Programujeme STM32

nízkopříkonové aplikace

Ing. Vojtěch Skřivánek

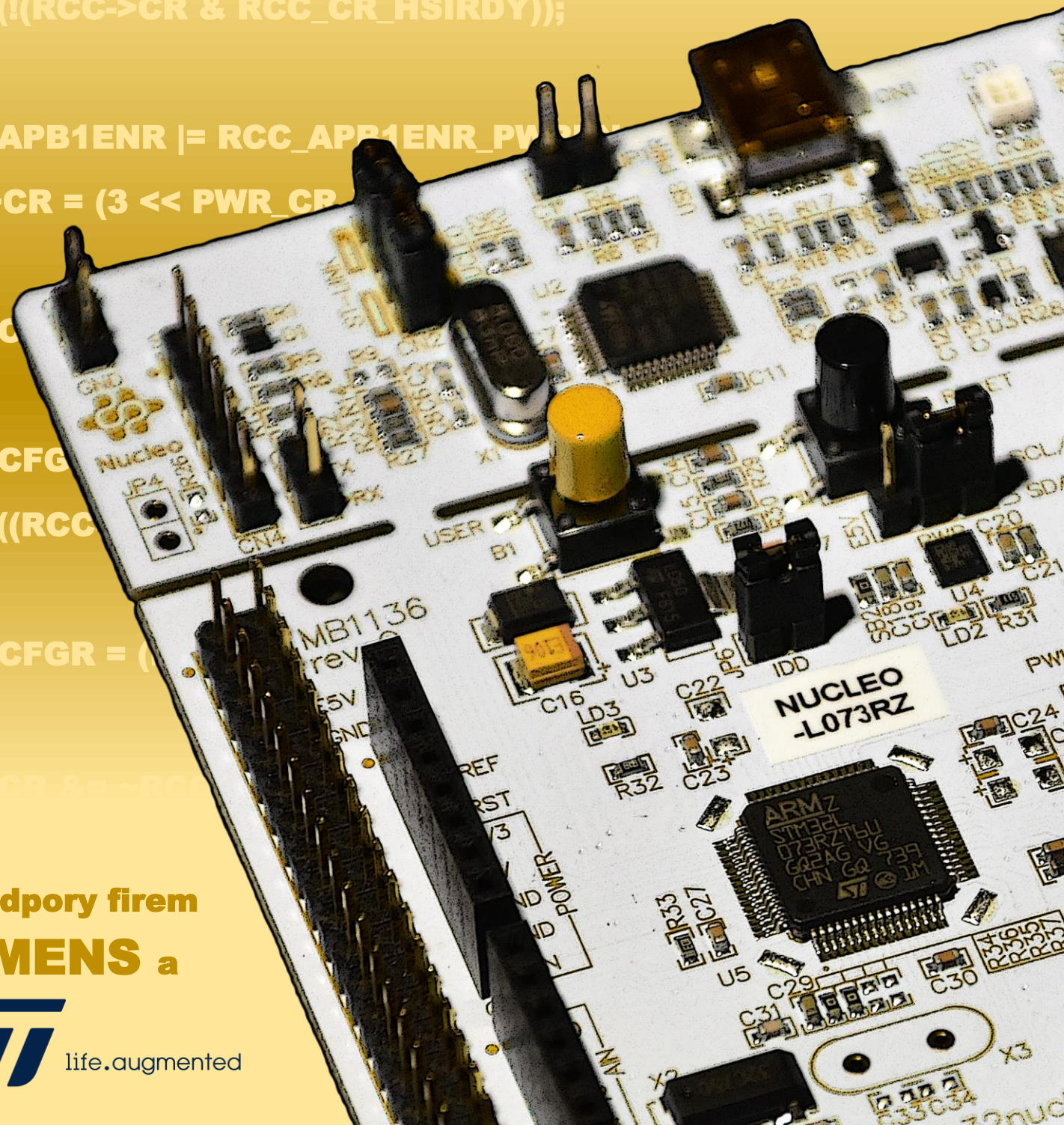
```
RCC->CR |= RCC_CR_HSION;
while (!(RCC->CR & RCC_CR_HSIRDY));

RCC->APB1ENR |= RCC_APB1ENR_PW
PWR->CR = (3 << PWR_CR

RCC->C

RCC->CFG
while ((RCC

RCC->CFGR = (
```



Za podpory firem

SIEMENS a

life.augmented

Poděkování

Děkuji firmám

SIEMENS

a

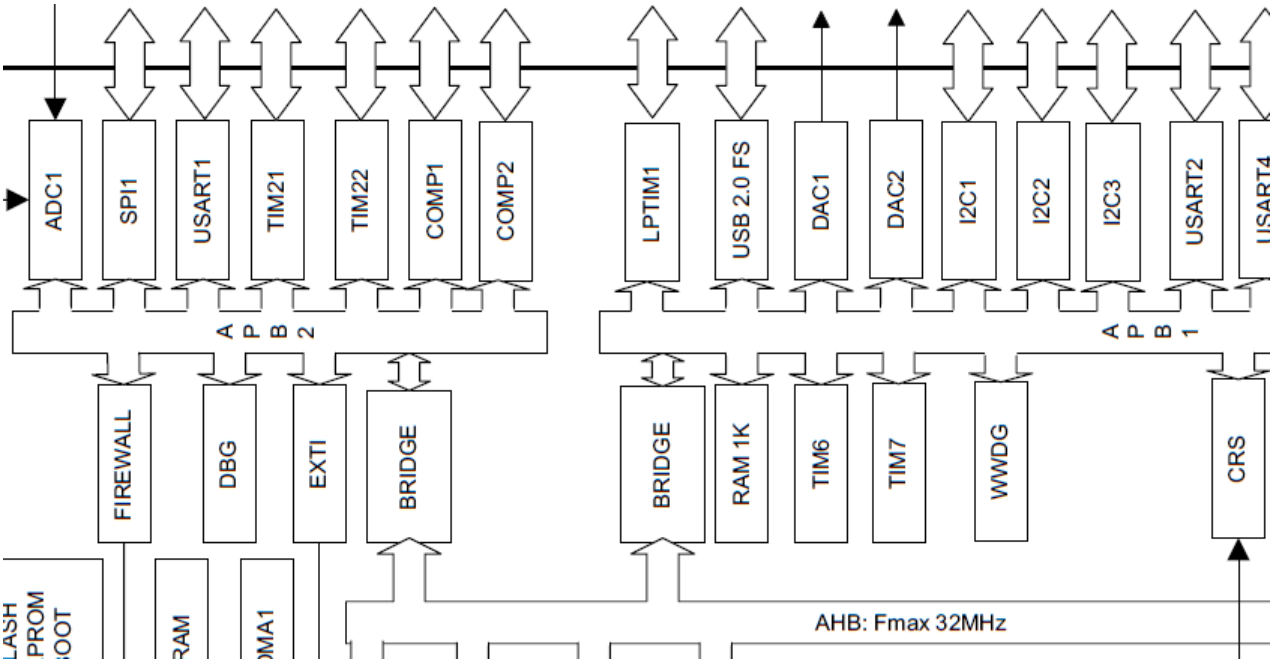
za podporu při psaní této knihy.

Obsah

4.3.2 Distribuce hodinového signálu

V předchozím příkladu jsme si ukázali, jak může spotřebu kontroleru ovlivnit zapnutá periferie. Její spotřeba je závislá na frekvenci hodinového signálu, který přijímá. Je tedy žádoucí udržovat tuto frekvenci co možná nejnižší u každé zapnuté periferie. Jaké máme možnosti a nástroje toho dosáhnout?

Veškeré periferie jsou připojené k jedné ze dvou sběrnic periferií ***APB1/2*** (***Advanced Peripheral Bus***). Obě tyto sběrnice jsou pomocí můstků připojeny k vysokorychlostní sběrnici ***AHB*** (***Advanced High-speed Bus***), která je připojena k jádru kontroleru.



Úryvek blokového diagramu kontrolerů STM32L073xx [1]

Každé sběrnici můžeme pomocí děliček nastavit jiný kmitočet. Toho se dá využít k optimalizaci odběru jednotlivých periferií.

Většina periferií využívá jako zdroj hodinového signálu právě hodinový signál sběrnice, ke které jsou připojeny. U některých periferií ale můžeme zdroj hodinového signálu změnit. I to se dá skvěle využít.

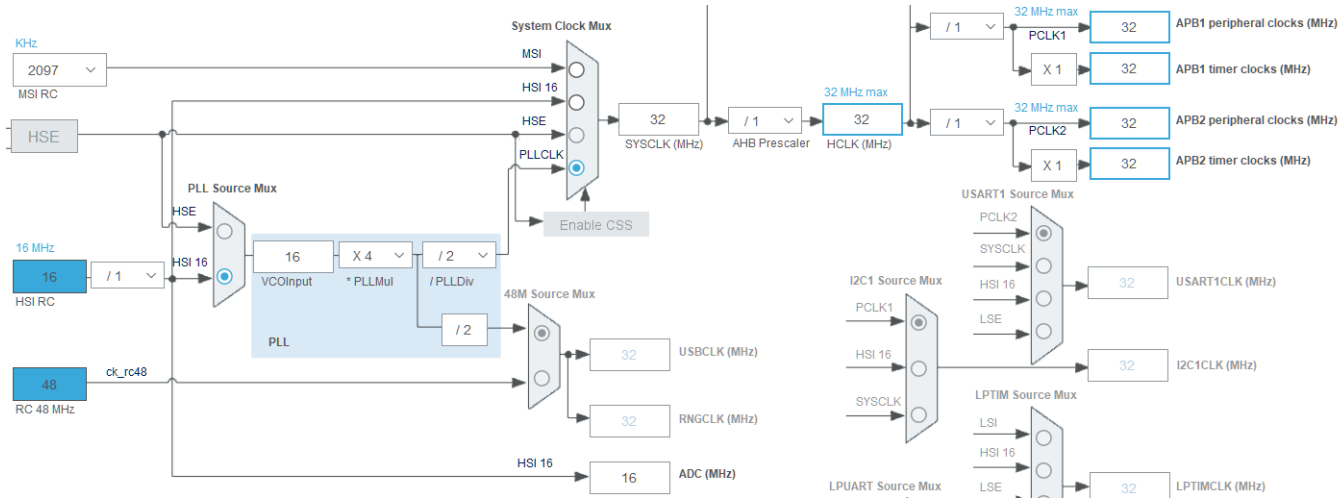
4.3.2.1 Příklad 10 – Distribuce hodinového signálu

Na tomto příkladu si ukážeme, jak může chytré využití distribuce hodinového signálu do různých periferií snížit spotřebu kontroleru.

Představme si, že náš kontroler musí používat následující periferie – ***SPI/I2S*** s co možná nejvyšší přenosovou rychlostí, 2x ***USART*** s rychlostí alespoň 115200 Baud, ***ADC*** k měření pomalých signálů, ***I2C*** v rychlém režimu (400 kb/s) a časovač s alespoň milisekundovým rozlišením. Nemůžeme použít externí zdroje hodinového signálu.

Podíváme-li se do katalogového listu se znalostmi s předchozího příkladu, dojdeme k následujícímu výběru periferií – ***SPI2*** (jediné podporuje ***I2S*** režim), ***USART4*** a ***USART5*** (nejnižší odběr), ***I2C*** (nejnižší odběr) a časovač ***TIM6*** (nejnižší odběr). ***ADC*** má kontroler pouze jeden. Mimo právě ***ADC*** jsou všechny tyto periferie na sběrnici ***APB1***.

Naše nastavení hodin bude vypadat následovně:



Využijeme ***HSI*** s ***PLL***. Získáme kmitočet 32 MHz pro maximální rychlost přenosu pomocí ***SPI2***.

Program s nastavením hodinového signálu a spuštěním jednotlivých periferií vypadá takto:

```
void nastavPll()
{
        // zapne interni vysokorychlostni oscilator
        RCC->CR |= RCC_CR_HSION;

        // cekej, dokud HSI neni pripraveny
        while (!(RCC->CR & RCC_CR_HSIRDY));

        // vydeli HSI 4x -> 4 MHz
        RCC->CR |= RCC_CR_HSIDIVEN;

        // nastavi napetovou uroven 1
        PWR->CR = (1U << PWR_CR_VOS_Pos);

        // zapne cekaci stav
        FLASH->ACR |= FLASH_ACR_LATENCY;

        // PLL nasobi 16x -> 64 MHz
        RCC->CFGR |= RCC_CFGR_PLLMUL16;
```

```
        // PLL deli 2x -> 32 MHz
        RCC->CFGR |= RCC_CFGR_PLLDIV2;

        // Zapne PLL
        RCC->CR |= RCC_CR_PLLON;

        // cekej, dokud neni PLL pripraven
        while ((RCC->CR & RCC_CR_PLLRDY) != RCC_CR_PLLRDY);

        // nastavi PLL jako zdroj hodinoveho signalu
        RCC->CFGR |= RCC_CFGR_SW_PLL;

        // cekej, dokud neni zdroj prepnuty
        while ((RCC->CFGR & RCC_CFGR_SWS_PLL) != RCC_CFGR_SWS_PLL);

        // vypne interni vicerychlostni oscilator
        RCC->CR &= ~RCC_CR_MSION;
}

int main(void)
{
        nastavPll();

        // zapne SPI2, USART4, USART5, I2C3 a TIM6
        RCC->APB1ENR |= (RCC_APB1ENR_SPI2EN | RCC_APB1ENR_USART4EN |
                         RCC_APB1ENR_USART5EN | RCC_APB1ENR_I2C3EN |
                         RCC_APB1ENR_TIM6EN);

        // zapne ADC
        RCC->APB2ENR |= RCC_APB2ENR_ADCEN;

        while(1);
}
```

Program je velmi jednoduchý. Jeho spotřeba je přibližně 5329 µA.

Nevýhodou tohoto programu je, že ač využívá nejúspornější periferie, jsou připojeny na zbytečně rychlý hodinový signál. Pojďme se nyní zamyslet, jak k periferiím připojit hodinový signál s co možná nejnižším kmitočtem.

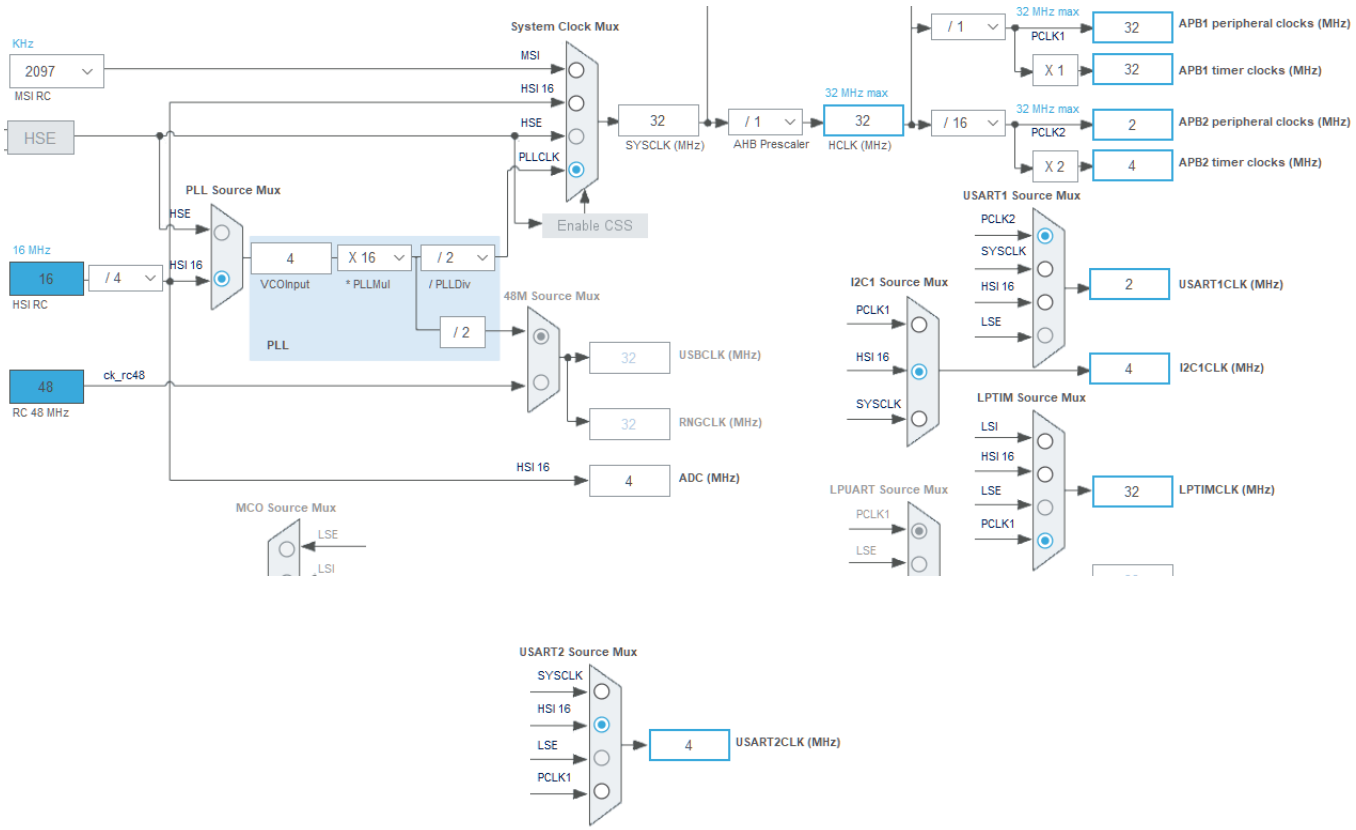
Nejprve se zaměřme na časovač. Sice jsme zvolili ten s nejnižší spotřebou, ale je připojen na hodinový signál o kmitočtu 32 MHz. Mohli bychom vybrat například časovač ***TIM21*** nebo ***TIM22***, které jsou připojeny ke sběrnici ***APB2***. Té bychom nastavili nižší frekvenci hodinového signálu na 4 MHz, čímž bychom spotřebu snížili. Můžeme ale také použít časovač ***LPTIM1***. Ten umožňuje volbu zdroje hodinového signálu. Můžeme zvolit mnohé zdroje, ale nejúspornější je vnitřní nízkorychlostní oscilátor ***LSI*** s přibližným kmitočtem 38 kHz. Využijme tedy místo ***TIM6*** právě tento časovač ***LPTIM1***.

Dále upravíme periferie ***USART4*** a ***USART5***. Zde bohužel nemůžeme použít ***LPUART1***, který také nabízí připojení k ***LSI***. Tím bychom nesplnili podmínku rychlosti komunikace. Můžeme ale použít periferii ***USART1***, jejímž zdrojem hodinového signálu může být sběrnice ***APB2***. Zvolíme jej a kmitočet sběrnice pomocí děličky upravíme na 2 MHz.

Co ale s druhou periferií ***USART***? Periferie ***USART2*** nabízí tyto možnosti volby zdroje hodinového signálu – ***LSE*** (není k dispozici), ***SYSCLK*** (32 MHz), ***PCLK1*** (***APB1*** – 32 MHz při maximální rychlost ***SPI2***) nebo ***HSI*** (16 MHz). Jak vidíme, nejvýhodnější je pro nás ***HSI***, čímž se spotřeba sníží na polovinu. Jenomže podle katalogového listu má ***USART2*** více než dvakrát větší spotřebu, než má ***USART4*** nebo ***USART5***. Proto tato změna není k lepšímu. My si však můžeme pomoci vydělením kmitočtu ***HSI*** čtyřmi. V ten moment do periferie ***USART2*** vstupuje signál o kmitočtu 4 MHz.

Jelikož jsme kmitočet ***HSI*** vydělili čtyřmi, je i hlavní hodinový signál pomalejší. To se však dá napravit ve fázovém závěsu ***PLL***, kde kde hodinovému signálu kmitočet zvýšíme. S periferií ***I2C3*** provedeme to samé, co s druhou periferií ***USART***. Místo ní vybereme ***I2C1***, které nastavíme zdroj hodinového signálu ***HSI*** již nastavený na 4 MHz. Zároveň díky tomuto nastavení klesne i rychlost a spotřeba ***ADC***.

Naše nastavení tedy vypadá takto:



Dle tabulkových hodnot spotřeby bychom měli dojít k takovémuto snížení odběru:

Periferie	Spotřeba periferie [μA/MHz]	Kmitočet [MHz]	Spotřeba periferie [μA]	Rozdíl spotřeby [μA]	Úspora [%]
USART4	5	32	160	-	-
USART1	14,5	2	29	131	81,9
USART5	5	32	160	-	-
USART2	14,5	4	58	102	63,8
I2C3	11	32	352	-	-
I2C1	11	4	29	308	87,5
TIM6	3,5	32	112	-	-
LPTIM1	10	0,037	0,37	111,6	99,6
ADC(1)	5,5	16	88	-	-
ADC(2)	5,5	4	29	66	75,0
Celkový rozdíl spotřeby				718,6	82,4

Jak vidíme, největší podíl na úspoře by měla být změna nastavení ***I2C***. Největší relativní rozdíl je vidět u časovače, kde by relativní úspora měla být téměř stoprocentní. U celkového rozdílu spotřeby vyjádřeného v procentech mějme na paměti, že se jedná pouze o rozdíl odběru periferií, nikoliv celého kontroleru.

Program bude nyní nepatrně složitější. V nastavení zdroje signálu vydělíme kmitočet ***HSI*** čtyřmi, zato jej však v ***PLL*** vynásobíme šestnácti.

```
void nastavPll()
{
        // zapne interni vysokorychlostni oscilator
        RCC->CR |= RCC_CR_HSION;

        // cekej, dokud HSI neni pripraveny
        while (!(RCC->CR & RCC_CR_HSIRDY));

        // nastavi napetovou uroven 1
        PWR->CR = (1 << PWR_CR_VOS_Pos);

        // zapne cekaci stav
        FLASH->ACR |= FLASH_ACR_LATENCY;

        // PLL nasobi 4x -> 64 MHz
        RCC->CFGR |= RCC_CFGR_PLLMUL4;

        // PLL deli 2x -> 32 MHz
        RCC->CFGR |= RCC_CFGR_PLLDIV2;

        // Zapne PLL
        RCC->CR |= RCC_CR_PLLON;

        // cekej, dokud neni PLL pripraven
        while ((RCC->CR & RCC_CR_PLLRDY) != RCC_CR_PLLRDY);

        // nastavi PLL jako zdroj hodinoveho signalu
        RCC->CFGR |= RCC_CFGR_SW_PLL;

        // cekej, dokud neni zdroj prepnuty
        while ((RCC->CFGR & RCC_CFGR_SWS_PLL) != RCC_CFGR_SWS_PLL);

        // vypne interni vicerychlostni oscilator
        RCC->CR &= ~RCC_CR_MSION;
}
```

V hlavním programu přidáme navíc pár řádek, které nastaví zdroje hodinového signálu jednotlivým periferiím.

```
int main(void)
{
        nastavPll();

        // zapne LSI
        RCC->CSR |= RCC_CSR_LSION;

        // nastavi delicku APB2 /16
        RCC->CFGR |= RCC_CFGR_PPRE2_DIV16;

        // nastavi zdroje hodinoveho signalu: LPTIM1 -> LSI
        // USART2 -> HSI; USART1 -> APB2 (vychozi); I2C -> HSI
        RCC->CCIPR |= (RCC_CCIPR_LPTIM1SEL_0 | RCC_CCIPR_USART2SEL_1 |
                       RCC_CCIPR_I2C1SEL_1);

        // zapne SPI2, USART2, I2C1 a LPTIM1
        RCC->APB1ENR |= (RCC_APB1ENR_SPI2EN | RCC_APB1ENR_USART2EN |
                         RCC_APB1ENR_I2C1EN | RCC_APB1ENR_LPTIM1EN |
                         RCC_APB1ENR_USART5EN);

        // zapne USART1 a ADC
        RCC->APB2ENR |= (RCC_APB2ENR_USART1EN | RCC_APB2ENR_ADCEN);

        while(1);
}
```

Po spuštění programu je spotřeba přibližně 4502 µA. Úspora je tedy 830 µA což přibližně odpovídá dříve vypočítanému odhadu.

V tomto příkladu tvoří 830 µA přibližně 15,5% úsporu, což není zanedbatelná hodnota. Samozřejmě je nutno brát v potaz, že jde o ilustrativní, idealizovaný příklad.

Ne vždy si můžeme zvolit, kterou z ekvivalentních periferií použijeme z důvodu použití pinů. Dále, pokud bychom pomocí ***ADC*** měli měřit více kanálů s rychlými signály, nemohli bychom kmitočet ***HSI*** vydělit. Spotřeba ***USART2*** a ***I2C*** by tím byla čtyřnásobná.

Periferie	Spotřeba periferie [μA/MHz]	Kmitočet [MHz]	Spotřeba periferie [μA]	Rozdíl spotřeby [μA]	Úspora [%]
USART4	5	32	160	-	-
USART1	14,5	2	29	131	81,9
USART5	5	32	160	-	-
USART2	14,5	16	232	-72	-45,0
I2C3	11	32	352	-	-
I2C1	11	16	176	176	50,0
TIM6	3,5	32	112	-	-
LPTIM1	10	0,037	0,37	111,6	99,6
ADC(1)	5,5	16	88	-	-
ADC(2)	5,5	16	88	0	0,0
Celkový rozdíl spotřeby				346,6	60,2

Jak vidíme v tabulce, bylo by v tomto nastavení použití ***USART2*** neúsporné. Naměřená úspora odběru s takovýmto nastavením činí pouze 352 µA z celkového odběru kontroleru (cca 6,6 % z celkového odvběru). Při ponechání ***USART5*** je úspora 503 µA – 9,4 % z celkového odběru.

Tento příklad jasně dokázal, že správná distribuce hodinového signálu jednotlivým periferiím vede k úspoře energie, a to bez vlivu na funkci a běh programu.