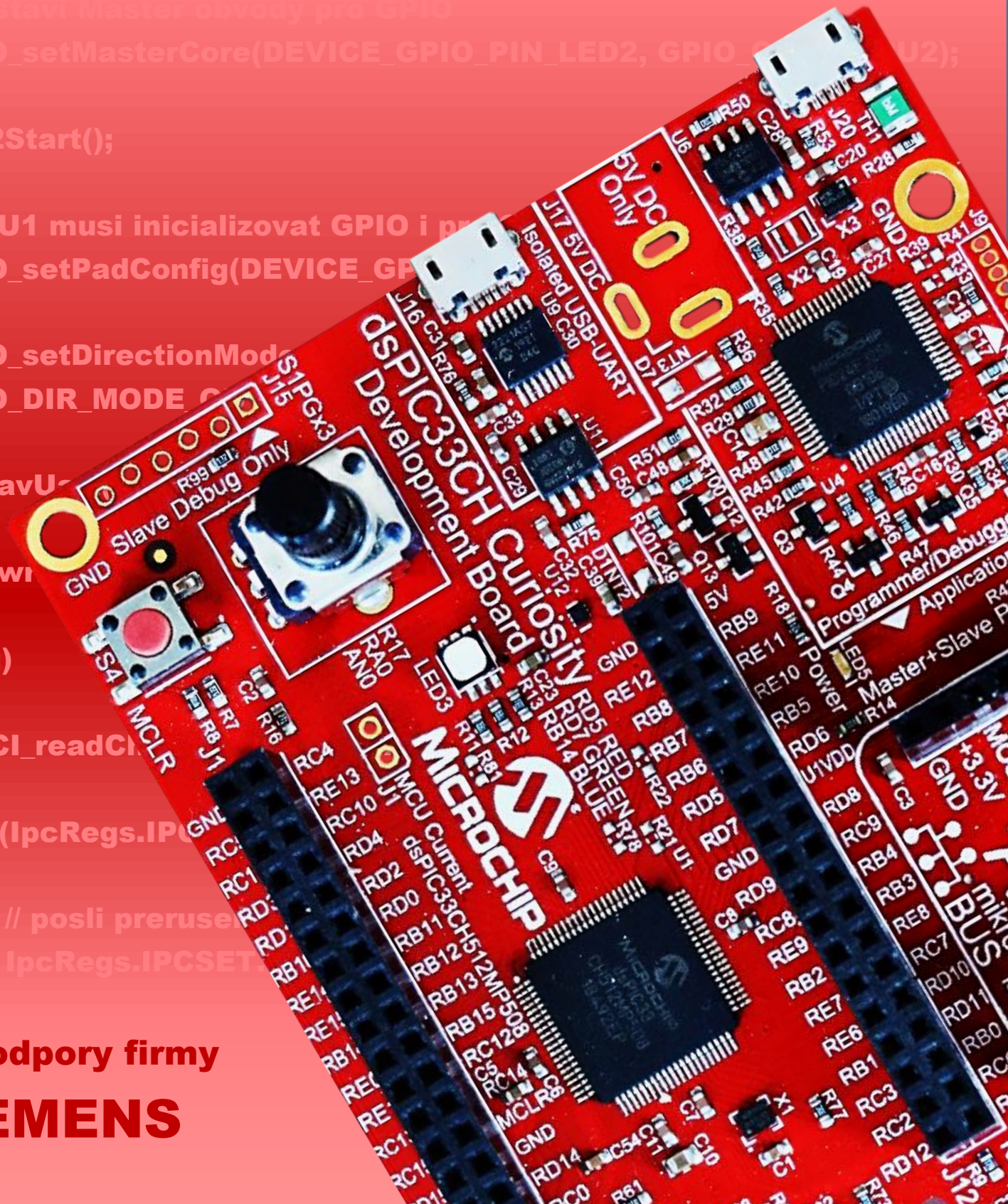
# Programujeme dvoujádrové kontrolery

**Ing. Vojtěch Skřivánek**

```
// nastavi Master obvody pro GPIO
GPIO_setMasterCore(DEVICE_GPIO_PIN_LED2, GPIO_CORE_CPU2);

cpu2Start();

// CPU1 musi inicializovat GPIO i pr
GPIO_setPadConfig(DEVICE_GP

GPIO_setDirectionMod
GPIO_DIR_MODE_

nastavU

SCI_wr

for(;;)
{
    SCI_readCh

    if((IpcRegs.IPC
    {
        // posli preruse
        IpcRegs.IPCSET
```




za podpory firmy

**SIEMENS**

# Poděkování

Děkuji firmě

**SIEMENS**

za podporu při psaní této knihy.

# Obsah

### 6.4.8 Příkazový registr

V kapitole o sdílených datových registrech bylo řečeno, že tyto sdílené registry nemají i přes svůj název žádnou speciální funkci. Můžeme je používat zcela libovolně. U některých z nich to ale za jistých podmínek není pravda.

Jako příklad může být použití příkazového registru ***IPCSENDCOM*** v naší funkci ***cpu2Start***, kterou používáme vždy na začátku programu jádra ***CPU1***. Zde do tohoto registru vkládáme hodnotu ***BROM_IPC_EXECUTE_BOOTMODE_CMD***, čímž po odeslání příznaků 0 a 31 zresetujeme jádro ***CPU2*** (provedeme opětovné spuštění - ***boot***).

Podíváme-li se do tabulky referenčního manuálu [7] do kapitoly 4.10.8.1 či 4.10.8.2 (***CPU1/2 IPC Commands***), zjistíme, že se tyto sdílené registry dají využít i jinými způsoby. Nejčastěji se jedná o příkazy k zápisu či čtení jednotlivých bitů či dat na adresu paměti jádra ***CPU1*** či ***CPU2*** (záleží, které jádro příkaz volá). Příkazový registr pak využívá adresový a datový registr pro určení adresy a dat zápisu či čtení.

Funkce příkazů jsou v obou směrech stejné až na tři příkazy ve směru z jádra ***CPU1*** do ***CPU2***. První rozdíl je již zmíněná možnost přimět jádro ***CPU2*** znovu spustit.

Další možností je přikázání jádru ***CPU2*** okamžitě skočit na určené místo v paměti programu, kde bude program pokračovat v běhu.

Poslední možností je skok na funkci, kdy jádro ***CPU2*** skočí na funkci svého programu, využije parametr uložený ve sdíleném registru, návratovou hodnotu uloží sdíleného registru, a nakonec se program vrátí na původní místo běhu před skokem na funkci.

Poslední dvě zmiňované možnosti mohou být vynikajícím nástrojem například v momentě, kdy chceme, aby bylo jádro ***CPU2*** většinu doby uspáno, ale občas provedlo nějakou krátkou funkcionalitu na popud jádra ***CPU1***.

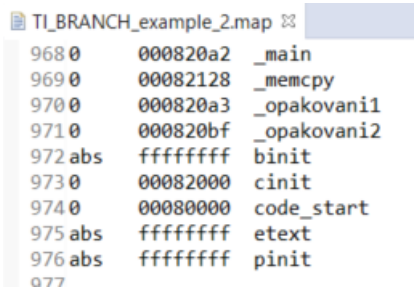
#### 6.4.8.1 Skok CPU2 na program

V prvním příkladu si ukážeme, jak přinutit jádro ***CPU2*** skočit na jinou adresu paměti programu, díky čemuž začne jádro vykonávat jiný program.

Začneme programem pro jádro ***CPU2***. To bude obsahovat tři funkce (spíše bychom měli říci procedury, jelikož tyto funkce nemají žádné parametry). První z nich bude hlavní funkce *main*, která bude prázdná. Další dvě funkce budou blikat diodou, každá však jinou rychlostí.

```
#include "gpio.h"

void main(void)
{
    for(;;){}
}

void opakovani1()
{
    for(;;)
    {
        for(uint16_t x = 0U; x < 10000U; x++);

        // vypni LED
        GPIO_writePin(DEVICE_GPIO_PIN_LED2, 1U);

        for(uint16_t x = 0U; x < 10000U; x++);

        // zapni LED
        GPIO_writePin(DEVICE_GPIO_PIN_LED2, 0U);
    }
}

void opakovani2()
{
    for(;;)
    {
        for(uint16_t x = 0U; x < 20000U; x++);

        // vypni LED
        GPIO_writePin(DEVICE_GPIO_PIN_LED2, 1U);

        for(uint16_t x = 0U; x < 20000U; x++);

        // zapni LED
        GPIO_writePin(DEVICE_GPIO_PIN_LED2, 0U);
    }
}
```

Abychom věděli, kde jsou tyto funkce uložené v paměti programu, nahlédneme do mapového souboru *nazevProjektu.map*, který po přeložení programu najdeme ve složce *Debug*. Po otevření nalezneme názvy funkcí a adresu jejich počátku v paměti programu.




Tyto hodnoty použijeme v programu jádra ***CPU1***.

Program jádra ***CPU1*** obsahuje funkci, v níž do sdíleného registru ***IPCSENDADDR*** vložíme její vstupní parametr, který představuje adresu v paměti programu, kam má jádro ***CPU2*** skočit. Dále v ní nastavíme hodnotu ***IPC_BRANCH*** do sdíleného registru ***IPCSENDCOM***. Tato hodnota značí příkaz skoku programu jádra ***CPU2***. Na závěr nastavíme příznaky 0 a 31, aby jádro ***CPU2*** příkaz provedlo.

```
#include "F2837xD_Ipc_drivers.h"
#include "sysctl.h"

void cpu2Branch(uint32_t adresaProgramu)
{
    // Priprav programovou adresu pro pro CPU2 pro skok na funkci
    IpcRegs.IPCSENDADDR = adresaProgramu;

    // Nastav prikaz na BRANCH_CALL
    IpcRegs.IPCSENDCOM = IPC_BRANCH;

    // nastaveni priznaku, aby CPU2 provedlo prikaz
    IpcRegs.IPCSET.all = (IPC_FLAG31 | IPC_FLAG0);
}
```

V hlavním programu jádra ***CPU1*** nastavíme výstupní pin, a poté v nekonečné smyčce střídavě používáme výše popsanou funkci, do jejíhož vstupního parametru vkládáme hodnotu počáteční adresy paměti programu funkcí jádra ***CPU2*** - *opakovani1* a *opakovani2*. Adresy jsou shodné s adresami uvedeným v mapovém souboru.

```
#include "F2837xD_Ipc_drivers.h"
#include "sysctl.h"
#include "gpio.h"

void main(void)
{
    cpu2Start();

    // nastavi ridici jadra pro GPIO
    GPIO_setMasterCore(DEVICE_GPIO_PIN_LED2, GPIO_CORE_CPU2);

    // CPU1 musi inicializovat GPIO i pro CPU2
    GPIO_setPadConfig(DEVICE_GPIO_PIN_LED2, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED2, GPIO_DIR_MODE_OUT);

    for(;;)
    {
        // adresa funkce opakovani1 nalezena v .map souboru
        cpu2Branch(0x820A3U);

        for(uint32_t x = 0U; x < 150000U; x++);

        // adresa funkce opakovani2 nalezena v .map souboru
        cpu2Branch(0x820BFU);

        for(uint32_t x = 0U; x < 150000U; x++);
    }
}
```

Po přeložení a nahrání programu do obou jader vidíme, jak se rychlost blikání uživatelské ***LED*** mění, jelikož jádro ***CPU1*** periodicky přikazuje jádru ***CPU2*** skákat z jedné funkce (procedury) do druhé.

#### 6.4.8.2 Skok CPU2 na funkci

V předchozím příkladu jsme si ukázali, jak může jádro ***CPU1*** přikázat jádru ***CPU2*** skok na konkrétní místo programu. V našem případě šlo o procedury. Tímto způsobem jsme ale nebyli schopni do procedur jádra ***CPU2*** přenést vstupní a získat výstupní parametry.

Samozřejmě je pro tento účel možné využít kteroukoliv z dříve demonstrovaných možností (které často budou použity z důvodu přenosu většího množství dat). Pro přímý přenos jednoho vstupního a jednoho výstupního parametru bez alokace dalších prostředků je však možné využít příkaz na volání funkce spolu se sdílenými registry. Pokud totiž použijeme příkaz na volání funkce, můžeme použít sdílený registr ***IPCSENDDATA*** pro nastavení hodnoty vstupního parametru. V registru ***IPCREMOTEREPLY*** můžeme po dokončení volané funkce nalézt návratovou hodnotu.

Využití v praxi si ukážeme na příkladu, který je podobný předchozímu. Jádro ***CPU1*** přikáže jádru ***CPU2*** skok na funkci. Rozdílem je, že nyní bude jádro ***CPU2*** skákat vždy na stejnou funkci, ale bude od ***CPU1*** dostávat pokaždé jinou hodnotu vstupního parametru.

Hodnota vstupního parametru určí kmitočet blikání ***LED***. Po dokončení funkce získá jádro ***CPU1*** návratovou hodnotu, která bude rovna dvojnásobku hodnoty vstupního parametru. Tato návratová hodnota bude při dalším příkazu skoku na funkci použita jako vstupní parametr. Kmitočet blikání se tedy bude postupně snižovat.

Po dokončení funkce se jádro ***CPU2*** vždy navrátí do hlavní funkce *main*, v níž bude blikat ***LED*** konstantní rychlostí.

Jako pozorovatelé tedy uvidíme, jak ***LED*** bliká nejprve rychle (funkce *main*), poté pomaleji (skok na funkci), opět rychle (návrat do *main*), ještě pomaleji (skok na funkci s nižším kmitočtem) a tak dále.

Podívejme se na jádro ***CPU2***. V jeho hlavním programu neustále blikáme ***LED***. Ve funkci *opakovani* vždy 3krát blikneme ***LED*** s frekvencí, která je dána vstupním parametrem funkce. Návratová hodnota je rovna dvojnásobku hodnoty vstupního parametru.

```
#include "gpio.h"

void main(void)
{
    for(;;){
        GPIO_togglePin(DEVICE_GPIO_PIN_LED2);

        for(uint32_t x = 0U; x < 5000U; x++);
    }
}

uint32_t opakovani(uint32_t cas)
{
    for(uint8_t x = 0U; x < 3U; x++){
        for(uint32_t x = 0U; x < cas; x++);

        // vypni LED
        GPIO_writePin(DEVICE_GPIO_PIN_LED2, 1U);

        for(uint32_t x = 0U; x < cas; x++);

        // zapni LED
        GPIO_writePin(DEVICE_GPIO_PIN_LED2, 0U);
    }

    return (cas * 2U);
}
```

U programu jádra ***CPU1*** se nejprve podívejme na funkci, která přikazuje jádru ***CPU2*** skok na funkci. V ní vidíme, že do sdíleného registru ***IPCSENDADDR*** dáváme, stejně jako v předchozím příkladu, počáteční adresu paměti programu ***CPU2***. Nyní navíc vkládáme do sdíleného registru ***IPCSENDDATA*** hodnotu vstupního parametru funkce, kterou jádro ***CPU2*** vykoná.

```
void cpu2Function(uint32_t adresaProgramu, uint32_t parametrFunkce)
{
    // priprav programovou adresu pro pro CPU2 pro skok na funkci
    IpcRegs.IPCSENDADDR = adresaProgramu;

    // priprav hodnotu parametru funkce pro pro CPU2
    IpcRegs.IPCSENDDATA = parametrFunkce;

    // nastav prikaz na FUNCTION_CALL
    IpcRegs.IPCSENDCOM = IPC_FUNC_CALL;

    // nastaveni priznaku, aby CPU2 provedlo prikaz
    IpcRegs.IPCSET.all = (IPC_FLAG31 | IPC_FLAG0);
}
```

V nekonečné smyčce hlavního programu jádra ***CPU1*** vidíme volání výše popsané funkce. Jejím prvním vstupním parametrem je počáteční adresa paměti programu, na které je uložená funkce *opakovani* jádra ***CPU2***. Adresa byla opět dohledána v mapovém souboru programu jádra ***CPU2***. Druhým parametrem je proměnná nazvaná *čas*. Ta se předá do funkce programu ***CPU2*** a určuje kmitočet blikání ***LED***. Tato proměnná se v programu mění. Mění se tím způsobem, že po zavolání funkce jádra ***CPU2*** se čeká na její dokončení (v našem příkladu je z důvodu jednoduchosti použita zpožďovací smyčka namísto např. příznaku), a poté se do proměnné *čas* vloží hodnota sdíleného registru ***IPCREMOTEREPLY***. Do tohoto registru se totiž automaticky ukládá návratová hodnota volané funkce jádra ***CPU2***. V našem příkladu je tedy každou iteraci hodnota proměnné *čas* dvojnásobná (dokud nedosáhne maxima určeného podmínkou). Každé další volání funkce *opakovani* má za následek blikání ***LED*** s nižším kmitočtem.

```
#include "F2837xD_Ipc_drivers.h"
#include "sysctl.h"
#include "gpio.h"

void main(void)
{
    cpu2Start();

    // nastavi ridici jadra pro GPIO
    GPIO_setMasterCore(DEVICE_GPIO_PIN_LED2, GPIO_CORE_CPU2);

    // CPU1 musi inicializovat GPIO i pro CPU2
    GPIO_setPadConfig(DEVICE_GPIO_PIN_LED2, GPIO_PIN_TYPE_STD);
    GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED2, GPIO_DIR_MODE_OUT);

    uint32_t cas = 10000U;

    for(;;)
    {
        // adresa funkce opakovani nalezena v .map souboru
        cpu2Function(0x82072U, cas);

        // cekej na dokonceni funkce CPU2
        for(uint32_t x = 0U; x < 150000U; x++);

        cas = IpcRegs.IPCREMOTEREPLY;

        if(cas > 150000U)
        {
            cas = 10000U;
        }
    }
}
```

Po překladu a nahrání programů do obou jader uvidíme, jak se kmitočet blikání ***LED*** mění. Buď pravě probíhá funkce *main*, která vyplňuje čas mezi voláními funkce *opakovani*, a kdy ***LED*** bliká velice rychle, nebo probíhá funkce *opakovani*, kdy se rychlost tří bliknutí ***LED*** mění s každým dalším voláním funkce.