



ale od tohoto čísla, a přeteče tedy dřív. Jako zdroj hodinového signálu většinou slouží vlastní hodinový signál procesoru. Protože však hodinový signál procesoru kmitá miliónkrát za vteřinu (1 MHz), přetekl by časovač téměř 4 000x za jednu vteřinu. Protože však potřebujeme nastavit mnohem delší časy, musíme použít takzvanou předděličku. Předdělička je zařazená mezi zdrojem hodinového signálu a samotným časovačem. Jak již její název napovídá, předdělička slouží k tomu, aby hodinový signál pro časovač vydělila nějakým číslem. Funguje to tak, že například předdělička osmi propustí každý osmý impuls, předdělička 64 propustí každý 64. impuls, atd. K řízení časovače a předděličky se používá registr TCCR (opět na konci opatřen číslem časovače). Podívejme se nyní, jak se nastavuje například časovač 0: `*/ /* TCCR0 = 0` - čítač je vypnut `TCCR0 = 1` - čítač je zapnut, a pracuje přímo s hodinovým signálem (bez předděličky) `TCCR0 = 2` - čítač je zapnut s předděličkou osmi (hodinový signál/8) `TCCR0 = 3` - čítač je zapnut s předděličkou 64 `TCCR0 = 4` - čítač je zapnut s předděličkou 256 `TCCR0 = 5` - čítač je zapnut s předděličkou 1024 `*/ /*` Nyní tedy umíme zapnout časovač a nastavit mu předděličku (TCCR). Dokážeme také jemně nastavit čas, za který nám časovač přeteče. To děláme pomocí registru časovače (TCNT), do kterého nahrajeme hodnotu, od které má začít počítat. Jediné co tedy potřebujeme ještě udělat, je povolit přerušení od časovače. Musíme vlastně procesoru říct, aby toto přerušení akceptoval. Implicitně je totiž mikrokontrolér nastaven tak, že všechna přerušení (kromě resetu) ignoruje. Atmega8 má totiž 19 různých možných zdrojů přerušení jako: reset, čítače, vnější přerušení, A/D převodník, sériové kanály a další. Všechna tato zařízení mohou vyvolat přerušení. K povolování nebo zakazování přerušení od časovačů slouží registr TIMSK. Nastavením některých bitů v registru TIMSK na "1" lze povolit přerušení od jednotlivých časovačů: `*/ /* bit TIMSK: časovač: vektor: 0` časovač 0 `ISR(TIMER0_OVF_vect)` 2 časovač 1 `ISR(TIMER1_OVF_vect)` 6 časovač 2 `ISR(TIMER2_OVF_vect)` Nakonec je ještě potřeba nastavit 7. bit registru SREG. Tento bit slouží ke globálnímu povolení nebo zakázání všech přerušení najednou. `*/ /*` Vektor je místo, kam program přeskočí, když je aktivováno přerušení. Zápis je následující: `ISR(TIMER0_OVF_vect) { příkazy obsluhy přerušení; }` Po provedení příkazů obsluhy přerušení se program vrátí zpět na místo, odkud byl přerušen, a pokračuje v normální činnosti. `*/ /*` Nyní bychom se tedy mohli pustit do psaní programu, ale jak již bylo minule patrné, čím delší program píšeme, tím je méně přehledný. Proto je lépe si jej rozložit na jednotlivé dílčí problémy, a ty pak zapsat pomocí funkcí. Například program: `*/ /* ... PORTB = 0xff; //nastav všechny bity portu "B" na "1" waitms(5); //počkej 5 milisekund PORTD = 0b00000001; //nastav nultý bit portu "D" na "1" waitms(2); //počkej 2 milisekundy PORTD = 0b00000010; //nastav první bit portu "D" na "1" PORTB = 0x00; //nastav všechny bity portu "B" na "0" waitms(5); //počkej 5 milisekund ... */ /*` Tento program by šel přepsat jednodušeji a přehledněji takto: `*/ /* ... zapni_portb(); //nastav všechny bity portu "B" na "1" a počkej 5 milisekund prepni_portd(); //přepni první a druhý bit portu "D" s prodlevou 2 milisekundy vypni_portb(); //nastav všechny bity portu "B" na "0" a počkej 5 milisekund ... */ /*` Program je nyní jednodušší. Řada příkazů byla nahrazena jednoduchými funkcemi. Aby však program mohl fungovat, musíme tyto funkce nadefinovat. Definice funkce se provádí tak, že napíšeme hlavičku funkce (jméno funkce a typy hodnot, které funkce přijímá, popřípadě vrací. Definice funkce vypadá takto: `typ_navratove_hodnoty nazev_funkce (definice proměnných, které funkce přijímá) { tělo funkce }` Protože nyní pro náš příklad nepotřebujeme předávat žádné parametry, napíšeme místo návratové hodnoty a přijímaných hodnot slovo "void". Definice našich nových funkcí je tedy následující: `*/ /* void zapni_portb(void) //hlavička funkce "zapni_portb" { //začátek těla funkce PORTB = 0xff; //nastav všechny bity portu "B" na "1"`

```
waitms(5); //počkej 5 milisekund } //konec těla funkce */ /* Nyní máme nadefinovanou
funkci "zapni_portb". Pokud tedy v hlavním programu napíšeme: zapni_portb(); //spustí se
tato funkce, a provede příkazy ve svém těle: PORTB = 0xff; //nastav všechny bity portu "B" na
"1" waitms(5); //počkej 5 milisekund //po provedení těchto příkazů se program vrátí zpět a
pokračuje dalším příkazem, //tedy: prepni_portd() //Tuto funkci zatím ale nemáme
definovanou. Její definice by vypadala takto: */ /* void prepni_portd(void) //hlavička {
PORTD = 0b00000001; //nastav nultý bit portu "D" na "1" waitms(2); //počkej 2 milisekundy
PORTD = 0b00000010; //nastav první bit portu "D" na "1" } */ /* Podobným způsobem by se
nadefinovala i funkce "vypni_portb". Tento program by se pak choval stejně, jako původní
program bez funkcí. */ /* Nyní se podíváme na to, jak by se zapisovala funkce, která přebírá a
předává parametry: Mějme proměnné "cislo1", "cislo2" a "vysledek": unsigned char cislo1,
cislo2, vysledek; Nyní si představme, že v programu budeme potřebovat vykonat tuto operaci:
vysledek = (cislo1 + cislo2) * cislo1; //do proměnné "vysledek" se uloží součet
//proměnných "cislo1" a "cislo2" vynásobený //proměnnou "cislo1" Tuto operaci
můžeme nahradit funkcí: vysledek = funkce_vypocet(); Protože však potřebujeme, aby tato
funkce počítala s nějakými proměnnými ("cislo1" a "cislo2") musíme jí tyto proměnné předat
jako parametry: vysledek = funkce_vypocet(cislo1, cislo2); Tento příkaz by se nyní vykonal
následovně: Program zjistí, že má něco přiřadit do proměnné "vysledek" jde tedy do prava, a
narazí na funkci s parametry. Zapamatuje si tedy tyto 2 parametry ("cislo1", "cislo2") a přeskočí
do funkce "funkce_vypocet()". Této funkci předá oba parametry a funkci vykoná. Na konci
funkce obdrží jedno číslo (návrátovou hodnotu). Tuto hodnotu si zapamatuje, a skočí zpátky k
našemu příkazu: vysledek = funkce_vypocet(cislo1, cislo2); Nyní již má ale funkci
vypočtenou, a má v paměti uloženou její návratovou hodnotu. Nahradí tedy původní zápis:
"funkce_vypocet(cislo1, cislo2)" pouze návratovou hodnotou a vznikne: vysledek =
navratova_hodnota; Tento příkaz pak vyřeší jako prosté přiřazení - zapíše do proměnné
"vysledek" návratovou hodnotu funkce. Takže příkaz: vysledek = funkce_vypocet(cislo1,
cislo2); se vykoná stejně jako příkaz: vysledek = (cislo1 + cislo2) * cislo1; */ /* Nyní si ale
musíme naší funkci ještě nadefinovat: Napíšeme zase hlavičku funkce, a blok příkazů do těla
funkce: char funkce_vypocet(char vstup1, char vstup2); { char vystup; //nadefinujeme si
výstupní proměnnou (sem budeme ukládat výsledek) vystup = (vstup1 + vstup2) * vstup1;
//do proměnné "vystup" vypočteme operaci return vystup; //a řekneme, že se má jako
návratová hodnota použít hodnota //proměnné "vystup" } */ /* Nyní si rozebereme
hlavičku funkce: Slovo "char" před názvem funkce říká, jakého typu bude návratová hodnota
(kolik místa zabere v paměti - char = 8 bitů) Pak následuje název funkce "funkce_vypocet" (to
je název, pomocí kterého funkci voláme v programu). Za názvem následují v závorkách
parametry funkce. Zápis: char funkce_vypocet(char vstup1, char vstup2); říká, že funkce
očekává 2 parametry typu char a vrací číslo typu char. Tělo funkce je vytvořeno stejně, jako
tomu bylo v minulých případech. Jedinou novinkou je příkaz "return", za který se napíše, co má
funkce vrátit. Tento příkaz zároveň funkci ukončuje. */ /* Nyní už tedy víme, jak používat
časovač a jak rozepsat program do jednotlivých funkcí. Ještě by se hodila jedna poznámka, a
to že každá funkce musí být před svým použitím definovaná. V opačném případě hlásí
překladač chybu, že tuto funkci nezná. Prozatím to tedy vyřešíme tak, že všechny funkce
budeme definovat ještě před hlavní funkcí ("main"). Funkce main (ve které budeme naše
nadefinované funkce používat) bude tedy nadefinovaná až jako poslední (na konci zdrojového
kódu). */ /* Můžeme se tedy pustit do psaní programu. Naš program by měl umět blikat
ledkami v různých režimech a rychlostech. Režimy blikání by mohly být: - jednoduché blikání
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

(svítí nesvítí) - stroboskopické blikání (rozsvícení ledek na velmi krátkou dobu) - maják (ledky dvakrát rychle po sobě krátce bliknou) - běžící světlo (něco podobného, jako měl KIT - světlo běhá zleva doprava a zprava doleva) - had (náš starý známý) Program by se měl ovládat pomocí 2 tlačítek (PD2 a PD3), přičemž jedním tlačítkem by se měnil režim blikání, a druhým rychlost. Program by měl fungovat tak, jednotlivé režimy blikání budou uloženy jako samostatné funkce. Hlavní smyčka pak zjistí, jaký režim blikání je vybrán, a podle toho spustí požadovanou funkci. Funkce pak provede jeden krok (posunutí hada, bliknutí, atd.) a skončí. Čekání mezi jednotlivými kroky bude zajišťovat hlavní smyčka. Mezitím se bude pomocí časovače0 provádět testování klávesnice, a případné nastavování proměnných "rychlost" a "režim". \*/ //Náš program by tedy mohl vypadat takto:

```
#define F_CPU 1000000UL
// 1 MHz (základní frekvence) kvůli delay.h
#include <avr/io.h>
//Knihovna vstupů a výstupů (PORT, DDR, PIN)
#include <util/delay.h>
//Knihovna čekacích funkcí
#include <avr/interrupt.h>
//Knihovna přerušení (kvůli vektoru ISR(TIMER0_OVF_vect))
#define RYCHLOST_MIN 2
//Minimální počet kroků (bliknutí) za sekundu
#define RYCHLOST_MAX 100
//Maximální počet kroků (bliknutí) za sekundu
#define POCET_REZIMU 15
//počet režimů (režimů je 5, zbylé režimy slouží k

//prodlužování hada)
#define BLIKANI 0b00010111
//určuje, které ledky se mají střídát při režimu

//jednoduchého blikání
#define STROBO_CAS 10
//(ms) - Určuje, kolik milisekund budou svítit ledky při

//strobo efektu. Využívá se i v režimu "maják".
#define MAJAK 50
//definuje, kolik milisekund je mezi dvojicí bliknutí ve funkci

//maják //Vytvoříme si tzv. globální proměnné (jsou deklarované mimo všechny funkce, //díky
čemuž jsou ve všech funkcích "viditelné" //Běžná proměnná totiž platí jen mezi složenými
závorkami {}, mezi kterými byla //vytvořena (to znamená pouze uvnitř funkce).
unsigned
char
rychlost;
//globální proměnná - rychlost blikání
unsigned
char
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
rezim;
//globální proměnná - režim blikání
unsigned
char
zmena;
//globální proměnná - Sem budeme ukládat informaci, že byl

//právě změněn režim blikání (některé funkce se budou při

//svém prvním spuštění potřebovat inicializovat(nastavit

//základní hodnoty) //Tyto proměnné musejí být globální, protože se nastavují ve vektoru
přerušení, //a čtou se v hlavní funkci. Musejí tedy být přístupné z obou těchto míst. //nyní si
tedy nadefinujeme jednotlivé funkce, které pak budeme v hlavní funkci volat:
/*****/ /* Vektor přerušení */ /***/
ISR(TIMER0_OVF_vect)
{
static
unsigned
char
tlacitko;
//statická proměnná - uchovává si hodnotu mezi

//jednotlivými voláními funkce

//Pro nás slouží jako informace, že minule bylo

//stisknuto tlačítko (abychom předešli nechtěnému

//přečtení jednoho stisku 2x)
TCNT0=
157
;
//Časovač začne počítat od 157 (255-157=98) předdělička je 1024.

//Hodinový kmitočet procesoru je 1MHz. Takže:

//1 000 000 / 1024 / 98 = 10 Hz (klávesnice se kontroluje přibližně

//10x za vteřinu)
if
(tlacitko)
//pokud bylo minule stisknuto tlačítko...

{
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
tlacitko=  
0  
;  
//...tak proměnnou "tlacitko" vynulujeme...  
  
return  
;  
  
//...a pro jistotu ukončíme cyklus (uživatel by mohl tlačítko  
  
}  
  
//stále držet)  
  
if  
(!(PIND&0b00000100))  
  
//pokud je stisknuté tlačítko 1 (režim)  
  
{  
  
    rezim++;  
    //změníme režim  
  
    zmena=  
    1  
    ;  
    //právě jsme provedli změnu režimu  
  
    tlacitko=  
    1  
    ;  
    //uložíme si informaci, že bylo stisknuto tlačítko  
  
}  
  
if  
(!(PIND&0b00001000))  
//pokud je stisknuté tlačítko 2 (rychlost)  
  
{  
  
    rychlost*=  
    2  
    ;  
    //přidáme rychlost (2x zrychlíme je to totéž jako: rychlost<<1;)
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
tlacitko=
1
;
//uložíme si informaci, že bylo stisknuto tlačítko

}
if
(rezim>POCET_REZIMU)
//pokud jsme již překročili povolený počet režimů...

{

rezim=
1
;

//...začneme zase od začátku (1. režim)

}
if
(rychlost>RYCHLOST_MAX)
//pokud jsme již překročili maximální rychlost...

{

rychlost=RYCHLOST_MIN;
//...nastavíme rychlost zpátky na minimum

}

}
//Konec vektoru přerušení - program se může vrátit zpět ke své práci.
/*****/ /*funkce jednoduchého blikání*/ /***/
void
blik
(
void
)

{
if
(zmena)

//pokud je funkce spuštěna poprvé
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
{

PORTB=
BLIKANI;
//nastav do portu "B" počáteční hodnotu

zmena=
0
;
//vynuluj proměnnou "zmena" (příště už to nebude poprvé)

}
else

//pokud to není poprvé...

{

PORTB=~PORTB;
//do poru "B" dáme jeho bitový komplement - vyměníme "0" za "1"

}
}
// konec funkce "blik"  /******/* /* funkce stroboskopu */
/******/
void
strobo
(
void
)
{
PORTB=
0xff
;
//rozsvítíme všechny ledky...
_delay_ms(STROBO_CAS);
//...počkáme malou chvíli...
PORTB=
0
;
//...a zhasneme všechny ledky
}
/******/* /* Maják */ /******/

void
majak
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
(
void
)
{
for
(
char
n=
0
;n<
2
;n++)
//cyklus o 2 opakováních:

{

PORTB=
0xff
;

//rozsvítíme všechny ledky...

_delay_ms(STROBO_CAS);
//...chvilku počkáme...

PORTB=
0
;

//...zhasneme všechny ledky...

_delay_ms(MAJAK);

//...a počkáme delší chvilku (50 ms)

}
}
//konec funkce "majak"  /*****/ /* Běžící světlo */
/*****/

void
semtam
(
void
)
{
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
static
unsigned
char
pozice;
//statická proměnná - uchovává si hodnotu mezi

//jednotlivými voláními funkce

//nám bude sloužit jako ukazatel, kde zrovna je ledka

//1-8 = 1.-8. ledka. 9-15 = 6.-1. ledka (jede zpátky)
if
(zmena)
//pokud je funkce volána poprvé

{

PORTB=
1
;
//zhasneme všechny ledky a rozsvítíme pouze jednu (PB0)...

pozice=
1
;

//...a nastavíme pozici na začátek (1.ledka)

zmena=
0
;
//vynuluj proměnnou zmena (příště už to nebude poprvé)

return
;

//ukončíme funkci (1. ledka už svítí, příště ji budeme posouvat

}
if
(pozice<
8
)
//pokud pozice ještě není 8 (ještě jsme nedošli k 8. ledce)

{
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
PORTB<<=
1
;
//posuň ledku o 1 doleva.

}
else
//pozice je větší nebo rovna 8, (už jsme dojeli k levému kraji)

{

PORTB>>=
1
;
//posuň ledku o 1 doprava (jedeme zase na začátek)

}

pozice++;
//posunuli jsme ledku, tak musíme také posunout ukazatel
if
(pozice>=
15
)
//pokud je ledka na 15. pozici (zpátky na začátku)

{

pozice=
1
;
//přepíšeme ukazatel na "1" (začátek)

}
}
//konec funkce "semtam"  /******/ /*          Had          */
/*****/

void
had
(
unsigned
char
delka)
//náš starý známý had
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
{

//Funkce "had" přebírá parametr délka. Při volání funkce tedy

//můžeme (nebo spíš musíme) zadat délku hada.

static
unsigned
char
pozice;
//statická proměnná - ukazatel na pozici hada

//(stejně jako u běžícího světla)
if
(zmena)
//pokud je funkce volána poprvé (stejně jako u běžícího světla)

{

PORTB=
0
;
//zhasneme všechny ledky ...

pozice=
0
;

//...a nastavíme pozici na začátek (1.ledka)

zmena=
0
;
//vynuluj proměnnou "zmena" (příště už to nebude poprvé)

}
//Dosavadní část je vlastně stejná, jako byla použita ve funkci "Běžící světlo" //Následující část
funkce "had" je víceméně stejná, jako v našem minulém programu.

if
(pozice<delka)
//pokud jsme rozsvítli méně ledek než je délka hada

{

PORTB
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
<<=
1
;
//přidáme článek hada (bity se posunou doleva a

//vpravo se doplní "0"

PORTB
++;
//přičteme "1" (nastavíme nultý bit na "1")

}

else

//pokud není počet opakování menší než délka hada

{

//to znamená, že hada už jsme nakreslili..

PORTB
<<=
1
;
//pouze posuneme bity (hada) doleva a doplní se nula...

}

pozice++;
//posunuli jsme ledku, tak musíme také posunout ukazatel

if
(pozice>(delka+
8
))
//pokud had dolezl na konec (poslední článek už zmizel,

{

pozice=
0
;
//přepíšeme ukazatel na "0" (začátek)

}
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
}  
//konec funkce "had"  /*****/  
  
/*****/  
  
/**   Hlavní funkce   **/  
  
/*****/  
  
int  
main  
(  
void  
) { DDRB  
=  
0xff  
;  
  
//Nastavíme port "B" jako výstupní  
TIMSK|=  
1  
;  
  
//nastavíme nultý bit na "1" a ostatní necháme (povolíme přerušení  
  
//od časovače "0")  
TCCR0  
=  
5  
;  
//Zapneme časovač "0" s předděličkou 1024.  
SREG  
  
|=  
(  
1  
<<  
7  
);  
//povolení přerušení (nastavíme 7. bit registru SREG)  
rezim=  
1  
;  
//nastavíme režim blikání na 1. možnost (jednoduché blikání)  
zmena=
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
1
;
//nastavíme, jako že jsme právě změnili režim (funkce blik bude spuštěna

//poprvé)
rychlost=Rychlost_MIN;
//nastavíme rychlost na minimum // nyní máme vše potřebné nastaveno, a můžeme spustit
hlavní smyčku:
for
(;;)
//hlavní smyčka
{
//Tato smyčka bude pouze zajišťovat výběr správné funkce (podle nastaveného

//režimu blikání, a čekání mezi jednotlivými kroky.

//Výběr správné funkce provedeme pomocí příkazu "switch".Switch je vlastně
//něco jako vícenásobný "if"

switch
(rezim)

//budeme rozhodovat podle toho, co je v proměnné "rezim"...

{

case
1
:

//...v případě že je to číslo "1"...

blik();
//...zavoláme funkci "blik()" (1. režim)...

break
;

//...a vyskočíme z příkazu switch (jinak by se provedly všechny

//následující instrukce - switch by už další podmínky netestoval.

case
2
:
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
//...v případě že je to číslo "2"...
```

```
strobo();
```

```
//...zavoláme funkci "strobo()" (2. režim)...
```

```
break
```

```
;
```

```
case
```

```
3
```

```
:
```

```
//...v případě že je to číslo "3"...
```

```
majak();
```

```
//...zavoláme funkci "majak()"
```

```
break
```

```
;
```

```
case
```

```
4
```

```
:
```

```
//...v případě že je to číslo "4"...
```

```
semtam();
```

```
//...zavoláme funkci "semtam()"
```

```
break
```

```
;
```

```
default
```

```
:
```

```
//Pokud nevyhovoval ani jeden z předchozích případů - zbývá už
```

```
//jen funkce "had" (5. a vyšší režim)
```

```
had(rezim-
```

```
4
```

```
);
```

```
//zavoláme funkci "had" a jako parametr jí předáme číslo
```

```
//režimu bez 4. To znamená, že funkce může obdržet číslo
```

## AVR - LED panel - #6 Program

Napsal uživatel Vašek Král  
Neděle, 10 Duben 2011 07:33

---

```
//jedna až něco, které pak použije k nastavení délky hada

}
//konec příkazu switch

//úspěšně jsme tedy vybrali jakou funkci má program provést a nyní nám už zbývá
//jen počkat požadovanou dobu, než se program pustí do dalšího kroku:
_delay_ms
(
1000
/rychlost);
//Čekání (1000 ms = 1 sekunda)

}
//konec hlavní smyčky
}
//konec funkce main //Pro radioklub OK1KVK napsal Vašek Král
```