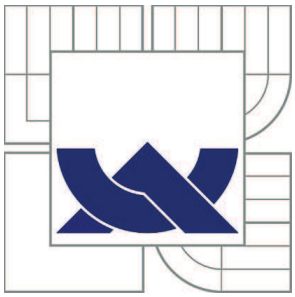
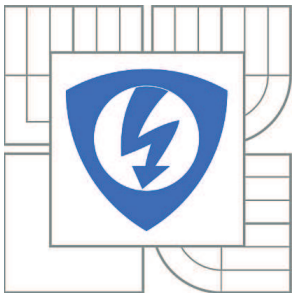


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

BEZDRÁTOVÝ MESH PROTOKOL PRO FREERTOS

WIRELESS MESH PROTOCOL FOR FREERTOS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

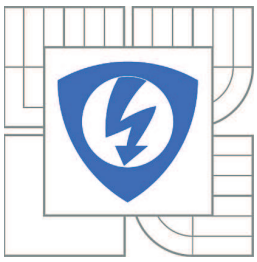
AUTOR PRÁCE — MARTIN HORÁK

AUTHOR

VEDOUCÍ PRÁCE — Ing. VLADIMÍR ČERVENKA

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Martin Horák	***ID:***	146010
Ročník:	3	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Bezdrátový mesh protokol pro FreeRTOS

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je implementace bezdrátového mesh protokolu do prostředí FreeRTOS. Implementace bude respektovat architekturu RTOS a jednotlivé funkce protokolové sady budou vhodně rozděleny do samostatných vláken. Součástí práce je analýza a podrobné srovnání vytvořené implementace s portací daného mesh protokolu na reálném zařízení. Implementace proběhne na platformě ARM Cortex-M.

DOPORUČENÁ LITERATURA:

[1] BARRY, Richard. Using the FreeRTOS real time Kernel /: Richard Barry [online]. Bristol: Real Time Engineers, c2010, xvi, 172 s. [cit. 2013-10-08]. ISBN 978-1-4461-6914-8.

[2] VALVANO, Jonathan W. Embedded Systems: Real-Time Interfacing to Arm® Cortex(TM)-M Microcontrollers. CreateSpace, 2011, xiv, 570 s. ISBN 978-1463590154.

[3] VALVANO, Jonathan W. Embedded Systems: Real-Time Operating Systems for Arm Cortex M Microcontrollers. s.l.: CreateSpace, 2012, xi, 366 s. ISBN 978-1466468863.

Termín zadání:	10.2.2014	***Termín odevzdání:***	4.6.2014

Vedoucí práce: Ing. Vladimír Červenka
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Bakalářská práce si klade za cíl v teoretické části představit princip fungování mesh sítí, jejich praktického využití, stručnému představení mesh protokolu a standartu 802.15.4. Dále představuje operační systémy reálného času, které lze využít pro tvorbu programů, založených na architektuře RTOS. V praktické části je vytvořeno několik programů, a provedeno měření spotřeby energie a počtu cyklů a na základě dosažených výsledků je nalezena nejvhodnější implementace do prostředí FreeRTOS. Závěrečná kapitola je věnována celkovému přehledu dosažených výsledků, které jsou přehledně uvedeny do tabulek.

KLÍČOVÁ SLOVA

FreeRTOS, implementace, mesh sítě, mesh protokol, 802.15.4

ABSTRACT

Bachelor thesis aims to introduce in therеretical part operating principe of mesh network, their practical use, brief introduction of mesh protokol and standard 802.15.4. Also present a real-time operating systems, that can be used for programming applications based on real-time operating systems architecture. In practical part is to create several progres and made measurements of energy consumption and the number of cycles. Based on the results is found most appropriate implementation into the FreeRTOS. The final chapter is devoted to the overview of the achieved results, which are summarized in the tables

KEYWORDS

FreeRTOS, implementation, mesh network, mesh protocol, 802.15.4

HORÁK, M. *Bezdrátový mesh protokol pro FreeRTOS.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 36 s. Vedoucí bakalářské práce Ing. Vladimír Červenka.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Bezdrátový mesh protokol pro FreeRTOS“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení S 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno . . . . . .

. . . . . .
(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Vladimíru Červenkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno . . . . . .

. . . . . .

(podpis autora)

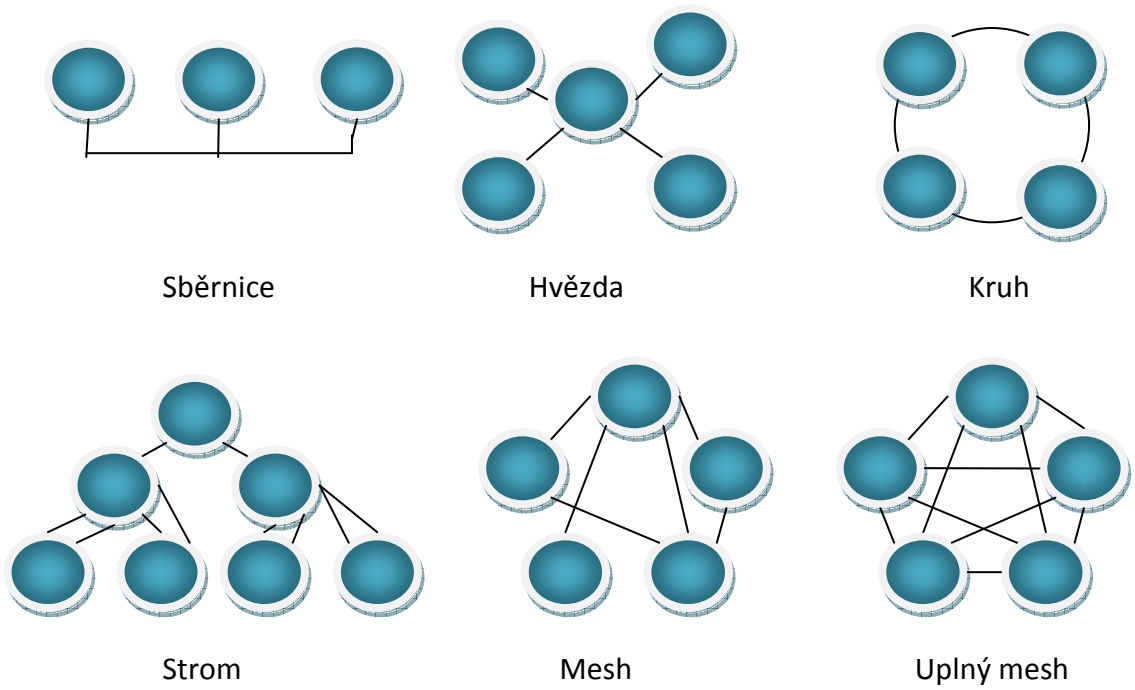
Obsah

1 Úvod

Cílem bakalářské práce bylo implementovat mesh protokol do prostředí operačního systému reálného času, konkrétně do distribuce FreeRTOS. Bakalářská práce je rozdělena do dvou stěžejních částí. První část tvoří teoretická část, která se zabývá problematikou a popisem mesh sítí (kap. 2) a mesh protokolu (kap. 3). Následující teoretická kapitola 4 se věnuje popisu operačním systémům reálného času a v podkapitole také popisu konkrétního operačního systému reálného času, tedy FreeRTOS. V praktické části, která začíná od 5. kapitoly, je podrobně popsán modifikovaný operační systém reálného času FreeRTOS, který je v implementaci použit. V podkapitole 5.1 je popsán postup vytvoření komunikační vrstvy mezi transceiverem a mikrokontrolérem a poté postup nalezení vhodné implementace mesh protokolu do prostředí FreeRTOS(kap. 5.2). V podkapitole 5.3 je popsána implementace nízkoenergetického časovače, který slouží k periodickému zapínání a vypínání transceiveru, tedy k redukování celkové spotřeby energie. V podkapitole 5.5 je popsána analýza vytvořené implementace s portací (kap. 5.4). V poslední kapitole 6 je přehled dosažených naměřených výsledků.

2 Bezdrátové mesh sítě

Hlavní vlastnosti bezdrátové mesh sítě vyplývají z jednotlivých částí názvů sítě. Termín bezdrátové odkazuje na fakt, že se jedná o bezdrátovou síť. Jednotlivé prvky sítě tedy nejsou propojeny pomocí metalických, optických či jiných spojů.
Druhý termín mesh odkazuje na typ topologie sítě. V komunikačních sítích rozlišujeme 5 typů topologií [1] : sběrnice (bus), hvězda (star), kruh(ring), strom (tree) a poslední typ je mřížka, což je právě mesh. Typy topologií jsou graficky znázorněny na obrázku 2.1.



Obr. 2.1: Přehled síťových topologií

Mezi vlastnosti mesh sítí patří i to, že mohou být samokonfigurované a samoorganizované. To znamená, že připojení nového prvku do sítě je možné bez konfiguračního zásahu do sítě. Při výpadku jedné z cest umí systém automaticky vyhledat novou cestu a to díky vícebodovému propojení jednotlivých prvků sítě. S tím souvisí další vlastnost sítě a to je multihop. To znamená, že odeslaná data od zdroje mohou "přeskákat" přes jednotlivé uzly sítě k příjemci dat. Samozřejmě tento princip je běžně uplatňován u drátových sítí, ale už není běžný u bezdrátových prvků, tak jak jsou v současnosti implementovány.

2.1 Uplatnění bezdrátových mesh sítí

Velkou výhodou bezdrátových mesh sítí oproti obyčejným sítím s pevnými spoji, jsou relativně nízké náklady na výstavbu, resp. na vytvoření nového uzlu dané sítě. Pokud na sebe dané prvky "vidí", tedy nestojí mezi nimi překážka znemožňující spojení a jsou ve vzájemném signálovém dosahu, tak může být vytvořen nový spoj, aniž by bylo nutné pokládat dodatečnou kabeláž.Toto řešení je z ekonomického hlediska velmi výhodné.

Pokud má daný prvek sítě řešeno napájení pomocí baterie, není potřebná v daném místě infrastruktura, která řeší rozvod elektrické energie. Na druhou stranu kapacita baterií je do jisté míry omezená.

V praxi se hledá uplatnění pro bezdrátové mesh sítě tam, kde je možné daný prvek připojit k elektrické síti (například pokrytí měst). V roce 2004 byl například zahájen projekt " South African wireless community networks", který řeší připojení oblastí Jižní Afriky, kde není dosah mobilní sítě [2].

2.2 IEEE 802.15

Jako IEEE 802 je označována rodina standardů standardizační neziskové organizace Institute of Electrical and Electronics Engineers. Skupina standardů 802.15 se zabývá definicí standardů pro WPAN- wireless personal area network, tedy malých bezdrátových osobních sítí [3]. Z těchto standardů vychází pak množství známých bezdrátových technologií, jako je například IrDA, Bluetooth nebo Zigbee. Tato skupina je pak dále rozdělena na 7 podskupin. Pro nás je konkrétně zajímavá jedna skupina, a to skupina IEEE 802.15.4.

2.2.1 IEEE 802.15.4

802.15.4 Low rate WPAN (Bezdrátové osobní sítě) - Jedná se o standard, který definuje první dvě vrstvy z OSI modelu- fyzickou a linkovou. Z toho vyplývá, že v tomto standardu není definován způsob směrování, který v OSI modelu odpovídá 3. směrovací vrstvě, a je možné jej doplnit tak, aby splňoval vlastnosti meshové sítě. Ve standardu jsou definována frekvenční pásma a kanál. Přenosová rychlost pak záleží na použitém pásmu - je to buď 20 kbit/s pro pásmo 868 MHz nebo pro pásmo 2450 MHz je přenosová rychlost 250 kbit/s [4]. Mezi meshové technologie postavené na standardu 802.15.4 patří ZigBee a DigiMesh [4].

3 Lightweight Mesh

Lightweight mesh byl navrhnut s ohledem na široké spektrum aplikací, které využívají bezdrátovou konektivitu. Mezi tyto aplikace patří například:

1. Bezdrátové ovládání
2. Alarmy a bezpečností aplikace
3. Automatické měřící systémy (např. termostat)

Mezi základní vlastnosti konkrétní implementace light weight mesh protokolu patří [6]:

1. Jednoduchost konfigurace a následné použití
2. 65535 uzlů v sítí (teoretický limit)
3. 65535 oddělených PAN na jednom komunikačním kanálu
4. Možnost AODV směrování
5. Možnost multicastové komunikace
6. Duplikované rámce jsou zahazovány
7. Malá náročnost na hardware mikrokontroléru, konkrétně na flash paměť (8 kB) a na RAM paměť (4 kB)

Zdrojový kód Lightweight Mesh stacku je rozdělen do několika logických úrovní. Stack je navrhnut tak, že nabízí pouze funkce, které jsou absolutně nezbytné pro bezdrátovou komunikaci. Je očekáváno, že zbytek bude vytvořen uživatelem nebo jej poskytnou knihovny třetích stran, je-li to požadováno.

4 RTOS

4.1 Vlastnosti RTOS

Operační systém reálného času (RTOS), je takový systém, u kterého správnost výsledku nezáleží pouze na logickém výsledku, ale také na čase, za který je výsledku dosaženo. Dělení systémů na soft a hard (kap. 4.2) je jedním ze základních dělení RTOS.

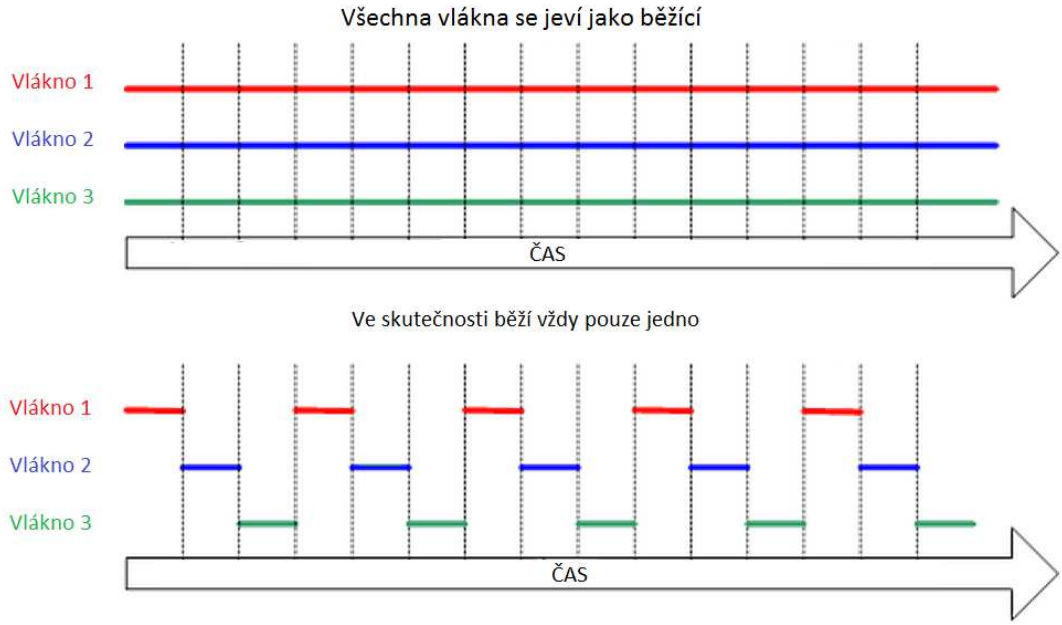
Použití RTOS umožňuje, aby byla aplikace napsána jako sada nezávislých vláken. Pro napsání dobré aplikace není RTOS důležitý, je to možnost volby programátora, ale použití RTOS přináší řadu výhod.

4.1.1 Multitasking

Je technika, která umožňuje běh více nezávislých procesů jedním procesorem. Jádro je základním prvkem operačního systému. Operační systém jako je Linux, používají jádra, která umožňují uživatelům přistupovat k počítači zdánlivě současně. Každý prováděný proces je vlákno pod kontrolou operačního systému. Pokud je operační systém schopen provádět několik procesů současně, jedná se o **multitasking**.

Použití multitasking operačního systému může zjednodušit návrh systému, co by jinak mohla být složitá aplikace. Multitasking a mezi-vláknová komunikace umožňují složité aplikaci, aby mohla být rozdělena na několik menších a více přehlednějších vláken. Rozdělení ve výsledku přinese jednodušší testování, opětovné použití kódu a například jednodušší práci v týmech. Složité načasování a sekvenční detaily mohou být odstraněny z kódu aplikace a stanou se povinností operačního systému.

Běžné procesory mohou vykonávat pouze jeden úkol současně. Rychlé přepínání mezi vlákny se jeví, jako kdyby byly prováděny současně. Průběh je znázorněn na obrázku 4.1, který znázorňuje tři vlákna, která běží současně. V horní polovině obrázku je znázorněn průběh, jak je vnímán. V dolní části obrázku je zobrazen skutečný model multitaskingu.



Obr. 4.1: Porovnání průběhu zdánlivého multitaskingu a reálného multitaskingu

4.1.2 Plánovač

Plánovač je funkce operačního systému, která postupně zpracovává běžící vlákna. Pro představu: plánovač si vytvoří seznam vláken, která jsou připravena k běhu. Když neběží nějaký proces, tedy je procesor volný, tak si plánovač vybere jedno vlákno z vytvořeného seznamu a přiměje vlákno ke spuštění. V **preemptivním** plánování jsou vlákna pozastavována v určitých periodách přerušení. Plánovač vybere nové vlákno, a jakmile se vrátí z přerušení, tak bude spuštěno nové vlákno. V tomto případě si systém sám určuje, kdy bude vlákno pozastaveno a kdy se vrátí do běžícího stavu.

V **kooperativním** plánovači se vlákna samy rozhodují, kdy se vrátí z pozastaveného stavu do běžícího. Distribuce FreeRTOS, která je používána v projektu, podporuje preemptivní a také kooperativní plánování [10].

4.1.3 Kontextové přepínání

Když vlákno běží, využívá registrů RAM a ROM paměti mikrokontroléru, stejně jako jakýkoliv jiný program. Tyto zdroje dohromady (registry procesoru, stack, atd.) tvoří kontext běžícího vlákna. Vlákno je část kódu, které neví, kdy bude pozastaveno nebo obnoveno od jádra a také neví, kdy se to stane. Například vlákno je pozastaveno těsně předtím, než běží

instrukce, která sečte 2 hodnoty registrů. Když je vlákno pozastaveno, ostatní vlákna budou běžet a můžou změnit hodnoty registrů procesoru. Po obnovení vlákno nebude vědět, že byly hodnoty registru změněny a následkem toho by byla špatná hodnota výsledku.

Aby bylo zabráněno tomuto typu chyby, je nezbytné, aby po obnovení vlákna mělo vlákno identický kontext, jako těsně před jeho pozastavením. Jádro operačního systému je zodpovědné za to, že uloží kontext vlákna, když je pozastaveno. Jakmile je opět obnoveno, tak jádro operačního systému obnoví uložený kontext těsně předtím, než je vlákno spuštěno. Proces ukládání kontextu vlákna, které je pozastaveno a obnovování kontextu vlákna, které je obnoveno, se nazývá **kontextové přepínání**.

4.2 Klasifikace RTOS

- **Hard**: Jedná se o tak zvané "tvrdé systémy". Úloha se musí stihnout do stanoveného časového limitu. Nestihnutí může mít katastrofální následky. Příklad může být ABS systém v automobilu.
- **Soft**: Úloha se musí stihnout do daného limitu, ale je povolen statický limit. Například jedna úloha z tisíce se nestihne v daném časovém limitu. Přiklad: Seizmické senzory.

4.3 FreeRTOS

FreeRTOS je operační systém reálného času pro vestavěné zařízení. Operační systém je celý napsán v programovacím jazyce C, jenom několik funkcí je vytvořeno v assembleru. Jádro je složeno ze tří souborů a to list.c, queue.c a task.c. FreeRTOS je jednou ze tří distribucí výrobce, která je zdarma. Další 2 verze jsou OpenRTOS(komerční použití) a dále je to SafeRTOS, který splňuje bezpečností normu SIL3. Distribuce FreeRTOS byla zvolena, protože se jedná o verzi, která je zdarma a podporuje procesor ARM Cortex M4, který je součástí používaného mikrokontroléru. Dalším důvodem výběru této distribuce je fakt, že se jedná o jednu z nejvíce rozšířených distribucí. Tím pádem je k ní nejvíce podkladů, jak se systémem pracovat, a také je nejvíce dostupných materiálů [11].

Velkou výhodou systému je jednoduchost a podpora velkého množství platform např. ARM (ARM7, ARM9,Cortex-M3, Cortex-M4,Cortex-A), Atmel AVR,AVR32, HCS12,MicroBlaze, Cortus(APS1, APS3, APS3R, APS5, FPF3, FPS6, FPS8). RTOS je založen na přepínání vláken. Každému vláknu je přiřazena určitá priorita. Čím menší číslo tím je menší priorita.

4.4 Vlákna

Samotná vlákna jsou klasifikována do 3 kategorií:

- **Periodic thread (periodické)** : Mělo by běžet jako konstantní časový interval. Například analogově digitální převodníky a digitálně analogové převodníky.
- **Aperiodic thread (neperiodické):** Obvykle běží, ale čas, kdy je potřeba, aby vlákno běželo, není předvídatelný. Vlákna, která jsou vyvolána lidským zásahem, spadají do této kategorie. Typický příklad: úder tlačítka klávesnice.
- **Sporatic thread (ojedinělý):** Jsou taková vlákna, která běží málo nebo někdy vůbec. Do této kategorie může spadat například: selhaní počítačového hardware, přehřátí.

Vlákno se může nacházet v jednom z následujících stavů:

- **Running** (běžící) : Vlákno je aktuálně běžící, vytěžuje procesor.
- **Ready** (připraveno) : V tomhle stavu jsou vlákna připravena. Nejsou blokována nebo pozastavena. Ale neběží, protože aktuálně běží jiné vlákno se stejnou nebo vyšší prioritou.
- **Blocked** (blokováno): Vlákno v tomto stavu je blokováno a čeká na nějakou externí událost. Například když zavoláme funkci vTaskDelay(), tak tělo funkce bude blokováno, než uběhne perioda zpoždění.
- **Suspended** (pozastaveno): Vlákna v tomto režimu nejsou dostupná plánovači. Vlákna pouze opouští nebo vstupují do pozastaveného stavu a to příkazy vTaskSuspend() a xTaskResume(). Časový limit periody nelze specifikovat.

Nastavení priority:

Čím vyšší číslo priority, tím vyšší priorita vlákna. Rozsah čísla priority pro dané vlákno může být od 0 do (configMAX_Priorities-1). Hodnota *configMAX_Priorities* se nastavuje v hlavičkovém souboru *FreeRTOSConfig.h*.

Implementace vlákna:

Vlákno by mělo mít takovou strukturu:

```
void vATaskFunction( void *pvParameters )
{
    for( ;; )
    {
        -- Task application code here. --
    }
}
```

Vlákna jsou vytvořena zavoláním API funkce *xTaskCreate*() a smazána zavoláním funkce *vTaskDelete*().

4.5 Synchronizační nástroje

Fronta:

Jedná se o základní typ komunikace mezi vlákny. Může být použita k poslaní zprávy mezi vlákny a mezi přerušeními a vlákny. Ve většině případů je používáno FIFO bufferu. Jeho princip lze přirovnat například k automatu na cigarety. První krabička co jde do zásobníku, musí jít také ze zásobníku první ven. Princip je stejný i v tomto případě. Do fronty pošle první vlákno pomocí funkce *xQueueSenttoback*() 3 položky (vytvořená fronta je na 5 položek), tudíž nemůže dojít k jejímu naplnění a druhé vlákno příjme 3 položky pomocí funkce *xQueueReceive*() a fronta se vyprázdní.

Binární semafor:

Binární semafor se využívá pro synchronizaci dvou vláken, nebo také hardwarového přerušení a vlákna. Funkcí *xSemaphoreGive*() se jeho hodnota nastaví na jedna, i když už před tím tuto hodnotu měl. Funkcí *xSemaphoreTake*() se hodnota semaforu nastaví na nulu, ale jen v případě, že měl hodnotu jedna. Pokud se v semaforu již v nule nachází, vlákno se přesune do blokovaného stavu a bude opět spuštěno, až některé z vláken hodnotu semaforu nastaví na jedna.

Čítací semafor:

Je velice podobný binárnímu semaforu, ale binární semafor si představujeme jako frontu o maximální délce 1. U čítacího semaforu ale délka muže být větší než 1. U semaforu není důležité, jaká data jsou uložena ve frontě, ale je důležité, jestli je prázdná nebo ne. Hodnota semaforu se inkrementuje pomocí funkce *xSemaphoreGive*() a pomocí funkce *xSemaphoreTake*() se hodnota dekrementuje. Jak bylo řečeno, nezajímá nás konkrétní uložená hodnota. Pokud je hodnota semaforu 0, tak je vykonávání vlákna zablokováno. Pokud je hodnota větší než 0, tak vykonání vlákna nebude zablokováno.

Mutex:

Jedná se o synchronizační nástroj, jehož princip je jednoduchý: když mám 2 vlákna, tak může pracovat pouze vlákno, které vlastní mutex. Druhé vlákno je zablokováno, dokud mutex nebude volný, aby mohlo začít pracovat. První vlákno si vezme pomocí příkazu *xSemaphoreTake*() a pracuje, druhé vlákno je blokováno. První vlákno dokončí práci a vrátí mutex pomocí příkazu *xSemaphoreGive*(). Mutex je nyní volný, druhé vlákno pomocí funkce *xSemaphoreTake*() vezme mutex a následně zahajuje činnost. Po dokončení činnosti mutex zase vrátí pomocí dříve uvedeného příkazu.

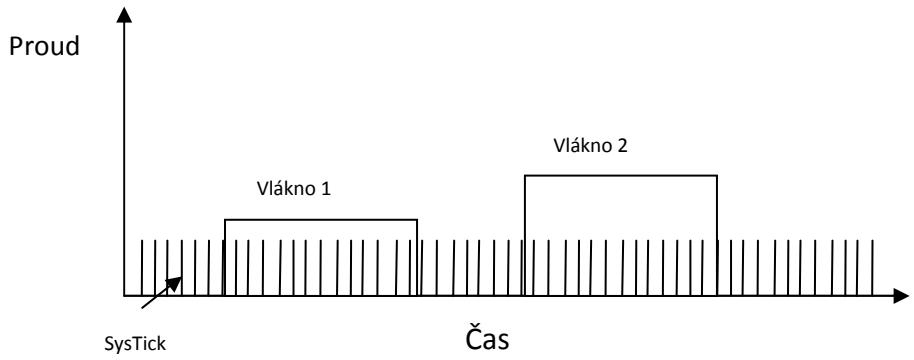
4.6 Tickless mode

Je běžné, že ke snížení energie spotřebované mikrokontrolérem, na kterém běží FreeRTOS, je úspory energie dosaženo za pomocí tak zvaného "Idle task hook" (kap. 4.6.2), ve kterém je definována funkce, která přepne mikrokontrolér do stavu s nízkou spotřebou. Úspora energie, které může být dosaženo tímto jednoduchým způsobem, je omezena nutností periodicky opouštět a znovu vstupovat do režimu s nízkou spotřebou za pomocí přerušení. Frekvence tiků přerušení je velmi vysoká. Jedná se o neefektivní metodu.

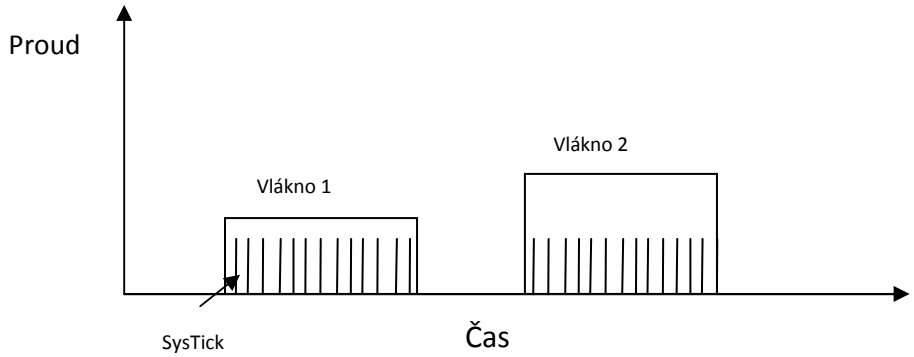
FreeRTOS tickless mód eliminuje periodické tiky během idle periody (perioda, během které běží vlákno idle (kap. 4.6.1) s nejnižší prioritou). A pak provede korekční nastavení hodnoty RTOS čítače tiků, když je přerušení tiků restartováno. Zastavení přerušení tiků umožňuje mikrokontroléru zůstat v režimu nízké spotřeby, dokud se neobjeví přerušení, nebo nastal čas k přechodu vlákna do stavu připraven.

Povolení tickless se provádí v *FreeRTOSConfig.h* a to konkrétně nastavení hodnoty u *configUSE_TICKLESS_IDLE* na 1. Když je tickless vlákno povoleno, tak jádro zavolá makro *portSUPPRESS_TICKS_AND_SLEEP()*. Hodnota *portSUPPRESS_TICKS_AND_SLEEP()* je parametr, který se rovná součtu period tiků předtím, než je vlákno přesunuto do běžícího stavu. Hodnota parametru je tedy čas, při kterém může mikrokontrolér bezpečně zůstat ve stavu nízké spotřeby, kde jsou potlačeny periodické tiky.

Pro lepší představu a pochopení problematiky RTOS tickless módu, je na obrázku 4.2 zobrazen režim bez tickless módu a na obrázku 4.3 RTOS s režimem tickless.



Obr. 4.2: Průběh znázorňující RTOS bez aktivního režimu tickless



Obr. 4.3: Průběh znázorňující RTOS s aktivním režimem tickless

4.6.1 Idle vlákno

Idle vlákno je vytvořeno automaticky, když je RTOS plánovač spuštěn. Zajistí se, že je vždy alespoň jedno vlákno, které je připraveno k běhu. Je vytvořeno na nejnižší možné prioritě.

4.6.2 Idle task hook

Idle vlákno může volitelně volat funkci definující hook funkci- idle hook. Idle vlákno běží na nejnižší prioritě, takže idle hook funkce může být prováděna, když neběží vlákna s vyšší prioritou. Tak je docíleno, že idle hook funkce je ideální k uvedení procesoru do stavu s nízkou spotřebou. Poskytuje automatickou úsporu, když nejsou žádné jiné vlákna k provedení.

Idle hook funkce bude zavolána, když je v hlavičkovém souboru *FreeRTOSConfig.h* nastavena u parametru *configUSE_IDLE_HOOK* hodnota 1. Idle hook je volána opakovaně tak dlouho, jak běží idle vlákno.

5 Implementace bezdrátového mesh protokolu do prostředí FreeRTOS

Jednou z důležitých částí implementace bezdrátového mesh protokolu do prostředí FreeRTOS bylo zachování logické struktury původního mesh protokolu. Bylo nutné vytvoření vlastní HAL vrstvy pro komunikaci mezi mikrokontrolérem a transceiverem. Podrobnému popisu HAL vrstvy je věnována kapitola 5.1. Poté bylo důležité nalézt vhodnou variantu implementace do prostředí FreeRTOS (kap. 5.2).

Pro implementaci bezdrátového mesh protokolu byla použita modifikovaná verze FreeRTOS. Modifikovaná verze se od původní verze liší tím, že je rozšířena o nové funkce.
V hlavičkovém konfiguračním souboru *FreeRTOSConfig.h* je na rozdíl od původní verze přidáno několik definicí. První z nich je definice *configUSE_TICKLESS_IDLE*, jejíž hodnota říká, zda systém bude používat tickless režim. Pokud systém bude používat tickless režim, tak v následujícím parametru *configSLEEP_MODE* je možnost z volby jednoho z tří nízko energetických režimů.

Jestliže je nastaven *configSLEEP_MODE* na hodnotu 1, tak je jako zdroj systémových hodin použit SYSTICK.

Jestliže je hodnota *configSLEEP_MODE* nastavena na 2, tak jako zdroj systémových hodin je použita periferie RTC. Pro snížení spotřeby energie je pro periferii RTC použita dělička kmitočtu dvěma. V definici *configCRYSTAL_IN_EM2* je také možné zvolit si jakožto zdroj hodinového signálu pro periferii RC oscilátor nebo krystalický oscilátor. Je-li použit RC oscilátor, tak je dosaženo lepší spotřeby energie, ale je-li použit krystalický oscilátor, tak je dosaženo přesnějšího časování.

V parametru *configTICK_RATE_HZ* je možné nastavit si frekvenci signálu, nachází- li se systém v nízkoenergetickém režimu.

Jestliže je hodnota *configSLEEP_MODE* nastavena na 3, tak je jakožto zdroj systémových hodin použita periferie BURTC.

V souboru *low_power_tick_management.c* se nachází inicializace periferií RTC a BURTC, obsluhy přerušení periferií RTC a BURTC a také přepisy původních funkcí, a to funkcí *vPortSetupTimerInterrupt()* a *vPortSuppressTicksAndSleep().*

Ve funkci *vApplicationIdleHook* je přepsána původní definice této funkce. Je-li v konfiguračním souboru nastavena definice *configUSE_SLEEP_MODE_IN_IDLE* na hodnotu 1 a zároveň *configUSE_TICKLESS_IDLE == 0* je zavolána funkce *SLEEP_Sleep*, která uspí systém do požadovaného nízkoenergetického režimu. Pokud tomu tak není a je nastaven parametr *configUSE_TICKLESS_IDLE == 1*, tak funkce *vApplicationIdleHook* není nikdy volána a je použit tickless režim.

Ve funkci *RTC_IRQHandler* je definována obsluha přerušení pro RTC, který je použit jakožto zdroj systémových hodin, nachází-li se systém v nízkoenergetickém režimu 2, tedy je-li *configSLEEP_MODE == 2*. V obsluze přerušení je vykonáno nastavení jedné periody tiku, restartování RTC, nastavení flagu, že bylo přerušení vykonáno a také k inkrementaci tiku.

Ve funkci *vPortSetupTimerInterrupt* je přepsána původní definice funkce *vPortSetupTimerInterrupt*. Funkce stanovuje zdroj přerušení systémových tiků. V modifikované verzi FreeRTOS zde proběhne inicializace periferií, které slouží jako zdroj systémových hodin. Je-li nastaven v hlavičkovém souboru *FreeRTOSConfig.h* parametr *configSLEEP_MODE == 2* proběhne zde nainicializování RTC, nastavení jeho priority přerušení a také povolení jeho přerušení v NVIC. Proběhne zde také nainicializování RC oscilátoru nebo krystalického oscilátoru, záleží na volbě uživatele, který zvolí v konfiguračním souboru *FreeRTOSConfig.h*. Je-li parametr *configSLEEP_MODE == 3* proběhne nainicializování BURTC, nastavení jeho priority přerušení a také povolení jeho přerušení v NVIC.

Ve funkci *vPortSuppressTicksAndSleep* je přepsána původní definice této funkce. Vstupní parametr této funkce je parametr *xExpectedIdleTime*, volně přeloženo jako „očekávaný stav nečinnosti“. Funkce je použita pouze tehdy, je-li v konfiguračním souboru *FreeRTOSConfig.h* nastaveno *configUSE_TICKLESS_IDLE == 1* a zároveň *configSLEEP_MODE == 2* nebo *configSLEEP_MODE == 3*. Je-li nastaven parametr *configSLEEP_MODE* na hodnotu 2, tak je použit jako zdroj systémových hodin RTC. V prvním kroku dojde ve funkci k ujištění, zda vstupní hodnota není větší než maximální možná hodnota, pokud ano je vstupní hodnota snížena na maximální možnou hodnotu. Poté je spočítána hodnota *ulReloadValue,* parametr, který říká jak dlouho má systém spát. Po té je zastaveno RTC a zakázáno přerušení a nastavena hodnota *intTickFlag* na false. V následujícím kroku je nastavena nová hodnota *ulReloadValue* a zavolána funkce *RTC_CompareSet*, pomocí které je nastaveno do jaké hodnoty má RTC počítat, jehož vstupní parametr bude hodnota *ulReloadValue*. Poté dojde k samotnému uspání mikrokontroléru. V následujícím kroku dojde k *ulRemainingCounter = RTC_CounterGet(),* tedy načtení aktuální hodnoty RTC časovače do proměnné *ulRemainingCounter*, zastavení časovače a povolení přerušení. Nastane-li v průběhu nastavené periody, kdy se systém nachází v jednom z nízkoenergetických režimů, jiné přerušení, tak dojde k přepočtu parametru *ulReloadValue,* aby nedošlo ke změně definované periody.

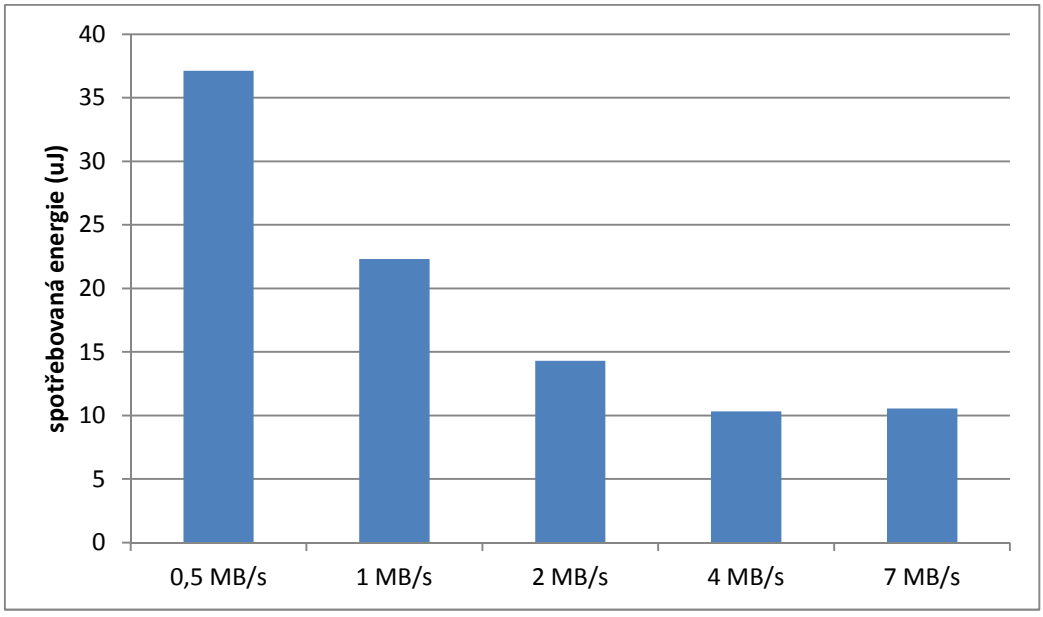
Ve funkci *BURTC_IRQHandler* je definována obsluha přerušení BURTC, které je použito jako zdroj systémových hodin, nachází-li se systém v nízkoenergetickém režimu 3, tedy je-li nastaven v hlavičkovém souboru *FreeRTOSConfig.h* parametr *configSLEEP_MODE == 3*. V obsluze přerušení jsou funkce, které slouží k nastavení jedné periody tiku, restartování čítače BURTC, nastavení flagu, že bylo přerušení vykonáno také k inkrementaci tiků.

5.1 Implementace hardwarové vrstvy

V pohledu celkové struktury programu je HAL vrstva nejnižší vrstvou protokolu. Jak již bylo zmíněno v úvodu kapitoly 5, tak v HAL vrstvě je řešena komunikace transceiveru s mikrokontrolérem, tedy správné nastavení portů, periferií, přerušení a nastavení SPI sběrnice.

SPI sběrnice:

Pro datovou komunikaci transceiveru a mikrokontroléru byla využita SPI sběrnice. Inicializace SPI sběrnice probíhá ve funkci *halPhyInit().* Pro nalezení optimální přenosové rychlosti, a to z hlediska spotřeby a přenosové rychlosti, SPI sběrnice, bylo provedeno několik měření pro rychlosti sběrnice 0,5 MB/s až 7 MB/s. Jednalo se o přenos 109 bajtů do bufferu transceiveru. Na obrázku 5.1 jsou graficky znázorněny naměřené hodnoty celkové spotřebované energie.



Obr. 5.1: Závislost přenosové rychlosti SPI na proudovém odběru

Tab. 5.1: Vypočtené hodnoty energie pro jednotlivé rychlosti přenosu

Rychlost přenosu	Počet cyklů	Celkový čas (us)	Proud (mA)	Spotřebovaná energie (uJ)
0,5 MB/s	33360	2400	4,59	37,12
1 MB/s	20171	1440	4,60	22,32
2 MB/s	12977	926	4,58	14,29
4 MB/s	9380	668	4,58	10,31
7 MB/s	9380	668	4,69	10,55

Z naměřených hodnot (obr. 5.1) a vypočtených hodnot (tab. 5.1) byla zvolena jakožto optimální rychlost nejvyšší rychlost 4 MB/s. Transceiver je schopen pracovat s rychlostmi sběrnice do 8 MB/s.

Následující řádky ukazují, jak je řešeno nastavení SPI sběrnice:

```
spi->CTRL |= (USART_CTRL_SYNC) |
  (USART_CTRL_CLKPOL_IDLELOW | USART_CTRL_CLKPHA_SAMPLELEADING)
|(USART_CTRL_MSBF);
  spi->CLKDIV = 256 * (SPI_PERCLK_FREQUENCY /(2*SPI_BAUDRATE) - 1);
  spi->CMD = USART_CMD_MASTEREN;
  spi->CMD = (uint32_t) (USART_CMD_RXEN | USART_CMD_TXEN);
  spi->CTRL &= ~USART_CTRL_TXBIL;
  spi->ROUTE = USART_ROUTE_TXPEN | USART_ROUTE_RXPEN |
USART_ROUTE_CLKPEN | (1 << LOCATION)
```

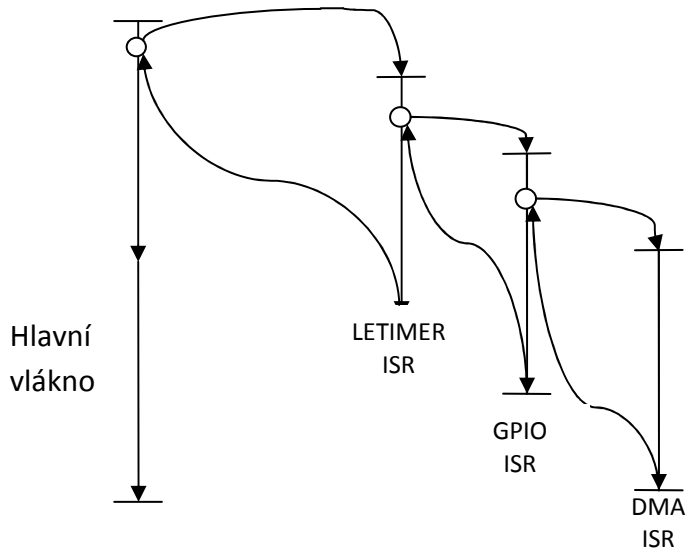
Proměnná „spi“ je struktura, která přestavuje jeden ze tří možných voleb USARTů, možnost volby je definována pomocí makra „spi“ které je definováno v hlavičkovém souboru *halPhy.h.* Stejné nastavení platí pro parametr „LOCATION“.

Nastavení vstupních a výstupních pinů:

Pro komunikaci je definováno 5 výstupních a 2 vstupní piny. Definice je pinů je provedena v hlavičkovém souboru *halPhy.h* a jejich nastavení ve funkci *halPhyInit().*

Nastavení přerušení

Přerušení je nastaveno na pinu PD4, který odpovídá pinu přerušení. Pin je nastaven tak, že reaguje na náběžnou hranu. Povolení přerušení je nastaveno pomocí příkazu *NVIC_EnableIRQ(GPIO_EVEN_IRQn).* Vzhledem k počtu zdrojů přerušení bylo nutné nastavit správně jejich priority (obr. 5.2)., aby nedocházelo k ovlivňování plynulosti běhu programu, nastane-li situace, že dojde k přerušení s vyšší prioritou, nebo nachází-li se program v obsluze přerušení periferie s nižší prioritou.



Obr. 5.2: Nastavení priorit přerušení pro jednotlivé periferie

Časovač

Časovač je nastaven v čítacím módu, při přetečení časovače je vygenerováno přerušení a následně obsluha přerušení *TIMER0_IRQHandler()* inkrementuje hodnotu halTimerIrqCount, s kterou dále pracuje systémový časovač ve funkci *SYS_TimerTaskHandler(),* který se nachází v souboru sysTimer.c. V souboru hal.c se nachází definice „PERFREQ“ ,kde je definována frekvence vysokofrekvenčních hodin periferií.

$$T = \frac{PRESCALER}{\mathrm{PERFREQ}} * TOPVAL \quad (s;\ -;\ Hz;\ -) \tag{5.1}$$

Podle vzorce (5.1) je vypočtena hodnota intervalu, který časovač periodicky generuje. Hodnota intervalu je stanovena na 10 ms. Jedná se o nejmenší hodnotu, se kterou je schopen systémový časovač pracovat. Hodnota $PRESCALER$ je proměnná která je nastavena v závislosti na hodnotě frekvence vysokofrekvenčních hodin periferií. $PRESCALER$ je dělička, která dělí zdroj hodinového signálu časovače, tedy signál vysokofrekvenčních hodin. $TOPVAL$ je hodnota do které časovač čítá, než dojde k přetečení. Dojde-li k přetečení, je vygenerováno přerušení, následně proběhne jeho obsluha.

AES šifrování

Jedná se algoritmus, který používá sdílený klíč. Stejný klíč slouží k zašifrování i dešifrování dat. Délka klíče je buď 128, 196 nebo 256 bitů.

LWM stack byl rozšířen o možnost hardwarového AES šifrování, které provádí přímo periferie mikrokontroléru. Na výběr jsou 3 možnosti šifrování. Softwarové šifrování XTEA nebo hardwarové šifrování AES. Hardwarové šifrování může provádět přímo transceiver nebo nově také přímo mikrokontrolér. Z bezpečnostního hlediska je bezpečnější, když provádí AES šifrování přímo mikrokontrolér. Provádí-li jej transceiver, je značným bezpečnostním rizikem, že komunikace mezi mikrokontrolérem a transceiverem probíhá po SPI sběrnici, která není šifrována.

5.2 Implementace mesh protokolu

Kapitola se zaobírá porovnáním jednotlivých implementací a nelezením nejoptimálnější implementace mesh protokolu do prostředí FreeRTOS. Za tímto účelem bylo vytvořeno několik aplikací a poté změřeny jejich charakteristiky pro příjem a odesílání dat, které znázorňují průběh programu, tedy periodické odesílání dat nebo jejich příjem. Následně bylo provedeno porovnání naměřených charakteristik a poté z naměřených a vypočtených hodnot nalezena nejoptimálnější varianta implementace mesh protokolu do FreeRTOS.
V grafických průbězích odesílání dat je celkový proud roven proudu odebíraným mikrokontrolérem, tedy procesorem a aktivními periferiemi. A v případě grafických průběhů příjmu dat se jedná také proud odebíraný mikrokontrolérem, pouze s tím rozdílem, pro lepší přehlednost grafu nebyl nainicializován nízkoenergetický časovač (kap. 5.3).

5.2.1 Aplikace A

Aplikace A je rozdělena do 2 vláken:

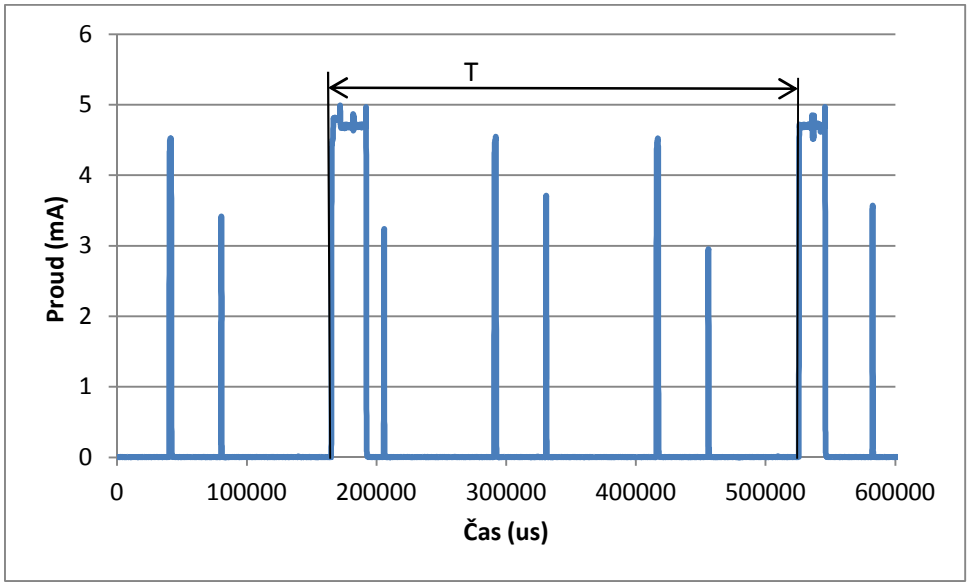
1. systaskhandler
2. apptaskhandler

První vlákno tvoří samostatný stack mesh protokolu a druhé vlákno tvoří samotná aplikace. Pro příjem dat je využito přerušení, které nastane, indikuje-li transceiver příchozí data. Nastavení požadovaného režimu přerušení od transceiveru proběhne v inicializační funkci *PHY_Init(),* jak je zobrazeno v níže uvedeném zdrojovém kódu:

```
phyWriteRegister(IRQ_MASK_REG, 0x00);
phyReadRegister(IRQ_STATUS_REG);
phyWriteRegister(IRQ_MASK_REG, (1 << 2));
```

Zápisem hodnoty do registru transceiveru IRQ_MASK_REG nastavím požadovaný režim přerušení. Zápisem hodnoty (1 << 2) nastavím režim přerušení RXIND, tedy indikuje-li transceiver nový příchozí rámec, tak nastane přerušení.

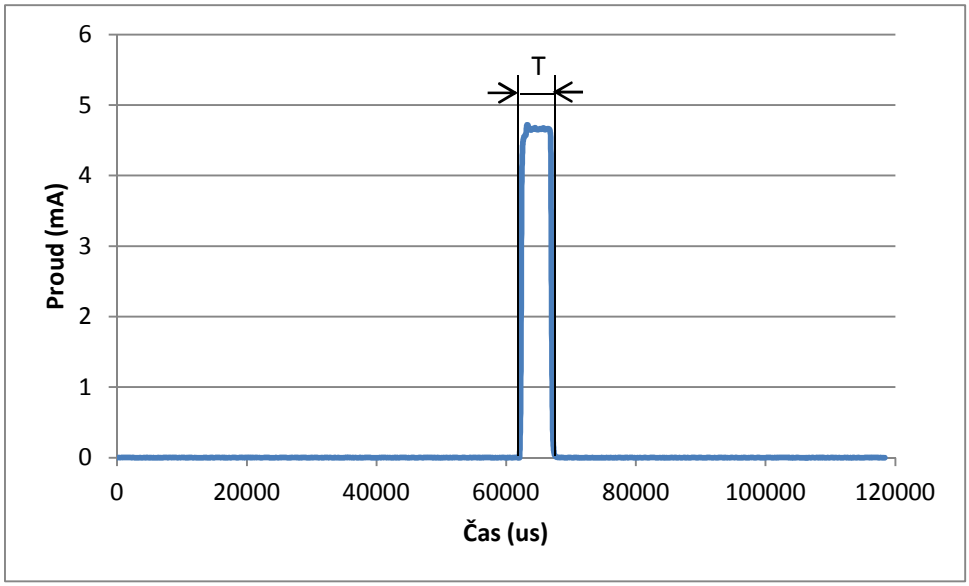
V obsluze přerušení *GPIO_EVEN_IRQHandler()* jsou pomocí SPI sběrnice vyčtena data z bufferu transceiveru do bufferu v RAM paměti mikrokontroléru. Následně ve funkci *PHY_TaskHandler()* zpracována a uložena do struktury PHY_DataInd, ze které jsou data dále zpracována ve funkci *nwkRxTaskHandler().* Na obrázku 5.4 je graficky zobrazen průběh příjmu dat, na obrázku 5.3 je zobrazen průběh odesílání dat. Vypočtené hodnoty celkové spotřebované energie při odesílání dat za jednu periodu a změřený počet cyklů jsou uvedeny v tabulce 5.2 Vypočtené hodnoty celkové spotřebované energie při příjmu dat za dobu obsluhy přerušení a změřený počet cyklů jsou uvedeny v tabulce 5.3.



Obr. 5.3: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 48 MHz

Tab. 5.2: Počet cyklů a celková spotřebovaná energie za periodu při odesílání dat

Počet cyklů za dobu jedné periody	5064347
Celková spotřebovaná energie za dobu jedné periody (uJ)	381



Obr. 5.4: Závislost proudového odběru při přijmu dat na čase při frekvenci jádra 48 MHz

Tab. 5.3: Počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat

Počet cyklů za dobu jedné periody	13347
Celková spotřebovaná energie za dobu jedné periody (uJ)	4,2

5.2.2 Aplikace B

Aplikace B je stejně jako aplikace A rozdělena do dvou vláken, ale na rozdíl od Aplikace A je v Aplikaci B využito pro odesílání a příjem dat DMA.

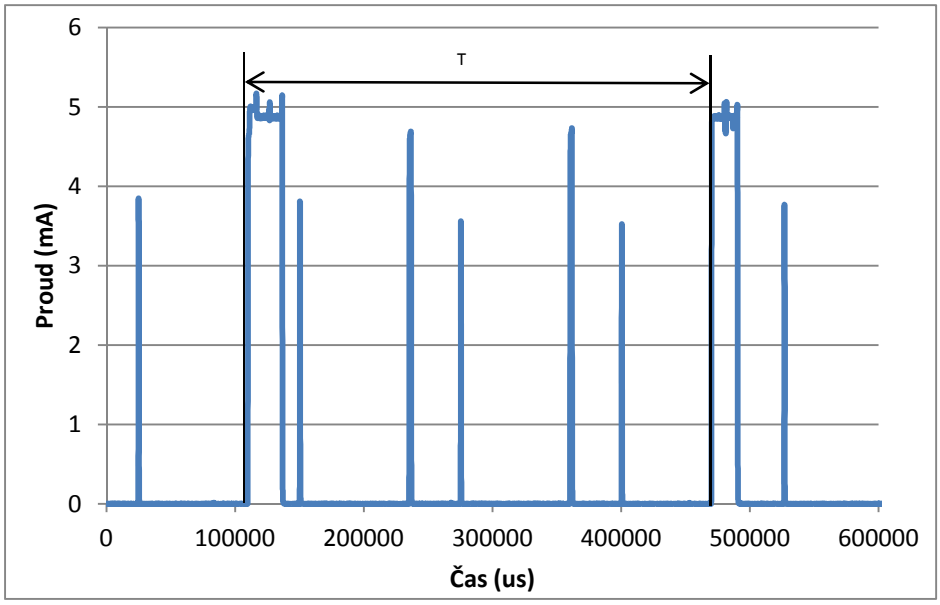
Výhoda DMA řadiče je ta, že je schopen provádět paměťové operace nezávisle na procesoru. To přináší výhodu snížení spotřeby energie a zátěže CPU a povoluje systému zůstat v režimu s nízkou spotřebou, když přenáší data z USARTu do RAM.

Pro příjem dat je také použit **cyklický buffer**. Cyklický buffer je oblast v RAM paměti, která je určena pro uložení příchozích dat, když je buffer plný, nová data jsou zapsána na začátek bufferu a stará data jsou přepsána.

V obsluze přerušení *GPIO_EVEN_IRQHandler()* jsou z bufferu transceiveru vyčtena pomocí DMA do bufferu v RAM paměti mikrokontroléru. Probíhá-li přenos dat z bufferu transceiveru do bufferu v RAM paměti mikrokontroléru, tak se mikrokontrolér nachází v nízkoenergetickém režimu, což umožňuje úsporu energie mikrokontroléru. Poté jsou v obsluze přerušení do cyklického bufferu zapsána data. Data jsou z bufferu ve vlákně vyčtena ve funkci *PHY_TaskHandler()* a následně jsou ve funkci data zapsána do struktury

PHY_DataInd a stejně jako v Aplikaci A dále zpracována ve funkci *nwkRxTaskHandler().* Měřením bylo zjištěno, že výše popsané řešení, které je implementované v Aplikaci B, je na rozdíl od řešení, které je implementované v Aplikaci A, efektivnější, protože obsluha přerušení trvá 6115 strojových cyklů (tab. 5.5). Na rozdíl od řešení v aplikaci A, které trvá 13347 strojových cyklů (tab. 5.3).

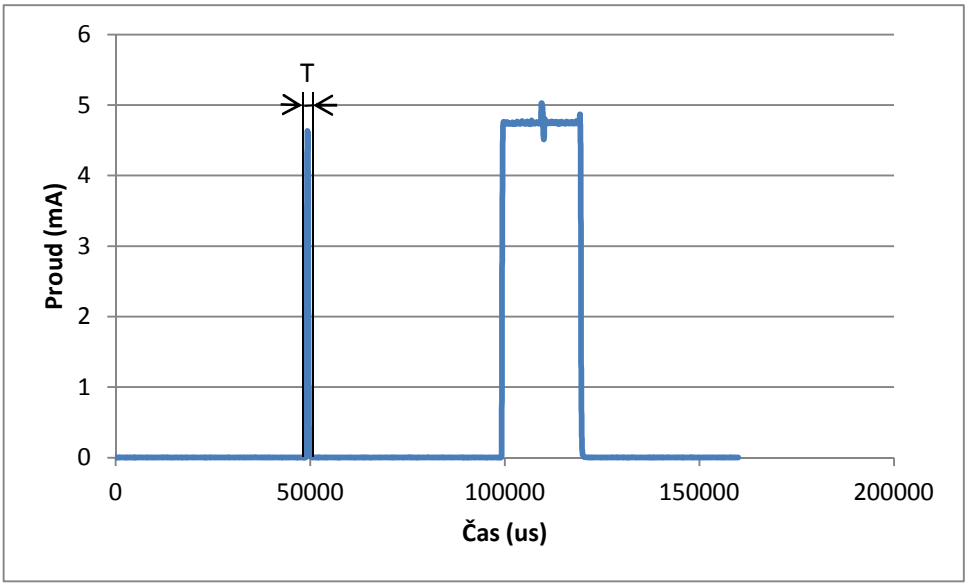
Na obrázku 5.5 je zobrazeno odesílání dat a na obrázku 5.6 příjem dat.



Obr. 5.5: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 48 MHz

Tab. 5.4: Počet cyklů a celková spotřebovaná energie za periodu při odesílání dat

Počet cyklů za dobu jedné periody	4855128
Celková spotřebovaná energie za dobu jedné periody (uJ)	360



Obr. 5.6: Závislost proudového odběru při příjmu dat na čase při frekvenci jádra 48 MHz

Tab. 5.5: Počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat

Počet cyklů za dobu jedné periody	6720
Celková spotřebovaná energie za dobu jedné periody (uJ)	2,11

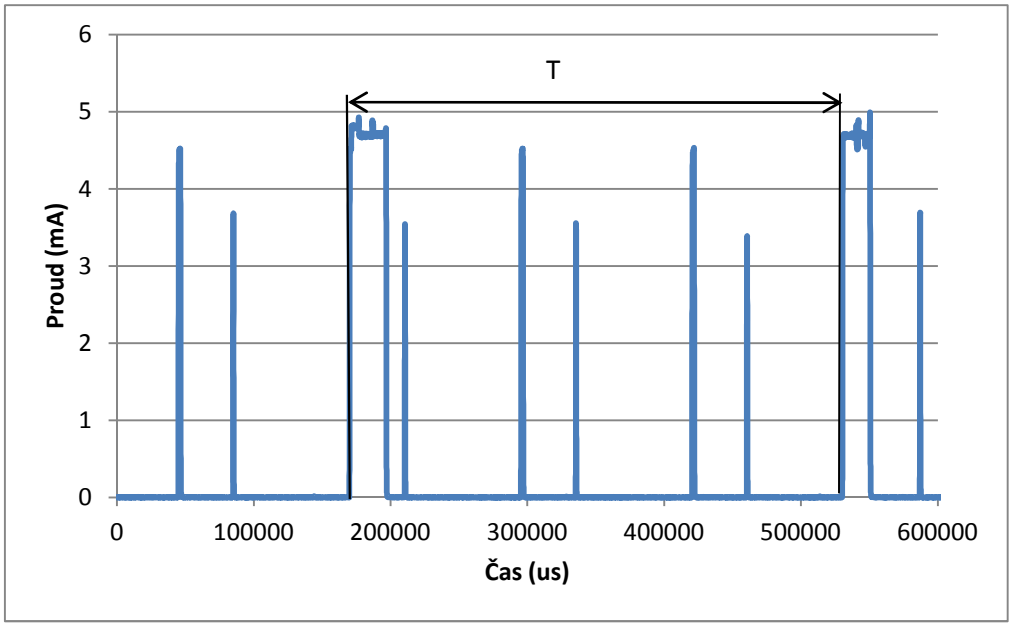
5.2.3 Aplikace C

Aplikace C je na rozdíl od předchozích dvou aplikací rozdělena do sedmi vláken:

I. PHY_TaskHandler
II. nwkRxTaskHandler
III. nwkTxTaskHandler
IV. nwkDataReqTaskHandler
V. nwkSecurityTaskHandler
VI. SYS_TimerTaskHandler
VII. apptaskhandler

Princip odesílání a příjmu dat je stejný jako v Aplikaci A.

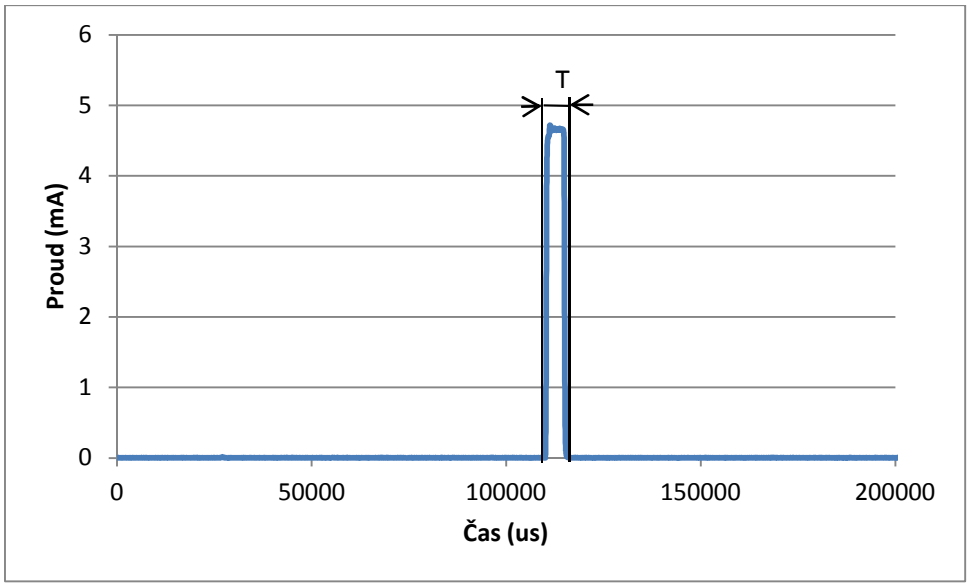
Na obrázku 5.7 je graficky zobrazen průběh odesílání dat a na obrázku 5.8 je graficky zobrazen průběh příjmu dat.



Obr. 5.7: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 48 MHz

Tab. 5.6: Počet cyklů a celková spotřebovaná energie za periodu při odesílání dat

Počet cyklů za dobu jedné periody	5564846
Celková spotřebovaná energie za dobu jedné periody (uJ)	400



Obr. 5.8: Závislost proudového odběru při příjmu dat na čase při frekvenci jádra 48 MHz

Tab. 5.7: Počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat

Počet cyklů za dobu jedné periody	13347
Celková spotřebovaná energie za dobu jedné periody (uJ)	4,2

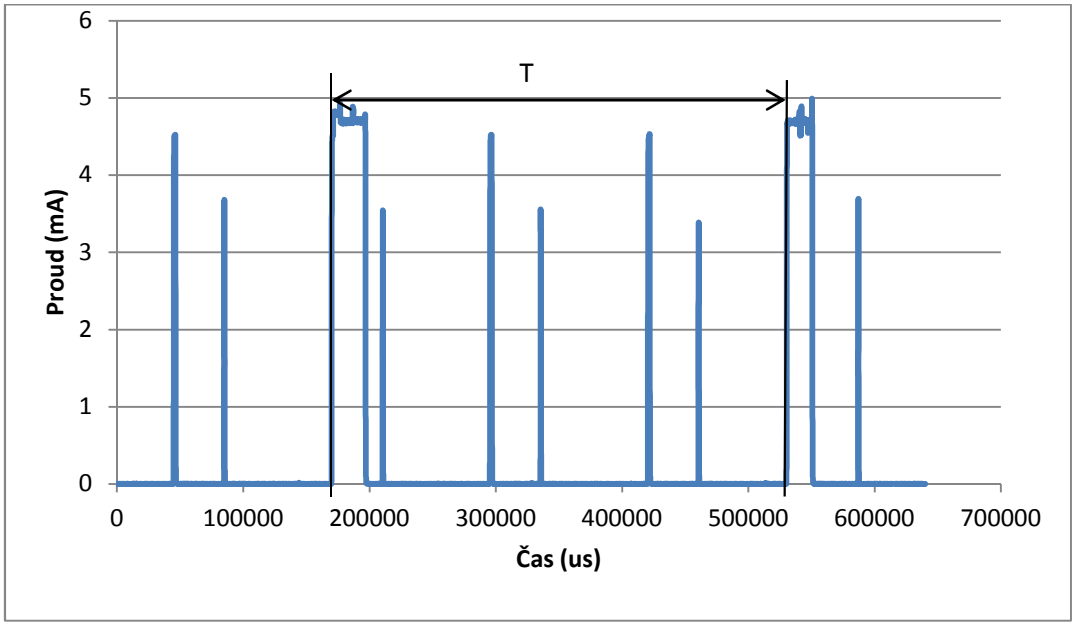
5.2.4 Aplikace D

Aplikace D je stejně jako Aplikace C rozdělena do sedmi vláken:

I. PHY_TaskHandler
II. nwkRxTaskHandler
III. nwkTxTaskHandler
IV. nwkDataReqTaskHandler
V. nwkSecurityTaskHandler
VI. SYS_TimerTaskHandler
VII. apptaskhandler

Na rozdíl od předchozí aplikace je ale zde použit jako v Aplikaci B DMA a cyklický buffer.

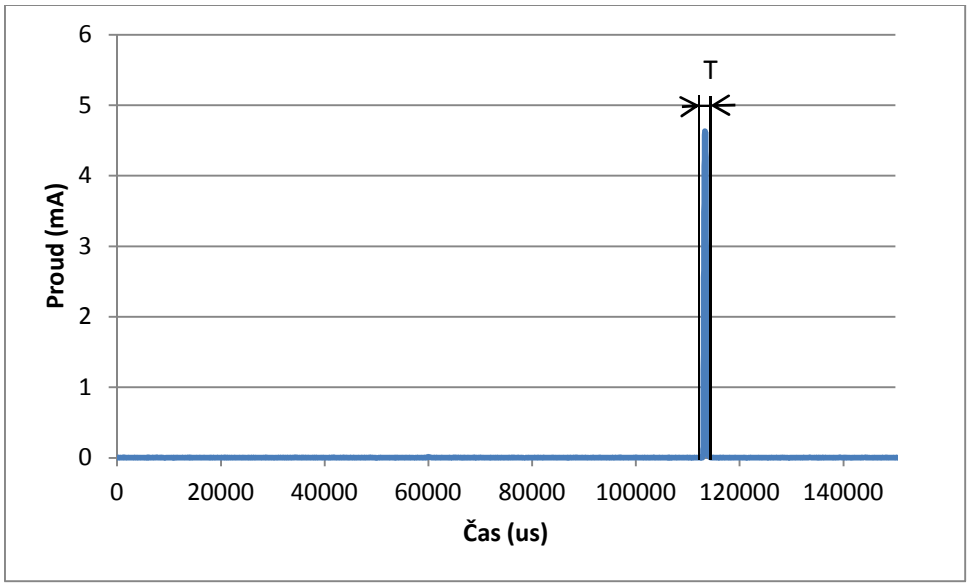
Průběh přijímání dat je zobrazen na obrázku 5.10 a průběh přijímání dat je zobrazen na obrázku 5.9.



Obr. 5.9: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 48 MHz

Tab. 5.8: Počet cyklů a celková spotřebovaná energie za periodu při odesílání dat

Počet cyklů za dobu jedné periody	5435185
Celková spotřebovaná energie za dobu jedné periody (uJ)	383



Obr. 5.10: Závislost proudového odběru při příjmu dat na čase při frekvenci jádra 48 MHz

Tab. 5.9: Počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat

Počet cyklů za dobu jedné periody	6115
Celková spotřebovaná energie za dobu jedné periody (uJ)	1,92

5.2.5 Finální aplikace

Finální aplikace je rozdělena do 5 vláken:

I. Dispather
II. Phytask
III. Nwktask
IV. Sys timer
V. App task

V souboru main.c je definován parametr TASK_MAX_NUM, jehož hodnota určuje maximální možnou hodnotu počtu vláken.

V řešení finální aplikace byla vytvořena struktura *TaskWorkerParameters*, která obsahuje 2 vstupní parametry. První z parametrů je parametr *xSemaphoreHandle mutex,* pomocí funkce

xSemaphoreHandle dojde k definování mutexu. Druhý z parametrů struktury, je parametr *void(*handler)().* Jedná se o ukazatel na funkci.

Pomoc funkce *TaskCreate* vytvoříme vlákno. Funkce TaskCreate má dva vstupní parametry a to jsou *void(*handler)()* a druhý parametr *unsigned portBASE_TYPE prio*. První ze dvojice vstupních parametrů je ukazatel na funkci a druhý parametr je priorita daného vlákna.

Funkce vytvoří pomocí funkce *vSemaphoreCreateBinary* mutex. Pomocí další funkce *xSemaphoreTake* si vezme vytvořený mutex a pomocí funkce *xTaskCreate* dojde k vytvoření funkce pro RTOS plánovač a v posledním kroku funkce dojde k inkrementaci hodnoty *task_num*, která určuje počat vláken.

Níže uvedený kód zobrazuje tělo funkce TaskCreate:

```
 xtasks[task_num].handler = handler;
  vSemaphoreCreateBinary(xtasks[task_num].mutex);
  xSemaphoreTake(xtasks[task_num].mutex, portMAX_DELAY);
  xTaskCreate(TaskWorker, "", STACK_SIZE_FOR_TASK,&xtasks[task_num],
prio, NULL);
  task_num++;
```

Přepínání vláken je realizováno pomocí mutexů, a to tak že, vlákno TaskWorker vyčítá parametry struktury, tedy mutex a funkci. Čeká na mutex a poté spustí funkci a poté odevzdá mutex řídícímu vláknu dispatcher.

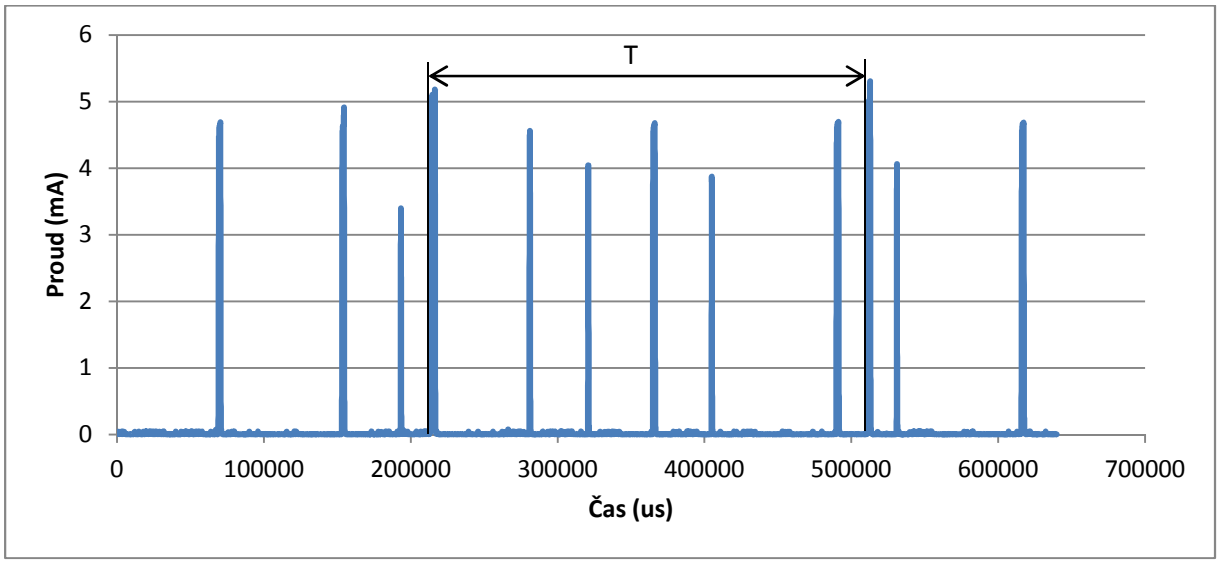
Vlákno dispatcher, jehož tělo je zobrazeno níže:

```
  while(1)
  {
    uint32_t i;
    for(i = 0; i < task_num; i++)
    {
      xSemaphoreGive(xtasks[i].mutex);
      xSemaphoreTake(dispatcher_mutex, portMAX_DELAY);
    }
```

Z výšce uvedeného kódu lze vidět, že princip vlákna dispatcher je založen na odevzdávání a přijímání mutexů. Vlákno dispatcher má za úkol přepínání mezi vlákny, tedy řídí běh celé aplikace.

Na základě dosažených výsledků u Aplikace A a Aplikace D bylo u finální aplikace použito také DMA, a také byl pro zápis přijímaných dat do bufferu použit cyklický buffer, stejně jako u předchozích dvou aplikací.

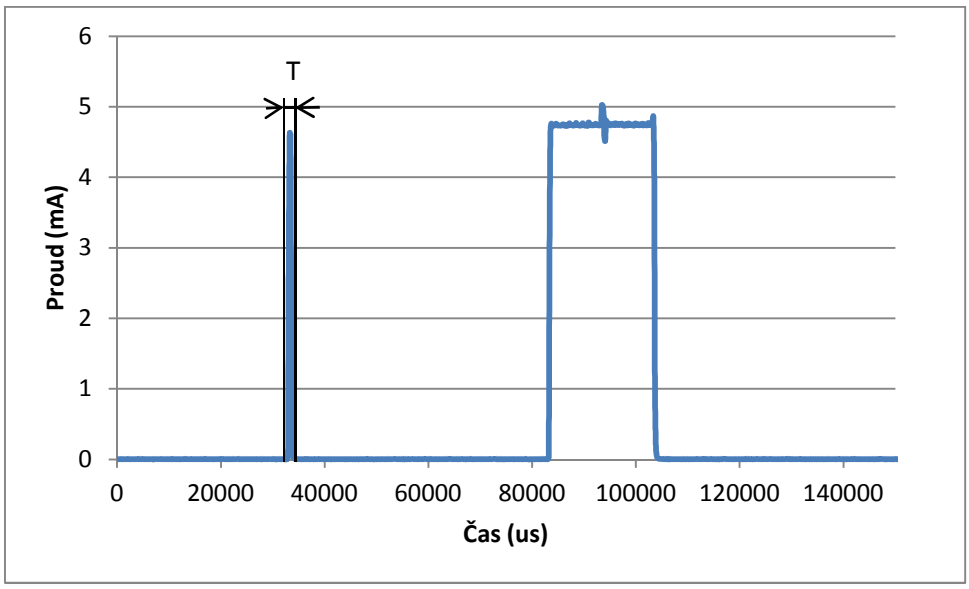
Na obrázku 5.11 je zobrazen průběh odesílání a na obrázku 5.12 je zobrazen průběh příjmu dat. A v tabulce 5.10 počet cyklů a celková spotřebovaná energie za periodu při odesílání dat a v tabulce 5.11 počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat.



Obr. 5.11: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 48 MHz

Tab. 5.10: Počet cyklů a celková spotřebovaná energie za periodu při odesílání dat

Počet cyklů za dobu jedné periody	4673716
Celková spotřebovaná energie za dobu jedné periody (uJ)	139



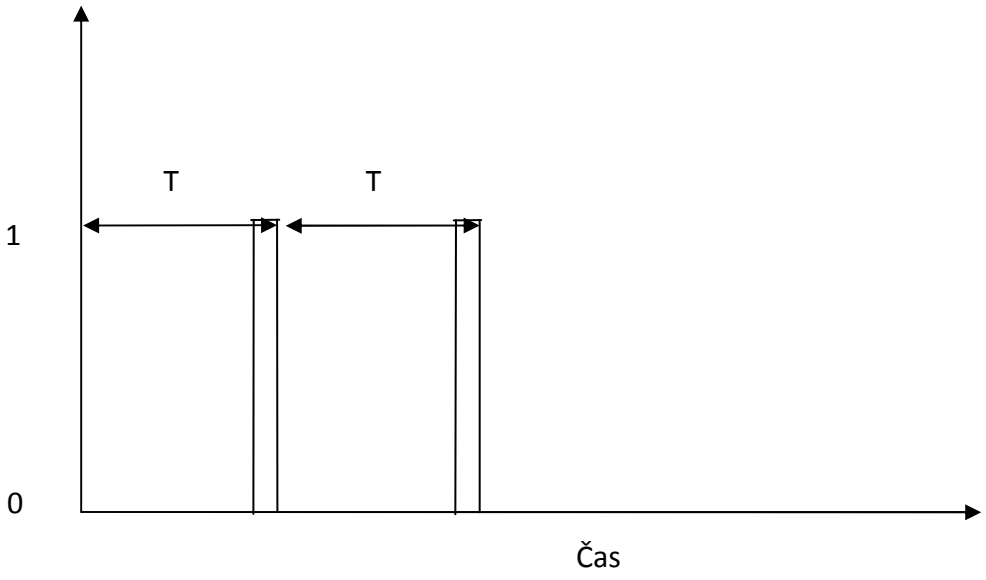
Obr. 5.12: Závislost proudového odběru při příjmu dat na čase při frekvenci jádra 48 MHz

Tab. 5.11: Počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat

Počet cyklů za dobu jedné periody	6115
Celková spotřebovaná energie za dobu jedné periody (uJ)	1,92

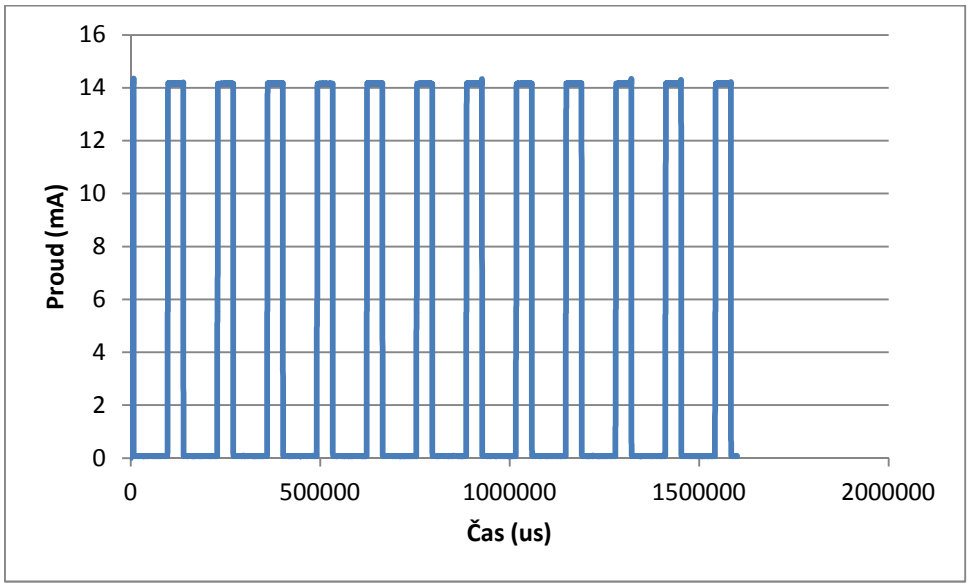
5.3 Řízení spotřeby pomocí nízkoenergetického časovače

Za účelem snížení spotřeby transceiveru byl v projektu implementován nízkoenergetický časovač. Jednou z jeho charakteristických vlastností je ta, že je schopen pracovat v nízkoenergetickém režimu. Nízkoenergetický časovač vytváří střídu, tj. výstupní periodický signál, přechází z jedné úrovně do druhé a naopak. Dochází tedy k periodickému vypínání a zapínání transceiveru, s libovolně definovatelnou střídou. Z hlediska maximální úspory je nejvhodnější nastavit střídu co nejvyšší. Je ale potřeba vzít v úvahu fakt, že jednotlivé uzly nejsou synchronní. Kdyby byla nastavena velmi vysoká střída, docházelo by k velké ztrátovosti přijímaných paketů, protože transceiver by se delší dobu z periody nacházel v režimu spánku (obr. 5.13), tudíž by nemohl přijímat data.

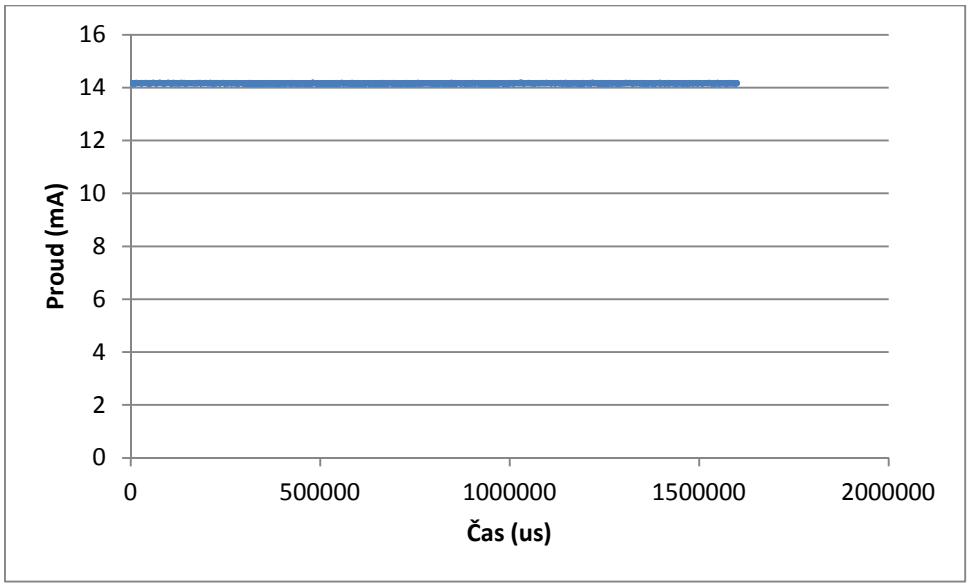


Obr. 5.13: Grafický průběh signálu s vysokou střídou

Proto byla nastavena vhodná střída 1/3. Z hlediska spotřeby energie se jedná o horší řešení, než kdyby byla nastavena velká střída, ale z hlediska potlačení možnosti velké ztrátovosti přijímaných paketů se jedná o lepší řešení, protože jednotky jsou na sebe asynchronní. Větší střídou se minimalizuje ztrátovost příchozích paketů. Na obrázku 5.14 je zobrazen průběh periodického zapínání a vypínání přijmu transceiveru a na obrázku 5.15 je zobrazen průběh bez implementovaného nízkoenergetického časovače, tedy je-li příjem pořád zapnut.



Obr. 5.14: Závislost proudového odběru transceiveru a mikrokontroléru na čase s implementovaným nízkoenergetickým časovačem



Obr. 5.15: Závislost proudového odběru transceiveru a mikrokontroléru na čase bez implementovaného nízkoenergetického časovače

Z grafických průběhů (Obr. 5.14 a Obr. 5.15) je vidět, že rozdíl proudového odběru s implementací nízkoenergetického časovače a bez implementace nízkoenergetického časovače je v desítkách mA. Tudíž bylo vhodné nízkoenergetický časovač implementovat.

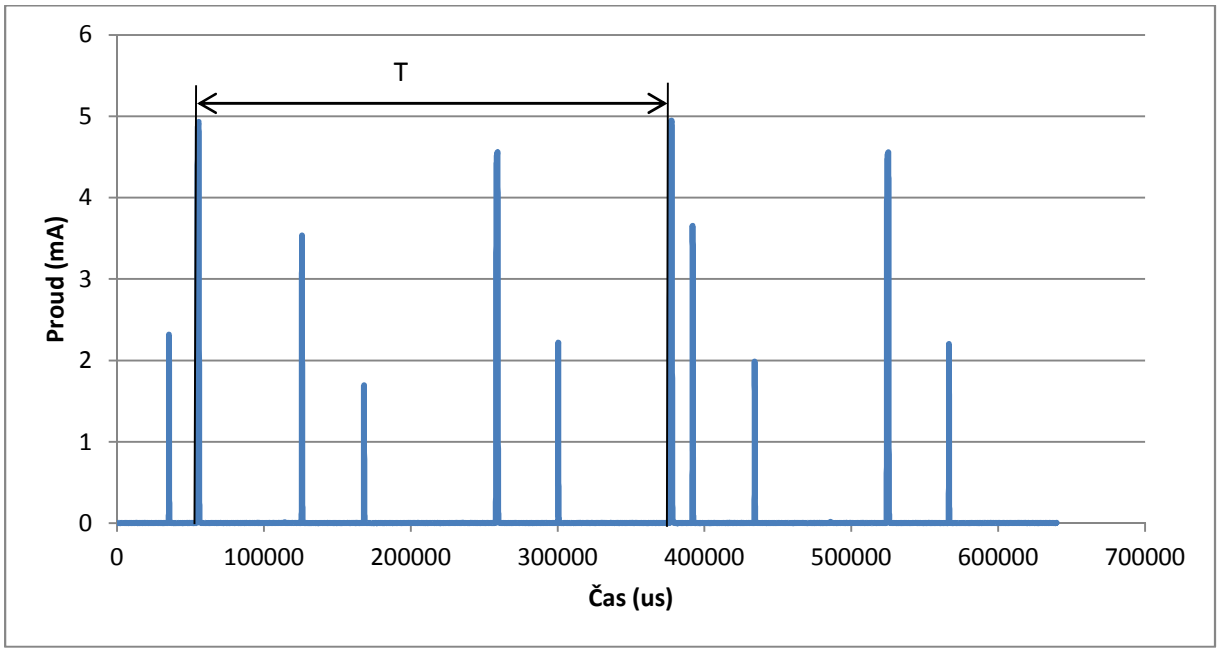
5.4 Portace

Za účelem srovnání vytvořených aplikací v kapitole 5.2.1 až kapitole 5.2.5 s aplikacemi, které nejsou implementovány do FreeRTOS, byly vytvořeny 2 aplikace, které nebyli implementované ve FreeRTOS.

5.4.1 Aplikace PA

Aplikace PA pracuje na stejném principu jako aplikace A (kap. 5.2.1).

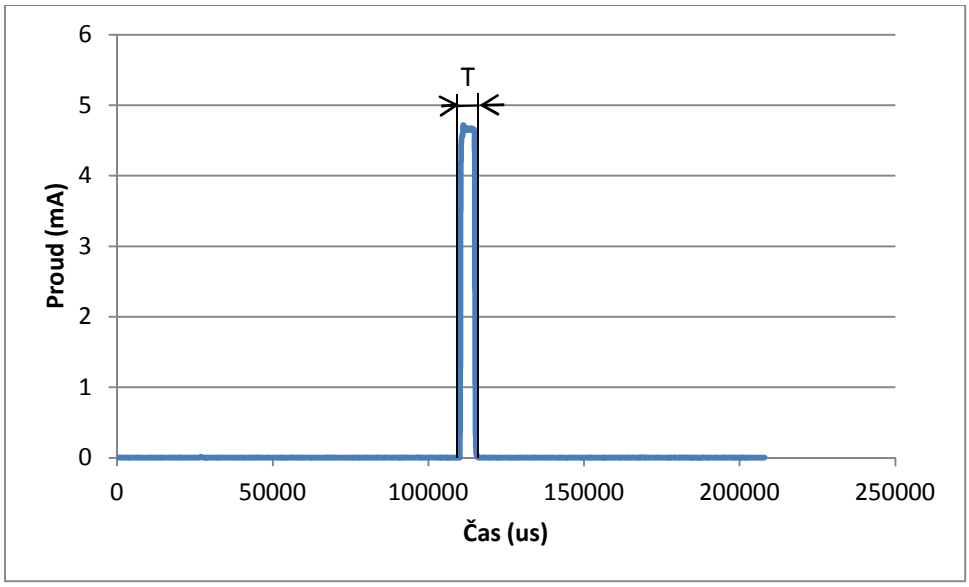
Na obrázku 5.17 je graficky zobrazen průběh příjmu dat, na obrázku 5.16 je zobrazen průběh odesílání dat. Vypočtené hodnoty celkové spotřebované energie při odesílání dat za jednu periodu a změřený počet cyklů jsou uvedeny v tabulce 5.12. Vypočtené hodnoty celkové spotřebované energie a počet cyklů za dobu obsluhy přerušení při příjmu dat jsou uvedeny v tabulce 5.13.



Obr. 5.16: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 48 MHz

Tab. 5.12: Počet cyklů a celková spotřebovaná energie za periodu při odesílání dat

Počet cyklů za dobu jedné periody	4288491
Celková spotřebovaná energie za dobu jedné periody (uJ)	45,25



Obr. 5.17: Závislost proudového odběru při příjmu dat na čase při frekvenci jádra 48 MHz

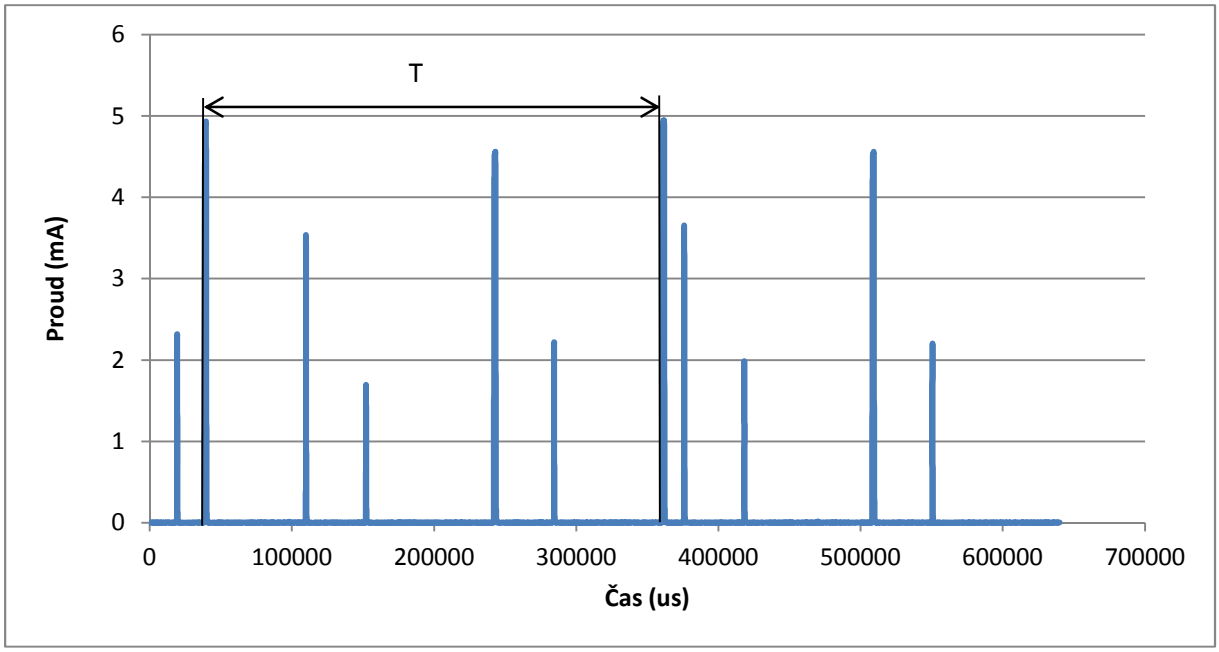
Tab. 5.13: Počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat

Počet cyklů za dobu jedné periody	15486
Celková spotřebovaná energie za dobu jedné periody (uJ)	4,86

5.4.2 Aplikace PB

Aplikace PA pracuje na stejném principu jako aplikace B (kap. 5.2.2). je zde také využito DMA a cyklický buffer.

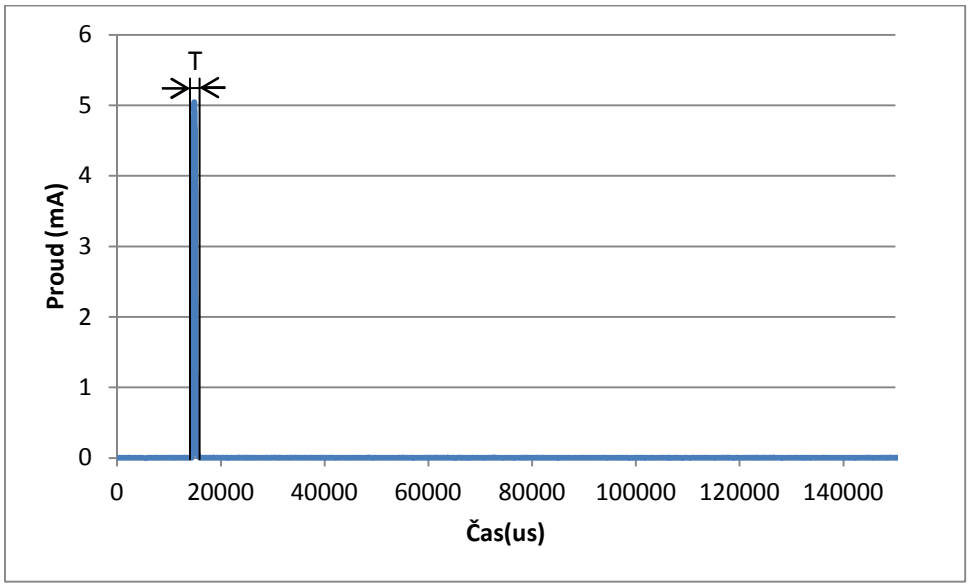
Na obrázku 5.19 je graficky zobrazen průběh příjmu dat, na obrázku 5.18 je zobrazen průběh odesílání dat. Vypočtené hodnoty celkové spotřebované energie při odesílání dat za jednu periodu a změřený počet cyklů jsou uvedeny v tabulce 5.14. Vypočtené hodnoty celkové spotřebované energie při příjmu dat za dobu obsluhy přerušení sou uvedeny v tabulce 15.15.



Obr. 5.18: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 48 MHz

Tab. 5.14: Počet cyklů a celková spotřebovaná energie za periodu při odesílání dat

Počet cyklů za dobu jedné periody	4268497
Celková spotřebovaná energie za dobu jedné periody (uJ)	53,26



Obr. 5.19: Závislost proudového odběru při příjmu dat na čase při frekvenci jádra 48 MHz

Tab. 5.15: Počet cyklů a celková spotřebovaná energie za dobu obsluhy přerušení při příjmu dat

Počet cyklů za dobu jedné periody	6632
Celková spotřebovaná energie za dobu jedné periody (uJ)	2,08

5.5 Analýza vytvořené implementace s portací

Za účelem analýzy implementace mesh protokolu do prostředí FreeRTOS a portace mesh protokolu byla vytvořena aplikace, která periodicky po 0,5 s vzorkuje pomocí světelného senzoru frekvenci světla. Po navzorkování 512 vzorků je vyhledána nejvyšší frekvence vzorku, poté je zobrazena na displej a po několika vzorcích odeslána. Na základě dosažených výsledků u předchozích programů (kap. 5.2.2 kap. 5.2.4) bylo zde pro přenos dat mezi bufferem RAM a bufferem transceiveru použito DMA.

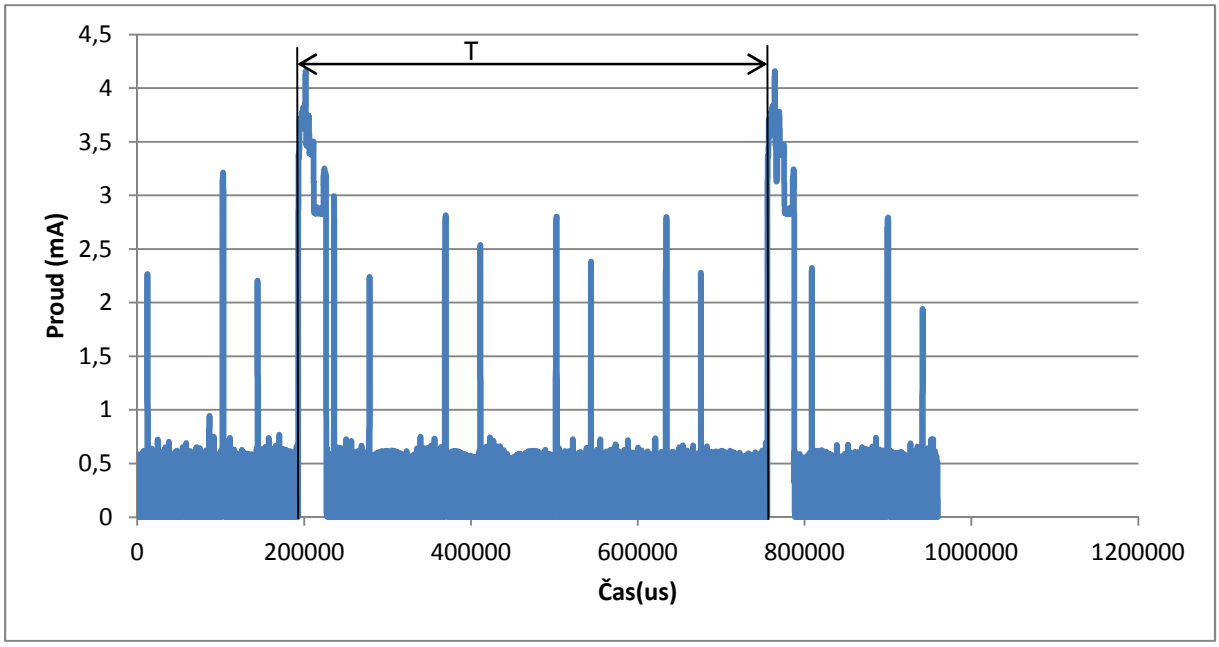
Portace

Light weight mesh je navrhnut jako stavový automat, jehož vlákna běží v sekvenci. U aplikace bylo použito jako zdroj přerušení RTC. Procesor se mezi jednotlivými vzorky nachází v režimu spánku a čeká na přerušení od RTC. Jakmile nastane přerušení, proběhne navzorkování jednoho vzorku, tento krok se děje 512krát. Po navzorkování je zobrazen výsledek na displeji a odeslán. Na obrázku 5.20 je zobrazen průběh vzorkování a poté zobrazení a odeslání dat. V tabulce 5.17 je zobrazen počet cyklů a celková spotřebovaná energie za jednu periodu běhu programu.

Za účelem optimalizace spotřeby mikrokontroléru byla změřena celková energie při různých frekvencích hodin jádra (tab. 5.16) za dobou jedné periody běhu programu. Z naměřených hodnot byla zvolena jako nejoptimálnější frekvence hodin jádra 24 MHz. Na obrázku 5.20 je zobrazen průběh vzorkování a poté zobrazení a odeslání dat. V tabulce 5.18 je zobrazen počet cyklů a celková spotřebovaná energie za jednu periodu běhu programu.

Tab. 5.16: Závislost celkové spotřebované energie na frekvenci hodin jádra

Frekvence hodin jádra [MHz]	Celková spotřebovaná energie za jednu periodu (uJ)
48	653,20
24	632,20
12	773,60
6	1111
3	1800
1,5	3310



Obr. 5.20: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 24 MHz

Tab. 5.17: Počet cyklů a celková spotřebovaná energie za jednu periodu

Počet cyklů za dobu jedné periody	3796184
Celková spotřebovaná energie za dobu jedné periody (uJ)	632,20

Implementace

V implementaci mesh protokolu do FreeRTOS nebylo nutné nastavovat RTC, protože RTC je nakonfigurováno v souboru *low_power_tick_management.c*. RTC slouží jako zdroj hodinového signálu nachází-li se mikrokontrolér v nízkoenergetickém režimu 2. Podobně jako v portaci se mikrokontrolér mezi jednotlivými vzorky nachází v nízkoenergetickém režimu a čeká na přerušení RTC. Frekvence RTC byla nastavena v hlavičkovém konfiguračním souboru FreeRTOS v souboru FreeRTOSConfig.h. Nastavením parametru configTICK_RATE_HZ byla nastavená požadovaná frekvence 500 Hz. Jakmile nastane přerušení od RTC, tak proběhne navzorkování jednoho vzorku. Po 512 vzorcích je nalezena nejvyšší frekvence vzorku a poté zobrazena na displej a odeslána.

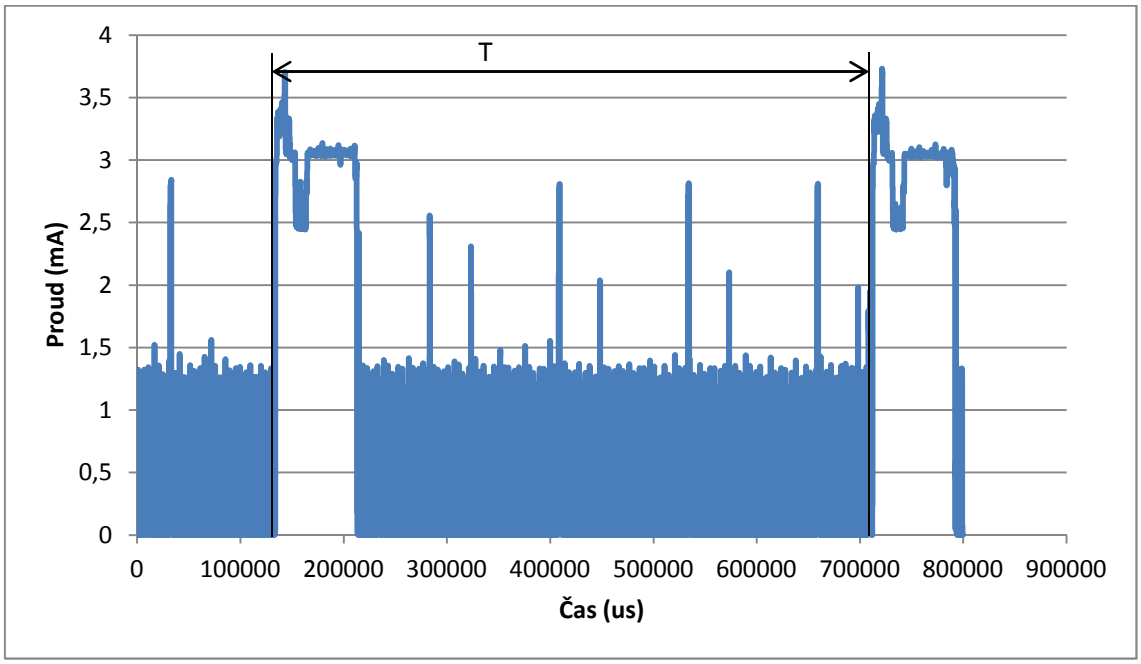
Na obrázku 5.21 je zobrazen průběh vzorkování a poté zobrazení a odeslání dat. V tabulce 5.19 je zobrazen počet cyklů a celková spotřebovaná energie za jednu periodu běhu programu.

Stejně jako u portace byla změřena celková energie při různých frekvencích hodin jádra (tab. 5.18) za dobou jedné periody běhu programu. Z naměřených hodnot byla zvolena jako nejoptimálnější frekvence hodin jádra 24 MHz.

Tab. 5.18: Závislost celkové spotřebované energie na frekvenci hodin jádra

Frekvence hodin jádra [MHz]	Celková spotřebovaná energie za jednu periodu (uJ)
48	1 290
24	1 140
12	1 770
6	3 340
3	4 050
1,5	5 040

Na obrázku 5.21 je zobrazen průběh vzorkování a poté zobrazení a odeslání dat.



Obr. 5.21: Závislost proudového odběru při odesílání dat na čase při frekvenci jádra 24 MHz

Tab. 5.19: Počet cyklů a celková spotřebovaná energie za jednu periodu

Počet cyklů za dobu jedné periody	3992154
Celková spotřebovaná energie za dobu jedné periody (uJ)	1 140

6 Diskuse

Kapitola se zabývá zhodnocením naměřených a vypočítaných výsledků v předchozí kapitole 5.

V tabulce 6.1 je zobrazeno srovnání celkové spotřebované energie mikrokontrolérem a počtů cyklů, jak při příjmu dat, tak pro jejich odesílaní. Tabulka 6.1 zobrazuje aplikace PA a aplikace PB, které jsou v kapitolách 5.4.1 a 5.4.2.

Tab. 6.1: Srovnání aplikace PA a aplikace PB

	Odesílání dat		Příjem dat	
	Počet cyklů	Spotřebovaná energie (uJ)	Počet cyklů	Spotřebovaná energie (uJ)
Aplikace PA	4288491	45,25	15486	4,86
Aplikace PB	4268497	53,26	6632	2,08

Z naměřených a vypočtených hodnot lze říct, že aplikace PB má lepí spotřebu energie a menší počet cyklů. Lepšího výsledku je dosaženo z toho důvodu, že u aplikace PB je použito DMA, jak pro příjem tak odeslání. A pro příjem dat je zde použití cyklický buffer.

V tabulce 6.2 je zobrazeno srovnání celkové spotřebované energie mikrokontrolérem a počtů cyklů, jak při příjmu dat, tak pro jejich odesílaní. Tabulka 6.2 zobrazuje aplikace A až finální aplikace, které jsou ve výše uvedených kapitolách 5.2.1 až 5.2.5.

Tab. 6.2: Srovnání aplikací A až finální aplikace

	Odesílání dat		Příjem dat	
	Počet cyklů	Spotřebovaná energie (uJ)	Počet cyklů	Spotřebovaná energie (uJ)
Aplikace A	5064347	381	13347	4,2
Aplikace B	4855128	360	6720	2,11
Aplikace C	5564846	400	13347	4,2
Aplikace D	5435185	383	6115	1,92
Finální aplikace	4673716	139	6115	1,92

Z naměřených a vypočítaných výsledků, které jsou zobrazeny v tabulce 6.2 lze říct, že nejlepší aplikace z hlediska spotřeby a počtu cyklů je finální aplikace. Spotřebovaná energie mikrokontroléru při odesílání dat je třetinová oproti aplikacím A až D. Při přijmu dat je u finální aplikace dosaženo stejného výsledku jako u aplikace B a aplikace D, protože je použito použito DMA a cyklický buffer, jako u aplikace B a aplikace D.

V tabulce 6.3 je zobrazeno srovnání celkové spotřebované energie mikrokontrolérem a počtů cyklů, jak při příjmu dat, tak pro jejich odesílání pro aplikace v kapitole 5.5, tedy portace a implementace, po nalezení optimální frekvence 24 MHz.

Tab. 6.3: Srovnání portace a implementace

	Odesílání dat	
	Počet cyklů	Spotřebovaná energie (uJ)
Portace	3796184	632,20
Implementace	3992154	1140

Z naměřených a vypočítaných hodnot, které jsou zobrazeny v tabulce 6.3 lze říct, že implementace má z hlediska spotřeby vyšší hodnotu spotřebované energie a také počet cyklů. Spotřeba energie je vyšší z toho důvodu, že samotná režie FreeRTOS využije několik počtů cyklů a také se zvýší spotřebovaná energie. Operační systém nabízí určité funkce a také zajistí rychlejší časovou odezvu, ale na úkor celkové spotřeby energie.

7 Závěr

Úkolem této bakalářské práce bylo získat patřičné teoretické základy v oblasti mesh sítí a operačních systémů reálného času a tyto teoretické základy využít pro vytvoření implementace mesh protokolu do operačního systému reálného času, konkrétně do distribuce FreeRTOS. Pro realizaci implementace byla použita modifikovaná distribuce FreeRTOS. Popis této distribuce je věnována kapitola 5. V kapitole 5.1 je popis nastavení komunikační vrstvy mezi transceiverem a mikrokontrolérem, tedy správné nastavení portů, periferií, přerušení a nastavení SPI sběrnice a výběr optimální rychlosti přenosu SPI sběrnice, jehož výběr byl ověřen měřením. Na základě nalezení nejlepší implementace bylo vytvořeno několik nezávislých aplikací (kap. 5.2.1 až kap. 5.2.5). Z naměřených hodnot proudu byly vytvořeny grafické průběhy a do tabulek zaneseny změřené počty cyklů a také celková spotřebovaná energie. V kapitole 5.2.5 je popsána finální aplikace, jejíž měřené parametry byly z vytvořených implementací nejlepší. Cíl byl tedy splněn, byla nalezena nejlepší varianta implementace mesh protokolu do FreeRTOS. V kapitole 6 je v tabulkách uvedeno přehledné srovnání vytvořených programů.

8 Seznam literatury

[1] Počítačové sítě - základní topologie. *Samuraj-cz* [online]. 07.04.2009 [cit. 2013-12-29]. Dostupné z: http://www.samuraj-cz.com/clanek/pocitacove-site-zakladni-topologie/

[2] South African wireless community networks. *Wikipedia: the free encyclopedia.* [online]. 2001- [cit. 2013-12-29]. Dostupné z: http://en.wikipedia.org/wiki/South_African_wireless_community_networks

[3] IEEE 802.15™: WIRELESS PERSONAL AREA NETWORKS (PANs). *IEEE Standards association.* [online]. 29.12.2013 [cit. 2013-12-29]. Dostupné z:*http://standards.ieee.org/about/get/802/802.15.html*

[4] Přispěvatelé Wikipedie, IEEE 802.15.4 [online], Wikipedie: Otevřená encyklopedie, c2014, Datum poslední revize 27. 03. 2014, 12:29 UTC, [citováno 4. 06. 2014] <*http://en.wikipedia.org/wiki/802.15.4*>

[5] Atmel Lightweight Mesh. ATMEL. *Atmel* [online]. 2013 [cit. 2013-12-30]. Dostupné z: http://www.atmel.com/tools/lightweight_mesh.aspx

[6] *Atmel AVR2130: Lightweight Mesh Developer Guide*. 2013. Dostupné z: http://www.atmel.com/Images/Atmel-42028-Lightweight-Mesh-Developer-Guide_Application-Note_AVR2130.pdf

[7] RTOS Program Models Used in Embedded Systems [online]. Budapest, Hungary, 2011 [cit. 2014-01-30]. Dostupné z: http://www.uni-obuda.hu/e-bulletin/Kopjak_Kovacs_2.pdf. Scientific article. Óbuda University.

[8] Software architectures: Software for Embedded Systems [online]. Portugal, 2011 [cit. 2014-01-30]. Dostupné z: https://dspace.ist.utl.pt/bitstream/2295/910339/1/ase-09-h.pdf. Scientific article. Instituto Superior Técnico.

[9] SIMON, David E. An embedded software primer. Boston: Addison-Wesley, 1999, xix, 424 s. ISBN 02-016-1569-X.

[10] BARRY, Richard. *Using the FreeRTOS Real Time Kernel*. 1. vyd. London: Real Time Engineers Ltd., 2010. ISBN 978-1-4461-7030-4.

[11] CLARKE, Peter. Android, FreeRTOS top EE Times' 2013 embedded survey. In: *EE Times* [online]. 2013 [cit. 2013-12-30]. Dostupné z: http://www.eetimes.com/document.asp?doc_id=1263083

[12] VALVANO, Jonathan W. *Embedded Systems: Real-Time Operating Systems for Arm Cortex M Microcontrollers*. Charleston: CreateSpace, 2012, xi, 366 s. ISBN 978-1466468863.

[13] VALVANO, Jonathan W. *Embedded Systems: Real-Time Interfacing to Arm® Cortex(TM)-M Microcontrollers.* Charleston: CreateSpace, 2011, xiv, 570 s. ISBN 978-1463590154.

[14] *RF231 datasheet* [online]. 2009 [cit. 2.1.2014]. Dostupné z: http://www.atmel.com/images/doc8111.pdf

9 Abecední přehled použitých zkratek

AES	(Advanced Encryption Standard), standard pokročilého šifrování
AODV	Ad hoc On-demand Distance Vector
ARM	(Advanced RISC Machine), architektura procesorů
AVR	architektura procesorů
BURTC	(backup real time clock), záložní hodiny reálného času
DMA	(Direct Memory Access), přímý přístup do paměti
HAL	(Hardware abstraction layer), abstraktní vrstva ovládající hardware
IEEE	(Institute of Electrical and Electronics Engineers), „Institut pro elektrotechnické a elektronické inženýrství
IrDA	(Infrared Data Association), komunikaci pomocí infračerveného světla
ISR	(Interrupt Service Routine), obsluha přerušení
MB	(Megabyte), datová jednotka
OSI	(Open Systems Interconnection), Referenční model
PAN	(Personal Area Network), osobní síť
RAM	(random-access memory), paměť s přímým přístupem
RTC	(real time clock), hodiny reálného času
RTOS	(real-time operating systém), operační systém reálného času
USART	(Universal Synchronous / Asynchronous Receiver and Transmitter), Synchronní / asynchronní sériové rozhraní
WPAN	(Wireless Personal Area Network), osobní bezdrátová síť

10 Seznam příloh

A Obsah přiloženého média

B Příručka pro použití vytvořených aplikací

A Obsah přiloženého média

Přiložené médium obsahuje bakalářskou práci v elektronické podobě. Médium také obsahuje vyexportované .csv soubory s jednotlivými měřeními pro aplikace, soubory je možno otevřít v programu excel pro připadnou kontrolu grafů.

B Příručka pro použití vytvořených aplikací

Aplikace byly vytvořeny ve vývojovém prostředí IAR.

Pro spuštění aplikace je vývojovém prostředí je nutné nainstalovat vývojové prostředí IAR Embedded Workbench for ARM.

Po připojení vývojového kitu k PC a instalaci IAR Embedded Workbench for ARM je pro spuštění aplikace nutné v kořenovém adresáři aplikace spustit soubor *LWM.eww*. Poté se otevře zvolená aplikace ve vývojovém prostředí.

V dalším kroku v záložce *Project* zvolíme *Download*, a poté *Download active application*.