Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

7.7 Program 7 – voltmetr

V předchozích programech jsme programovali obsluhu externích periferií jako jsou tlačítka, klávesnice či LCD displej nebo diody LED. Jedinou vnitřní periferií, kterou jsme zatím programově obsluhovali, byl UART v *programu3*. Nyní si ukážeme jak programově obsloužit A/D převodník, který je vnitřní periferií mikrokontroléru AT90S8535. Jeho činnost je ovládána zápisem do jeho registru **ADCSR**. Činnost tohoto převodníku je povolena nastavením nejvýznamnějšího bitu (bit 7) tohoto registru na 1. Pro tento, sedmý bit je používáno označení **ADEN**. Další, nižší bit (bit 6) je označován **ADSC**, jeho nastavením na 1 je nastartován převod A/D. Po dokončení tohoto převodu bude hodnota tohoto bitu rovna 0. Pokud probíhá převod A/D, nemá případné zapsání nuly to tohoto bitu nějaký vliv na převod. Nastavení bitu 3, nazývaného **ADIE**, na jedničku znamená povolení (aktivaci) přerušení **ADC Conversion Complete**. K tomuto přerušení dojde po dokončení A/D převodu. Nejnižší tři bity registru ADCSR určují dělicí poměr předděličky, kterým je dělen kmitočet krystalu mikrokontroléru. Výstupní signál z předděličky je přiveden na hodinový vstup A/D převodníku. Po dokončení A/D převodu je získaná naměřená digitální hodnota uložena v registrech ADCL a ADCH převodníku A/D. Obsah obou registrů vytváří obsah ADCW. Ten je již přímo úměrný měřenému napětí. V našem případě měříme napětí v rozsahu 0 až 5 V. Protože A/D převodník v AT90S8535 je 10bitový, a $2^{10} = 1024$, odpovídá toto číslo 5 V, tj. 5000 mV. Proto musíme naměřený údaj v ADCW vynásobit konstantou $5000/1024 \equiv 4{,}88$. Tomu odpovídá příkaz

```
napeti = (int)(ADCW*4.888);
```

příkazy

```
if (napeti <1000) lcd_gotoxy(1,1);
if (napeti <100) lcd_gotoxy(2,1);
if (napeti <10) lcd_gotoxy(3,1);
```

pak zajišťují umístění naměřené hodnoty na pozici LCD nezávisle na počtu cifer naměřeného napětí – zarovnání doprava. Zdrojový kód voltmetru je velice jednoduchý:

```
/*
na PORTC je připojen LCD displej
nastavení :
- VTarget=5.0V
- ARef=5.0V
- Oscillator=8 MHz

SS měřené napětí 0 až 5V se přes odpor cca 1k připojí
+ na PORTA pin PA0 a - na GND pin
*/
```

```
// I/O definice registru pro AT90S8535
#include <90s8535.h>
#include <delay.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm

#include <lcd.h>
#include <stdlib.h>

#define ADC_VREF_TYPE 0x00

unsigned char pom[5];
unsigned int napeti;

// obsluha přerušení pro ADC
interrupt [ADC_INT] void adc_isr(void)
{
lcd_gotoxy(0,1);
lcd_putsf(" ");
napeti = (int)(ADCW*4.888); // 0.005 kvůli zaokrouhlení
itoa(napeti,pom);
lcd_gotoxy(0,1);
if (napeti <1000) lcd_gotoxy(1,1);
if (napeti <100) lcd_gotoxy(2,1);
if (napeti <10) lcd_gotoxy(3,1);
lcd_puts(pom);

// 20 ms delay
delay_ms(20);
// začátek nového převodu AD
ADCSR|=0x40;
}

void main(void)
{

lcd_init(16);
lcd_gotoxy(0,0);
lcd_putsf("Voltmetr demo");
// ADC inicializace

// ADC Interrupts: On
ADCSR=0x8E;
```

```
// povolení přerušení
#asm("sei")

// výběr vstupu ADC = vstup 0
ADMUX=0;

// začátek prvního převodu, jeho dokončení vyvolá přerušení ADC
ADCSR|=0x40;

// veškerou činnost zajišťuje obsluha přerušení ADC
while (1);
}
```

Kód programu je záměrně velice jednoduchý, stejně jako kód ostatních programů v této knize. Má se tím usnadnit pochopení jeho funkce začátečníkům. Může se však stát základem složitějších projektů doplněných např. o komfortnější obsluhu voltmetru, jeho propojení s PC pomocí RS232 umožňující zpracování naměřených hodnot na osobním počítači. Dokonce lze takto realizovat i osciloskop. Příkladem může být konstrukce A. Jelisejeva na www.telesys.ru včetně zdrojového kódu pro AT90S8535 v IAR C a zdrojového kódu pro PC v Delphi (najdeme i na doprovodném CD v adresáři nápady).

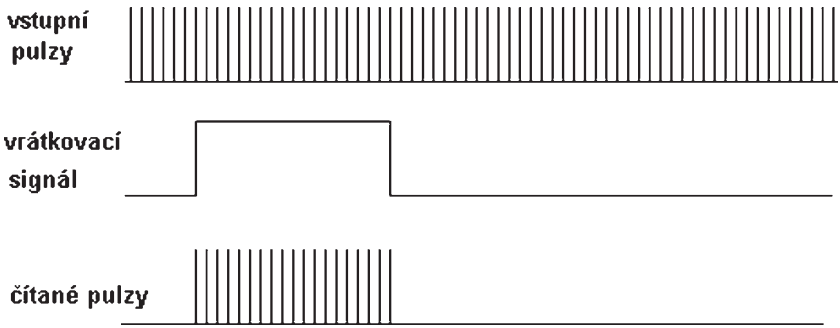
7.8 Program 8 – čítač

Již více než 30 let patří mezi oblíbené konstrukce měřič kmitočtu – čítač. V prvních desetiletích realizovaný zcela hardwarově, číslicovými obvody TTL, později i CMOS či ECL. Nyní již zcela převládají konstrukce čítačů realizovaných jednočipovými mikrokontroléry či mikropočítači, v našich poměrech často s PIC16F84 či jednočipem kompatibilním s x51. V poslední době se objevují i konstrukce s MCU AVR. Bohužel autoři těchto konstrukcí často uveřejňují jen schéma zapojení a obrazec plošných spojů. Pokud jde o program, nabízejí za úplatu prodej naprogramovaného mikrokontroléru.

Existují však i výjimky, viz např. kniha [17 str. 159–178]. Popisující čítač s AT90S2313, včetně zdrojového kódu v assembleru, propojený přes RS232 s PC, na němž pod Windows běží program sloužící jednak jako displej čítače, jednak k ovládání čítače. Výhoda znalosti zdrojového kódu je zřejmá – můžeme si zkontrolovat, že v programu není chyba a můžeme ho dále upravovat. Pro takové účely je výhodné mít zdrojový kód ve vyšším programovacím jazyce. Ukážeme si jednoduchý program v CodeVisionAVR C. Nejprve si však vysvětlíme použitou metodu měření. Tou bude metoda přímá. Její princip je zřejmý z *obr. 7.27*.

Měřený signál prochází zesilovačem/tvarovačem, který vstupní signál přemění na pulzy, které jsou jedním ze vstupních signálů vrátkovacího obvodu. Jeho druhým vstupem je vrátkovací signál, který slouží k řízení vrátkovacího obvodu – určuje kdy

propustí vstupní pulzy na výstup. Bude-li doba, po kterou propouští vstupní pulzy rovna např. 1 s, bude počet pulzů propuštěných na výstup vrátkovacího obvodu roven již kmitočtu vstupního signálu v Hz, v případě délky vrátkovacího signálu 1 ms bude počet pulzů na výstupu vrátkovacího obvodu odpovídat kmitočtu v kHz atd. Proto stačí pulzy na výstupu vrátkovacího obvodu čítat a po skončení čítání zobrazit na displeji, který tak bude zobrazovat kmitočet měřeného signálu. Při klasické realizaci měřiče kmitočtu číslicovými obvody může být vrátkovacím obvodem např. hradlo AND, pulzy jsou pak čítány dekadickými čítači. Po skončení čítání se stav čítačů přenese do nějaké vyrovnávací paměti a poté se stav čítačů vynuluje. Obsah vyrovnávací paměti se přes převodník kódu přenese na displej.



Obr. 7.27 Průběhy pulzů v měřiči kmitočtu pracující přímou metodou

Nyní si ukážeme jak realizovat měřič kmitočtu mikrokontroléru ATMEL AT90S8515. K vrátkování využijeme **čítač/časovač0**, který je součástí mikrokontroléru. Nastavením `TCCR0=0x05;` docílíme nastavení dělicího poměru předděličky 1024. Na jejím vstupu budou hodinové pulzy z krystalového oscilátoru mikrokontroléru o kmitočtu 8 MHz, takže čítač/časovač0 bude čítat pulzy o kmitočtu 8000000/1024 = 7812,5 Hz. Protože tento čítač je 8bitový, bude po načítání 256 pulzů docházet k jeho přetečení, což vyvolá přerušení. Obsluha tohoto přerušení `interrupt [TIM0_OVF] void timer0_ovf_isr(void)` softwarově realizuje vrátkovací obvod s kmitočtem vrátkování 7812,5/1024 což je cca 30,5 Hz. Proto po skončení vrátkování nebude stav čítače čítající vstupní, měřený signál odpovídat přímo kmitočtu v Hz, či jeho dekadických násobcích. Bude však přímo úměrný kmitočtu a proto k získání správného kmitočtu můžeme vypočítat pomocí konstanty přímé úměrnosti. Získáme ji jako převratnou hodnotu kmitočtu vrátkování, tj. přibližně 1/30,5. Přesná hodnota konstanty úměrnosti je 0,032768. Předpokládá ovšem, že hodinový kmitočet mikrokontroléru je nastaven přesně na 8 MHz. Pokud použijeme běžný krystal 8 MHz připojený k oscilátoru, který je součástí MCU dostaneme obvykle kmitočet o něco odlišný od 8 MHz. Místo pracného nastavování kmitočtu tohoto oscilátoru na přesnou hodnotu je snadnější změnit konstantu přímé úměrnosti.

K čítání pulzů odvozených od měřeného signálu použijeme 16bitový **čítač/časovač1**, který je rovněž součástí AT90S8515. Měřený signál, zpracovaný vstupním zesilovačem/tvarovačem, je přiveden na vstup T1 AT90S8515. Ten je v inicializační části programu nastaven tak, že **čítač/časovač1** bude měnit svůj stav s náběžnou hranou signálu na T1. Stav tohoto čítače, tj. údaj o počtu načtených pulzů je součástí registrů `TCNT1L` a `TCNT1H`. Proto při obsluze přerušení **čítače/časovače0** nejprve překopírujeme obsah těchto registrů do pomocných proměnných

```
poc1 = TCNT1L;
poc2 = TCNT1H;
```

a poté tyto registry vynulujeme. Před opuštěním obslužné rutiny příkazem `TCCR1B=0x07;` spustíme čítání měřeného signálu. Protože k dalšímu přerušení dojde za přibližně 1/30,5 sec, budou globální proměnné `poc1` a `poc2` obsahovat hodnoty závislé na měřeném kmitočtu. Příkazem

```
pocitadlo = poc1 + 256*poc2;
```

umístíme v proměnné `počitadlo` hodnotu již přímo úměrnou měřenému kmitočtu. Přesnou hodnotu kmitočtu poskytne příkaz `počitadlo = pocitadlo * 0.03053;` Proměnná `pocitadlo` je globální, takže je přístupná příkazům v nekonečné smyčce `while (1)`, zabezpečujícím zobrazování hodnot naměřeného kmitočtu na LCD připojeném k portu A.

Protože **čítač/časovač1** je 16bitový, tj. může načíst max. 65536 pulzů a vrátkovací kmitočet je cca 30,5 Hz, je maximální měřený kmitočet 65536 × 30,5 tj. cca 1,998 MHz. Výsledný zdrojový kód měřiče kmitočtu do cca **2 MHz** je:

```
MERIC KMITOCTU do 2,0 MHz 16bitovy

Chip type           : AT90S8515
Clock frequency     : 8,000000 MHz
Memory model        : Small
Internal SRAM size  : 512
External SRAM size  : 0
Data Stack size     : 128
*****************************************/

#include <90s8515.h>
#include <delay.h>
#include <stdlib.h>

// LCD displej na PORT A
#asm
.equ __lcd_port=0x1B
#endasm
```

```
#include <lcd.h>        // knihovna pro LCD

float pocitadlo=0;
unsigned int poc1 = 0;
unsigned int poc2 = 0;

unsigned char pom[8];

// obsluha prerušení - přetečení Timer 0
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
#asm("cli")         //zákaz přerušení
TCCR1B=0x00;        //konec čítání měřeného signálu
TCCR0=0x00;         //konec čítání vrátkovacího obvodu

poc1 = TCNT1L;      //přečtení hodnoty čítače TIMER1
poc2 = TCNT1H;

TCNT1H = 0;         //vynulovani citace TIMER1
TCNT1L = 0;
TCNT0=0x00;
TCCR1B=0x07;        //začátek čítání měřeného signálu na T1
TCCR0=0x05;         //dělení kmitočtu hodin 1024 ma
#asm("clv")         //vynulování příznaku přetečení
#asm("sei")         //povoleni přerušení
//
}

void main(void)
{

// Port B initialization - B je vstup
PORTB=0x00;
DDRB=0x00;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x05;
TCNT0=0x00;
// Timer/Counter 1 initialization
// Clock source: T1 pin Rising Edge
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
```

```
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x07;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x02;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;

// LCD module initialization
lcd_init(16);
lcd_gotoxy(0,0);
lcd_putsf("meric kmitoctu ");
// Global enable interrupts
#asm("sei")

while  (1)
       {
       lcd_gotoxy(0,1);
       lcd_putsf(" ");
       pocitadlo = poc1 + 256*poc2;   //poc1 dolních 8bitů
                                      //poc2 horních 8bitů
                                  //16bitoveho citace TIMER1
       //konstanta sloužící k nastaveni přesného kmitočtu
       pocitadlo = pocitadlo * 0.03053;
       if (pocitadlo<10) lcd_gotoxy(5,1);
```

```
        if ((pocitadlo<100)&(pocitadlo>9.99)) lcd_gotoxy(4,1);
        if ((pocitadlo<1000)&(pocitadlo>99.99)) lcd_gotoxy(3,1);
        if ((pocitadlo<10000)&(pocitadlo>999.99)) lcd_gotoxy(2,1);
        if ((pocitadlo<100000)&(pocitadlo>9999.99))
        lcd_gotoxy(1,1);
        if ((pocitadlo<1000000)&(pocitadlo>99999.99))
        lcd_gotoxy(0,1);
        ftoa(pocitadlo,1,pom);
        lcd_puts(pom);
        lcd_gotoxy(9,1);
        lcd_putsf("kHz ");
        delay_ms(200);

        };
}
```

Náš měřič kmitočtu měří do 2 MHz. Čítače mikrokontrolérů AVR však mohou měřit kmitočet až do poloviny kmitočtu hodin, v případě, kdy měřené pulzy jsou synchronní s kmitočtem hodin. Jinak je maximální kmitočet o něco nižší. V našem případě by se tedy dal měřit kmitočet o něco nižší než 4 MHz. Bylo by však nutné buď zvýšit kmitočet vrátkování nebo zvýšit počet bitů čítače měřeného signálu. Protože další vestavěný čítač/časovač již nemáme, můžeme počet bitů tohoto čítače zvýšit již jen softwarově. To lze velice snadno. Jeho zdrojový kód, v CD příloze označený jako *Program8B* se od výše uvedeného zdrojového kódu liší jen tím, že obsahuje další dvě globální proměnné `pomocny` a `poc3` a obsluhu přerušení vyvolaného přetečením **čítače/časovače1**, ke kterému dochází při překročení cca 2000000 pulzů/sec do **čítače/časovače1**. Při obsluze tohoto přetečení

```
interrupt [TIM1_OVF] void timer1_overflow(void) {
pomocny = pomocny + 1;
#asm("clv")
}
```

se zvýší o 1 pomocná proměnná, a tím při obsluze přerušení **čítače/časovače0** i hodnotu pomocného sw čítače `pom3`. Obsluha přerušení **čítače/časovače0** obsahuje proti výše uvedenému kódu ještě příkazy

```
poc3 = pomocny;
pomocny = 0;
```

Poslední změnou je výpočet kmitočtu v nekonečné smyčce `while (1)`, který nyní bude

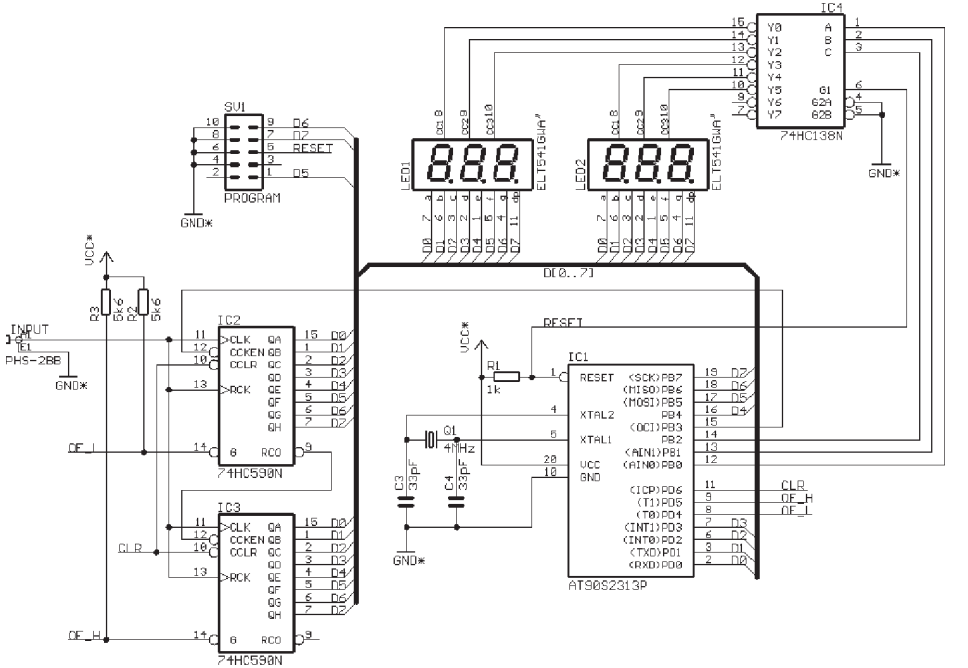
```
pocitadlo = poc1 + 256*poc2
pocitadlo = pocitadlo * 0.03053 + poc3*2000;
```

Od výpočtu v *Program8A* se liší tím, že při kmitočtech nad 2 MHz se ke kmitočtu zjištěnému pomocí 16bitového čítače přidávají právě 2 MHz. Pro nižší kmitočty bude totiž `poc3` obsahovat 0, při překročení 2 MHz bude obsahovat 1. Větší číslo než 1 nebude obsahovat, protože **čítač/časovač1** má mezní kmitočet o něco nižší než 1 kmitočet hodin MCU, tj. 4 MHz.

7.8a Program 8 – měřiče kmitočtu

Pokud chceme měřit kmitočty vyšší, než je mezní kmitočet vnitřních čítačů/časovačů MCU, musíme použít vnější čítače s vyšším mezním kmitočtem. Jednou z možností je použít předdělička a tak získat pulzy s kmitočtem, který již náš čítač zvládá a údaj měřený našim AVR čítačem násobit dělicím poměrem předděličky. Tento způsob najdeme např. v již zmiňované knížce D. Matouška [17]. Jinou možností je použít vnější čítače o vyšším mezním kmitočtu přímo k čítání měřených pulzů a AVR MCU použít ke zpracování údaje o stavu těchto čítačů získaného na jejich výstupech a přivedených v paralelním tvaru na porty AVR MCU. To je případ konstrukce, kterou jsem objevil někde na internetu. Autorem je Jesper Hansen, čítač měří do 50 MHz a výsledek zobrazuje na 7segmentovém LED displeji *obr. 7.28*.

Autor uvedl i zdrojový kód v GCC.



Obr. 7.28 Zapojení měřiče kmitočtu do 50 MHz se číslovkami LED podle Jespera Hansena

7.9 Program 9 – hodiny

V předchozím příkladě jsme využili vnitřní čítače/časovače ATMEL AT90S8515 jako základ měřiče kmitočtu. Počítali jsme přitom pulzy měřeného signálu po dobu otevření vrátkovacího obvodu, která při kmitočtu krystalu 8 MHz odpovídá kmitočtu (8000000/1024)/256 tj. přibližně 30,5 Hz. Počet načítaných pulzů jsme potom násobili konstantou, abychom dostali hodnotu odpovídající měřenému kmitočtu. V řadě případů je výhodnější získat z vnitřních čítačů MCU přímo kmitočet, který je nějakým násobkem deseti. V tom případě provádíme přednastavení čítačů tak, aby k jejich přetečení docházelo za požadovanou dobu. Ukážeme si to na příkladě hodin, kdy budeme potřebovat čítat „sekundové tiky". Opět použijeme AT90S8515 s krystalem 8 MHz a k jejich vytvoření použijeme **čítač/časovač1**. Nastavením TCCR1B=0x04 docílíme toho, že tento čítač bude čítat hodinové pulzy z předděličky 256, tj. na jeho vstupu bude kmitočet 8 000 000/256 = 31 250 Hz.

K získání kmitočtu 1 Hz tedy potřebujeme, aby k přetečení čítače/časovače1 došlo vždy po 31 250 pulzech.

Protože však je tento čítač 16bitový, potřebuje k přetečení 65536 pulzů. Proto ho musíme přednastavit na 65 536 – 31 250 = 34 286. Protože 34 286 = 133 × 256 + 238 musí být TCNT1H=133 a TCNT1L=238 na začátku čítání. K volání obsluhy přerušení vyvolané přetečením **čítače/časovače1** pak bude docházet vždy za 1 sekundu. Součástí této obsluhy přerušení bude zvýšení obsahu proměnné obsahující údaj o počtu sec o jedničku. Bude-li však výsledek roven 60, musí se o 1 zvýšit počet minut a údaj o počtu sekund nastavit na nulu. Obdobně v případě dosažení 60 minut či 24 hodin apod. Pro snazší udržování kódu je proto výhodnější vytvořit vlastní typ, kterým je v našem kódu struktura time se složkami second, minute, hour, date, month a year jejichž obsahem bude úplný časový údaj obsahující rok, měsíc, den, hodina, minuta a sekunda. Součástí obsluhy přerušení vyvolané přetečením **čítače/časovače1** každou sekundu bude tedy zvýšení úplného časového údaje o tuto jednu sekundu, tj. nejen respektování toho, že minuta má 60 sekund, hodina 60 minut, den 24 hodin, ale i počet dnů v jednotlivých měsících včetně uvažování přestupných roků. Program ještě musí obsahovat kód zabezpečující zobrazení času např. na LCD displeji a dále vyřešit nějak nastavení či opravu zobrazovaného údaje. Po spuštění programu se totiž čas měří od 0 sekund, 0 minut, 0 hodin atd. Pro snazší pochopení kódu jsem základní kód obsahující čítání času po 1 sekundě doplnil o zobrazování jen hodin a minut na LCD připojeném k portu A a šesti tlačítek připojených k portu C a sloužících k opravě zobrazovaného údaje. Funkce těchto šesti tlačítek je zvýšení údaje o minutách o 1, snížení údaje o minutách o 1, zvýšení údaje o minutách o 10 minut, snížení údaje o minutách o 10 minut, zvýšení údaje o hodinách o 1 a snížení údaje o hodinách o 1. Součástí kódu reagujícího na stisk tlačítka zvýšením či snížením hodnoty hodin či minut je i respektování toho, že minut je nejvýše 60 a hodin nejvýše 24. Výsledný zdrojový kód bude:

```
#include <90S8515.h>
#include <delay.h>
#include <stdlib.h>

// LCD displej na PORT A
#asm
   .equ __lcd_port=0x1B
#endasm
#include <lcd.h> // knihovna pro LCD

unsigned int poc = 0;
unsigned char pom[8];
char not_leap(void);

typedef struct{
unsigned char second;
signed char minute;
signed char hour;
unsigned char date;
unsigned char month;
unsigned int year;
            }time;

   time t;

void main(void)
{
   PORTC=0xFF;
   DDRC=0x00;
   DDRB=0xFF;

// inicializace
TCCR1A=0x00;
TCCR1B=0x04; //zdrojem clk jsou vnitřní hodiny 8MHz vydělené 256
TCNT1H=133;
TCNT1L=238;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
GIMSK=0x00;
MCUCR=0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x80; //

    //prescale the timer to be clock source / 128 to make it
  #asm("sei")
    //set the Global Interrupt Enable Bit
  lcd_init(16);
  lcd_gotoxy(0,0);
  lcd_putsf("hodiny ");
  while(1){
       //obsluha tlačítek nastavujících čas
       if (PINC.0 == 0) t.minute = t.minute + 1;
       if (PINC.1 == 0) t.minute = t.minute - 1;
       if (PINC.2 == 0) t.minute = t.minute + 10;
       if (PINC.3 == 0) t.minute = t.minute - 10;

       if (t.minute==60) t.minute = 0;
       if (t.minute==-1) t.minute = 59;
       if (t.minute>=60) t.minute = t.minute - 60;
       if (t.minute< 0) t.minute = t.minute + 60;

       if (PINC.4 == 0) t.hour = t.hour + 1 ;
       if (PINC.5 == 0) t.hour = t.hour - 1 ;
       if (t.hour>=24) t.hour = t.hour - 24;
       if (t.hour < 0) t.hour = t.hour + 24;

     lcd_gotoxy(0,1);
     lcd_putsf(" ");
     lcd_gotoxy(1,1);
     poc = t.hour;
     itoa(poc,pom);
     lcd_puts(pom);

     lcd_gotoxy(4,1);
     poc = t.minute;
     itoa(poc,pom);
     lcd_puts(pom);

     lcd_gotoxy(7,1);
     poc = t.second;
     itoa(poc,pom);
     lcd_puts(pom);
```

```
     delay_ms(200);
   }
}

interrupt [TIM1_OVF] void counter(void) //overflow interrupt vector
{
#asm("cli")
     if (++t.second==60)
     {
       t.second=0;
       if (++t.minute==60)
       {
         t.minute=0;
         if (++t.hour==24)
         {
           t.hour=0;
           if (++t.date==32)
           {
             t.month++;
             t.date=1;
             }
           else if (t.date==31)
           {
             if ((t.month==4) || (t.month==6) ||
(t.month==9) || (t.month==11))
             {
                  t.month++;
                  t.date=1;
           }
         }
         else if (t.date==30)
         {
              if(t.month==2)
              {
                   t.month++;
                   t.date=1;
              }
         }
         else if (t.date==29)
         {
              if((t.month==2) && (not_leap()))
              {
                   t.month++;
```

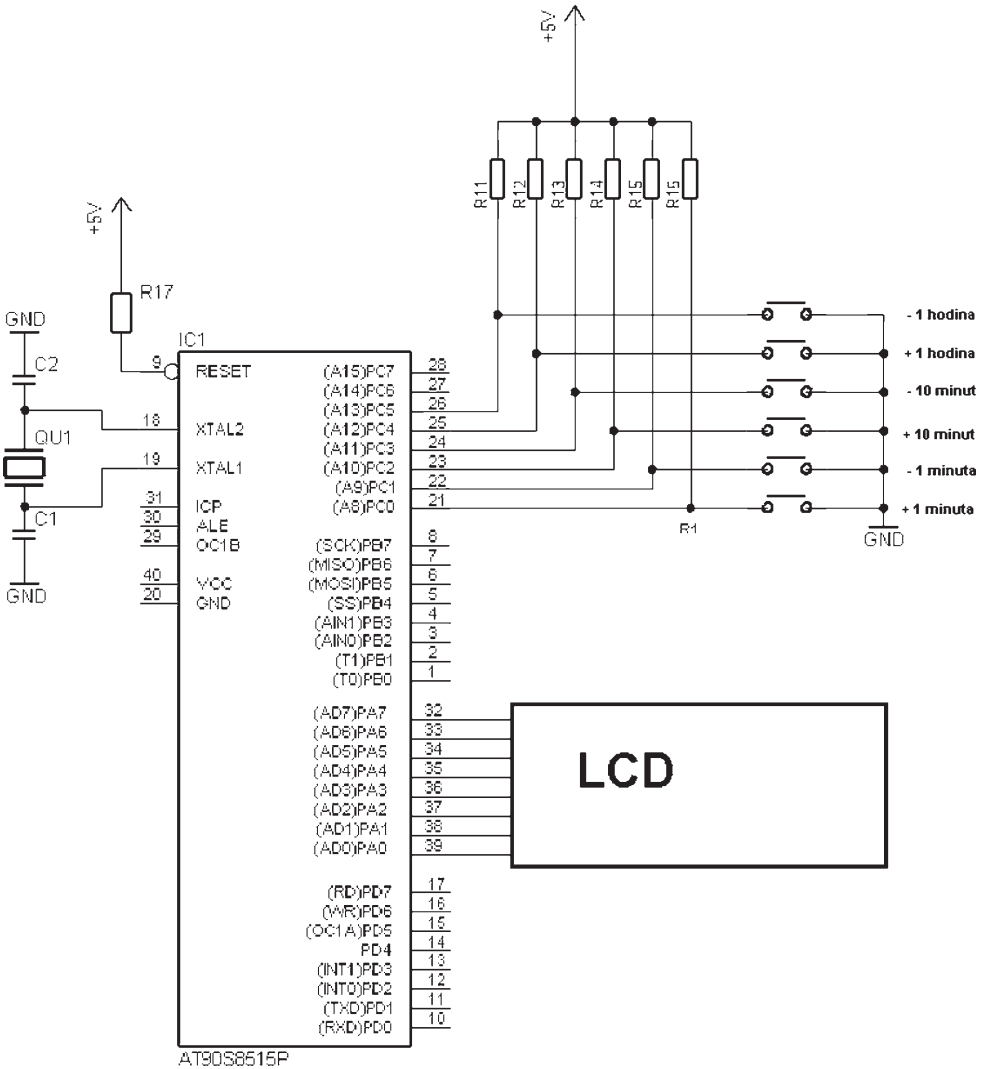
```
                    t.date=1;
                }
            }
            if (t.month==13)
            {
                t.month=1;
                t.year++;
              }
            }
          }
      }

TCNT1H=133;
TCNT1L=238;
#asm("clv") //vynulování příznaku přetečení
#asm("sei")

}

char not_leap(void) //přestupný rok ?
{
      if (!(t.year%100))
        return (char)(t.year%400);
      else
        return (char)(t.year%4);
}
```

Pro úplnost uvedeme ještě zjednodušené schéma hodin *obr. 7.29*.

7.10 Program 10 – sběrnice MicroWire

V předchozích příkladech jsme si ukázali jednoduché příklady, ve kterých mikrokontrolér AVR komunikuje s okolím prostřednictvím osmibitových paralelních portů, nebo pomocí vestavěných periferií, jako je UART či A/D převodník. Další možností je komunikace pomocí sériové sběrnice. Jako *sériové sběrnice* obvykle označujeme propojovací systémy, které spojují mikrokontrolér s pomocnými obvody v rámci jednoho zařízení. Sériová sběrnice šetří počet vývodů mikrokontroléru a periferních obvodů, zjednodušuje konstrukci. Je typicky tvořena dvojicí signálových vodičů. Jeden přenáší hodinový signál, hrany hodinového signálu definují časové okamžiky, ve kterých jsou na druhém vodiči prezentovány jednotlivé bity přenášených dat.

Pro některé obvody, jako jsou malé paměti EEPROM (např. AT24C02) je použití sériové sběrnice typické. Sériová sběrnice je pochopitelně pomalejší než sběrnice paralelní.



Obr. 7.29 Principiální schéma zapojení hodin s LCD displejem – příklad č. 9

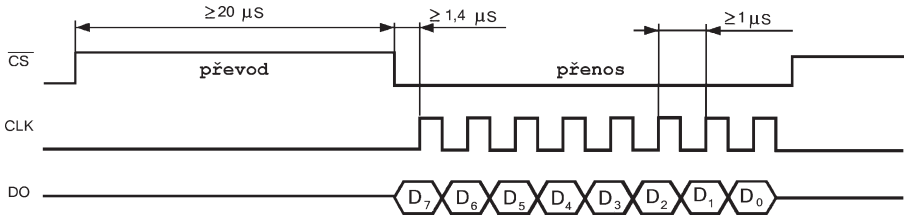
Přítomnost hodinového signálu určuje, že půjde o synchronní přenos. Tento přenos po sériové sběrnici je z obvodového hlediska jednodušší než přenos asynchronní.

Příkladem jednoduché sériové sběrnice je sběrnice **MicroWire** firmy National Semiconductor. Tato sběrnice dovoluje připojit skupinu periferních obvodů k mikrokontroléru. Je tvořena trojicí vodičů CLK, SO/DI a SI/DO. Hodinový signál CLK řídí přenos po dvou datových vodičích. První propojuje výstup mikrokontroléru *serial out* **SO** se vstupy *data in* **DI** periferních obvodů, druhý připojuje výstupy *data*

out **DO** periferních obvodů na vstup *serial in* **SI** mikrokontroléru. K jednomu mikrokontroléru lze připojit skupinu periferních obvodů. Výběr periferního obvodu vyžaduje použití dalších výběrových vodičů **CS**. Polarita hodinového signálu a časování datových signálů je definováno tak, že úrovně na datových vodičích se mění se sestupnou hranou hodinového signálu CLK, signály jsou čteny s náběžnou hranou hodin CLK. Sběrnice MicroWire nemá pevně definovanou délku předávaného slova. Pro většinu periferních obvodů je definován určitý primitivní protokol. U jednoduchých periferií však takový protokol ani není zapotřebí a data jsou předávána jako pouhé posloupnosti bitů. Takovým obvodem je 8bitový A/D převodník TLC549 od firmy Texas Instrumens. Jeho maloobchodní cena je asi 65 Kč.

Obvod TLC549 obsahuje posuvný registr do něhož umísťuje naměřenou hodnotu. Vlastní A/D převod trvá méně než 20 mikrosekund, takže jsou možná i rychlá měření. Signál chip celect CS přepíná mezi režimy převodu a čtení. Vlastní převod se odehrává při jedničce na pinu CS a trvá 20 mikrosekund. Poté lze na CS přivést nulu. Tím se na datový vodič přivede bit s největší váhou. Každým hodinovým impulzem se vysune bit s následující nižší váhou.

Z popisu funkce TLC549 je také zřejmé, že z datových signálů DI a DO sběrnice MicroWire využívá jen výstup dat. Průběhy signálů ukazuje následující *obr. 7.30*.

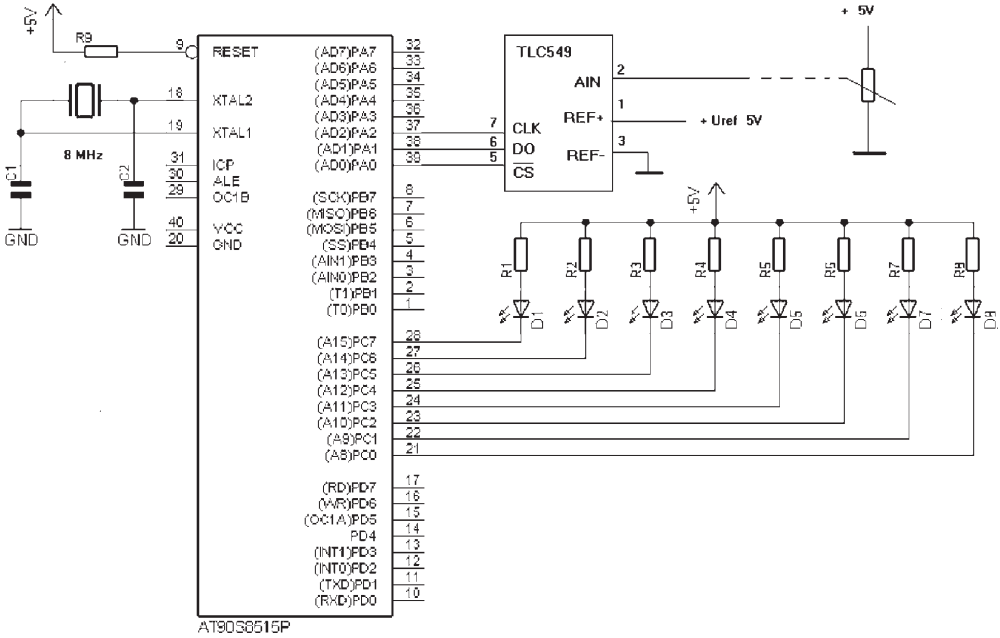


Obr. 7.30 Průběhy signálů sběrnice Microwire u obvodu TLC549

Zdrojový kód v assembleru ASM51 pro mikrokontrolér AT89C2051 je uveden v [20]. My si teď ukážeme jednoduchý program napsaný v CodeVisionC pro riscový mikrokontrolér AT90S8515. K zobrazení 8bitových dat pak použijeme pro jednoduchost programu jen LEDky. Zjednodušené schéma zapojení bude na *obr. 7.31*.

Zdrojový kód může být např.:

```
//TLC549 připojen k PORTA
//CS připojen k PA0, D0 k PA1, CLK k PA2
//výstup na PORTC – např. LEDky
#include <90s8515.h>
#include <delay.h>

#define cs PORTA.0
#define clk PORTA.2
```



Obr. 7.31 Principiální schéma A/D převodníku s obvodem TLC549 – příklad č. 10

```
unsigned char readtlc()
{
unsigned char napeti = 0, Ux = 128, loopi = 8;
clk = 0;
cs = 0;
delay_us(2);
while(loopi--)
     {
     if(PINA.1 == 1)    //čtení dat D0
          {
          napeti = napeti + Ux;
          }
     Ux=Ux/2;
     delay_us(2);
     clk = 1;           //hodinový pulz, náběžná hrana
     delay_us(2);
     clk = 0;           //hodinový pulz,sestupná hrana
     delay_us(2);
     }
cs = 1;
```

```
return napeti;
}

void main(void)
{
PORTC=0x00;
DDRC=0xFF;

PORTA=0x00;
DDRA=0x05;          //pin0 výstup, pin1 vstup, pin2 výstup
//inicializace vytvořená wizardem:
TCCR0=0x00;TCNT0=0x00;TCCR1A=0x00;TCCR1B=0x00;TCNT1H=0x00;
TCNT1L=0x00;OCR1AH=0x00;
OCR1AL=0x00;OCR1BH=0x00;OCR1BL=0x00;GIMSK=0x00;MCUCR=0x00;
TIMSK=0x00;ACSR=0x80;
cs = 1;
while (1)
       {
       PORTC = readtlc(); // zobrazení výsledku na PORTc – LEDky
       delay_ms(100);
       };
}
```

Program běží v nekonečné smyčce `while (1)`, ve které volá funkci `readtlc()` vracející číslo v rozmezí 0 až 255 v závislosti na naměřeném napětí a tuto hodnotu posílá na PORTC s připojenými diodami LED. Činnost funkce `readtlc()` je velice jednoduchá. Příkazem `cs = 1;` se spustí převod A/D. Před přečtením výsledku je třeba nastavit `cs = 0;` a nějakou dobu pomocí `delay_us(2);` počkat na ustálení stavu na CS. Hodinový signál byl přitom na nule. Poté již můžeme pomocí

```
clk = 1;        //hodinový pulz,náběžná hrana
delay_us(2);
clk = 0;        //hodinový pulz,sestupná hrana
delay_us(2);
```

vytvářet hodinové impulzy. Protože na začátku máme proměnnou `loopi` nastavenou na osm a impulzy generujeme ve smyčce `while(loopi--)`, bude počet průchodů touto smyčkou a tedy počet hodinových impulzů právě 8. Při každém průchodu smyčkou bude před vygenerováním hodinového impulzu provedeno zkontrolování úrovně signálu na DO. Bude-li to jednička, přičte se k proměnné **napeti** číslo, odpovídající váze příslušného bitu. Data vysílá TLC549 počínajíc nejvýznamnějším bitem. V tom případě bude tato váha `Ux = 128`. Pro následující bit bude tato váha poloviční, pro další bity čtvrtinová, osminová atd. Toho docílíme příkazem `Ux=Ux/2;` prováděným při každém průchodu smyčkou.