Vysoké učení technické
v Brně
Fakulta informačních
technologií

Mikroprocesorové a vestavěné systémy

Přerušení od modulu Periodic Interrupt Timer (PIT)

20. prosince 2020

Michal Koval(xkoval17)

Obsah

1 Úvod **2**
1.1 Autori a zdroje 2

2 Rozbor tématu **3**
2.1 Mikrokontroler 3
2.2 Časovače - Timers 3
2.2.1 PIT - Periodic Interrupt Timer 4

3 Návrh aplikácie **6**

4 Implementácia **7**
4.1 Zaujímavé časti kódu 7

5 Záver **10**

1 Úvod

Zadanie projektu bolo vytvoriť projekt demonštrujúci generovanie prerušenia modulom Periodic Interrupt Timer (PIT) dostupnom na mikrokontroléry Kinetis K60 [1] z dosky platformy FITkit 3 [2]. Aplikácia má umožňovať interakciu s uživateľom, prinajmenšom vhodná signalizácia prerušenia (LED, piezzo bzučák, terminál). Aplikáciu treba nastaviť tak, aby demonštrovala schopnosť generovania prerušení s čo najkratšou a potom s čo najdlhšou periodou. Generáciu PIT prerušení je potrebné demonštrovať v nasledujúcih režimoch:

- nezreťazený s jedným časovačom
- nezreťazený s viacerými časovačmi (2 a viac)
- zreťazený s 2 časovačmi
- zreťazený s maximálnym počtom časovačov (4)

1.1 Autori a zdroje

Projekt vypracoval Michal Koval študent FIT VUT v Brne. Ako študijné zdroje boli použité prednášky a slidy kurzu Mikroprocesorové a vestavěné systémy [3], referenčný manuál mikrokontroleru Kinetis K60 [1]. Pri tvorbe projektu nebol k dispozícii FITkit3, takže aplikácia nebola testovaná a vychádza iba zo štúdia kurzu a referenčného manuálu.

2 Rozbor tématu

2.1 Mikrokontroler

Mikrokontroler alebo MCU (microcontroller unit) je jednočipový počítač optimalizovaný pre využitie vo vstavaných (embedded) aplikáciach/systémoch. Slúži na riadenie zariadenia, príjma spracuváva a odosiela signály, ktorými sa riadi chovanie zariadenia. Programovať mikrokontroler znamená popisovať chovanie zariadenia pomocou softwaru. Zovšeobecnene sa skladá z procesoru, pamäti, modulov a vstupno výstupných zariadení (UART, AD prevodníky, porty).

Moduly

Modul mikrokontroleru je typicky hardwarová implementácia niejakej funkcie. Túto funkciu potom nie je potrebné programovať. Stačí ich z jadra softwarovo spúšťať keď su potrebné. S modulmi pracujeme pomocou definovaných adries ich rôznych vstupných a výstupných registrov. Do nich zapisujeme čo má modul vykonať a čítame z nich stav a výsledok práce modulu. Výhody využívania modulov:

- rýchlosť - obvod, ktorý implementuje konkrétny algoritmus je rýchlejší ako keby sa ten istý algoritmus napísný softwarovo vykonával na univerzálnom počítači
- energetická efektivita - výrazne menšia spotreba
- zaberajú menej kremíku - menej plochy na čipu
- bežia paralelne - umožňujú jádru vykonávať inú prácu zatiaľ čo moduly si robia tú svoju, odbremeňujú samotné a umožňujú istý level paralelizmu

2.2 Časovače - Timers

Časovač je hardwarový modul na počítanie času. Umožňuje sledovať čas nezávysle od chodu CPU, čím ho odbremeňuje a zároveň aj šetrí energiu. Principiálne jeho fungovanie je triviálne. Inkrementuje alebo dekrementuje register pri každom pulse, väčšinou pulsy hodinového signálu, po uplnutí nastaveného počtu pulsov vyvolá výnimku. Toto by sa samozrejme dalo vykonávať aj v procesore, ale bolo by to energeticky omnoho náročnejšie a zároveň by sa zbytočne plýtvali takty procesora na tieto operácie, namiesto ktorých by mohol robiť niečo iné. V prípade, kedy by procesor nemusel robiť nič iné, tak sa procesor môže uspať, čím dôjde k ďalšiemu šetreniu energie, a až po uplninutí času, ktorý rátal časovač sa prebudí na vyvolanie prerušenia. Dôležitosť tíchto časovačov a merania času všeobecne je zcela zrejmá. Zariadenie

funguje v reálnom svete, kde plynie čas a je potrebné od mikrokontroleru, aby bol schopný vykonávať niejaké veci v niejakých požadovaných časoch, závisiach od konkrétnej aplikácie. Väčšina mikrokontrolerov má k dispozícii hneď niekoľko typov časovačov pre rôzne potreby.

2.2.1 PIT - Periodic Interrupt Timer

PIT periodicky generuje prerušenie (akcia v software) alebo trigger (akcia v hardware). PIT časovač sa nastavuje na hodnotu TSV - timer start value. Táto hodnota sa dekrementuje každým časovým tikom kým nedosiahne nulu. V tom bode časovač vygeneruje prerušenie a opäť sa nastaví na počiatočnú TSV hodnotu. Tieto časovače najlepšie slúžia k časovaniu niejakých periodických stále rutín, napríklad obnovenie displeja.

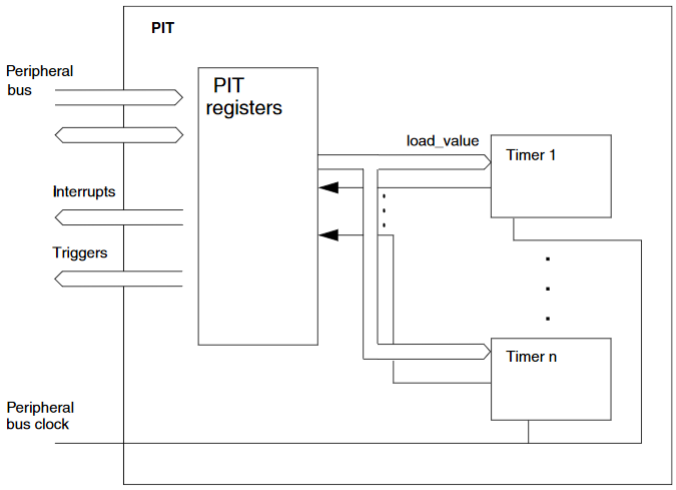


Figure 41-1. Block diagram of the PIT

Obrázek 1: Blokový diagram PIT modulu v Kinetis K60 [1]

Nastavenie času

Hodnota TSV predstavuje počet tikov hodiniek kým sa má vyvolať prerušenie a začať ďalší odpočet. V našom MCU Kinetis K60 [1] sa táto hodnota zapisuje do registru PIT_LDVALn, kde n predstavuje poradie časovača. Máme k dispozícii 4, takže n môže nadobúdať čísla 0 - 4. Ak chceme nastaviť časovač podľa reálneho času, hodnotu TSV môžme vypočítať podľa vzorca:

$$TSV = round(T * f_{count} - 1)$$

T = čas v ms

f_{count} = frekvencia časovača, zvyčajne rovnaká ako frekvencia MCU, v našom prípade 100MHz

Pri zmene TSV hodnoty a zápisu novej hodnoty do PIT_LDVALn registru, sa časovač nereštartuje, ani neodratáva od tejto novej hodnoty, ale ukončí najskôr aktuálny odpočet a až v ďalšom bude odrátavať od novej hodnoty. Ak chceme časovač prenastaviť hneď, je potrebné ho najskôr deaktivovať, potom nastaviť novú TSV a následne zase aktivovať.

Zreťazenie

Keďže register časovača je obmedzený, v našom prípade na 32 bitov [1], jeden časovač má limit koľko maximálne času dokáže "čakať". Pri použití vzorca z predošlej sekcie môžme vyrátať maximálny čas jedného našeho PIT časovača:

$$2^{32} = T * 100 - 1$$
$$2^{32} + 1 = T * 100$$
$$(2^{32} + 1)/100 = T$$
$$T = 42949672ms$$

T_{max} je teda približne necelých 12 hodín.

V prípade že potrebujeme niečo vykonávať s väčšou periodou, napr rutinná kontrola stavu zariadenie raz týždenne, jeden časovač nestačí a vtedy sa používa zretezenie časovačov. Zreťazenie praxi znamená, že po vypršaní času 1. časovača sa dekrementuje hodnota 2. časovača, ktorý je zreťazený s 1. Každý zo zreťazených časovačov môže mať nastavenú inú TSV hodnotu. Vďaka zreťazeniu máme k dispozícii maximálnu periodu pri využití všetkých 4 časovačov:

$$2^{128} = T * 100 - 1$$
$$2^{128} + 1 = T * 100$$
$$(2^{128} + 1)/100 = T$$
$$T = 3,4 * 10^{38}ms$$

Čo je už doba dlhšia ako zrejme bude životnosť samotnej aplikácie.

Teoreticky by sa čakacia doba dala ešte ďalej predĺžiť tým, že po vypršaní akumulovaného času všetkych časovačov prerušenie ošetríme tým, že procesor inkrementuje niejaký ďalší register v RAM, ktorý bude slúžiť ako časovač.

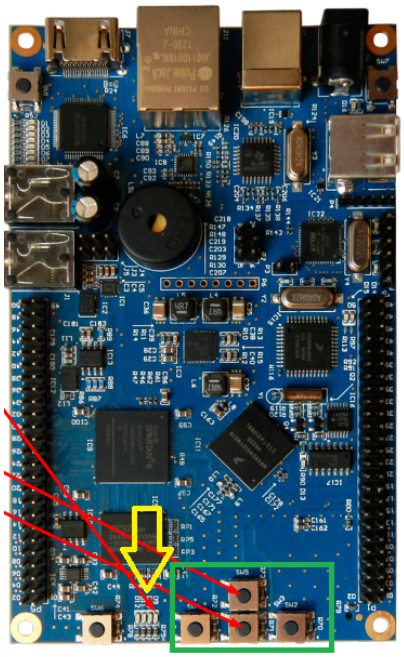
Zreťazenie sa nastavuje pomocou PIT_TCTRLn registru. Nastavením jeho 2. (CHN) bitu na 1 sa časovač zreťazí s predošlím časovačom, tzn. časovač 2 s 1. Ak je CHN bit na 0 časovač je v nezreťazenom režime.

3 Návrh aplikácie

Program po spustení bude v režime 0. Prerušenia v každom režime uživateľovi zobrazujú bliknutím príslušnej LEDky. Medzi módmi sa dá prepínať tlačítkami. Konfigurácia režimov ich tlačidiel a LEDiek:

- režim 0: 1 nezreťazený časovač, LED D9 (raz za 1 sekundu), tlačitko SW4
- režim 1: 2 nezreťazené časovače, LED D9 (raz za 1 sekundu) a LED D10 (raz za 2 sekundu), tlačítko SW3
- režim 2: 2 zreťazené časovače, LED D11 (raz za 1 sekundu), tlačítko SW2
- režim 3: 4 zreťazene časovače, LED D12 (raz za 2 sekundu), tlačítko SW5

V prípade stlacenia tlačítka na zmenu režimu do režimu, v ktorom program aktuálne je sa nedeje nič. Všetky periody na čakanie boli zámerne zvolené krátke, aby nebolo potrebné pri demonštrácii dlho čakať na prebliknutie príslušnej LEDky.



Obrázek 2: Znázornenie ovládacích tlačidiel (zelený rámček) a signalizačných LED (žltá šipka)

4 Implementácia

Projekt bol implementovaný v IDE Kinetis Design Studio ver. 3.0. KDS vygenerovalo hlavičkový súbor `MK60D10.h` obsahujúci makrá pre našu vývojovú platformu. Samotná implementácia projektu je napísana v súbore `main.c`. Projekt nebol testovaný pretože som nemal pri vývoji FITkit3.

4.1 Zaujímavé časti kódu

Hlavná smyčka - prepínanie režimov

Nekonečná šmyčka aby sa program nikdy nedostal z mainu. V nej sa prevádzajú kontroly na stlačenie prepínacích tlačidel. Ak je tlačidlo stlačené a aktuálny režim je iný ako cieľový režim daného tlačidla, deaktivujú sa aktúalne nestavené PIT časovače s funkciou `clear_current_setup()` a funkciou `setup_rezim_n()` sa aktivuje požadovaný režim.

```
// Hlavna slucka kde kontrolujeme stlacenie tlacitok a zmeny rezimov
while (1) {
    if ((GPIOE_PDIR & BTN_SW5) && rezim != 3){
        clear_current_setup(); // deaktivacia aktualneho rezimu
        setup_rezim_3(); // zapnutie rezimu 3
    } else if ((GPIOE_PDIR & BTN_SW4) && rezim != 0){
        clear_current_setup(); // deaktivacia aktualneho rezimu
        setup_rezim_0(); // zapnutie rezimu 0
    } else if ((GPIOE_PDIR & BTN_SW3) && rezim != 1){
        clear_current_setup(); // deaktivacia aktualneho rezimu
        setup_rezim_1(); // zapnutie rezimu 1
    } else if ((GPIOE_PDIR & BTN_SW2) && rezim != 2){
        clear_current_setup(); // deaktivacia aktualneho rezimu
        setup_rezim_2(); // zapnutie rezimu 2
    }

}
```

Nastavenie PIT časovačov

Nasledujúca časť kódu zobrazuje príklad nastavenia PIT časovača a jeho prerušenia. Tieto dve konkrétne funkcie ukazujú ako by sa teoreticky dal nastaviť časovač na maximálnu možno periódu a minimálnu možnú. V `setup_max_time()` sa všetky 4 časovače nastavia na maximálny možný počet tikov a na poslednom sa aktivuje prerušenie. V `setup_min_time()` sa nastaví prerušenie na každý jeden tik času. Predpokladám, že toto zrejme v praxi by robilo problémy. Pretože prerušenie sa tuna generuje v každom takte procesora a samotná obsuluha prerušenia trvá niekoľko taktov, buď by boli obsluhy značne oneskorené a ak by obsluhu trvala príliš dlho, možno by aj pretiekla fronta prerušení.

```
/* Demonstracna funkcia na ukazanie nastavenia maximalnej
   cakacej doby so zretazenim vsetkych casovacov.
   Nie je pouzita lebo by sme sa doslova nedockali kym by vyprsal cas.*/
void setup_max_time(void)
{
    PIT_LDVAL0 = 0xFFFFFFFF; // nastavenie TSV casovaca na maximum
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK; // zapnutie casovaca 0

    PIT_LDVAL1 = 0xFFFFFFFF; // nastavenie TSV casovaca na maximum
    PIT_TCTRL1 |= PIT_TCTRL_CHN_MASK; // zretazenie casovaca 1 s casovacom 0
    PIT_TCTRL1 |= PIT_TCTRL_TEN_MASK; // zapnutie casovaca 1


    PIT_LDVAL2 = 0xFFFFFFFF; // nastavenie TSV casovaca na maximum
    PIT_TCTRL2 |= PIT_TCTRL_CHN_MASK; // zretazenie casovaca 2 s casovacom 1
    PIT_TCTRL2 |= PIT_TCTRL_TEN_MASK; // zapnutie casovaca 2


    PIT_LDVAL3 = 0xFFFFFFFF; // nastavenie TSV casovaca na maximum
    PIT_TCTRL3 |= PIT_TCTRL_TIE_MASK; // aktivujeme prerusenie pre PIT casovac 3
    PIT_TCTRL3 |= PIT_TCTRL_CHN_MASK; // zretazenie casovaca 3 s casovacom 2
    PIT_TCTRL3 |= PIT_TCTRL_TEN_MASK; // zapnutie casovaca 3

}

/* Demonstracna funkcia na ukazanie nastavenia minimalnej cakacej doby pre PIT timer.
   Nie je pouzita lebo tuna by sme si to prerusenie zas na druhej strane nestihli vsimnut.*/
void setup_min_time(void)
{
    PIT_LDVAL0 = 0; // nastavenie na prerusenia kazdy tik hodin
    PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK; // aktivujeme prerusenie pre PIT casovac 0
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK; // zapnutie casovaca 0
}
```

Handlery PIT prerušení

PIT prerušenia sa ošetrujú zapnutím príslušnej LEDky, počkaním 0.1 sekundy, aby sme si sthli zapnutie ledky všimnúť, a následným vypnutím LEDky. Nakoniec sa odnastavý príznak prerušenia.

```
void PIT0_IRQHandler(void)
{
    GPIOB_PDOR ^= LED_D9; // Zapnutie ledky
    delay(10000000); // aktivne cakanie 0,1 sekundy
    GPIOB_PDOR ^= LED_D9; // Vypnutie ledky
    PIT_TFLG0 |= PIT_TFLG_TIF_MASK; // odnastavime interrupt flag
}

void PIT1_IRQHandler(void)
{
    GPIOB_PDOR ^= LED_D10; // Zapnutie ledky
    delay(10000000); // aktivne cakanie 0,1 sekundy
    GPIOB_PDOR ^= LED_D10; // Vypnutie ledky
    PIT_TFLG1 |= PIT_TFLG_TIF_MASK; // odnastavime interrupt flag
}

void PIT2_IRQHandler(void)
{
    GPIOB_PDOR ^= LED_D11; // Zapnutie ledky
    delay(10000000); // aktivne cakanie 0,1 sekundy
    GPIOB_PDOR ^= LED_D11; // Vypnutie ledky
    PIT_TFLG2 |= PIT_TFLG_TIF_MASK; // odnastavime interrupt flag
}

void PIT3_IRQHandler(void)
{
    GPIOB_PDOR ^= LED_D12; // Zapnutie ledky
    delay(10000000); // aktivne cakanie 0,1 sekundy
    GPIOB_PDOR ^= LED_D12; // Vypnutie ledky
    PIT_TFLG3 |= PIT_TFLG_TIF_MASK; // odnastavime interrupt flag
}
```

5 Záver

Projekt sa podarilo vypracovať iba z rešeršnej a konceptuálnej časti, vzhľadom k tomu, že som nemal k dispozícii potrebný hardware FITkit verzie 3. Kód, ktorý som napísal nebol nikdy otestovaný a vysokopravdepodobne bude aj nefunkčný, slúži skôr k ukážke toho akým smerom by som sa uberal pri skutočnej implementácii a k demonštrácii štúdia manuálu[1].

Reference

[1] K60 Sub-Family Reference Manual https://www.nxp.com/docs/en/reference-manual/K60P144M100SF2V2RM.pdf

[2] Schéma obvodového zapojení výukového kitu Minerva https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=\%2Fcourse\%2FIMP-IT\%2Fexcs\%2FFITkit3-schema.pdf&cid=13997

[3] Mikroprocesorové a vestavěné systémy FIT VUTBR https://www.fit.vut.cz/study/course/13997/.cs