Mikroprocesorová technika

Prednáška č. 12

Moderné aplikácie mikroradičov

1. časť – Metodika tvorby softvéru pre mikroradiče

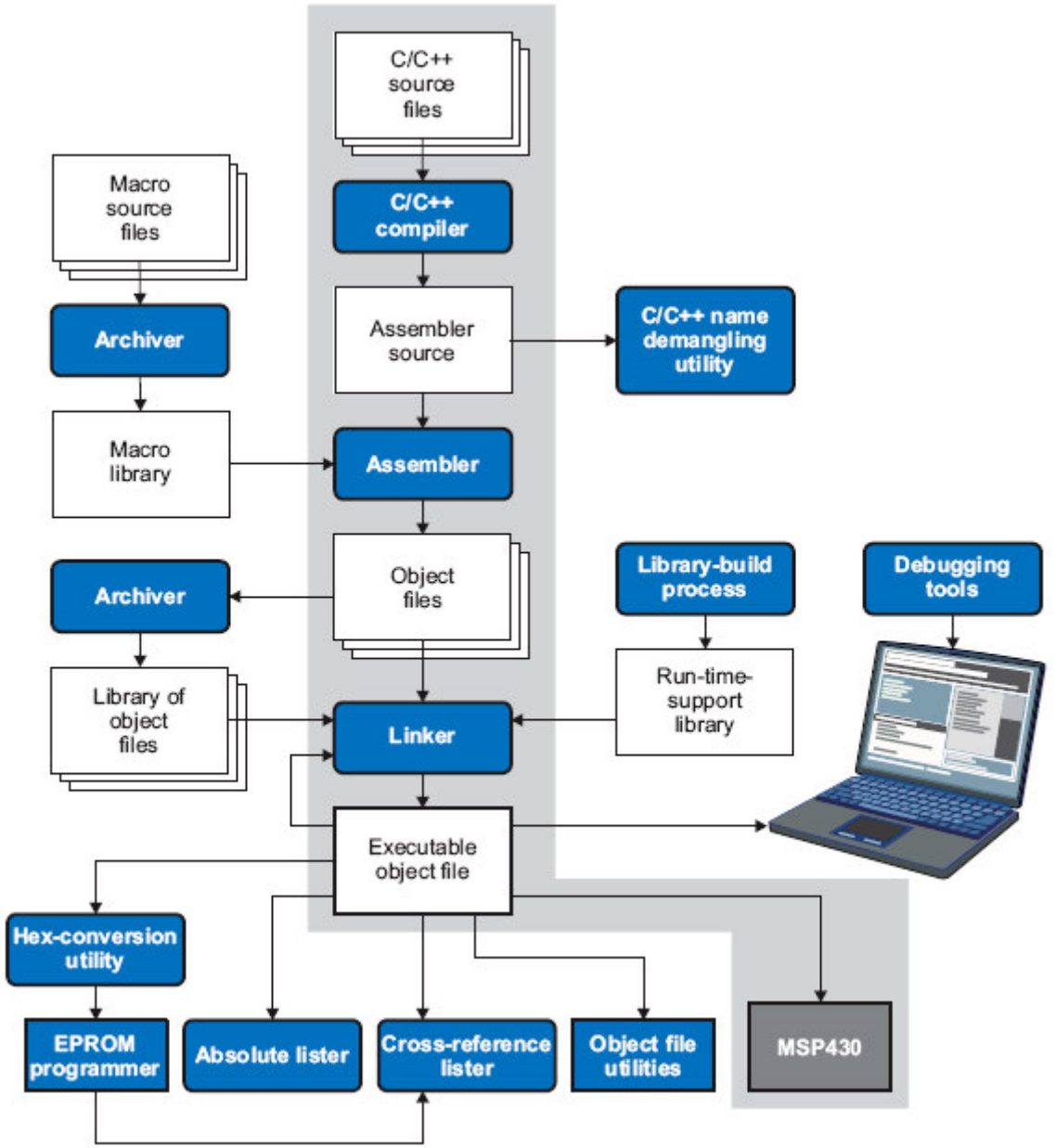
2. časť – RTOS (Real Time Operating System)

3. časť – Príklady aplikácii MSP430

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA RÁDIOELEKTRONIKY
Laboratórium DSP a mikroradičov

1. časť – Metodika tvorby softvéru pre mikroradiče

:: Postupnosť pri vývoji softvéru pre procesory



:: Poznámky k tvorbe softvéru

- architektúra zdrojového kódu moderných mikroradičov je obvykle založená na prerušeniach (interrupt driven), t.j. procesor sa nachádza väčšinu času vo zvolenom nízkopríkonovom režime a v aktívnom režime sa nachádza iba pri oblsuhe jednotlivých udalostí, ktoré spúšťajú prerušenia

- vieme, že v prípade MSP430 sa bity, ktorými volíme nízkopríkonový režim nachádzajú v stavovom registri SR

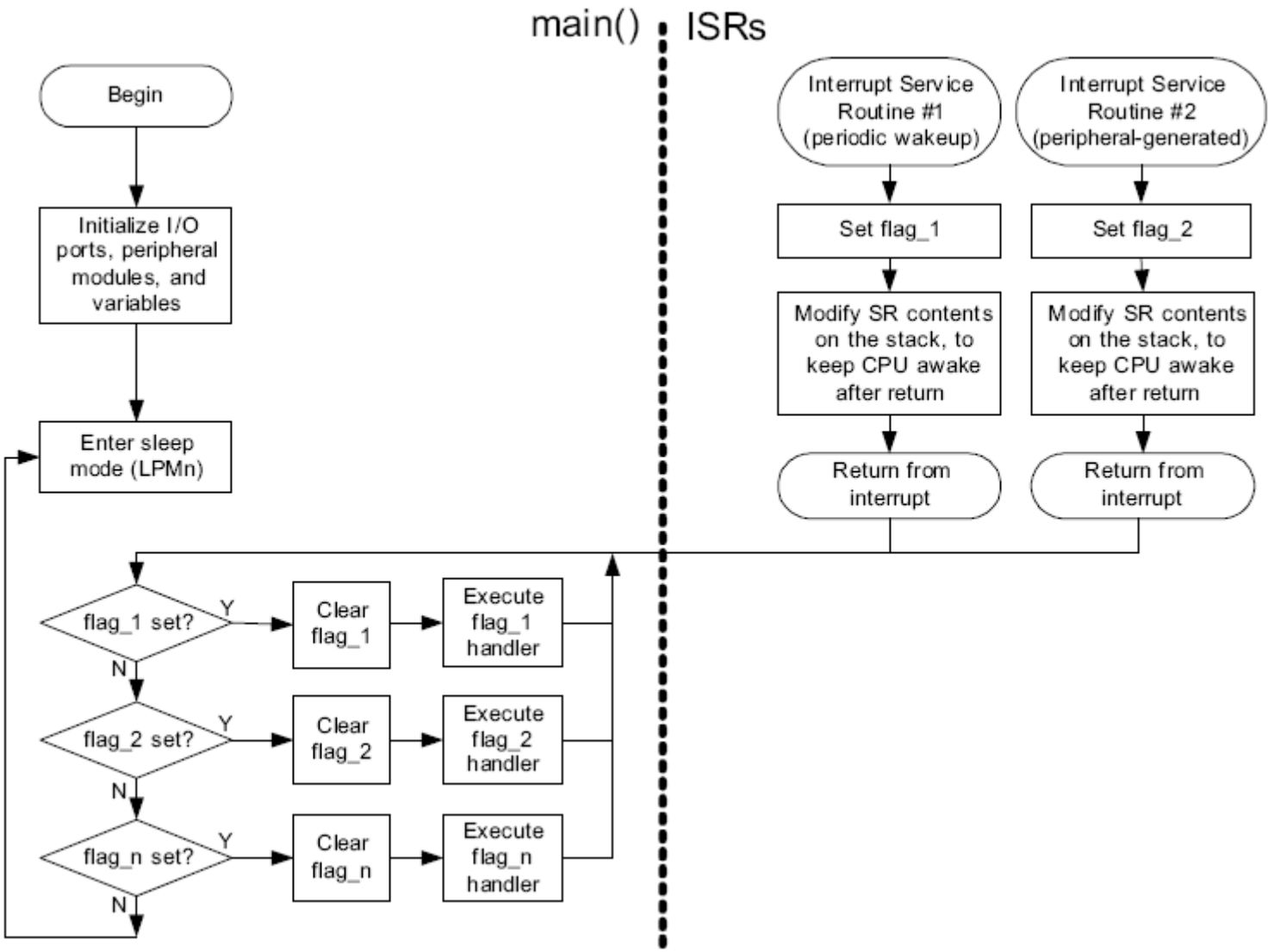
- výhoda tohto usporiadania spočíva v jednoduchom fakte: ak nastane prerušenie, odloží sa aktuálna hodnota SR a teda aj nastavený nízkopríkonový režim do zásobníka, stavový register SR sa vymaže a tým prejde procesor do aktívneho režimu, počas ktorého obslúži prerušenie

- po ukončení ISR dôjde k obnoveniu pôvodného obsahu registra SR a tým aj k opätovnému prechodu do zvoleného nízkopríkonového režimu

- tento tok programu však môže programátor ľahko zmeniť, ak počas vykonávania ISR zmení hodnotu registra SR, ktorá je odložená v zásobníku

- to znamená, že programátor môže zvoliť dve metódy spracovávania udalostí:
  - buď napíše zdrojový kód pre úplnú obsluhu udalosti priamo do ISR
  - alebo v ISR iba nastaví príznak udalosti a zmení odloženenú hodnotu SR tak, aby procesor po ukončení zostal v aktívnom režime a v rámci funkcie main() vykonal po otestovaní príznakov jednotlivých udalostí obsluhu udalosti, ktorá nastala

:: Architektúra zdrojového kódu MSP430



:: Poznámky k tvorbe softvéru

- pravidlo: first thing first
  - *jednou z prvých vecí po štarte programu by malo byť nastavenie WDT, ktorý je po resete aktívny a ak nezmeníme jeho nastavenie, resp. ak ho nedeaktivujeme, dôjde po jeho expirácii k resetu a následne k vytvoreniu nekonečnej slučky resetov*
  - *ďalšou dôležitou vecou je korektné nastavenie jednotlivých oscilátorov a celého systému generovania hodinových signálov*
- pravidlo: používame štandardné definície z hlavičkových súborov dodaných výrobcom procesora
  - *hlavičkové súbory obsahujú prehľadne usporiadané konštanty pre všetky registre a bity zvoleného procesora, pričom názvy registrov a bitov zodpovedajú názvom, ktoré sú uvedené v technickej špecifikácii procesora, čo výrazne uľahčuje písanie zdrojového kódu a zlepšuje jeho čitateľnosť*

- pravidlo: využívame intrinzické funkcie implementované v jazyku C pre daný procesor
  - *poskytujú možnosť vykonať niektoré nastavenia a kritické úlohy veľmi efektívne*
  - *najjednoduchším príkladom kritickej úlohy, ktorú môžeme vykonať efektívne s použitím intrinzickej funkcie je vstup/výstup do/z nízkopríkonového režimu*

```
_bis_SR_register(LPM3_bits + GIE);
```

  - *alebo ak potrebujeme modifikovať hodnotu bitov registra SR uloženú v zásobníku tak, aby zostal procesor po ukončení ISR v aktívnom stave:*

```
_bic_SR_register_on_exit(CPUOFF);
```

- pravidlo: využívame intrinzické funkcie implementované v jazyku C pre daný procesor
  - *môžeme ich využiť ak potrebujeme v zdrojovom kóde vytvoriť časove oneskorenie, ktoré bude trvať presne daný počet cyklov procesora (parameter definujúci počet cyklov musí byť konštanta v čase prekladu):*

```
_delay_cycles(unsigned long);
```

  - *alebo ak potrebujeme povoliť alebo zakázať prerušenia:*

```
_enable_interrupts(void);
_disable_interrupts(void);
```

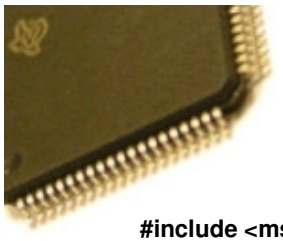
- pravidlo: využívame nízkoúrovňové inicializačné funkcie (Low-level Initailization Function)
  - *keď kompilátor prekladá zdrojový kód z jazyka C a generuje asembler, vytvorí na začiatku kód, ktorý inicializuje všetky deklarované pamäťové miesta a napĺňa ich požadovanými hodnotami. Tento kód je umiestnený pred prvou inštrukciou funkcie main().*
  - *ak je množstvo deklarovanej pamäte veľké (množstvo premenných, dlhé polia a pod.) môže tento postup predstavovať problém z hľadiska watchdogu*
  - *čas potrebný na inicializáciu dlhého zoznamu premenných môže byť totiž taký dlhý, že dôjde k expirácii časovača WDT ešte pred vykonaním prvej inštrukcie funkcie main()*

- pravidlo: využívame nízkoúrovňové inicializačné funkcie (Low-level Initailization Function)
  - *tzn. že nikdy nedôjde k vykonaniu zdrojového kódu, ktorý inicializuje WDT (obvykle umiestnený na začiatku funkcie main) následkom čoho vznikne nekonečná slučka, v ktorej sa bude procesor neustále resetovať*
  - *pozn.: z hľadiska rýchlosti počítadla WDT sa tento problém týka iba procesorov MSP430, ktoré disponujú RAM pamäťou väčšou ako 2kB*
  - najjednoduchší spôsob ako zabrániť vzniku tohto problému je použiť direktívu kompilátora, ktorá zakáže inicializáciu pamäťových elementov, ktoré nie je potrebné inicializovať, napr.:

```
__no_init int x_array[2500];
```

- pravidlo: využívame nízkoúrovňové inicializačné funkcie (Low-level Initailization Function)
  - *ak takáto direktíva nie je v danom vývojovom prostredí implementovaná, je možné využiť **nízkoúrovňovú inicializačnú funkciu**, ktorá zabezpečuje, že jej telo bude vykonané pred samotnou inicializačnou časťou zdrojového kódu, ktorú vkladá kompilátor*
  - *tým zabezpečíme, že incializácia WDT prebehne ešte pred inicializáciou premenných, napr.:*

```
void __low_level_init(void)
{
        WDTCTL = WDTPW+WDTHOLD;
}
```



:: Príklad použitia nízkoúrovňovej inicializačnej funkcie

```
#include <msp430x16x.h>

int x_array[2500];                          // inicializacia pola zaberie viac ako 32ms.  32ms je expiracna perioda WDT!
unsigned int i,x;

void main(void)
{
//WDTCTL = WDTPW+WDTHOLD;                   // ak umiestnime zakazanie WDT sem namiesto do funkcie __low_level_init()
                                            // dojde k vytvoreniu nekonecnej resetovacej slucky

  P1DIR = 0x01;                             // konfiguracia vystupu pre LED
  x_array[0] = 1;                           // ak pole nie je v aplikacii pouzite, kompilator by ho nevytvoril

  // nasledovna slucka zabezpecuje blikanie LED, najskor rychlo potom coraz pomalsie a cyklus sa opakuje
  while(1)
  {
    P1OUT ^= 0x01;                                      // zmenime stav na LED
    TACCR1=TACCR1+400;                                  // predlzime oneskorenie
    if(TACCR1>18000) TACCR1=0;                          // ak je blikanie uz prilis pomale, zacneme znova
    TACCTL1 = CCIE;                                     // komparacny rezim, povolene prerusenie
    TACTL = TASSEL_2 + TACLR + +ID_3 + MC_2;            // SMCLK/4, zmazanie TA, rezim kontinualneho pocitania
    _bis_SR_register(LPM0_bits+GIE);                    // vstup do rezimu LPM0, cakame na prerusenie CCR1
   }
}

// telo nizkourovnovej funkcie bude vykonane pred inicializaciou pamate
void __low_level_init(void)
{
  WDTCTL = WDTPW+WDTHOLD;                               // zastavenie casovaca WDT
}

// obsluzna rutina prerusenia casovaca A1
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A(void)
{
   _bic_SR_register_on_exit(LPM0_bits);                 // po ukonceni ISR zostane procesor v aktivnom stave
   CCTL1 &= ~CCIFG;                                     // zmazeme priznak prerusenia CCR1
}
```

2. časť – RTOS (Real Time Operating System)

:: Definícia RTOS

- operačný systém pracujúci v reálnom čase (RTOS – **R**eal-**T**ime **O**perating **S**ystem) je knižnica funkcií, ktorá implementuje pravidlá a stratégie týkajúce sa časovo kritického prideľovania zdrojov procesora aplikačnému softvéru

- primárnou funkciou RTOS je prideľovanie systémových zdrojov akými sú výpočtový čas CPU, pamäťové prostriedky, periférne moduly procesora a pod.

- RTOS určuje, ktoré aplikácie budú vykonávané a v akom poradí a aký časový interval bude každej aplikácii pridelený predtým, než sa riadenie odovzdá ďalšej aplikácii

:: Základné pojmy

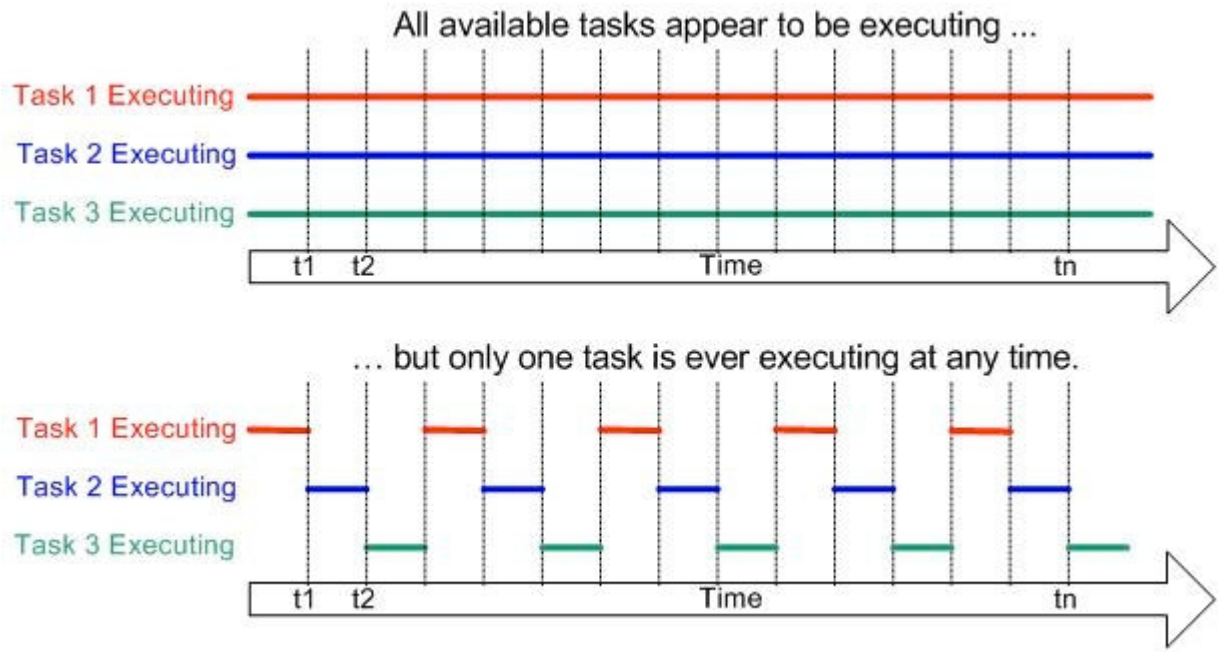
- **kernel** (jadro)
  - *predstavuje základný komponent operačného systému*
- **task** (úloha)
  - *v zásade každý vykonateľný program, ktorého vykonávanie riadi operačný systém*
- **multitasking**
  - *schopnosť operačného systému vykonávať pseudoparalelne viacero úloh*

:: Základné pojmy

- použitie RTOS s multitaskingom zjednodušuje návrh zložitých softvérových aplikácií:
  - *multitasking umožňuje rozdeliť zložitú aplikáciu na súbor menších a ľahšie manažovateľných úloh*
  - *takéto rozdelenie aplikácie umožňuje jednoduchšie testovanie softvéru, jednoduchšiu distribúciu úloh medzi členov vývojového tímu a ľahšie využívanie už napísaných častí kódu*
  - *je možné odstrániť zložité časovanie a detaily sekvenčného radenia jednotlivých úloh z aplikačného softvéru, pretože za tieto úlohy bude zodpovedať použitý RTOS*

:: Multitasking vs Concurrency

- bežné procesory sú schopné v danom okamihu vykonávať iba jednu úlohu
- pri rýchlom prepínaní medzi jednotlivými úlohami sa bude javiť, že všetky úlohy sú vykonávané paralelne



:: Scheduling (plánovanie)

- **plánovač (scheduler)** je časťou kernelu zodpovednou za rozhodovanie, ktorá úloha bude v danom čase vykonávaná

- **kernel** môže pozastaviť a neskôr znovu spustiť vykonávanie určitej úlohy viackrát počas jej úplného vykonania (task lifetime)

- **stratégia plánovania (scheduling policy)** predstavuje algoritmus, ktorý používa scheduler pri rozhodovaní, ktorá úloha má byť v danom čase vykonávaná

:: Scheduling (plánovanie)

- statické plánovanie
  - *kompletný plán je určený pred začatím vykonávania jednotlivých úloh*
  - *metódy statického plánovania:*
    - As-soon-as-possible (ASAP)
      - *úlohy sú vykonávané v najskoršom možnom čase*
    - As-late-as-possible (ALAP)
      - *pri danom maximálnom celkovom oneskorení reakcie jednotlivých úloh je možné naplánovať ich spúšťanie v najneskoršom možnom čase*
- dynamické plánovanie
  - *využíva priority priradené jednotlivým úlohám k dynamickému rozhodovaniu, ktorá úloha bude vykonávaná v ďalšom čase*

:: Scheduling (plánovanie)

- existujú dve základné stratégie plánovania:
  - *stratégia systému, ktorý nepracuje v reálnom čase, prideľuje každej úlohe rovnaký interval vypočtového času CPU a ostatných prostriedkov*
  - *stratégia systému, ktorý pracuje v reálnom čase (embedded systems), prideľuje vypočtový čas CPU a ostatné prostriedky na základe priority*

:: Scheduling (plánovanie)

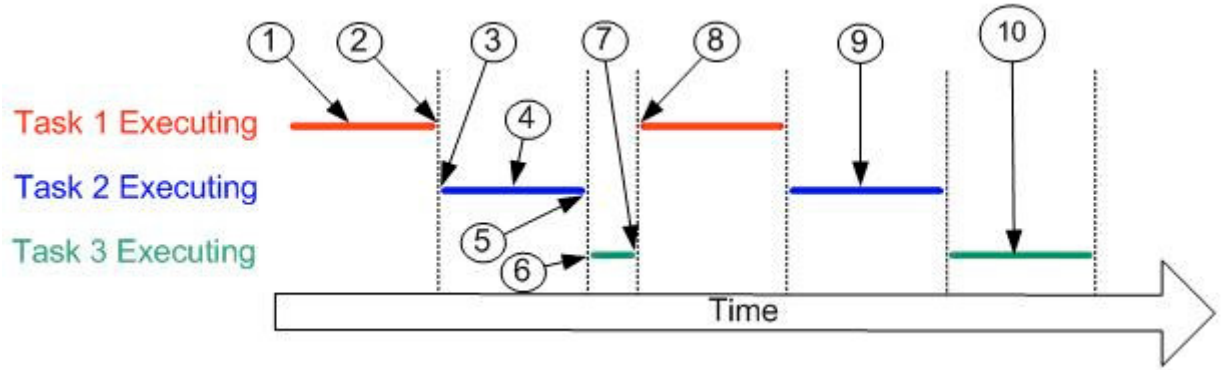
- z princípu multitaskingu vyplýva, že kernel RTOS môže pozastaviť vykonávanie každej úlohy

- navyše aj každá úloha môže pozastaviť svoje vlastné vykonávanie

- táto vlastnosť je dôležitá vtedy, keď daná úloha potrebuje čakať v rámci pevne daného intervalu (**sleep** – spiaci režim) alebo musí počkať (**block** - blokovanie), kým bude dostupný zdroj, s ktorým pracuje (napr. sériový port) alebo kým nenastane očakávaná udalosť (napr. stlačenie tlačidla)

- blokovaná alebo spiaca úloha nepokračuje vo vykonávaní zdrojového kódu a nevyužije tak procesorový čas

:: Scheduling (plánovanie) - príklad



- V (1) je vykonávaná úloha 1, v (2) jadro pozastaví úlohu 1 a v (3) znovu spustí úlohu 2.
- Počas vykonávania úlohy 2 (4), táto úloha uzamkne určité periférne časti procesora iba pre svoje použitie.
- V (5) jadro pozastaví úlohu 2 a v (6) znovu spustí 3.
- Úloha 3 sa pokúša pristupovať k rovnakým periférnym častiam procesora, zistí, že sú uzamknuté inou úlohou, nemôže teda pokračovať a sama sebe pozastaví činnosť (7).
- V (8) jadro znovu spustí úlohu 1, atď.
- V ďalšom časovom intervale je vykonávaná úloha 2 (9), ktorá dokončí prácu s uzamknutými perifémymi časťami procesora a odomkne ich.
- V ďalšom časovom intervale je vykonávaná úloha 3 (10), ktorá zistí, že už môže pristupovať k požadovaným perifémym častiam procesora, preto pokračuje vo vykonávaní svojho zdrojového kódu až kým ju nepozastaví jadro RTOS.

:: Context Switching (prepínanie kontextu)

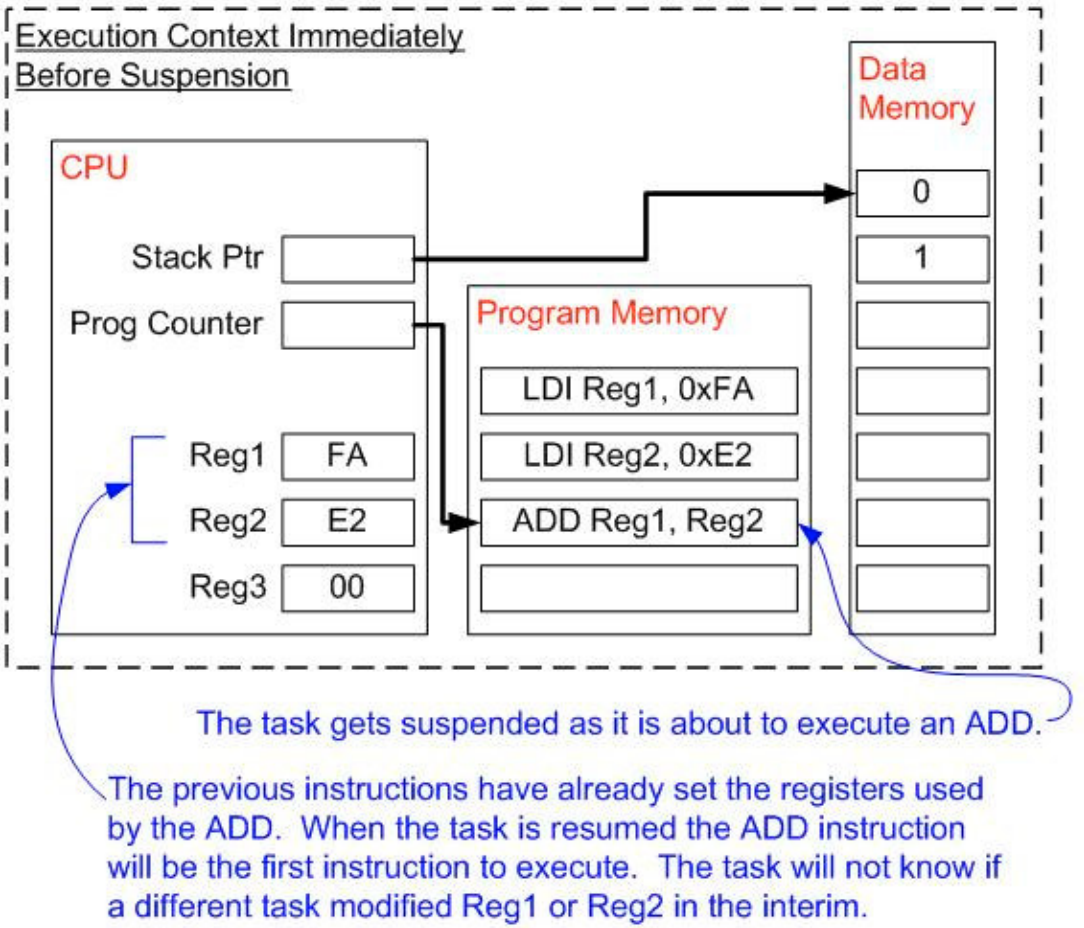
- každá úloha počas vykonávania prirodzene využíva registre, RAM a FLASH pamäť procesora tak ako každý iný program

- všetky tieto využívané zdroje spoločne predstavujú tzv. kontext úlohy (task execution context)

- úloha je sekvenčný úsek zdrojového kódu, preto nevie, že sa blíži okamih, kedy ju kernel pozastaví alebo znova spustí a dokonca ani nevie, kedy sa to stalo

:: Context Switching (prepínanie kontextu)

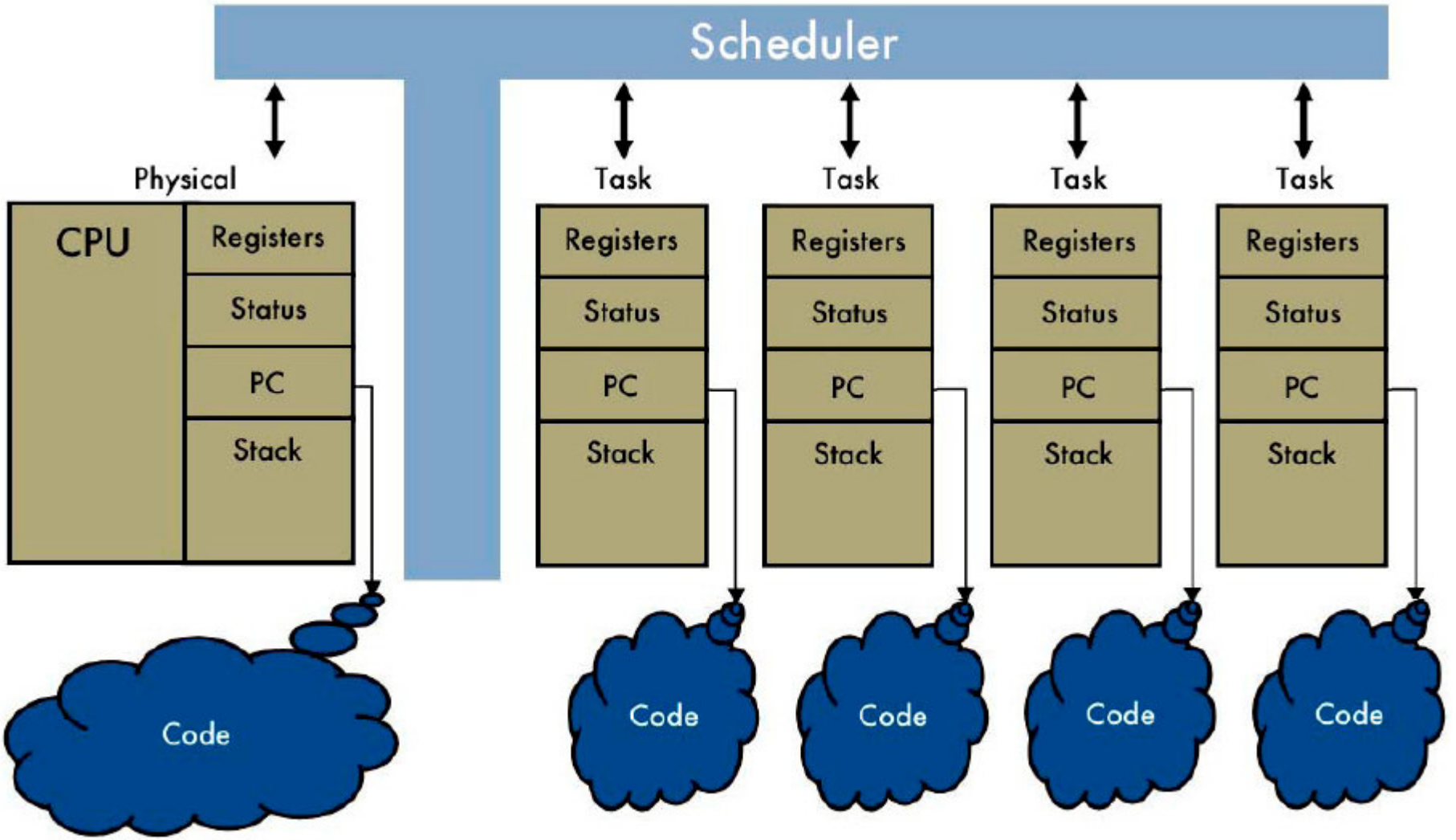
- zvážme príklad úlohy, ktorá je pozastavená okamžite predtým ako vykoná inštrukciu, ktorá sčíta hodnoty v dvoch registroch procesora:



:: Context Switching (prepínanie kontextu)

- pokým je daná úloha pozastavená, iná vykonávaná úloha môže modifikovať hodnoty v registroch procesora
- po opätovnom spustení pôvodnej úlohy táto nebude vedieť, že došlo k modifikácii hodnôt v registroch, s ktorými pracuje a inštrukcia sčítania poskytne nekorektné výsledky
- aby sme zabránili tomuto typu chýb, je dôležité, aby mala úloha po opätovnom spustení kontext identický tomu, ktorý existoval tesne predtým ako bolo pozastavené jej vykonávanie
- uvedený proces zabezpečuje kernel tým, že ukladá kontext úlohy po jej pozastavení a predtým ako kernel znovu úlohu spustí obnoví jej uložený kontext
- proces ukladania kontextu pozastavovanej úlohy a obnovovania spúšťanej úlohy nazývame prepínanie kontextu (context switching)

:: Context Switching (prepínanie kontextu)



:: Aplikácie pracujúce v reálnom čase

- základné rozdelenie aplikácií pracujúcich v reálnom čase (RTA - **R**eal **T**ime **A**pplication):
  - *Hard RTA, napr. riadiaci systém jadrového reaktora*
  - *Firm RTA, napr. riadiaci systém továrne na spracovanie potravín*
  - *Soft RTA, napr. animácie na grafickom displeji prebiehajúce v reálnom čase*

:: Aplikácie pracujúce v reálnom čase

- aplikácie pracujúce v reálnom čase sú navrhované tak, aby poskytovali čo najrýchlejšiu odpoveď na udalosti, ktoré v reálnom svete nastali, a na ktoré musí aplikácia reagovať

- je dôležité si uvedomiť, že mnohé udalosti, ktoré v reálnom svete nastanú zároveň majú definovaný čas, do uplynutia ktorého musí aplikácia zareagovať

- v prípade RTOS musí byť preto stratégia plánovania (scheduling policy) navrhnutá tak, aby RTOS bol schopný reagovať na všetky eventuálne udalosti v reálnom čase

:: Aplikácie pracujúce v reálnom čase

- aby bolo možné dosiahnuť tento cieľ, musí návrhár predovšetkým priradiť prioritu každej jednotlivej úlohe

- stratégia plánovania RTOS potom jednoducho zabezpečuje, aby úloha s najvyššou prioritou, ktorá je pripravená na vykonanie, dostala pridelený výpočtový čas CPU

- v prípade, kedy má viacero úloh rovnakú prioritu a zároveň sú všetky pripravené na vykonanie, musí RTOS zabezpečiť, aby bol všetkým týmto úlohám pridelený výpočtový čas CPU rovnomerne

:: Aplikácie pracujúce v reálnom čase - príklad

- predstavme si systém pracujúci v reálnom čase, ktorý zahŕňa klávesnicu a LCD displej

- užívateľ musí získať spätnú väzbu o každom stlačení tlačidla prostredníctvom displeja v akceptovateľnom čase, inak sa o takejto aplikácii dá sotva povedať, že pracuje v reálnom čase

- je potrebné zadefinovať najdlhší interval reakcie systému, ktorý je ešte akceptovateľný, napr. 100ms – potom každá reakcia v intervale 0 až 100ms bude akceptovateľná

:: Aplikácie pracujúce v reálnom čase - príklad

- uvedenú funkcionalitu môžeme implementovať ako autonómnu úlohu s nasledujúcou štruktúrou:

```
void vKeyHandlerTask (void *pvParameters )
{
// snimanie stavu klavesnice je kontinualny proces
// uloha je preto implementovana ako nekonecna slucka
// tak ako mnohe ulohy, ktore musia pracovat v realnom case
    for( ; ; ) {
            [Úloha pozastavená; čakanie na stlačenie tlačidla]
            [Spracovanie stlačenia tlačidla]
    }
}
```

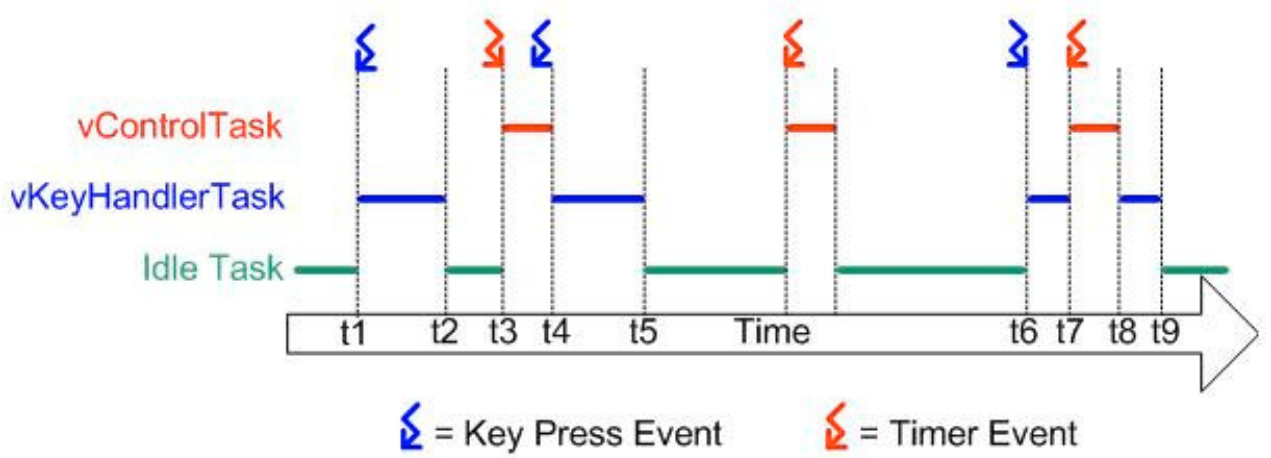
:: Aplikácie pracujúce v reálnom čase - príklad

- teraz predpokladajme, že systém pracujúci v reálnom čase tiež zároveň vykonáva riadiacu funkciu, ktorá závisí od číslicovo filtrovaného vstupu
- vstup je potrebné vzorkovať, filtrovať a riadiaci cyklus musí byť vykonaný každé 2ms
- aby filter správne pracoval, musí byť presnosť vzorkovacej periódy 0.5ms
- uvedenú funkcionalitu môžeme implementovať ako autonómnu úlohu s nasledujúcou štruktúrou:

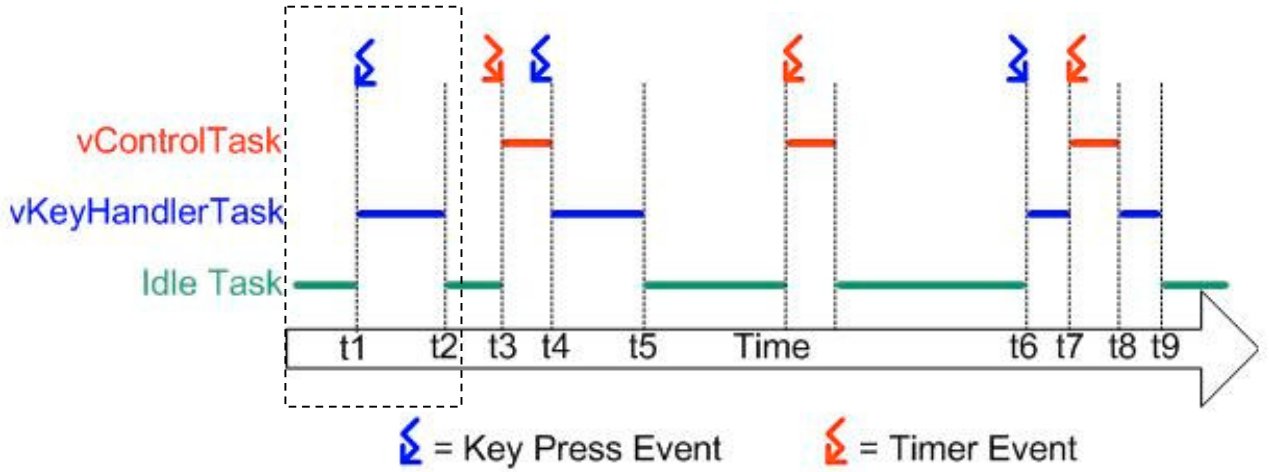
```
void vControlTask (void *pvParameters ) {
    for( ; ; ) {
            [Úloha pozastavená; čakanie v intervale 2ms od začiatku predchádzajúceho cyklu]
            [Zachytenie vzorky vstupného signálu]
            [Proces filtracie vstupných vzoriek]
            [Vykonanie riadiaceho algoritmu]
            [Výstup výsledkov]
    }
}
```

:: Aplikácie pracujúce v reálnom čase - príklad

- vývojár musí priradiť vyššiu prioritu riadiacej úlohe, keďže:
  - *časovanie riadiacej úlohy je striktnejšie než časovanie úlohy snímania stavu klávesnice*
  - *následky narušenia časovania sú výraznejšie v prípade riadiacej úlohy než v prípade úlohy snímania stavu klávesnice*
- nižšie uvedený diagram ukazuje ako budú vykonávané jednotlivé úlohy operačným systémom RTOS, ktorý navyše vytvoril ešte jednu úlohu – idle task, ktorá bude vykonávaná len v prípade, kedy žiadna iná úloha nebude pripravená na vykonanie
- **RTOS idle task** je úloha, ktorá je stále pripravená na vykonanie, ale má najnižšiu prioritu

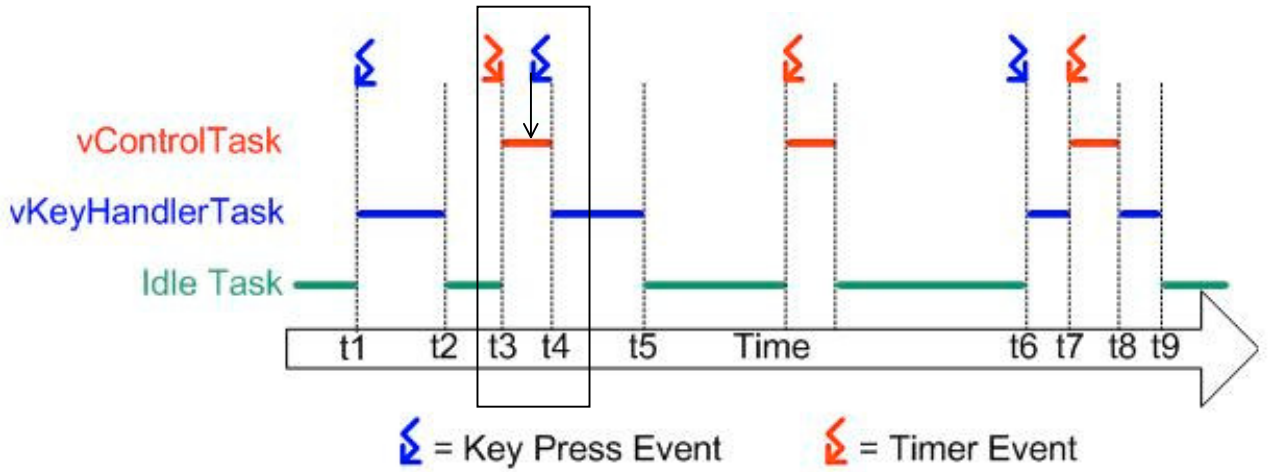


:: Aplikácie pracujúce v reálnom čase - príklad



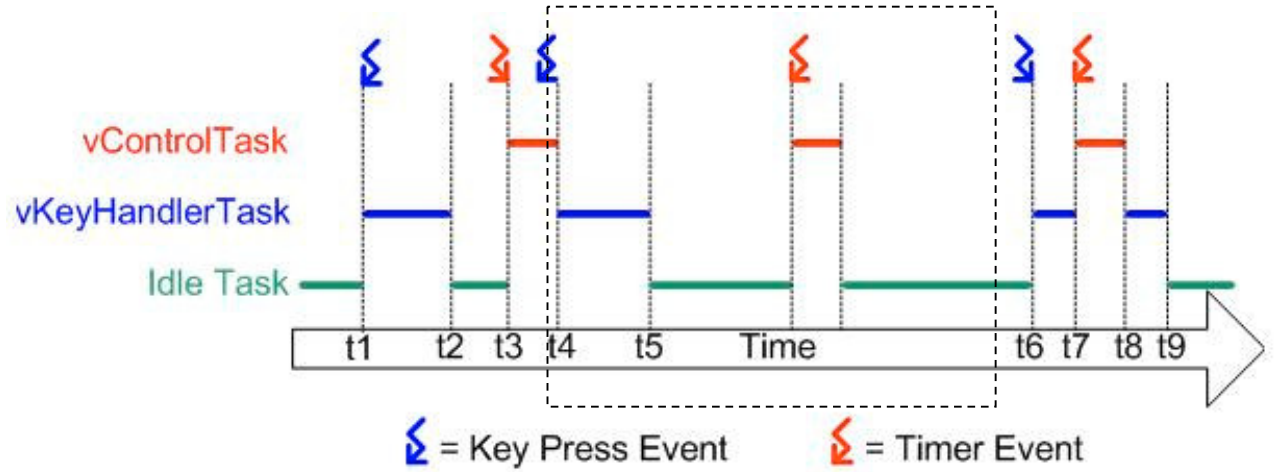
- Na začiatku programu ani jedna z našich dvoch úloh nie je pripravená na vykonanie – vControlTask čaká na správny čas začiatku vykonania riadiaceho cyklu a vKeyHandlerTask čaká na stlačenie tlačidla klávesnice. Výpočtový čas CPU preto RTOS pridelí úlohe idle.
- V čase t1, nastane stlačenie tlačidla. Úloha vKeyHandlerTask je teraz pripravená na vykonanie – má vyššiu prioritu než úloha idle a RTOS jej teda pridelí výpočtový čas CPU.
- V čase t2 dokončí vKeyHandlerTask spracovanie stlačenia tlačidla a obnovu dát zobrazovaných na displeji. Keďže nemôže pokračovať kým nebude stlačené opäť niektoré z tlačidiel klávesnice, pozastaví sama sebe činnosť a RTOS pridelí výpočotový čas CPU úlohe idle.

:: Aplikácie pracujúce v reálnom čase - príklad



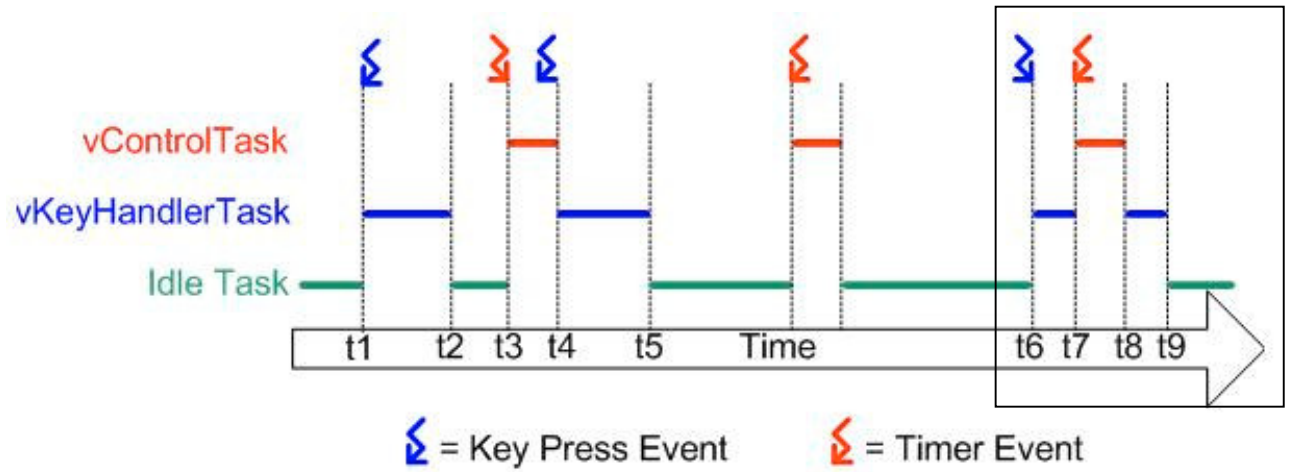
- V čase t3 došlo k udalosti časovača, ktorá indikuje, že nastal čas pre vykonanie ďalšieho riadiaceho cyklu. Úloha vControlTask je teraz pripravená na vykonanie a keďže má najvyššiu prioritu, RTOS jej okamžite pridelí výpočtový čas CPU.
- V intervale medzi okamžikmi t3 a t4, kým je úloha vControlTask stále vykonávaná, dôjde k stlačeniu tlačidla. Úloha vKeyHandlerTask je teraz pripravená na vykonanie, ale keďže má nižšiu prioritu než úloha vControlTask nie je jej zatiaľ pridelený výpočotový čas CPU.

:: Aplikácie pracujúce v reálnom čase - príklad



- V čase t4 dokončí úloha vControlTask riadiaci cyklus a nemôže pokračovať v činnosti, kým nedôjde k ďalšej udalosti časovača, preto pozastaví svoju činnosť. V tomto okamihu je úloha vKeyHandlerTask úloha s najvyššou prioritou, ktorá je pripravená na vykonanie, RTOS jej teda pridelí výpočtový čas CPU, aby mohla spracovať predchádzajúce stlačenie tlačidla.
- V čase t5 úloha vKeyHandlerTask dokončila spracovanie stlačenia tlačidla, pozastavila svoju činnosť a čaká na ďalšiu udalosť stlačenia tlačidla. Opäť ani jedna z našich úloh nie je pripravená na vykonanie, RTOS teda pridelí výpočtový čas CPU úlohe idle.
- V intervale medzi okamžikmi t5 a t6 došlo k spracovaniu udalosti časovača, ale nedošlo k žiadnemu stlačeniu tlačidla.

:: Aplikácie pracujúce v reálnom čase - príklad

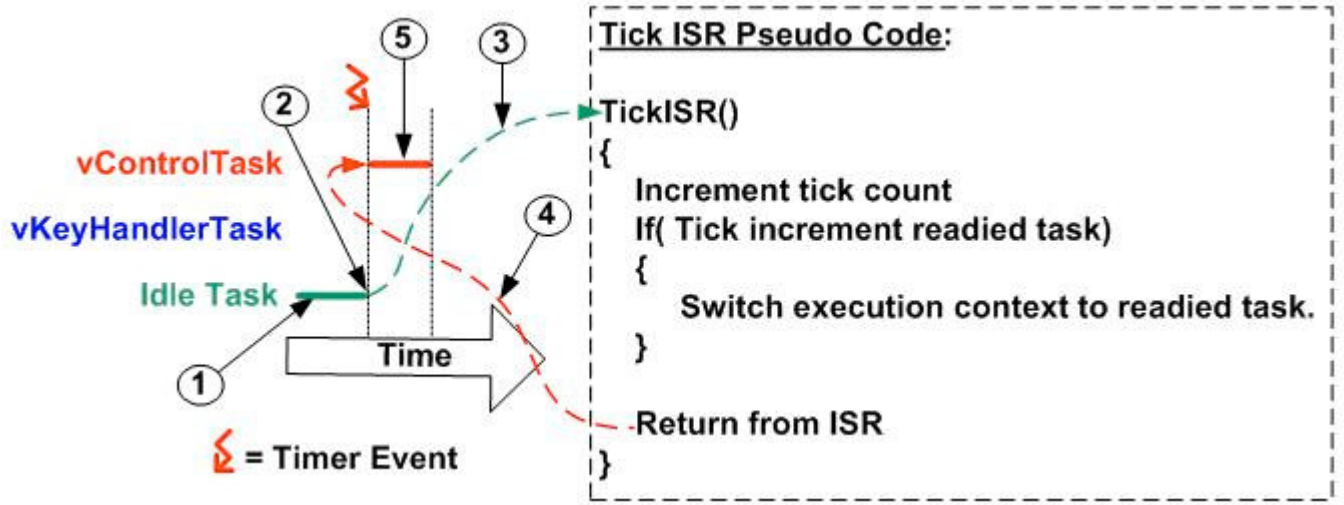


- K ďalšiemu stlačeniu tlačidla došlo v čase t6, ale predtým, než úloha vKeyHandlerTask stihla dokončiť spracovanie stlačenia tlačidla, došlo k udalosti časovača. V tomto okamihu sú obe úlohy pripravené na vykonanie. Keďže úloha vControlTask má vyššiu prioritu, RTOS pozastaví úlohu vKeyHandlerTask predtým ako dokončí spracovanie stlačenia tlačidla a pridelí výpočtový čas CPU úlohe vControlTask.
- V čase t8 úloha vControlTask dokončila riadiaci cyklus, pozastaví svoju činnosť a čaká na ďalšiu udalosť časovača. Úloha vKeyHandlerTask je teraz úlohou s najvyššou prioritou, ktorá je pripravená na vykonanie, RTOS jej teda pridelí výpočtový čas CPU a úloha dokončí rozpracovanú obsluhu stlačenia tlačidla.

:: Časové značky RTOS – RTOS Tick

- Úloha nachádzajúca sa v režime sleep špecifikuje časový interval, po uplynutí ktorého požaduje od RTOS aktiváciu (waking).
- Úloha nachádzajúca sa v režime block môže špecifikovať maximálny časový interval v rámci ktorého potrebuje čakať.
- Bežné RTOS, napr. FreeRTOS merajú reálny čas s využitím premennej označovanej ako tick count. Prerušenie od časovača (RTOS tick interrupt) inkrementuje premennú tick count s konštantnou presne danou periódou, čím umožňuje RTOS merať čas s rozlíšením daným frekvenciou prerušení od časovača.
- Zakaždým, keď dôjde k inkrementácii premennej tick count, musí RTOS skontrolovať, či nenastal čas k odblokovaniu alebo zobudeniu niektorej úlohy.
- Je dôležité si uvedomiť, že úloha, ktoré je odblokovaná/zobudená počas ISR obsluhujúcej premennú tick count, môže mať vyššiu prioritu než úloha, ktorá bola danou ISR prerušená. Ak je to tak, potom ISR obsluhujúca premennú tick count by sa mala vrátiť do znovu odblokovanej/zobudenej úlohy – t.j. k prerušeniu od časovača došlo počas vykonávania jednej úlohy, ale po ukončení ISR sa vrátime do inej úlohy.

:: Časové značky RTOS – RTOS Tick

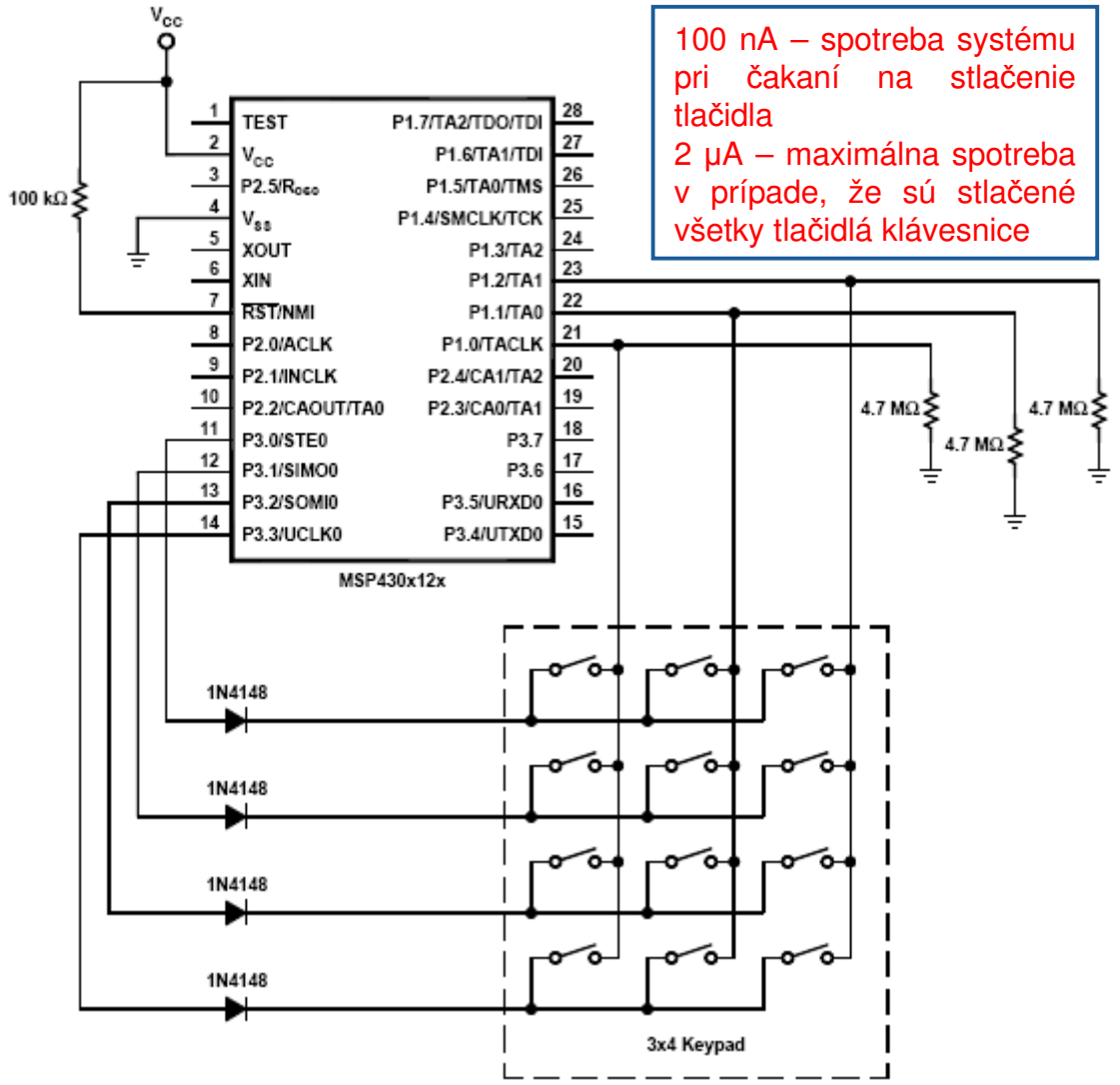


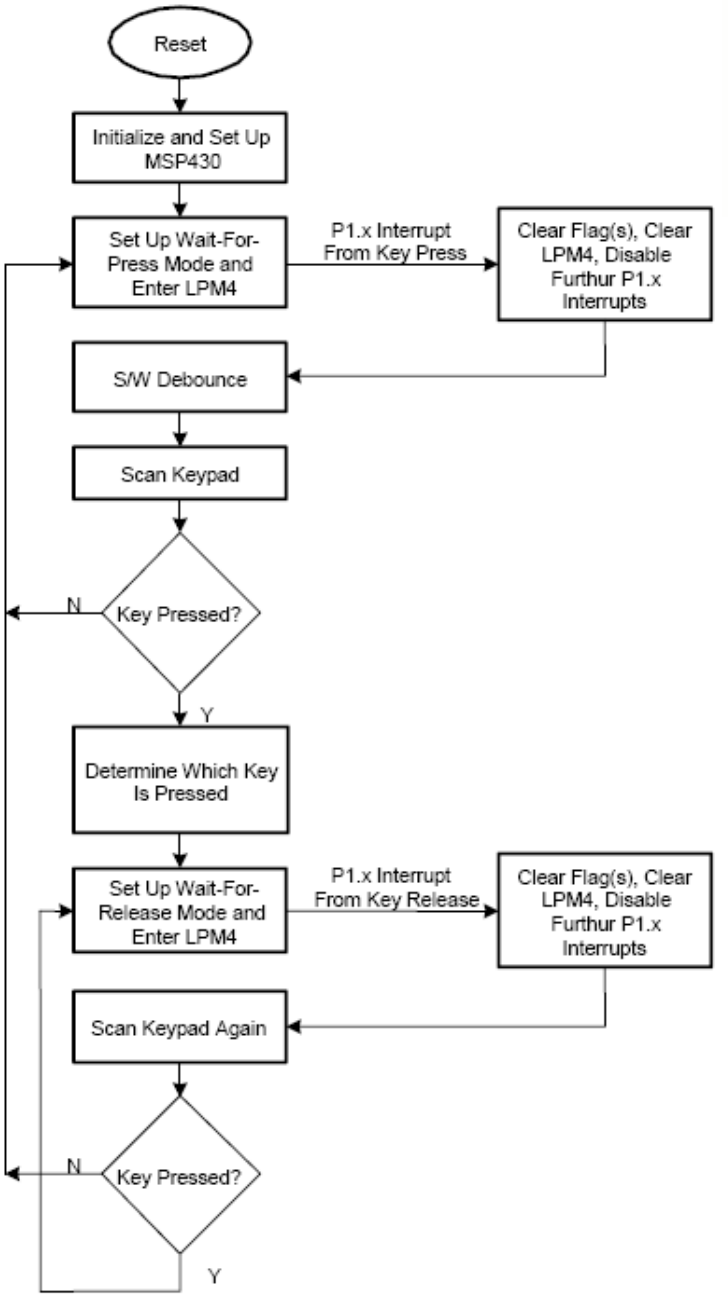
- V čase (1) je aktívna úloha idle.
- V čase (2) dôjde k udalosti RTOS tick a riadenie toku programu je odovzdané obsluhe ISR (3).
- RTOS tick ISR uvoľní úlohu vControlTask na vykonanie a keďže úloha vControlTask má vyššiu prioritu než úloha idle, dôjde k prepnutiu kontextu úlohy vControlTask.
- Keďže aktuálny kontext teraz zodpovedá kontextu úlohy vControlTask, odovzdá sa po návrate z ISR (4) riadenie úlohe vControlTask, ktorá začne okamžite vykonávať svoj zdrojový kód (5).
- Takýto spôsob prepínania kontextov nazývame preemptívne prepínanie kontextov, keďže prerušená úloha je nútene predbehnutá inou úlohou bez jej dobrovolného pozastavenia.

3. časť – Príklady aplikácii MSP430

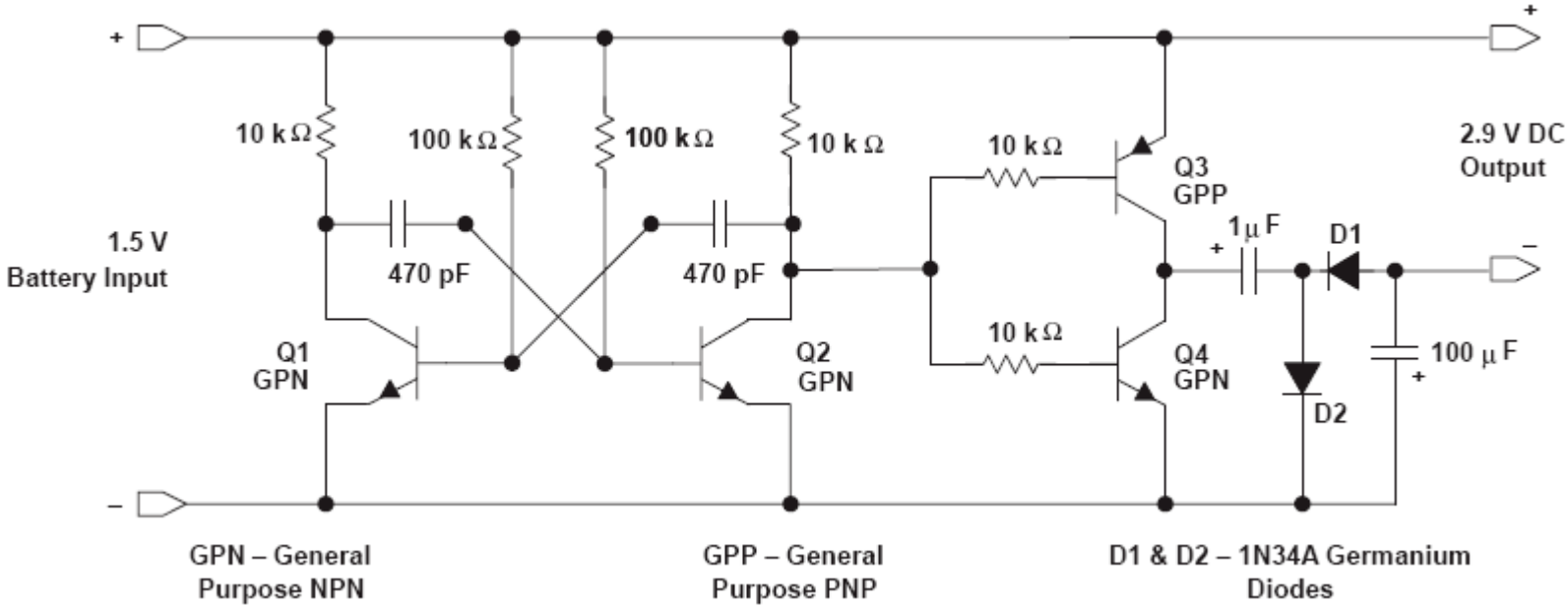
:: Nízkopríkonová maticová klávesnica 3x4

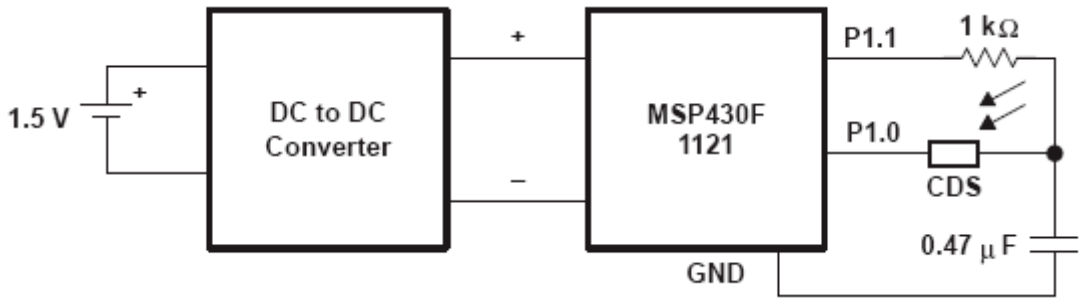
100 nA – spotreba systému pri čakaní na stlačenie tlačidla
2 µA – maximálna spotreba v prípade, že sú stlačené všetky tlačidlá klávesnice





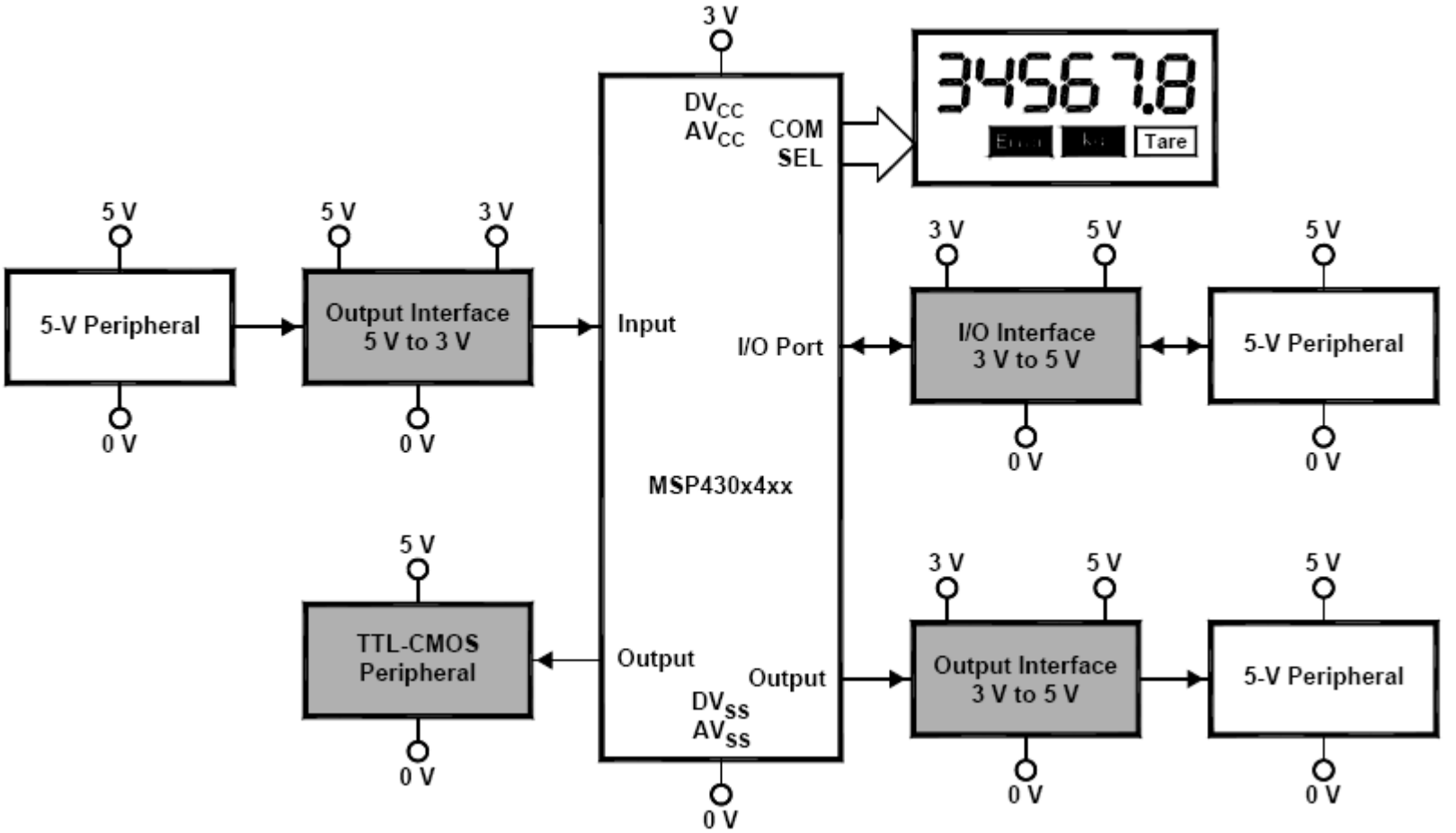
:: Step-Up DC-DC konvertor pre napájanie senzora svetla na báze MSP430





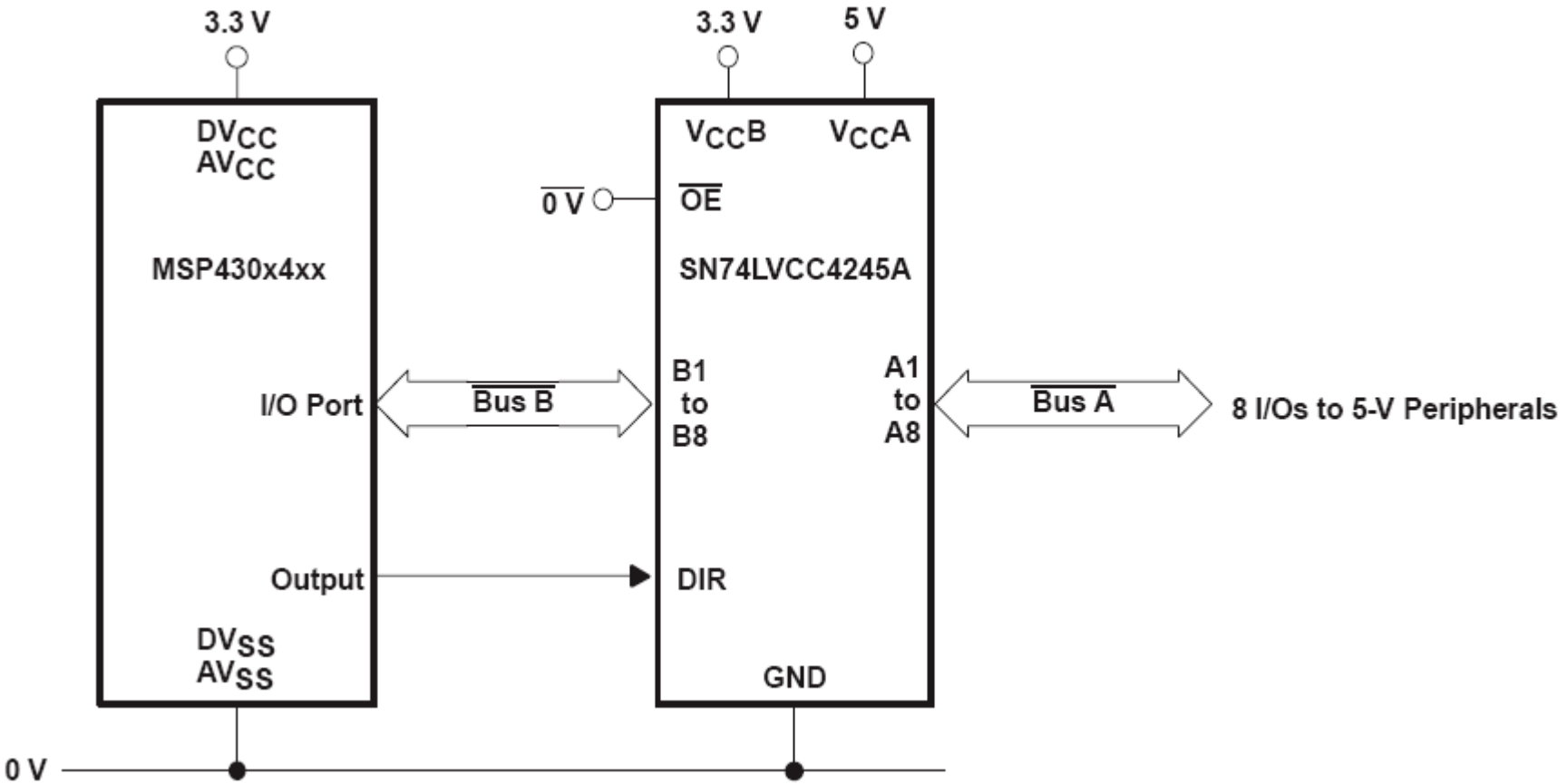
Pri použití štandardnej alkalickej batérie typu AA (s odhadovanou kapacitou 0.5 Ah) by uvedená aplikácia na báze MSP430 bola schopná pracovať nepretržite približne 1000 hodín, t.j. približne 42 dní!

:: Rozhranie medzi 3.3V a 5V systémom



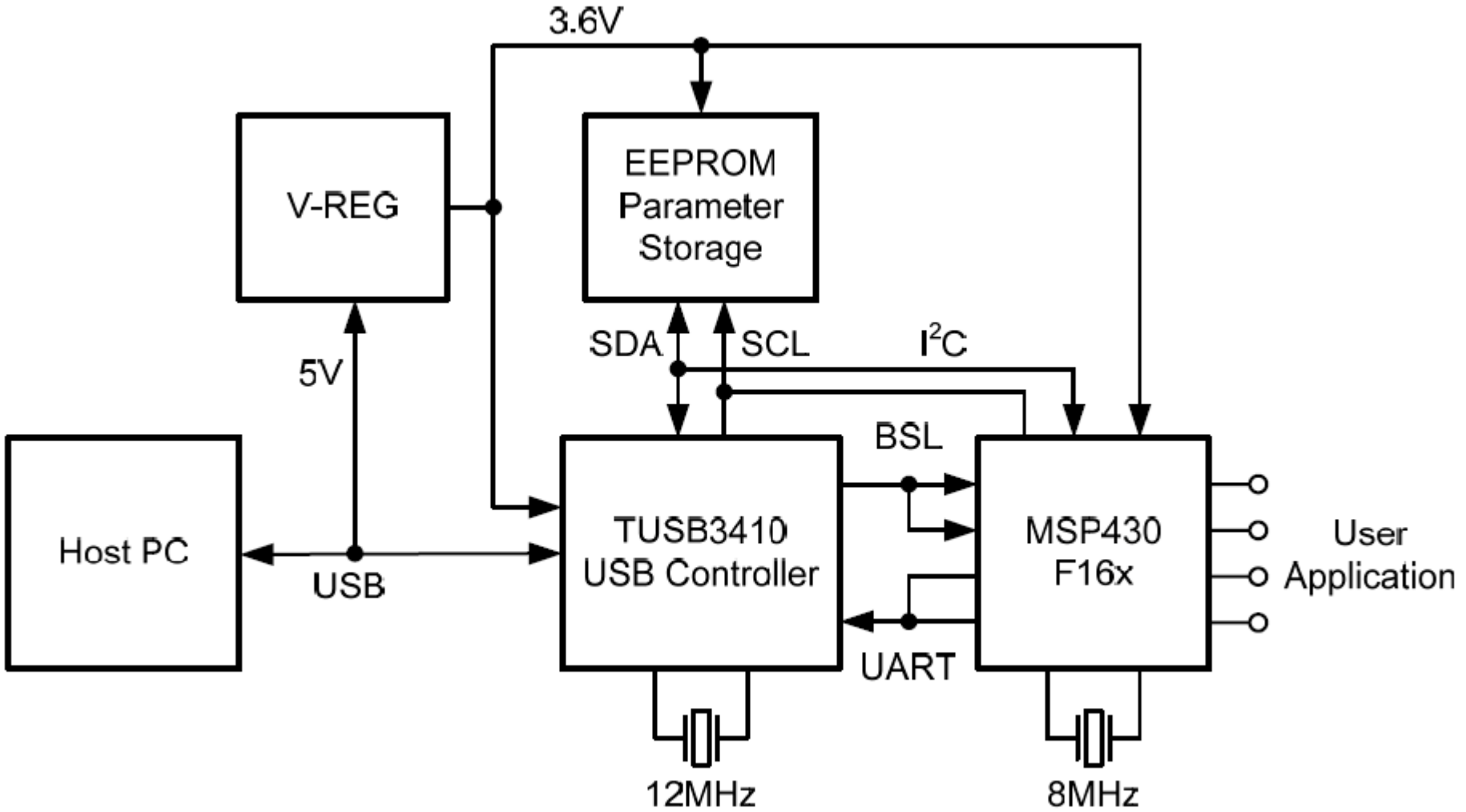
:: Rozhranie medzi 3.3V a 5V systémom

- využitie úrovňových prekladačov



:: Podpora zbernice USB pre procesory MSP430

- využitie obvodu TUSB3410



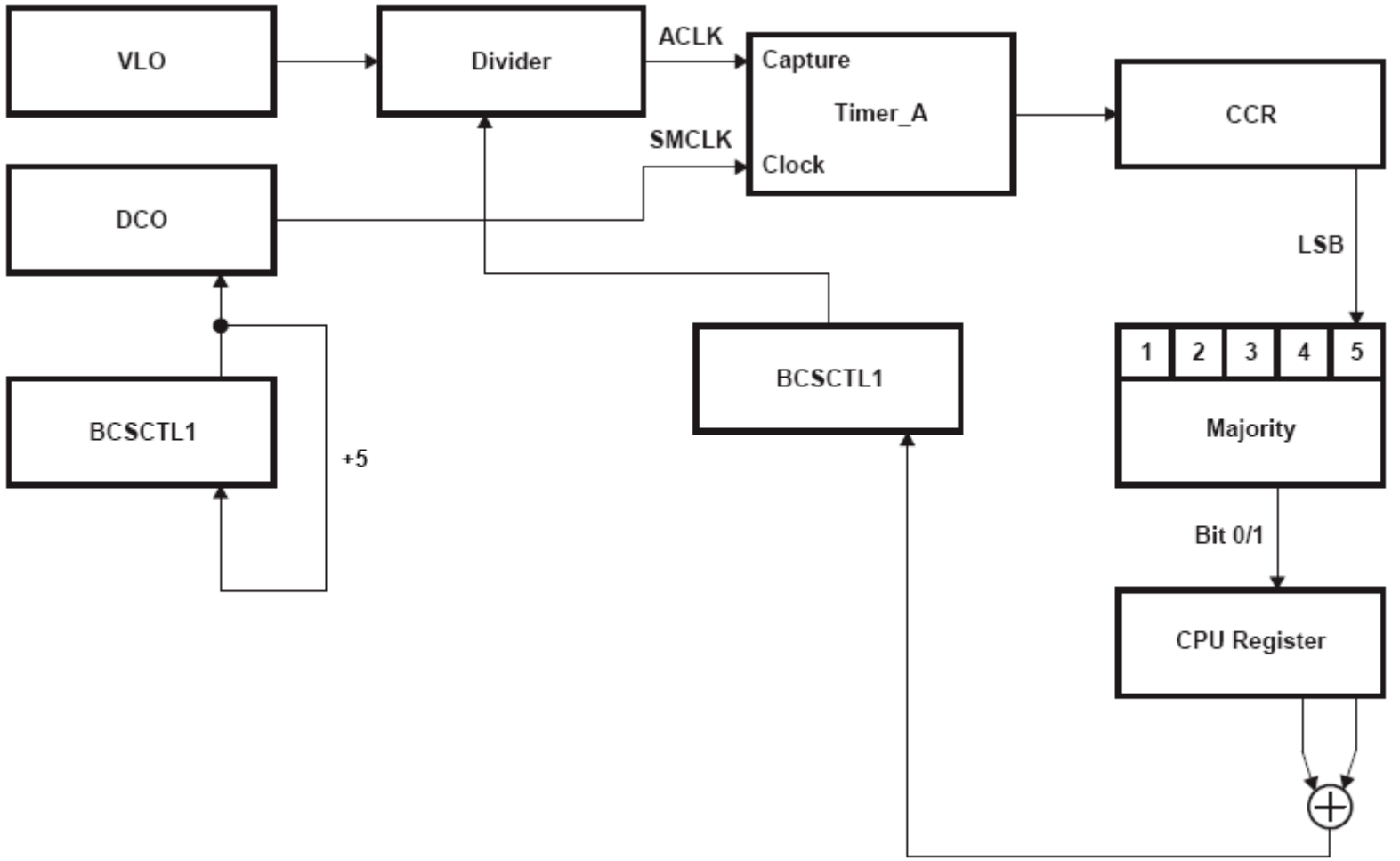
:: Generátor náhodných čísiel na báze MSP430

- nízkofrekvenčný oscilátor (VLO) a číslicovo riadený oscilátor (DCO) predstavujú dva nezávislé hodinové systémy, z ktorých každý má svoj vlastný zdroj časovania
- keďže tieto oscilátory sú nezávislé, časový rozdiel medzi prechodmi hrán týchto dvoch oscilátorov je náhodnou premennou času
- časový rozdiel medzi týmito dvomi hodinovými systémami je možné využiť pre generovanie toku náhodných bitov
- v rámci jedného cyklu oscilátora VLO bude počet cyklov oscilátora DCO približne rovnaký ale keďže oba oscilátory sú nezávislé, nie je možné predpovedať, či bude tento počet cyklov párny alebo nepárny
- dokonca nie je možné predpovedať počet týchto cyklov ani keď poznáme predchádzajúcu hodnotu
- v aplikácii je časovač A konfigurovaný tak, aby kontinuálne počítal jednotlivé cykly DCO v rámci jedného cyklu VLO a bit s najnižšou váhou (LSb) 16-bitového výsledku je použitý pri tvorbe 16-bitového náhodného slova

:: Generátor náhodných čísiel na báze MSP430

- aby sa zvýšila entropia generovaných náhodných čísiel, boli do systému zavedené ďalšie náhodné faktory:
  - *po každom cykle oscilátora VLO, je k registru BCSCTL1 pripočítané číslo 5, čím sa zmenia bity RSEL, a v dôsledku toho sa zmení rýchlosť DCO relatívne k rýchlosti VLO v rámci každého meracieho cyklu. Hodnota 5 bola určená experimentálne ako postačujúca na dosiahnutie významnej relatívnej zmeny rýchlosti DCO voči VLO.*
  - *po každom cykle oscilátora VLO, je medzi dvomi LSBs bitmi formovaného výsledku vykonaná operácia XOR a výsledok operácie je uložený na pozíciu bitov DIVA v registri BCSCTL1, ktoré riadia deliaci pomer deličky predradenej oscilátoru VLO*
  - *každý bit výsledku je v skutočnosti výsledkom väčšinového testu medzi piatimi cyklami VLO. Každý cyklus generuje náhodný bit ale až výsledok väčšinového testu piatich týchto bitov sa uloží ako platný bit výsledku.*

:: Generátor náhodných čísiel na báze MSP430



:: Kapacitný senzor ako dotyková klávesnica na báze MSP430

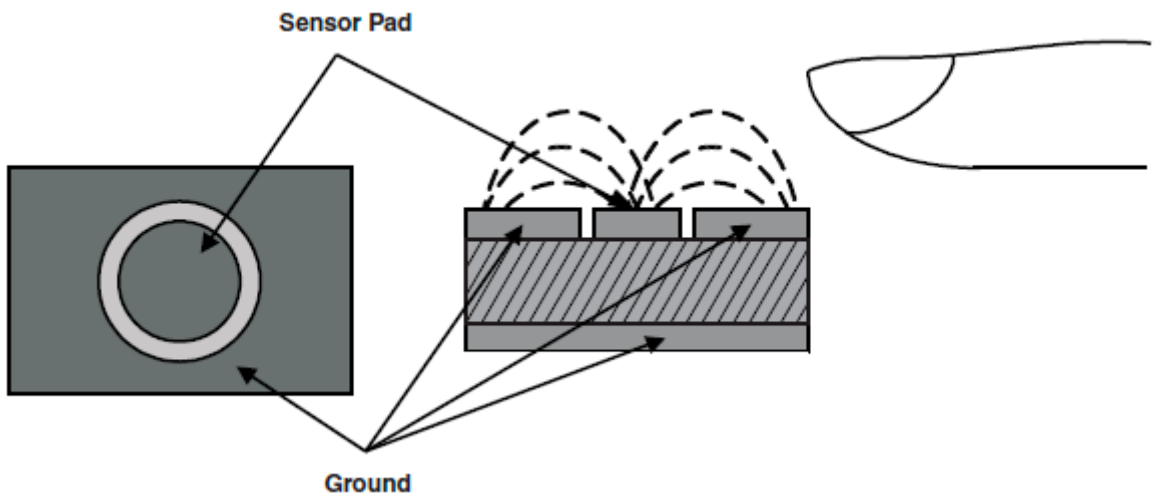
- procesory MSP430 majú niekoľko unikátnych vlastností, ktoré ich robia vhodnými pre implementáciu dotykových kapacitných senzorických systémov

- princíp činnosti dotykových kapacitných senzorických systémov je často založený na jednoduchej RC metóde, ktorá nevyžaduje žiadne špeciálne periférie procesora

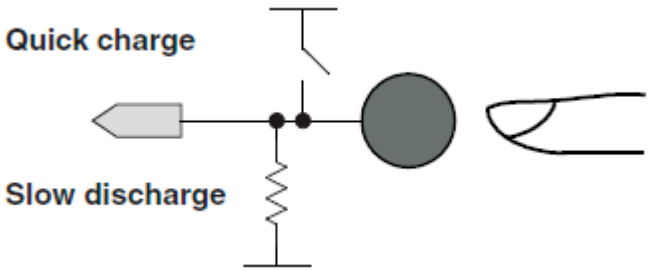
- navyše je táto metóda prirodzene „nízkopríkonová“ a teda v súčinnosti s procesormi MSP430 vytvára predpoklad pre nízkopríkonové implementácie dotykových kapacitných senzorických systémov

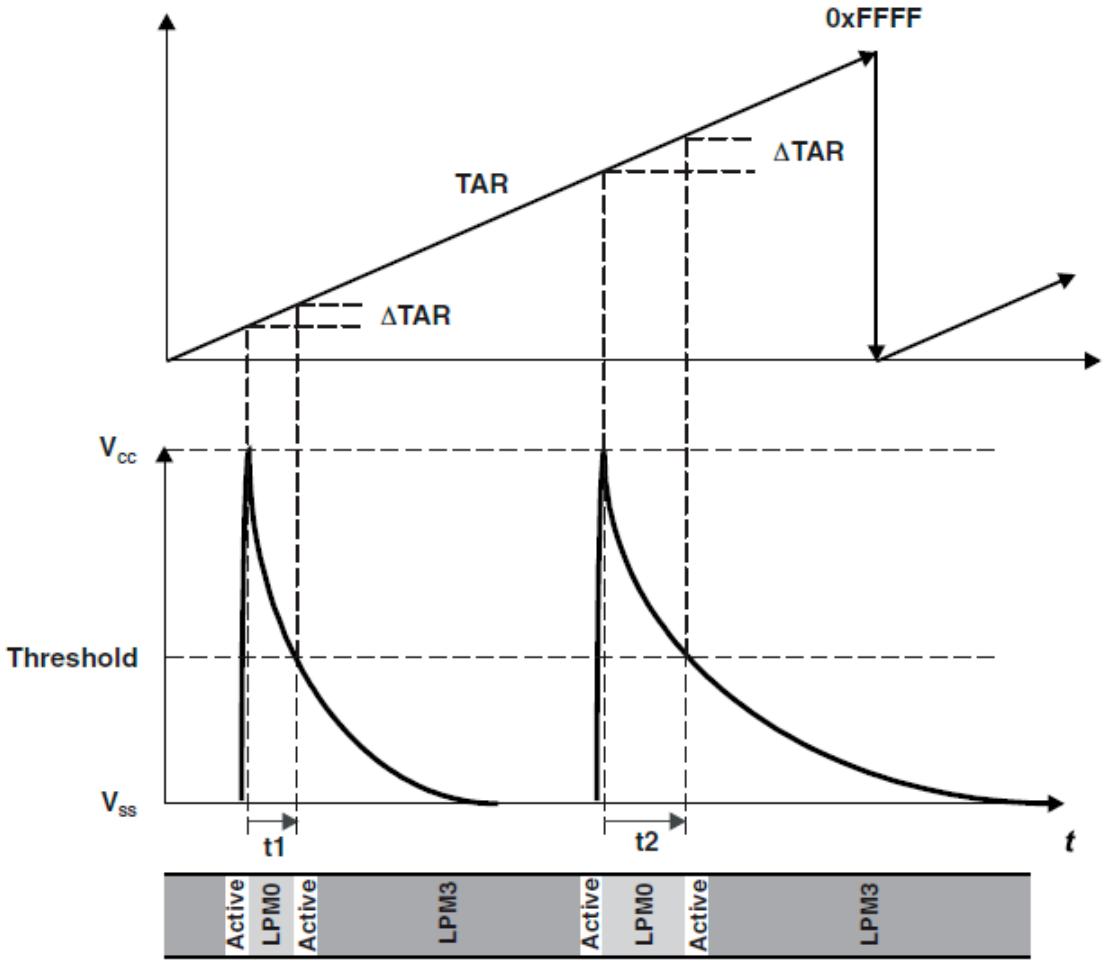
:: Kapacitný senzor ako dotyková klávesnica na báze MSP430

- kapacitný senzor je konštruovaný ako otvorený kapacitor, t.j. elektrické pole môže interferovať s vodivými objektami v okolí
- kapacita takéhoto dotykového senzora kolíše vplyvom zmeny teploty a vlhkosti, preto riadiaci systém musí tieto zmeny sledovať a kompenzovať

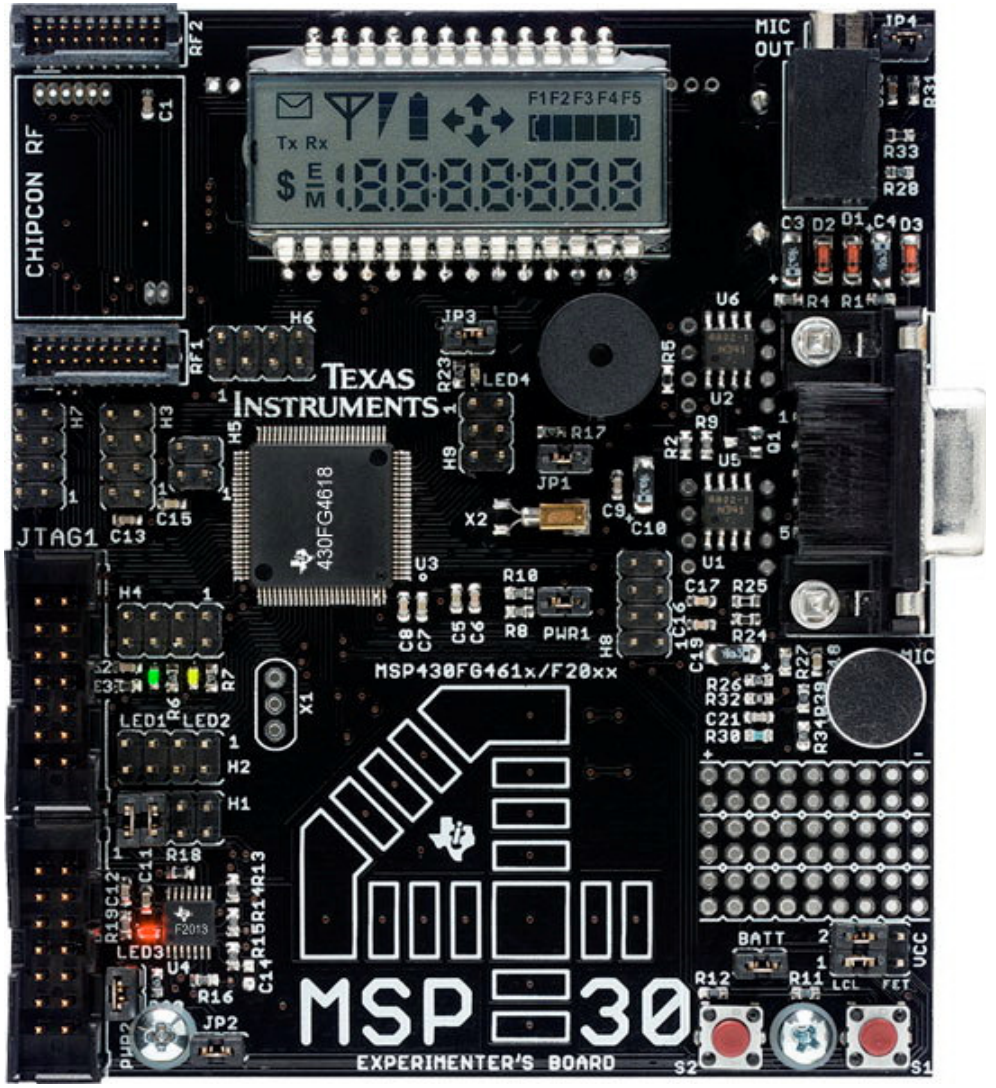


:: Kapacitný senzor ako dotyková klávesnica na báze MSP430 – meranie kapacity senzora





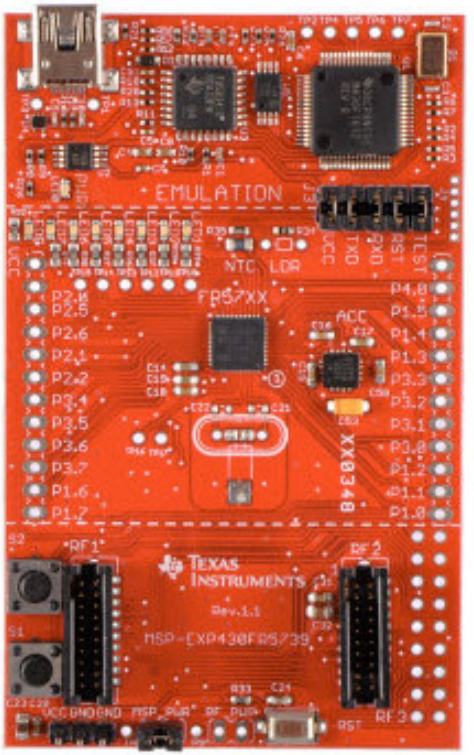
:: Kapacitný senzor ako dotyková klávesnica na báze MSP430 – vývojová doska



:: Gadgets for Launchpad :o)

MSP430 Capacitive Touch BoosterPack

MSP-EXP430FR5739 Experimenter Board



Koniec prednášky č. 12

Moderné aplikácie mikroradičov