0=right line 1
                         1=right aux 1
                         2=right line 2
                         3=right line 1 post-mixed loopback
                    b5-4: res
                    b3-0: right input gain x 1.5 dB  */
        0xc288,     /*
                    left aux 1 control reg
                    b7  : 1=left aux 1 mute
                    b6-5: res
                    b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB */
        0xc388,     /*
                    right aux 1 control reg
                    b7  : 1=right aux 1 mute
                    b6-5: res
                    b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB */
        0xc488,     /*
                    left aux 2 control reg
                    b7  : 1=left aux 2 mute
                    b6-5: res
                    b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB */
        0xc588,     /*
                    right aux 2 control reg
                    b7  : 1=right aux 2 mute
                    b6-5: res
                    b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB */
        0xc680,     /*
                    left DAC control reg
                    b7  : 1=left DAC mute
                    b6  : res
                    b5-0: attenuation x 1.5 dB */
        0xc780,     /*
                    right DAC control reg
                    b7  : 1=right DAC mute
                    b6  : res
                    b5-0: attenuation x 1.5 dB */
        0xc850,     /*
                    data format register
                    b7  : res
                    b5-6: 0=8-bit unsigned linear PCM
                         1=8-bit u-law companded
                         2=16-bit signed linear PCM
                         3=8-bit A-law companded
                    b4  : 0=mono, 1=stereo
                    b0-3: 0=  8.
                         1=  5.5125
```

```
                    2= 16.
                    3= 11.025
                    4= 27.42857
                    5= 18.9
                    6= 32.
                    7= 22.05
                    8=  .
                    9= 37.8
                    a=  .
                    b= 44.1
                    c= 48.
                    d= 33.075
                    e=  9.6
                    f=  6.615
                (b0) : 0=XTAL1 24.576 MHz; 1=XTAL2 16.9344 MHz */
        0xc909,    /*
                interface configuration reg
                b7-4: res
                b3  : 1=autocalibrate
                b2-1: res
                b0  : 1=playback enabled */
        0xca00,    /*
                pin control reg
                b7  : logic state of pin XCTL1
                b6  : logic state of pin XCTL0
                b5  : master - 1=tri-state CLKOUT
                      slave  - x=tri-state CLKOUT
                b4-0: res */
        0xccc0,    /*
                miscellaneous information reg
                b7  : 1=16 slots per frame, 0=32 slots per frame
                b6  : 1=2-wire system, 0=1-wire system
                b5-0: res */
        0xcd00;    /*
                digital mix control reg
                b7-2: attenuation x 1.5 dB
                b1  : res
                b0  : 1=digital mix enabled */;

.SECTION/DM       buf_var2;
/* tento cirkulacny bufer je vyuzivany na vysielanie dat z kodeku, data su zapisovane
   v tvare Command + L data + R data. Zapis sa realizuje synchronne s prijmom v obsluhe
   prerusenia od prijimaca, pricom vystupne udaje su citane z premennych
   _left_out, _right_out.
   Vysielacii buffer sa vyuziva aj pocas konfiguracie kodeka (call init_1847), pricom
   pocas tejto konfiguracie je aktivna aj obsluha prerusenia od prijimaca (jump next_cmd) */
.var      tx_buf[3] = 0xc000, 0x0000, 0x0000;     /* Cmd + L data + R data    */

/*****************************************************************************
 *
 *  Tabulka vektorov preruseni
 *
 *****************************************************************************/
 .SECTION/PM    interrupts;
        jump start;  rti; rti; rti;     /*00: reset */
        rti;            rti; rti; rti;     /*04: IRQ2 */
        rti;            rti; rti; rti;     /*08: IRQL1 */
        rti;            rti; rti; rti;     /*0c: IRQL0 */
        ar = dm(stat_flag);        /*10: SPORT0 tx */
        ar = pass ar;
        if eq rti;
        jump next_cmd;
        jump input_samples;         /*14: SPORT0 rx */
                rti; rti; rti;
        rti;            rti; rti; rti;     /*18: IRQE */
        rti;            rti; rti; rti;     /*1c: BDMA */
        jump irq1isr;              /* suvisi s monitorom EZ-KIT Lite */
                rti; rti; rti;     /*20: SPORT1 tx or IRQ1 */
        rti;            rti; rti; rti;     /*24: SPORT1 rx or IRQ0 */
        nop;            rti; rti; rti;     /*28: timer (aktivnt v monitore EZ-KIT Lite) */
        rti;            rti; rti; rti;     /*2c: power down */

 .SECTION/PM    seg_code2;
start:
        call            init_1847;      // inicializacia kodeka
```

```
/*------------------------------------------------------------------------
 -  Hlavna (nekonecna) slucka v ktorej je mozne spracovat udaje z premennych
 -  _left_in, _right_in (vstupne 16-bitove vzorky) a zapisat ich do
 -  _left_out, _right_out (vystupne 16-bitove vzorky).
 -  Kedze obsluha prerusenia pouziva tienovu banku registrov, je mozne pouzivat
 -  vsetky registre ADSP s vynimkou (i2,i3,l2,l3,m1=1, ktore su vyuzivane na
 -  pre autobufrovaci mod SPORT0)
 ------------------------------------------------------------------------*/
        set     fl1;            // zapnutie LEDky na doske EZ-KIT Lite
wt:     idle;                   // znizeny prikon po spracovni vzoriek

        ax0 = dm(_left_in);     // kopiruje data zo vstupu na vystup (L vzorka)
                                // tu je mozne vlozit spracovanie L vzorky
        dm(_left_out) = ax0;
        ax0 = dm(_right_in);    // kopiruje data zo vstupu na vystup (R vzorka)
                                // tu je mozne vlozit spracovanie R vzorky
        dm(_right_out) = ax0;

        jump wt;                // koniec nekonecnej slucky

/*------------------------------------------------------------------------
 -  obsluha prerusenia od SPORT0
 ------------------------------------------------------------------------*/
input_samples:
        ena sec_reg;            // pouzitie tienovej banky registrov
        ax0 = dm(rx_buf+1);     // citanie L vzorky
        ax1 = dm(rx_buf+2);     // citanie R vzorky
        dm(_left_in) = ax0;     // zapis nacitanej L vzorky do _left_in
        dm(_right_in) = ax1;    // zapis nacitanej R vzorky do _right_in

        ax0 = dm(_left_out);    // citanie aktualnej vystupnej L vzorky
        ax1 = dm(_right_out);   // citanie aktualnej vystupnej R vzorky
        dm(tx_buf+1) = ax0;     // zapis vystupnych vzoriek do vysielacieho
        dm(tx_buf+2) = ax1;     // bufra TX_BUF na spravne pozicie
        rti;

/*------------------------------------------------------------------------
 -  obsluha prerusenia od vysielaca SPORT0 (pouzity pre incializaciu kodeu ADSP 1847)
 ------------------------------------------------------------------------*/
next_cmd:
        ena sec_reg;
        ax0 = dm (i0, m1);      // vyber dalsieho riadiaceho slova
        dm (tx_buf) = ax0;      // a umiestnenie do slotu 0 (1 pozicia v tx_buf)
        ax0 = i0;
        ay0 = init_cmds;
        ar = ax0 - ay0;
        if gt rti;              // rti ak este treba vyslat dalsie riadiace slova
        ax0 = 0xaf00;
        dm (tx_buf) = ax0;      // remove MCE if done initialization */
        ax0 = 0;
        dm (stat_flag) = ax0;   // vynuluje premennu flag (koniec konfiguracie)
        rti;


/* Nasledujuce prerusenie suvisi s konstrukciou dosky EZ-KIT Lite a riadiacim
   programom MONITOR v EPROM EZ-KIT Lite. Tu je originalny komentar:
   A high to low transition on flag_in signifies the start bit; it also
   triggers IRQ1 ISR which then turn on timer if the timer is off.  This is
   at to most 1/3 bit period too late but we shoudl still catch the byte. */
irq1isr:
        pop sts;
        ena timer;              /* start timer now */
        rts;                    /* note rts */

/*************************************************************************
 *
 *  Incializacia kodeku ADSP 1847 (autobufering vyuziva i2,i3,l2,l3,m1=1)
 *
 *************************************************************************/

init_1847:
        /* incializacia procesora (SPORT0) */
        /*   shut down sport 0 */
        ax0 = b#0000100000000000;
```

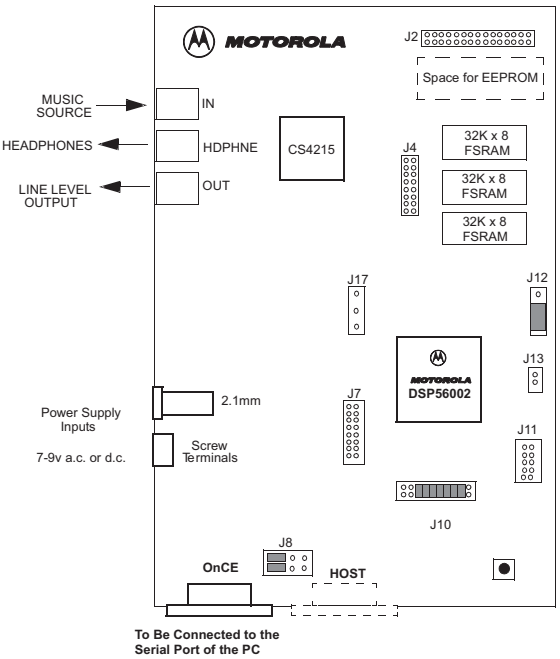
```
        dm (Sys_Ctrl_Reg) = ax0;
        ena timer;
        i2 = rx_buf;
        l2 = LENGTH(rx_buf);
        i3 = tx_buf;
        l3 = LENGTH(tx_buf);
        i0 = init_cmds;
        l0 = LENGTH(init_cmds);
        m1 = 1;

/*================== S E R I A L   P O R T   #0   S T U F F ==================*/
      ax0 = b#0000011010100111;   dm (Sport0_Autobuf_Ctrl) = ax0;
        /*  |||!|-/!/|-/|/|+- receive autobuffering 0=off, 1=on
            |||!|  ! |  | +-- transmit autobuffering 0=off, 1=on
            |||!|  ! |  +---- | receive m1
            |||!|  ! |       |
            |||!|  ! +------- ! receive i2
            |||!|  !         !
            |||!|  !         !
            |||!|  +========== | transmit m1
            |||!|            |
            |||!+------------ ! transmit i3
            |||!             !
            |||!             !
            |||+============== | BIASRND MAC biased rounding control bit
            ||+--------------- 0
            |+---------------- | CLKODIS CLKOUT disable control bit
            +---------------- 0 */

      ax0 = 0;    dm (Sport0_Rfsdiv) = ax0;
        /*   RFSDIV = SCLK Hz/RFS Hz - 1 */
      ax0 = 0;    dm (Sport0_Sclkdiv) = ax0;
        /*   SCLK = CLKOUT / (2  (SCLKDIV + 1) */
      ax0 = b#1000011000001111;   dm (Sport0_Ctrl_Reg) = ax0;
        /*  multichannel
            ||+--/!||+/+---/ | number of bit per word - 1
            |||   |!|||      | = 15
            |||   |!|||      |
            |||   |!|||      |
            |||   |!||+====== ! 0=right just, 0-fill; 1=right just, signed
            |||   |!||       ! 2=compand u-law; 3=compand A-law
            |||   |!|+------- receive framing logic 0=pos, 1=neg
            |||   |!+-------- transmit data valid logic 0=pos, 1=neg
            |||   |+========= RFS 0=ext, 1=int
            |||   +---------- multichannel length 0=24, 1=32 words
            ||+-------------- | frame sync to occur this number of clock
            ||                | cycle before first bit
            ||                |
            ||                |
            |+---------------- ISCLK 0=ext, 1=int
            +----------------- multichannel 0=disable, 1=enable */
        /*  non-multichannel
            |||!|||!|||!+---/ | number of bit per word - 1
            |||!|||!|||!      | = 15
            |||!|||!|||!      |
            |||!|||!|||!      |
            |||!|||!|||+===== ! 0=right just, 0-fill; 1=right just, signed
            |||!|||!||+------ ! 2=compand u-law; 3=compand A-law
            |||!|||!|+------- receive framing logic 0=pos, 1=neg
            |||!|||!+-------- transmit framing logic 0=pos, 1=neg
            |||!|||+========= RFS 0=ext, 1=int
            |||!||+---------- TFS 0=ext, 1=int
            |||!|+----------- TFS width 0=FS before data, 1=FS in sync
            |||!+------------ TFS 0=no, 1=required
            |||+============== RFS width 0=FS before data, 1=FS in sync
            ||+-------------- RFS 0=no, 1=required
            |+---------------- ISCLK 0=ext, 1=int
            +----------------- multichannel 0=disable, 1=enable */

/* THIS PROGRAM USES 16 SLOTS PER FRAME */
      ax0 = b#0000000000000111;   dm (Sport0_Tx_Words0) = ax0;
        /*   ^15          00^   transmit word enables: channel # == bit # */
      ax0 = b#0000000000000000;   dm (Sport0_Tx_Words1) = ax0;
        /*   ^31          16^   transmit word enables: channel # == bit # */
      ax0 = b#0000000000000111;   dm (Sport0_Rx_Words0) = ax0;
```

```
      /*   ^15         00^   receive word enables: channel # == bit # */
      ax0 = b#0000000000000000;   dm (Sport0_Rx_Words1) = ax0;
      /*   ^31         16^   receive word enables: channel # == bit # */

/*============== S Y S T E M   A N D   M E M O R Y   S T U F F ==============*/
      ax0 = b#0001100000000000;   dm (Sys_Ctrl_Reg) = ax0;
      /*  +-/!||+-----/+-/- | program memory wait states
          |  !|||          | 0
          |  !|||          |
          |  !||+---------- 0
          |  !||           0
          |  !||           0
          |  !||           0
          |  !||           0
          |  !||           0
          |  !||           0
          |  !|+----------- SPORT1 1=serial port, 0=FI, FO, IRQ0, IRQ1,..
          |  !+------------ SPORT1 1=enabled, 0=disabled
          |  +============== SPORT0 1=enabled, 0=disabled
          +---------------- 0
                           0
                           0 */

      ifc = b#00000011111110;        /* clear pending interrupt */
      nop;

      icntl = b#00010;
      /*    ||||+- | IRQ0: 0=level, 1=edge
            |||+-- | IRQ1: 0=level, 1=edge
            ||+--- | IRQ2: 0=level, 1=edge
            |+---- 0
            |----- | IRQ nesting: 0=disabled, 1=enabled */

      mstat = b#1100000;
      /*    ||||||+- | Data register bank select
            |||||+-- | FFT bit reverse mode (DAG1)
            ||||+--- | ALU overflow latch mode, 1=sticky
            |||+---- | AR saturation mode, 1=saturate, 0=wrap
            ||+----- | MAC result, 0=fractional, 1=integer
            |+------ | timer enable
            +------- | GO MODE */

/* komunikacia s kodekom ADSP 1847 (programovanie kodeku) */

      /*   clear flag */
      ax0 = 1;
      dm(stat_flag) = ax0;

      /*   enable transmit interrupt */
      ena ints;
      imask = b#0001000001;
      /*    |||||||+ | timer
            ||||||+- | SPORT1 rec or IRQ0
            |||||+-- | SPORT1 trx or IRQ1
            ||||+--- | BDMA
            |||+---- | IRQE
            ||+----- | SPORT0 rec
            |+------ | SPORT0 trx
            +------- | IRQL0
                     | IRQL1
                     | IRQ2 */

      ax0 = dm (i3, m1);             /* start interrupt */
      tx0 = ax0;

check_init:
      ax0 = dm (stat_flag);          /* wait for entire init */
      af = pass ax0;                 /* buffer to be sent to */
      if ne jump check_init;         /* the codec            */

      ay0 = 2;

check_aci1:
      ax0 = dm (rx_buf);             /* once initialized, wait for codec */
      ar = ax0 and ay0;              /* to come out of autocalibration */
```

```
        if eq jump check_aci1;          /* wait for bit set */
check_aci2:
        ax0 = dm (rx_buf);              /* wait for bit clear */
        ar = ax0 and ay0;
        if ne jump check_aci2;
        idle;

        ay0 = 0xbf3f;                   /* unmute left DAC */
        ax0 = dm (init_cmds + 6);
        ar = ax0 AND ay0;
        dm (tx_buf) = ar;
        idle;

        ax0 = dm (init_cmds + 7);       /* unmute right DAC */
        ar = ax0 AND ay0;
        dm (tx_buf) = ar;
        idle;
        ifc = b#00000011111110;         /* clear any pending interrupt */
        nop;

                        imask = b#0001100001;       /* enable rx0 interrupt */
        /*    |||||||+ | timer
              ||||||+- | SPORT1 rec or IRQ0
              |||||+-- | SPORT1 trx or IRQ1
              ||||+--- | BDMA
              |||+---- | IRQE
              ||+----- | SPORT0 rec
              |+------ | SPORT0 trx
              +------- | IRQL0
                       | IRQL1
                       | IRQ2 */

/*   end codec initialization */
        rts;
```

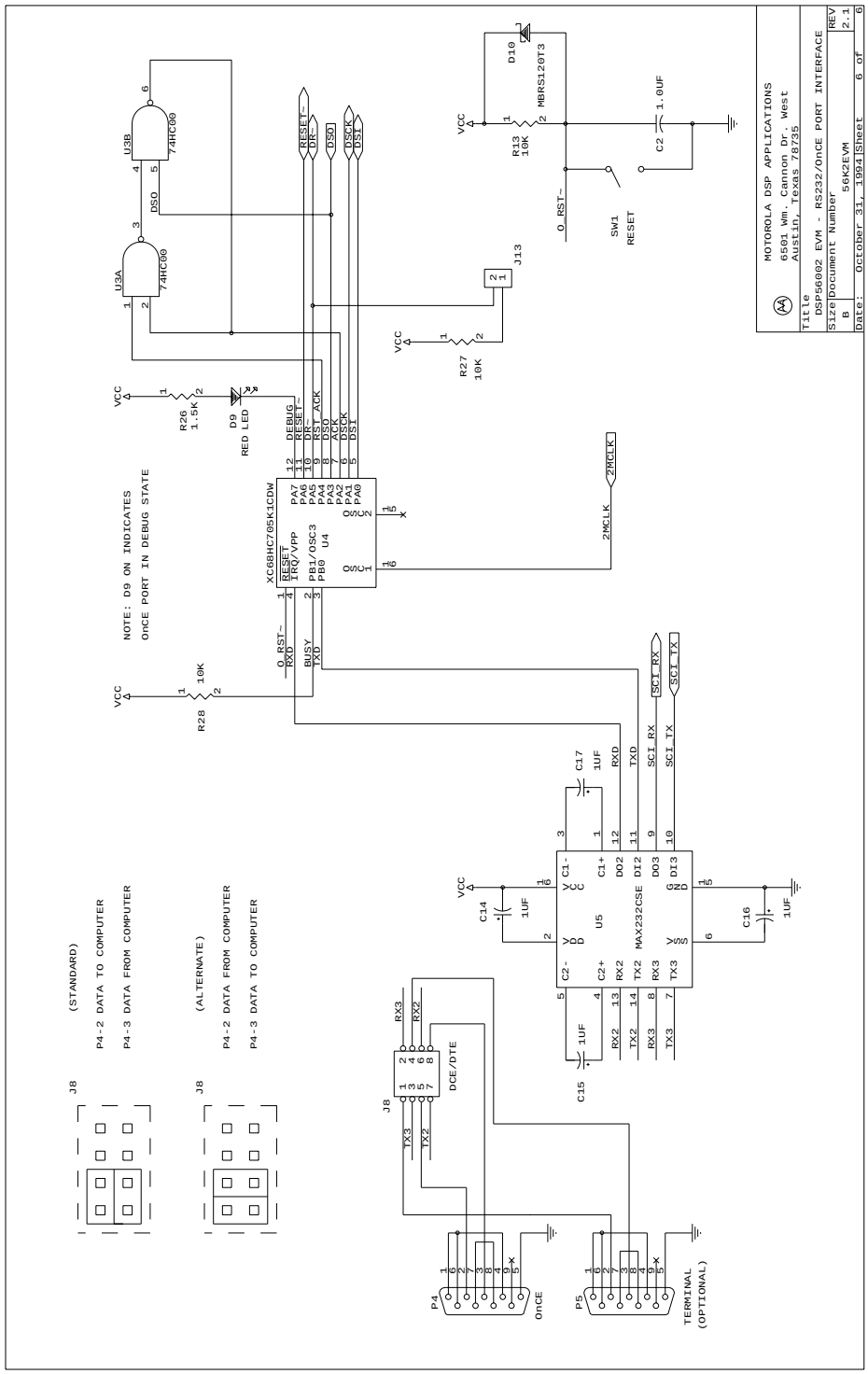
1.4 OnCE EMULÁCIA NA DOSKE MOTOROLA EVM56002

Umiestnenie hlavných prvkov a vývodov tejto vývojovej dosky je zobrazené na na obrázku 2 [4].



Obr.2 Vývojová doska Motorola EVM56002

Doska EVM56002 poskytuje podobné funkcie ako doska EZ-KIT2181 Lite, pričom naviac umožňuje využitie OnCE emulácie - t.j. využitie špecializovaných ladiacich obvodov, ktoré sú integrované v každom čipe DSP56002. Na ich využitie je potrebné k procesoru pripojiť ďalšie obvody, ktoré realizujú konverziu medzi príkazmi ovládacieho programu a hardverovým rozhraním. Tieto obvody sú zobrazené na obrázku 3 [4].



Obr.3 Obvody na využitie OnCE emulácie v procesoroch Motorola DSP56002

Na ovládanie dosky EVM (zavedenie programu, spustenie, krokovanie, prezeranie a modifikácia registrov, definovanie bodov zastavenia ...) je potrebné využiť program EVM56KW.EXE. Program má zabudovaný podobný **help systém** ako simulátor a po spustení programy **prehľadávajú rozhrania** COM a hľadajú pripojenú EVM dosku. Programy je možné spustiť aj v **demonštračnom režime** (zadaním prepínača **–d**). Základné príkazy a ovládanie programu bude demonštrované počas cvičenia.

1.5 VÝVOJOVÉ DOSKY PRE ADSP PROCESORY

Firma Analog Devices ponúka vývojové dosky pre všetky typy signálových procesorov. Ich aktuálny prehľad je možné nájsť na stránkach firmy [5] a niektoré základné informácie je možné nájsť aj na katedrových stránkach [6].

LITERATÚRA

[1] ADSP-2181 EZ-KIT Lite Evaluation System Manual. Analog Devices, Inc., January 2001, pp.1-40.

[2] Serial-Port 16-bit SoundPort Stereo Codec ADSP1847. Analog Devices, Inc. 1995.

[3] (dostupné v elektronickej forme **\SPvT\Cvicenia\ADDA_test.zip**)

[4] DSP56002 Evaluation Module Quick Start. DSP56002EVMUM/D, Rev.1.0, 3/1999 Motorola, Inc.

[5] www.analog.com

[6] www.kemt.fei.tuke.sk/adsp/

PRÍLOHA

V prílohe **EZ-KIT2181_Schemy.pdf** sú uvedené kompletné schémy vývojového modulu Analog Devices EZ-KIT2181 Lite [1]. Schémy sú rozdelené do 4 častí:

- DSP
- Kodek
- Analógová časť
- Konektory a napájacia časť