Tick

JEDNOTKA PRO PRÁCI SE SYSTÉMOVÝM ČASOVAČEM

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.30

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 20.04.2005
Datum posledního uložení dokumentu: 20.04.2005
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Hvo		První vydání.
1.10	2.XX	Tum	22.05.2003	Úprava dokumentu dle ISO9000.
1.20	2.XX	Wil	04.08.2004	Přidána konstanta TickInt = $08. Proceduru SetTimeUser1 lze volat i vícekrát za sebou, což do teď nebylo možné při korekci proměnné IRQTime. Navíc SetTimeUser1 provádí tuto korekci automaticky, tj. po volání procedury SetTimeUser1 **tuto korekci neprovádět v aplikaci**. Změna inicializace ExitProc, která se nastavuje až v procedurách InitTick a InitTickI a ne v inicializační části jednotky, jako tomu bylo dosud. Přidána procedura DoneTick, která obnovuje původní časovač a nastaví původní ExitProc. Přidány proměnné FastIRQxxx, které udávají údaje o "zrychleném" časovači. Přidána procedura SetTimeUser2 pro nastavení periody volání původní obsluhy časovače a UserTick2. Kompenzace zrychlení systémového časovače při vyšší komunikační rychlosti na KitV40 se provádí automaticky, tj. **tuto korekci neprovádět v aplikaci**.
1.30	2.XX	Wil	20.04.2005	Ve zrychlené větvi přerušení se inkrementuje lokální čítač, který je dostupný pomocí funkce GetFastTime. Pokud není nainicializován driver zrychleného časovače, vrací GetFastTime čítač standardního počtu přerušení (stejně jako AH=00h INT 1Ah).

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky pro práci se systémovým časovačem.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu není nezbytně nutné číst žádný další manuál, ale je potřeba se orientovat v používání programového vybavení SofCon. Pro bližší

porozumění funkce knihovny Tick je vhodné se seznámit s konstantami systémového časovače definovanými v knihovně „HwSyst“.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu „LibVer“.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu „Termíny a definice“.

3. Úvod

Jednotka umožňuje nastavení nové obsluhy přerušení, která volá uživatelsky nastavitelné procedury **UserTick1**, **UserTick2** a původní obsluhu přerušení. Dále umožňuje nastavení periody volání systémového přerušení INT $08, která lze nastavit přímo pomocí hodnoty systémového časovače nebo zadáním času volání procedury **UserTick1**, přičemž procedura **UserTick1** bude volána s nastavenou periodou systémového časovače a procedura **UserTick2** a původní obsluha přerušení systémového časovače budou volány s původní periodou. Všechny tyto výše popsané možnosti lze použít pro zrychlení odezvy na vstupní události.

U řídících jednotek KitV40 se tato jednotka používá i pro kompenzaci zrychlení systémového časovače v důsledku vyšších komunikačních rychlostí při komunikaci pomocí obvodu i8251 na procesoru V40, viz. manuál BIOS V40.

4. Inicializace ovladače systémového časovače

Instalace obsluhy přerušení systémového časovače, se provede zavoláním procedury **InitTick** nebo procedury **InitTickI**. Nastavená obsluha přerušení volá uživatelsky nastavitelné procedury **UserTick1**, **UserTick2** a původní obsluhu přerušení **OldInt08**. Procedura **InitTickI** nastaví reentrantní obsluhu přerušení. Tzn. v době vykonávání procedury **UserTick1** nebo **UserTick2** může přijít další přerušení od časovače nebo i jiné hardwarové přerušení. V případě dalšího (vnořeného) přerušení od systémového časovače se právě prováděná procedura **UserTick1** nebo **UserTick2** obchází. Tento způsob obsluhy přerušení umožňuje trasování programu i v procedurách **UserTick1** a **UserTick2**. Procedura **InitTick** nemá ošetřenu reentrantní obsluhu přerušení. Tzn. v době vykonávání obsluhy přerušení systémového časovače nemůže přijít další přerušení od systémového časovače ani jiné hardwarové přerušení.

Při instalaci obsluhy přerušení systémového časovače některou z procedur **InitTick** nebo **InitTickI** se rovněž nastaví příznak **FlInitTick** a nastaví se Exit procedura (Pascal **ExitProc**), která obnovuje původní systémový časovač. Aplikace by měla případnou vlastní Exit proceduru nastavovat ještě před **InitTick** nebo **InitTickI**, čímž se zajistí, že v době provádění aplikační Exit procedury bude již ovladač systémového časovače ukončen.

Před každým voláním procedury **UserTick1** je inkrementován lokální čítač počtu přerušení, jehož aktuální hodnota je aplikaci dostupná pomocí funkce **GetFastTime** (viz kapitola „10.8 GetFastTime“). Tento čítač je obdobou čítače inkrementovaného v původní obsluze přerušení časovače **OldInt08**, který je dostupný přes službu AH=00h INT 1Ah.

5. Nastavení periody přerušování systémového časovače

Implicitně je **UserTick1** a **UserTick2** přiřazena prázdná procedura (**Nic**). Při provádění obsluhy přerušení nastavené pomocí procedur **InitTick** nebo **InitTickI** je vyhodnocována lokální proměnná **TickDivider**, která určuje kolikrát pomaleji bude volána procedura **UserTick2** a původní obsluha přerušení systémového časovače

(**OldInt08**) oproti (zrychlené) proceduře **UserTick1**. Tzn. použitím vlastní procedury **UserTick1** lze dosáhnout zrychlení odezvy na vstupní událost. Proměnná **TickDivider** je implicitně nastavena na hodnotu 1. Zjištění hodnoty proměnné **TickDivider** je možno pomocí funkce **GetTickDivider**. Při potřebě zrychlit odezvu na vstupní události nebo kompenzaci zrychlení systémového časovače je možno použít buď proceduru **SetTickDivider**, která nastaví proměnnou **TickDivider** a provede nastavení systémového časovače, nebo proceduru **SetTimeUser1**, která nastaví periodu volání **UserTick1** a rovněž provede nastavení systémového časovače. Rozdíl **SetTickDivider** a **SetTimeUser1** je následující: Chceme-li, aby procedura **UserTick2** a **OldInt08** byly volány pravidelně po 55ms a procedura **UserTick1** byla volána X-krát rychleji, použijeme proceduru **SetTickDivider(X)**. Tím jsme ale schopni dosáhnout pouze určitých period volání procedury **UserTick1** – vždy pouze celočíselné podíly 55ms (55 div X). Pokud chceme nastavit přesné volání procedury **UserTick1**, použijeme proceduru **SetTimeUser1**. **UserTick1** je volána přesně dle nastaveného parametru a **UserTick2** a **OldInt08** je volána přibližně s 55ms. Tato přesnost závisí na parametru procedury **SetTimeUser1**, jelikož perioda **UserTick2** a **OldInt08** je odvozena z násobku volání **UserTick1**, který se nejvíce blíží 55ms. Přesnější hodnota volání **UserTick2** a **OldInt08** je nastavena do proměnné **ActIRQTime** z jednotky HWSyst. Např. Chceme-li volat **UserTick1** přesně s periodou 10ms, zavoláme **SetTimeUser1(0,010)**. Periodu volání **UserTick2** a **OldInt08** poté zjistíme z proměnné **ActIRQTime**.

6. Rady a triky

Následují rady a typy pro práci s touto jednotkou, které nemusí být na první pohled zřejmé:

- Při nastavení vlastní obsluhy přerušení systémového časovače *<u>před voláním</u>* **InitTick** nebo **InitTickI** je tato vlastní obsluha volána vždy s frekvencí, která je téměř stejná jako původní obsluha, tj. 55ms.

```
Př. GetIntVec(TickInt,OldTimeInt); {ulozime puvodni obsluhu}
    SetIntVec(TickInt{8},MyTimeInt); {MyTimeInt je interrupt
                              procedura s vlastní obsluhou
                              systemoveho casovace, která na konci
                              vola puvodni obsluhu OldTimeInt}
    UserTick1:=@MyUserTick1; {MyUserTick1 je vlastní FAR
                              procedura která se bude volat
                              zrychlene}
    UserTick2:=@MyUserTick2; { MyUserTick1 je vlastní FAR
                              procedura která se bude volat
                              standardne}
    InitTick;
    SetTickDivider(11);
    {nyní je nastaveno:
       zrychlený časovač 55/11 = 5ms
       standardní časovač 55ms
       MyTimeInt, OldTimeInt a MyUserTick2 se volá standardně
       MyUserTick1 se volá zrychleně
    }
```

- Při nastavení vlastní obsluhy přerušení systémového časovače *<u>po zavolání</u>* **InitTick** a **InitTickI** je tato obsluha volána s nastavenou frekvencí systémového časovače, tj. dle nastavených rychlostí **SetTickDivider** nebo **SetTimeUser1**.

```
Př. UserTick1:=@MyUserTick1; {MyUserTick1 je vlastní FAR
                                        procedura která se bude
                                        volat zrychlene}
    UserTick2:=@MyUserTick2; { MyUserTick1 je vlastní FAR
                               procedura která se bude volat
                               standardne}
    InitTick;
    SetTickDivider(11);
    GetIntVec(TickInt,OldTimeInt);
    SetIntVec(TickInt,MyTimeInt); {MyTimeInt je interrupt
                               procedura s vlastní obsluhou
                               systemoveho casovace, která na konci
                               vola puvodni obsluhu OldTimeInt }
    {nyní je nastaveno:
       zrychlený časovač 55/11 = 5ms
       standardní časovač 55ms
       MyUserTick2 se volá standardně
       MyUserTick1, MyTimeInt a OldTimeInt se volá zrychleně
    }
```

- Zavedení procedur **UserTick1** a **UserTick2** je z důvodu snížení režie na obsluhu přerušení INT $08. V případě použití vlastních obsluh přerušení systémového časovače je tato režie mnohem vyšší (práce se zásobníkem) a je třeba uvážit, kdy tuto obsluhu nainstalovat (viz výše).

- Používáme-li operační systém reálného času ReTOS, musí se uvážit, kdy inicializovat jádro ReTOS a kdy inicializovat driver systémového časovače voláním procedur **InitTick** nebo **InitTickI**. Při volání procedury **InitTick** nebo **InitTickI** *před inicializací jádra* ReTOS se zrychlí přepínání procesů v systému ReTOS, tj. přepínání jádra se děje se stejnou periodou jako volání **UserTick1**. V případě *provedení inicializace* **InitTick** *nebo* **InitTickI** *po inicializaci jádra* ReTOS se přepínání procesů v systému ReTOS nezrychlí, tj. přepínání jádra se děje se stejnou periodou jako **UserTick2**, tj. přibližně po 55ms.

- Nyní si ukažme způsob, jak zrychlit systémový časovač na požadovanou periodu (v příkladu na 2ms) a zároveň zrychlit i systém reálného času ReTOS na jinou požadovanou periodu (v příkladu na 50ms).

```
Př. Const
      cFTickPer =  2; {[ms] požadovaná perioda zrychleného časovače}
      cRetosPer = 50; {[ms] požadovaná perioda nezrychleného časovače
                       a o.s.ReTOS, MUSÍ být celistvým násobkem
                       konstanty cFTickPer}
    Begin
     {základním předpokladem pro následující řešení je, že perioda
      přepínání ReTOS musí být větší nebo rovna periodě zrychleného
      časovače}
      if cRetosPer<cFTickPer then RunError(255);

     {inicializace driveru zrychleného časovače}
      InitTick;

     {případné zrychleni časovače}
      if cFTickPer<DefIRQTime{55} then
      begin
        SetTimeUser1(cFTickPer/1000{[s]});
      end;

     {případné zrychlení standardního časovače a ReTOS}
      if cRetosPer<ActIRQTime then
      begin
        SetTimeUser2(cRetosPer/1000{[s]});
      end;
```

```
  {inicializace o.s.ReTOS}
   StartMain(Main_SPrio,Main_DPrio);
   InitInterruptStack(1,254);
   StartTimeSlicing(8);
   ...
    tělo programu
End.
```

- Jak již bylo řečeno, voláním procedury **InitTick** nebo **InitTickI** se nastaví Exit procedura, která prostřednictvím procedury **DoneTick** obnovuje činnost systémového časovače. Voláním **DoneTick** se také obnoví původní Exit procedura, která byla jako ExitProc nastavena PŘED voláním procedury **InitTick** nebo **InitTickI**. Proto POZOR na následující příklad.

```
Př.  Begin
       ...
       {do ExitProc nastavíme svoji obsluhu MyExit1}
        SaveExit1:=ExitProc;
        ExitProc:=MyExit1;
       {inicializujeme ovladač systémového časovače}
        InitTick;
       {do ExitProc nastavíme svoji obsluhu MyExit2}
        SaveExit2:=ExitProc;
        ExitProc:=MyExit2;
       {odinstalujeme ovladač systémového časovače}
        DoneTick;
```

  Nyní je v ExitProc nastavena MyExit1, jelikož volání DoneTick obnovilo stav před InitTick. Proto se doporučuje, aby aplikace svoji Exit proceduru nastavovala PŘED InitTick resp. InitTickI.

  Pozn: Obdobná situace je s operačním systémem reálného času ReTOS, který nastavuje svoji Exit proceduru voláním StartMain.

7. Kompenzace zrychlení systémového časovače při vyšší komunikační rychlosti na KitV40

Tato část je určena pouze systémům s řídící jednotkou KitV40 využívající komunikační kanál V40 (např. pomocí knihoven ChnV40, ChnV40P, ChnV40_, ChnV40T).

Při inicializaci komunikačního kanálu V40 s rychlostí 9600Bd a vyšší dochází ke změně frekvence časování systémového časovače. Při používání starších verzí systémových knihoven LIB bylo zapotřebí, aby programátor provedl kompenzaci této změny frekvence přímo ve své aplikaci pomocí ručního nastavení proměnné **TickDivider**. Pokud by tak neučinil, porušilo by se časování původní obsluhy systémového časovače a i knihovna **Timer** pro odměřování časových intervalů by měřila špatně. V současné verzi systémových knihoven se tyto kompenzace provádějí automaticky, tj. programátor nemusí (ani nesmí) ručně upravovat proměnnou TickDivider. Z tohoto důvodu byla **proměnná TickDivider přesunuta z veřejné (interface) sekce jednotky do privátní (implementation). Tj. při překladu starší již napsané aplikace (která ještě stále provádí ruční úpravu**

TickDivider) s novými knihovnami nahlásí překladač chybu „Unknown identifier“. Řešení je velice prosté: Odstraňte ruční nastavení TickDivider.

Příklady automatické kompenzace systémového časovače:

Př.1:
```
{inicializace ovladače systémového časovače}
 InitTick;

{zrychlení systémového časovače}
 SetTimeUser1(10*1e-3{10ms});

{nastavení vlastních procedur vyvolávaných v systémovém
 časovači}
 asm pushf; cli end;
 UserTick1:=@MyUserTick1; {MyUserTick1 se volá s periodou
                           FastIRQtime, v tomto případě 10ms}
 UserTick2:=@MyUserTick2; {MyUserTick2 se volá s periodou
                           ActIRQtime, v tomto případě 60ms}
 asm popf end;

{inicializace komunikačního kanálu V40 s rychlostí 19200Bd}
 viz postup popsaný v příslušné komunikační knihovně
 {pro rychlost 19200Bd na KitV40 16MHz dochází ke 4násobnému
  zrychlení systémového časovače, díky automatické kompenzaci
  bude FastIRQtime = 10ms a ActIRQtime = 60ms}
```

Př.2:
```
{inicializace ovladače systémového časovače}
 InitTick;

{zrychlení systémového časovače}
 SetTimeUser1(10*1e-3{10ms});

{nastavení vlastních procedur vyvolávaných v systémovém
 časovači}
 asm pushf; cli end;
 UserTick1:=@MyUserTick1; {MyUserTick1 se volá s periodou
                           FastIRQtime, v tomto případě 10ms}
 UserTick2:=@MyUserTick2; {MyUserTick2 se volá s periodou
                           ActIRQtime, v tomto případě 60ms}
 asm popf end;

{inicializace komunikačního kanálu V40 s rychlostí 38400Bd}
 viz postup popsaný v příslušné komunikační knihovně
 {pro rychlost 38400Bd na KitV40 16MHz dochází k 8mi násobnému
  zrychlení systémového časovače, díky automatické kompenzaci
  bude FastIRQtime = 10ms a ActIRQtime = 60ms}
```

Pozn: Nezáleží na tom, zda nejprve nainicializujeme systémový časovač a potom komunikaci V40 nebo naopak. Kompenzace se provádí i při odinicializování komunikace V40.

8. Odinstalování ovladače systémového časovače

Odinstalování ovladače systémového časovače, který byl inicializován některou z procedur **InitTick** nebo **InitTickI** se provede procedurou **DoneTick** (viz „10.3 DoneTick“). Tato procedura je rovněž volána jako Exit procedura této knihovny.

9. Popis konstant a typů

```
cVerNo = např. $0251; { BCD formát }
cVer   = např. '02.51,07.08.2003';
```

Číslo verze jednotky v BCD tvaru a v textové podobě včetně datumu změny.

```
TickInt = $08
```

Číslo přerušení pro obsluhu systémového časovače.

```
tUserTick = procedure;
```

Typ procedura.

```
UserTick1 : tUserTick = Nic;
```

Uživatelsky nastavitelná procedura, která je volána s novou (zrychlenou) frekvencí systémového časovače.

```
UserTick2 : tUserTick = Nic;
```

Uživatelsky nastavitelná procedura, která je volána s původní frekvencí systémového časovače.

```
FlInitTick : Boolean = False;
```

Příznak určující zda ovladač systémového časovače je či není nainstalován. Tato konstanta se nastavuje voláním procedur **InitTick** nebo **InitTickI** a nuluje voláním procedury **DoneTick**.

```
FlTick1Over : Boolean = False;
```

Příznak, který signalizuje, že před dokončením provádění procedury **UserTick1** bylo vygenerováno nové přerušení systémového časovače a procedura **UserTick1** nemohla být provedena, protože byla již obsluhována. Tj. doba provádění **UserTick1** je příliš dlouhá. Tento příznak se používá pouze v případě nastavení reentrantní obsluhy systémového časovače procedurou **InitTickI**.

```
FlTick2Over : Boolean = False;
```

Příznak, který signalizuje, že před dokončením provádění procedury **UserTick2** bylo vygenerováno nové přerušení systémového časovače a procedura **UserTick2** nemohla být provedena, protože byla již obsluhována. Tj. doba provádění **UserTick2** je příliš dlouhá. Tento příznak se používá pouze v případě nastavení reentrantní obsluhy systémového časovače procedurou **InitTickI**.

```
OldInt08 : Pointer = Nil;
```

Ukazatel na obsluhu přerušení od systémového časovače Int 08h při volání **InitTick** nebo **InitTickI**. Používá se pro volání původní obsluhy přerušení od systémového časovače.

```
FastIRQTime : word = DefIRQTime;
```

Aktuální perioda [ms] mezi přerušeními "zrychleného" systémového časovače.

```
FastIRQpSec : real = DefIRQpSec;
```

Aktuální počet přerušení za sekundu "zrychleného" systémového časovače.

```
FastIRQpDay : longint = DefIRQpDay;
```

Aktuální počet přerušení za den "zrychleného" systémového časovače.

Pozn: Proměnné **FastIRQxxx** se aktualizují po každém volání **InitTick, InitTickI, SetTickDivider, SetTimeUser1** a **DoneTick**. Tyto proměnné se vztahují na „zrychlený“ časovač, tj. platí pro uživatelskou proceduru **UserTick1**. Na rozdíl od toho proměnné **ActIRQxxx** z knihovny **HwSyst** se vztahují na původní, tj. „nezrychlený“ časovač **UserTick2**.

10. Procedury a funkce

10.1. InitTick

```
procedure InitTick;
```

Procedura **InitTick** nastaví nereentrantní obsluhu přerušení systémového časovače volající původní obsluhu přerušení a uživatelsky definovatelné obsluhy **UserTick1** a **UserTick2**. Během provádění procedur **UserTick1** a **UserTick2** není povoleno žádné hardwarové přerušení.

10.2. InitTickI

```
procedure InitTickI;
```

Procedura **InitTickI** nastaví reentrantní obsluhu přerušení systémového časovače volající původní obsluhu přerušení a uživatelsky definovatelné obsluhy **UserTick1** a **UserTick2**. Na rozdíl od procedury **InitTick**, která nastaví nereentrantní obsluhu přerušení, není během provádění procedur **UserTick1** a **UserTick2** zakázáno hardwarové přerušení. To znamená, že i v době, kdy je program vnořen do procedur **UserTick1** nebo **UserTick2**, může přijít jakékoliv hardwarové přerušení i další nové přerušení od systémového časovače. V případě dalšího (“vnořeného”) přerušení od systémového časovače program zajistí, aby již obsluhované procedury **UserTick1** a **UserTick2** byly obcházeny. To dovoluje trasování (ladění) programu i v procedurách **UserTick1** a **UserTick2**.

10.3. DoneTick

```
procedure DoneTick;
```

Procedura **DoneTick** ukončí činnost ovladače systémového časovače a nastaví jeho původní obsluhu, tj. **OldInt08**, která je volána se standardní periodou 55ms. To znamená, že vrátí systémový časovač do stavu, který byl před voláním procedur **InitTick** resp. **InitTickI**. Procedura **DoneTick** také obnoví původní Exit proceduru, která byla nastavena před voláním procedur **InitTick** resp. **InitTickI**.

Po proceduře **DoneTick** je možno opět volat proceduru **InitTick** resp. **InitTickI**.

Pozn: Procedura **DoneTick** je volána i v **ExitProc** (ukončovací proceduře) této knihovny, pokud uživatel nezavolal **DoneTick** ještě před ukončovacím řetězcem Exit procedur.

10.4. SetTickDivider

```
procedure SetTickDivider(TD:word);
```

Procedura nastavuje poměr **TickDivider** a dělič systémového časovače odpovídající dělicímu poměru **TickDivider**. Při dělicím poměru 1 je perioda volání procedur **UserTick1** a **UserTick2** 55 ms. Nastavením jiného dělicího poměru (>1) se učí, kolikrát častěji má být volána **UserTick1** oproti **UserTick2,** která je volána s 55ms. Například při poměru 2 je perioda 22.5 ms pro **UserTick1** a 55ms pro **UserTick2**. Před voláním procedury **SetTickDivider** je nutno inicializovat ovladač systémového časovače některou z procedur **InitTickI** nebo **InitTick**.

Pro kompenzaci zrychlení systémového časovače při vyšších komunikačních rychlostech na KitV40 by se měl používat postup uvedený v kapitole „Úvod“ v části zaměřené na rady a typy pro práci s jednotkou Tick.

Procedura **SetTickDivider** se může volat pouze při nainicializované obsluze systémového časovače, tj. po procedurách **InitTick**/**InitTickI**. V opačném případě se vygeneruje RunTime Error 199.

10.5. GetTickDivider

```
function GetTickDivider:word;
```

Funkce navrátí aktuální hodnotu proměnné **TickDivider**.

10.6. SetTimeUser1

```
procedure SetTimeUser1(aTime:Real);
```

Procedura nastaví proměnnou **TickDivider** a systémový časovač tak, aby uživatelsky nastavitelná procedura **UserTick1** byla přerušována v nastaveném čase dle parametru *aTime* [s]. Původní obsluha **OldInt08** a **UserTick2** nemusí být volány v pravidelné periodě 55ms, ale pouze přibližně (viz kapitola „Úvod“). Z tohoto důvodu je také nutno, aby se při volání **SetTimeUser1** nepoužíval (neměl „rozčasováno“) žádný časovač z jednotky **Timer**. Ten totiž předpokládá, že od doby volání metody **SaveTime** (objektu **tTimer** z jednotky **Timer**) až do vypršení nastaveného časového limitu metodou **TstTime** je frekvence volání obsluhy **OldInt08** konstantní. Pokud bychom tedy zavolali proceduru **SetTimeUser1** v polovině měření času objektem **tTimer**, mohli bychom rozhodit jeho správné vyhodnocení časového intervalu.

Parametr *aTime* musí být v rozmezí 0.0001s až 0,055s.

Procedura **SetTimeUser1** se může volat pouze při nainicializované obsluze systémového časovače, tj. po procedurách **InitTick**/**InitTickI**. V opačném případě se vygeneruje RunTime Error 199.

10.7. SetTimeUser2

```
procedure SetTimeUser2(aTime:Real);
```

Procedura upraví lokální proměnnou **TickDivider** tak, aby uživatelsky nastavitelná procedura **UserTick2** a původní obsluha přerušení **OldInt08** byla přerušována v přibližně nastaveném čase dle parametru *aTime* [s].

Procedura **SetTimeUser2** se může volat pouze při nainicializované obsluze systémového časovače, tj. po procedurách **InitTick**/**InitTickI**. V opačném případě se vygeneruje RunTime Error 199.

Výraz přibližné nastavení periody *aTime* znamená, že pokud tato perioda není v celistvých násobcích periody volání **UserTick1**, tj. **FastIRQtime/1000**, použije se nejbližší vyhovující hodnota (viz příklad níže).

Dále je také nutno, aby se při volání **SetTimeUser2** nepoužíval (neměl „rozčasováno“) žádný časovač z jednotky **Timer**. Ten totiž předpokládá, že od doby volání metody **SaveTime** (objektu **tTimer** z jednotky **Timer**) až do vypršení nastaveného časového limitu metodou **TstTime** je frekvence volání obsluhy **OldInt08** konstantní. Pokud bychom tedy zavolali proceduru **SetTimeUser1** v polovině měření času objektem **tTimer**, mohli bychom rozhodit jeho správné vyhodnocení časového intervalu.

Parametr *aTime* musí být v rozmezí 0.0001s až 0,055s, ale nesmí být menší než perioda volání **UserTick1**, tj. **FastIRQtime/1000**.

Př.1:
```
InitTick;

SetTimeUser1(10*1e-3);
SetTimeUser2(40*1e-3);
{UserTick1 je volán s periodou 10ms,
 UserTick2 a OldInt08 je volán s periodou 40ms}
```

Př.2:
```
InitTick;

SetTimeUser2(40*1e-3);
SetTimeUser1(10*1e-3);
{UserTick1 je volán s periodou 10ms,
 UserTick2 a OldInt08 je volán s periodou 60ms, tj. není volán
 s periodou 40ms => SetTimeUser2 volat až po SetTimeUser1}
```

Př.3:
```
InitTick;

SetTimeUser1(10*1e-3);
SetTimeUser2(25*1e-3);
{UserTick1 je volán s periodou 10ms,
 UserTick2 a OldInt08 je volán s periodou 30ms, jelikož
 požadovaných 25ms není celistvý násobek SetTimeUser1}
```

Pozn.: Správné nastavení výše zmíněných period si lze ověřit následovně: proměnná **FastIRQtime** by měla odpovídat parametru procedury **SetTimeUser1** (pozor na to, že FastIRQtime je v milisekundách, kdežto parametr procedury SetTimeUser1 je v sekundách) a proměnná **ActIRQtime** by měla odpovídat parametru procedury **SetTimeUser2** (rovněž pozor na jednotku času [ms] vs. [s])

10.8. GetFastTime

```
function GetFastTime:longint;
```

Funkce vrátí aktuální hodnotu čítače počtu zrychleného přerušení od časovače. Tento čítač se inkrementuje v přerušení od časovače před každým voláním procedury **UserTick1**. Inkrementace probíhá od hodnoty 0 do hodnoty 7FFFFFFFh (= High(longint)). Pokud není inicializován (některou z procedur InitTick, InitTickI) driver zrychleného systémového časovače, funkce GetFastTime vrací aktuální hodnotu původního časovače pomocí služby AH=00h INT 1Ah.

10.9. Nic

```
procedure Nic;
```

Procedura je prázdná, obsahuje pouze instrukci RETF. Tato procedura je implicitně přiřazena procedurálním proměnným **UserTick1** a **UserTick2**, dokavad uživatel neprovede jejich přenastavení.