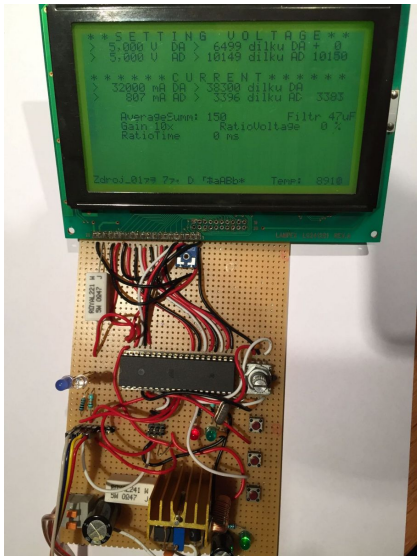
Laboratorní zdroj - 4. část

Publikované: 10.04.2016, Kategória: Silové časti

www.svetelektro.com

Komunikace po sériové lince a programování DA a AD převodníku

Aby bylo možné komunikovat s podřízeným procesorem na analogové desce laboratorního zdroje, připojil jsem k ní a naprogramoval provizorní řídící desku. Je osazena procesorem ATmega162 a alfanumerickým displejem s rozlišením 16 x 40 znaků. Displej je osazen řadičem HD61830. Rád takovou desku používám pro první oživování nových věcí. Na displej se vejde spousta informací o běžícím programu, takže není potřeba JTAG. Mezivýsledky výpočtů, které programuji, jsou pořád někde zobrazovány a já můžu kontrolovat jejich správnost. Nemusím se starat a do procesoru nahrávat obsáhlou znakovou sadu potřebnou pro řízení barevného displeje, který bude použit později, takže desítky přeprogramovávání při vývoji programu v AVR studiu jsou velmi rychlé. No a na univerzální desku si můžu připájet periferie, které zrovna potřebuji. Teď je na ní jenom stabilizátor, rotační kodér a sériová linka. V konečné verzi zdroje bude použita řídící deska s procesorem ATxmega128 a 4,3″ TFT displej.



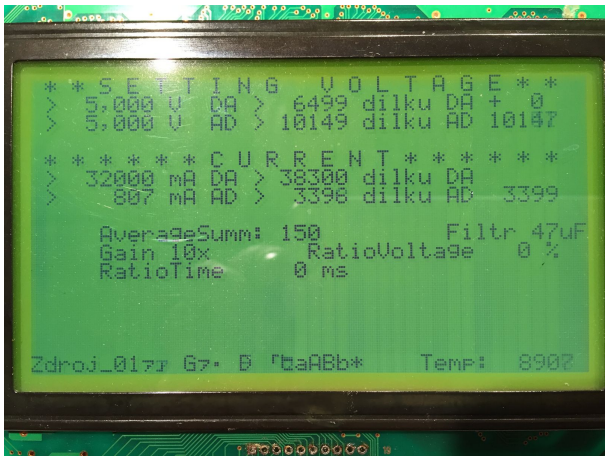
V hlavní smyčce programu, kterým ovládám laboratorní zdroj, se věnuji hlavně zobrazování na displeji:

```
if( RotStatus == 0 )    LCD_Write_String( 0, 0,  " * * S E T T I N G   V O L T A G E * *  " );
      else                LCD_Write_String( 0, 0,  " * * * * * * V O L T A G E * * * * * *  " );
      LCD_Write_String( 0, 1,  " > " );
      LCD_Write_Dec16_Point3( VoltageDaMilivolt );      // nastavene napeti pro DA - z rotacniho koderu
      LCD_Write_String( 9, 1,  " V  DA" );
      LCD_Write_String( 15, 1, " > " );
      LCD_Write_Dec16( VoltageDaOutput );               // vypocitane dilky pro DA prevod
      LCD_Write_String( 23, 1, " dilku DA ");
      LCD_Write_Signed_Dec8( VoltageDaMilivolt - VoltageAdMilivolt );
      LCD_Write_String( 0, 2,  " > " );
      LCD_Write_Dec16_Point3( VoltageAdMilivolt );      // vypocitane napeti z dilku AD
      LCD_Write_String( 9, 2,  " V  AD" );
      LCD_Write_String( 15, 2, " > " );
      LCD_Write_Dec16( VoltageAdInput );                // namerene dilky AD prevodniku napeti
      LCD_Write_String( 23, 2, " dilku AD ");
      LCD_Write_Dec16( VoltageAdInput1 );               // namerene dilky bez prumerovani !!!!!!!!!!!!!!!

      if( RotStatus == 1 )    LCD_Write_String( 0, 4,  " * * S E T T I N G   C U R R E N T * *  " );
      else                LCD_Write_String( 0, 4,  " * * * * * * C U R R E N T * * * * * *  " );
      LCD_Write_String( 0, 5,  " >  " );
      LCD_Write_Dec16( CurrentDaMiliamper );            // nastaveny proud pro DA - z rotacniho koderu
      LCD_Write_String( 9, 5,  " mA DA" );
      LCD_Write_String( 15, 5, " > " );
      LCD_Write_Dec16( CurrentDaOutput );               // vypocitane dilky pro DA prevod
      LCD_Write_String( 23, 5, " dilku DA ");
      LCD_Write_String( 0, 6,  " >  " );
      LCD_Write_Dec16( CurrentAdMiliamper );            // vypocitany proud z dilku AD
      LCD_Write_String( 9, 6,  " mA AD" );
      LCD_Write_String( 15, 6, " > " );
      LCD_Write_Dec16( CurrentAdInput );                // namerene dilky AD prevodniku proudu
      LCD_Write_String( 23, 6, " dilku AD ");
      LCD_Write_Dec16( CurrentAdInput1 );               // namerene dilky bez prumerovani !!!!!!!!!!!!!!!
```

Je potřeba zobrazit napětí nastavené na rotačním kodéru a z něj vypočítaný údaj odesílaný na DA převodník. Ten samozřejmě v budoucnu nebude potřeba, ale teď se hodí. Je potřeba hlídat, jestli dobře funguje rovnice, která hodnotu pro DA převodník počítá. Na dalším řádku je průměrná hodnota z AD převodníku a z něj vypočítané naměřené napětí v milivoltech. Na kraji displeje je ještě rozdíl nastavené a naměřené hodnoty a aktuální číslo z AD převodníku. Podobné údaje se opakují pro nastavení a měření proudu. Ve spodní části displeje jsou další věci, které zrovna programuji. No a na posledním řádku je výpis komunikace mezi procesory – zase pro jistotu, abych na první pohled viděl, že procesory mezi sebou komunikují.

Celé to pak vypadá takto:



Komunikace po sériové lince

Komunikaci bude řídit procesor ATmega162 na řídící desce. Podřízeným procesorem je ATmega16 na analogové desce zdroje. Na začátku komunikace Master vyšle nějaký textový řetězec, ze kterého je zřejmé, pro který procesor je informace určena a od koho se zároveň očekává odpověď. Já zvolil řetězec „Zdroj_01“. Za ním jde skupina dat:

- 16 bitů – 2 bajty: požadované napětí pro DA převodník
- 16 bitů – 2 bajty: požadovaný proud pro DA převodník
- 8 bitů – 1 bajt: nastavení relé, které jsou na desce zdroje
- 8 bitů – 1 bajt: počet měření na AD převodníku, která budou průměrována
- 16 bitů – 2 bajty: záloha
-

Slave procesor ví, že má přijmout 16 bajtů. Přitom kontroluje textový řetězec, aby bylo možno rozhodnout, že je informace pro něj. Pokud ano, tak přijme všechna data do pole dat **SerialRxArray**. Z něj budou údaje použity v dalších částech programu. Po přijetí posledního znaku začne slave procesor okamžitě vysílat svoji zprávu pro Master. Začíná stejně, řetězcem „Zdroj_01“ a pokračuje skupinou dat:

- 16 bitů – 2 bajty: aktuální naměřené napětí
- 16 bitů – 2 bajty: aktuální naměřený proud
- 16 bitů – 2 bajty: průměrné naměřené napětí
- 16 bitů – 2 bajty: průměrný naměřený proud
- 16 bitů – 2 bajty: naměřené napětí z teplotního čidla integrovaného v referenčním zdroji
- 8 bitů – 1 bajt: stav zdroje – zdroj napětí nebo proudu
- 40 bitů – 5 bajtů: záloha
-

Master data přijme a uloží do svého pole přijatých dat **SerialRxArray**. Vysílání z procesoru Master je řízeno podprogramem přerušení od časovače. Ten spouští vysílání zprávy přibližně 5x za sekundu. Na logickém analyzátoru to vypadá takto:

Program přerušení od časovače odešle jednou za 200ms první písmeno řetězce. Po jeho odeslání vznikne požadavek na přerušení od vysílače sériového kanálu, který odesílá zbytek.

```
ISR(TIMER1_CAPT_vect)
{
   SREG = 0x00;                    // zastavi globalni preruseni

   if ( TimCounter == 200 )        // pocita interval 0,2 sekunda k odeslani dat na seriovy kanal
   {
      TimCounter = 0;              // pocitadlo milisekund v casovaci
      SerialString = "Zdroj_01";
      while ( !( UCSR0A & (1<<UDRE0)));
      UDR0 = (*SerialString);      // zacatek odesilani
      SerialString++;
      SerialTxCounter = 0;
   }
   TimCounter++;
   SREG = 0x80;                    // pusti globalni preruseni
}
```

Začátek programu přerušení aktivovaný při dokončení odesílání znaku na sériové lince:

```
ISR(USART0_TXC_vect)
{
   if (*SerialString != '')    // hleda konec retezce
   {
      UDR0 = *SerialString;     // posle dalsi pismeno
      SerialString++;
   }
   else
   {
      switch (SerialTxCounter)
      {
         case 0:               // horni polovina DA prevodniku napeti
         {
            UDR0 = (unsigned char) (VoltageDaOutput >> 8);
            break;
         }
         case 1:               // spodni polovina DA prevodniku napeti
         {
            UDR0 = (unsigned char) (VoltageDaOutput);
            break;
         }
         case 2:               // horni polovina DA prevodniku proudu
         {
            UDR0 = (unsigned char) (CurrentDaOutput >> 8);
            break;
         }
         case 3:               // spodni polovina DA prevodniku proudu
         {
            UDR0 = (unsigned char) (CurrentDaOutput);
            break;
         }
```

Program přerušení aktivovaný při dokončení přijímání znaku na sériové lince:

```
ISR(USART0_RXC_vect)
{
   LED_G_Set;
   SerialRxArray[SerialRxCounter] = UDR0;
   switch (SerialRxCounter)
   {
      case 0:            // prijima znaky do pole SerialRxArray, prvni tri znaky kontroluje
      {
         if(SerialRxArray[0] == 'Z') SerialRxCounter++;
         break;
      }
      case 1:
      {
         if(SerialRxArray[1] == 'd') SerialRxCounter++;
         else
         SerialRxCounter = 0;
         break;
      }
      case 2:
      {
         if(SerialRxArray[2] == 'r') SerialRxCounter++;
         else
         SerialRxCounter = 0;
         break;
      }
      case 23:        // posledni prijaty znak
      {
         SerialRxCounter = 0;
         LED_G_Clr;
         break;
      }
      default:
      {
         SerialRxCounter++;
         break;
      }
   }
}
```

Nakonec ještě nastavení registrů mikroprocesoru:

```
// * * * * * * * * * * * * * * * * * * * * * * * * Nastaveni preruseni
GICR  = 0b10000000;        // povoleni preruseni INT1
MCUCR = 0b00001010;        // aktivace preruseni sestupnou hranou

// * * * * * * * * * * * * * * * * * * * * * * * * Nastaveni casovace T1
TCCR1A = 0b00000010;    // Frekvence = F krystal
TCCR1B = 0b00011001;    // rezim PWM 14
TIMSK  = 0b00001000;    // preruseni pri preteceni
ICR1 =     8000;            // 1000 Hz kmitocet PWM

// * * * * * * * * * * * * * * * * * * * * * * * * Nastaveni serioveho kanalu
UBRR0H = 0;
UBRR0L = 51;            // rychlost prenosu 9600 baudu
UCSR0B = 0b11011000;    // povoleni vysilace a prijimace, preruseni pri vysilani i pri prijmu
UCSR0C = 0b10001110;    // bez parity, osum bitu
```

Procesor ATmega16 na analogové desce laboratorního zdroje má zatím tyto úkoly:

- Po dvou sekundách od zapnutí připojit relé, které zkratuje 10Ω rezistor používaný pro pomalé nabíjení filtračního kondenzátoru. Je použita dvojice 10mF kondenzátorů.
- Přijímat data z Master procesoru, vyhodnotit zda jsou určena pro analogovou desku a odpovědět vysláním zprávy po sériové lince. To je řešeno funkcemi přerušení od sériového kanálu.
- Nastavovat DA převodníky podle přijatých dat, periodicky v hlavním programu.
- V hlavním programu periodicky měřit pomocí AD převodníků, získané údaje ukládat do pole a počítat průměrné hodnoty.
- Nastavovat relé přepínání vinutí transformátoru, když je požadováno větší napětí.
- Nastavovat relé přepínání filtrů na výstupních svorkách a relé, které ovládá zesilovač napětí z bočníku pro měření proudu.

DA převodník DAC8563

16-ti bitový převodník má dva výstupní kanály, jeden je použit k nastavení napětí, druhý pro nastavení maximálního proudu. Obvod komunikuje po SPI lince procesoru. Takže jsem odněkud opsal jednoduchou funkci pro ovládání SPI:

```
unsigned char SPI_Send(unsigned char val)
{
   SPDR = val;                    // odesilana data
   while(!(SPSR & (1<<SPIF)));
   return SPDR;                   // prijata data
}
```

Tuto funkci jsem použil v programu, který ovládá DA převodník:

```
void DAC8563_Send(unsigned char command, unsigned int val)
{
   SPCR  = 0b01010100;    // zmena dat na vzestupnou hranu CLK, cteni na sestupnou hranu
   DA_SYNC_clr;
   SPI_Send( command );   // data pro rizeni prevodniku
   SPI_Send( val >> 8 );  // horni bajt 16-ti bitoveho prevodniku
   SPI_Send( val );       // spodni bajt 16-ti bitoveho prevodniku
   DA_SYNC_set;
}
```

Funkce je pravidelně spouštěna v hlavní smyčce programu. Příkaz pro řízení převodníku má hodnotu 0b00011001 pro kanál napětí a hodnotu 0b00011000 pro kanál proudu.

Před zahájením práce, někde na začátku programu je potřeba resetovat a nastavit převodník:

```
DAC8563_Send( 0b00101000, 1 ); // Reset
DAC8563_Send( 0b00111000, 0 ); // Disable internal reference
DAC8563_Send( 0b00100000, 3 ); // power up DAC-A and DAC-B
DAC8563_Send( 0b00000010, 0 ); // Gain = 2
DAC8563_Send( 0b00110000, 3 ); // LDAC pin inaktive for DAC-A and DAC-B
```

AD převodník ADS8341

I tento obvod pro komunikaci s procesorem používá SPI linku. Akorát data jdou na sestupnou hranu signálu CLK, narozdíl od DA převodníku. Trvalo mi čtyři hodiny, než jsem na to přišel a na začátku funkcí ovládajících převodníky změnil stav registru SPCR, který ovládá SPI linku procesoru. Řídící bity převodníku jsou nastaveny tak, aby byl využíván interní oscilátor (PD1=1, PD0=0). Tomuto nastavení odpovídá polarita sledování signálu BUSY v programu.

Referenční napětí převodníku je 2,5V. Převodník tedy měří v rozsahu -2,5V až +2,5V vůči pinu COM. Když je pin COM připojen na referenční napětí, posune se rozsah měřeného napětí o 2,5V nahoru. Takže rozsah měření bude 0 – 5V s rozlišením 16 bitů.

```
unsigned int ADS8341_Send(unsigned char command)
{
   unsigned int AdVal;

   SPCR  = 0b01011100;    //zmena dat na sestupnou hranu CLK, cteni na vzestupnou hranu
   AD_CS_set;
   SPI_Send( command );
   while( PINB&0x02 );    // ceka na signal BUSY
   AD_CS_clr;
   while(~PINB&0x02 );
   AD_CS_set;
   AdVal =          SPI_Send( 0 ) << 8;
   AdVal = AdVal + (SPI_Send( 0 ));
   SPI_Send( 0 );
   AdVal = AdVal - 32768;
   AD_CS_clr;
   return ( AdVal );
}
```

Příkaz pro řízení AD převodníku může mít tyto hodnoty:

0b10010110 – kanál 0: měří napětí z bočníku, zobrazuje proud

0b11010110 – kanál 1: měří napětí z děliče, zobrazuje napětí

0b11100110 – kanál 3: referenční zdroj REF5025 má na jeden z pinů přiveden analogový teploměr.

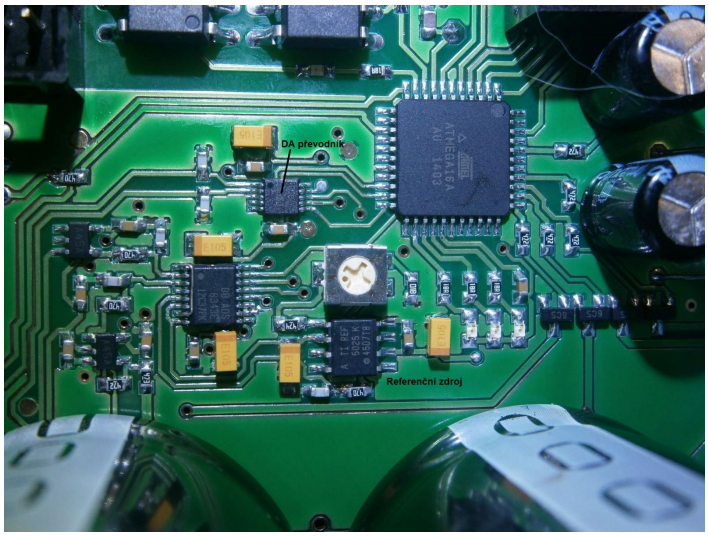
Výpočet průměrné hodnoty z naměřených dat

Hodnoty naměřené AD převodníkem jsou zatíženy šumem. Proto je nutné, aby program počítal průměrnou hodnotu z několika měření.

První nápad byl, uložit do pole dat např. 100 naměřených hodnot, ty sečíst a součet vydělit 100. Problém je, že změna mezi dvěma takto získanými průměry může být hodně velká. Chtělo by to vymyslet způsob, jak průměr měnit plynule, po každém měření. Po každém měření přičteme novou hodnotu k součtu VoltageAdd, nebo CurrentAdd. Zároveň je potřeba odečíst hodnotu z druhého konce pole dat. Periodicky spouštěný program pro měření a výpočet průměru je tady:

```
SREG = 0x00;
VoltageActual = ADS8341_Send( AdVoltage );      // data z AD prevodniku
CurrentActual = ADS8341_Send( AdCurrent );
SREG = 0x80;
VoltageArray[Pointer] = VoltageActual;          // ulozeny do pole
CurrentArray[Pointer] = CurrentActual;
VoltageAdd += VoltageArray[Pointer];            // pricte namerene k prumeru
CurrentAdd += CurrentArray[Pointer];
if( Pointer == MaxPointer ) Pointer = 0;        // posune ukazatel
else Pointer++;
VoltageAdd -= VoltageArray[Pointer];            // odecte data z druheho konce pole
CurrentAdd -= CurrentArray[Pointer];
VoltageAverage = (unsigned int) ( VoltageAdd / MaxPointer );
CurrentAverage = (unsigned int) ( CurrentAdd / MaxPointer );
```

Proměnné jsou deklarovány jako 16-ti bitové, jenom součty jsou 32-ti bitové *long int*.



Na obrázku je procesor, převodníky, referenční zdroj. Trimr slouží k nastavení referenčního napětí 2,5V. V horní části obrázku jsou optočleny, které oddělují sériovou linku.

Autori článku: František OK2JNJ a Michal OK2HAZ