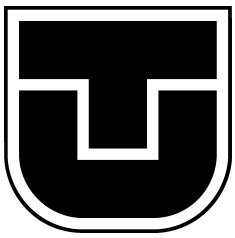
Program pre ADuC RF101, časový multiplex v2



Peter Šoltýs

Katedra elektroniky a multimediálnych telekomunikácií

Technická univerzita v Košiciach

Študijný odbor: Elektronika

Študijný program: Smartelektronika

Stupeň štúdia: druhý

Ročník: prvý

Akademický rok: 2015/2016

Obsah

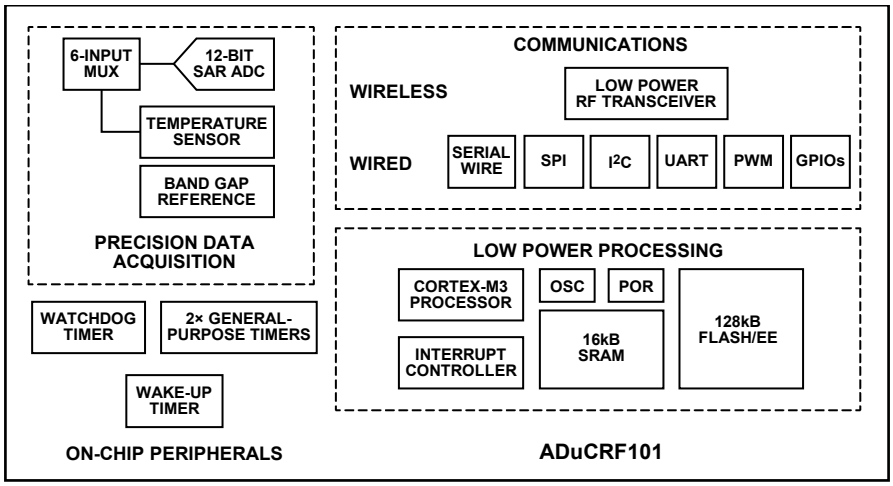
Zoznam obrázkov

1 Zadanie

Vytvorte pomocou komunikačného procesora **ADuC RF101** komunikačnú sieť s centrálnym zberom dát, z jednotlivých uzlov siete. Pre komunikáciu použite voľné komunikačné pásmo 433,92 MHz a dáta posielajte na centrálny uzol pomocou časového multiplexu. Dosiahnite pri tom minimálnu požadovanú priepustnosť 6 kbit/s pre každý uzol siete.

2 Procesor ADuC RF101

Tento procesor firmy Analog Devices [2] založený na jadre Cortex-M3 od firmy ARM. je vybavený veľkým množstvom periférnych obvodov čo vidno na obrázku 2.1. Taktiež obsahuje 12 bitový A/Č prevodník, čo ho predurčuje najmä k nasadeniu v senzorovej technike. Značne výhodnou vlastnosťou procesora je integrovaný bootloader, čo znamená že je ho možné naprogramovať aj pomocou sériového rozhrania. No hlavnou výhodou tohto procesora je v puzdre integrovaný rádio-frekvenčný komunikačný modul, spojený s procesorom pomocou internej komunikačnej zbernice SPI. Integrované jadro procesoru dodáva vysoký výkon čo je výhodné pre rýchle aplikácie. Jeho integrovaný RF modul má maximálny výstupný výkon vysielača +10 dBm pri maximálnej prenosovej rýchlosti 300 kbit/s. Podporuje, Menchester kódovanie, premenlivú dĺžku paketov až do 240 bajtov, pri použití FSK alebo GFSK modulácie. Jeho bloková schéma je na obrázku 2.1.



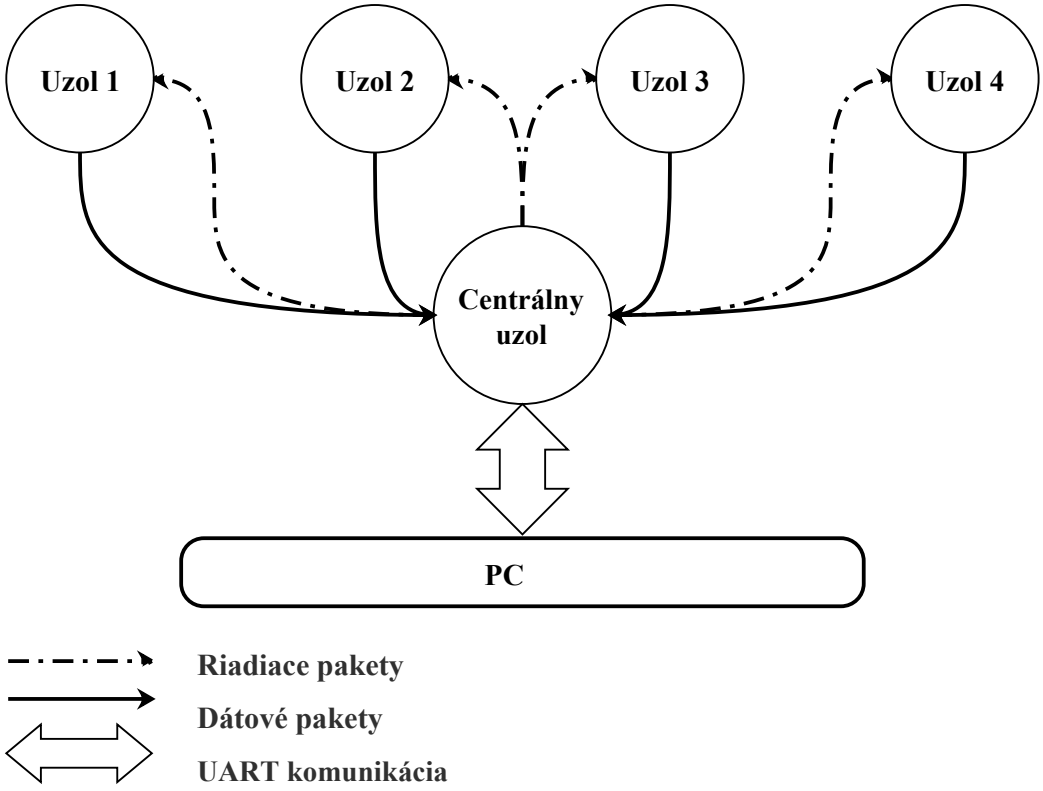
Obr. 2.1: Bloková schéma ADucRF101 [2]

3 Návrh riešenia

Pri navrhovaní riešenia bolo potrebné dodržať základné požadované parametre.

- minimálna priepustnosť 6 kbit/s na uzol
- odhadovaný minimálny počet ulov 4
- zaručiť čo najvyššiu spoľahlivosť prenosu
- použiť časový multiplex, kvôli šetreniu šírky frekvenčného pásma

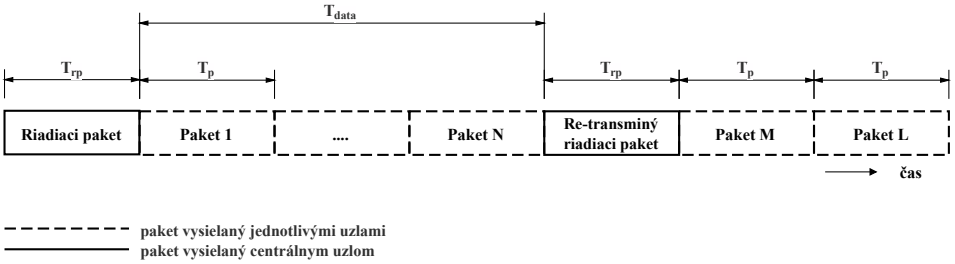
V princípe by celá komunikácia mala mať tvar zobrazený na obrázku 3.1.



Obr. 3.1: Základná schéma komunikácie

Pre dodržanie požadovaných parametrov bolo potrebné navrhnúť schému fungovania prenosu. Pre správny návrh riešenia je nutné poznať vlastnosti RF komunikačného modulu opísané v kapitole 2. Pre zabezpečenie postačujúcej prenosovej rýchlosti sa ponúka iba najvyššia prednastavená prenosová rýchlosť 300 kbit/s čo je značne viac než je potrebné. Pre zabezpečenie spoľahlivosti prenosu sa využíva hardvérové CRC (Cyclic Redundancy Check) ktoré používa prednastavený

polynóm $x^{16}+x^{12}+x^5+1$ [1] miesto samo-opravných kódov, čo zjednodušuje spracovanie a návrh. No v praxi použitie CRC znamená že paket s nesprávnym CRC je jednoducho zahodený, teda stratu dát. Preto je nutné ich znovu preposlať, čo si môžme dovoliť vďaka dostatočne veľkej prenosovej rýchlosti. Z tohto dôvodu bol časový multiplex navrhnutý následovne 3.2, kde riadiace pakety vysiela centrálny uzol s požiadavkou aby požadovaný uzol s priradeným **ID** (identifikátorom) vysielal nahromadené dáta, bez čakania na potvrdenie prijatia. Ak centrálny uzol zistí že nedošli všetky nahromadené dáta pošle riadiaci paket na retransmisiu.



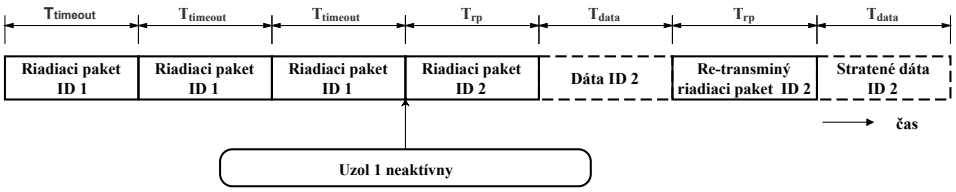
Obr. 3.2: Časová schéma multiplexu

Pred začatím vysielania má uzol v pamäti nazhromaždené pakety, maximálne v tomto prípade 20, z dôvodu obmedzenia RAM pamäte ktorá má len 16 kbajt, teda $2 * 20 * 240 = 9600$ bajt (pozri obrázky 3.4 a 4.2). Uzol vysiela pakety s hlavičkou ktorá obsahuje počet prijatých paketov ako aj poradové číslo paketu, jeho formát je na obrázku 3.4, týmto spôsobom poskytuje centrálnemu uzlu informáciu o počte paketov a číslo paketu ktorý došiel, teda dokáže určiť ktoré pakety chýbajú. Vo všeobecnosti všetky časy zobrazené na obrázku 3.2 okrem času T_{rp} keďže pakety môžu nadobúdať rôznu dĺžku.

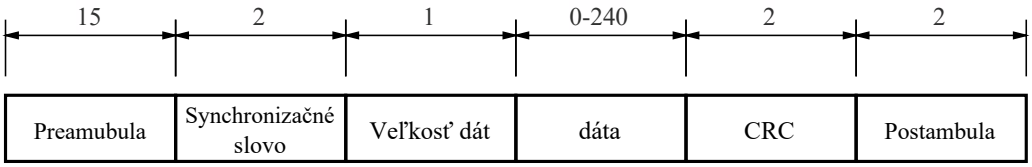
Pretože vždy existuje možnosť že sa budú strácať aj riadiace pakety, centrálny uzol tieto správy viacnásobne posiela, kým požiadaný uzol nepošle dáta, podľa schémy na obrázku 3.3. Ak uzol nereaguje ani po viac násobnej (v tomto prípade tretej) výzve s oneskorením $T_{timeout} = 5ms$ centrálny uzol bude pokračovať vo vysielaní riadiacich paketov pre ostatné uzly.

Vo všeobecnosti každý posielaný paket má tvar 3.4 a v jednotlivých paketoch je rozdielna iba dátová (ďalej označovaná ako paket) oblasť o dĺžke 240 bajtov ale pri dátovom pakete kvôli pridávanej hlavičke ostáva pre dáta 237 bajtov ako je vidno na obrázku 3.5. Ako nulový paket ktorý posiela uzol ak nemá nahromadené žiadne dáta sa používa dátový paket s nulovým označením paketu ako aj celkovým počtom paketov.

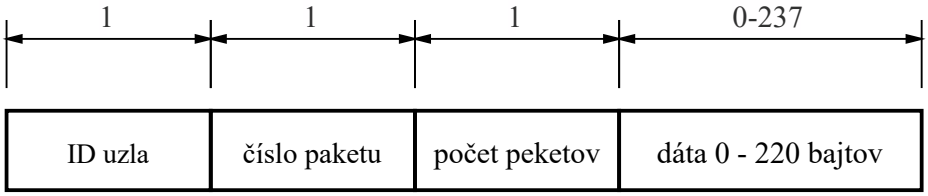
Identifikačný paket vysielaný centrálnym uzlom, ktorý signalizuje jednotlivým uzlom priradenie slotu sa používa paket v tvare zobrazenom na obrázku 3.6.



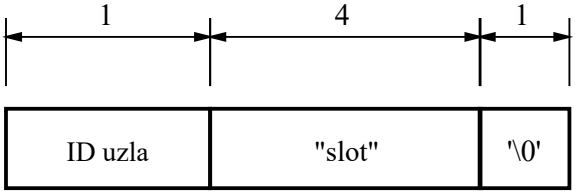
Obr. 3.3: Časová schéma viacnásobného posielania



Obr. 3.4: Všeobecný formát paketu

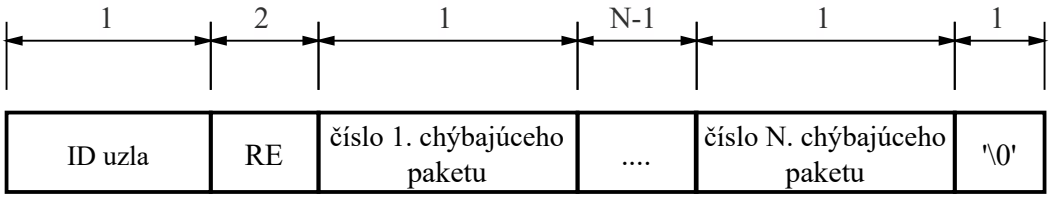


Obr. 3.5: Formát dátového paketu



Obr. 3.6: Formát identifikačného paketu

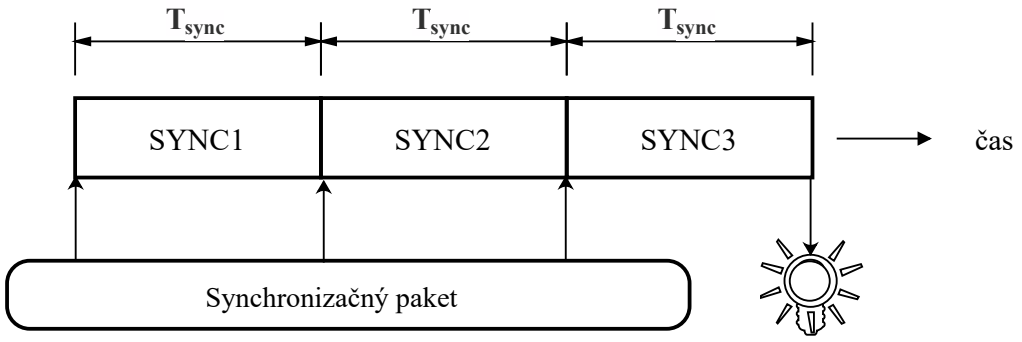
Re-transmisný paket, ktorým centrálny uzol žiada jednotlivé uzly o opätovné zaslanie stratených dát má formát zobrazený na obrázku 3.7.



Obr. 3.7: Formát re-transmisného paketu

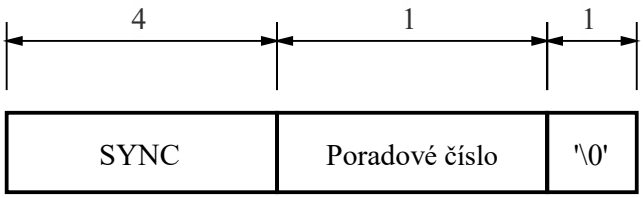
3.1 Synchronizácia

Aby bola dosiahnutá čo najväčšia spoľahlivosť synchronizácie, a čo najmenší časový rozptyl, bola navrhnutá ako zobrazuje obrázok 3.8, teda s viacnásobným vysielaním synchronizačného paketu, v konštantných časových rozostupoch $T_{sync} = 1.2ms$. Vďaka tejto schéme potenciálne postačuje ak každý uzol stratí dva z troch paketov. Teda príme aspoň jediný synchronizačný paket, ktorého poradové číslo je zároveň informácia, koľko času ostáva do synchronizácie ($T_{timeout} * N$, kde N =poradové číslo), z čoho je schopný presne nastaviť čas spustenia. Synchronizácia sa začína až po ukončení posielania dát.



Obr. 3.8: Schéma synchronizácie

Pri synchronizácií centrálny uzol používa paket v tvare zobrazenom na obrázku 3.9.



Obr. 3.9: Formát synchronizačného paketu

3.2 Sériová komunikácia

Na prenos dát medzi PC a procesorom je použité jednoduché rozhranie UART. Na zabezpečenie spoľahlivosti prenosu dát je použitý model komunikácie len pomocou vybranej množiny znakov. Sú to znaky vyjadrujúce hexadecimálnu sústavu čísel, teda ASCII znaky **1-9 A-Z**. Týmto spôsobom je zabezpečené že akýkoľvek prichádzajúci/odchádzajúci znak ktorý nepatrí do tejto množiny znamená posielanie/prijímanie riadiacich správ.

3.3 Kontrola integrity firmvéru

Pre zaistenie integrity a správnej funkčnosti softvéru, je kontrolovaná integrita pomocou softvérového CRC (Cyclic Redundancy Check), ktorý postupne načíta po blokoch 512 bajtov celú programovú pamäť, kde na koniec tejto pamäti je externým nástrojom **Srecord** zapísaný výsledok CRC, ktorý zabezpečí výsledok kontroly =0 .

4 Programová realizácia

Pri vytváraní programu pre **ADuC RF101** bola použitá oficiálna knižnica pre komunikačný modul, čo zjednodušovalo prácu s interným RF modulom ovládaným vnútornou SPI zbernicou. Počas vytvárania programu pokiaľ to bolo vhodné boli v prerušeniach používané, takzvané **flagy**, indikátory, ktoré v sebe nesú informáciu o vykonaní prerušenia. Takáto taktika používania indikátorov je hlavne s pohľadu stability programu značne výhodnejšia, ako vykonanie rutín priamo v prerušení. Pretože častým dôsledkom takéhoto princípu bolo vykonanie rutiny v nevhodný okamih.

Program je rozdielny pre centrálny uzol aj pre samostatné uzly. Oba programy možno konfigurovať v spoločnom hlavičkovom súbore **settings.h** kde sa dajú nastaviť všetky potrebné parametre, ako aj množstvo uzlov. V programových súboroch sú uzly označované ako **Slave** a centrálny uzol ako **Master**.

V oboch programoch bola snaha používať čo najviac spoločných bázových funkcií ako napríklad na inicializáciu periférií, rádiového modulu, funkcie na prijatie jedného paketu zobrazenej na 4.1, vyslanie jedného paketu a pod. Pričom nastavenia všetkých perifémych obvodov je možné zmeniť v spoločnom hlavičkovom súbore **settings.h**

4.1 Program pre centrálny uzol

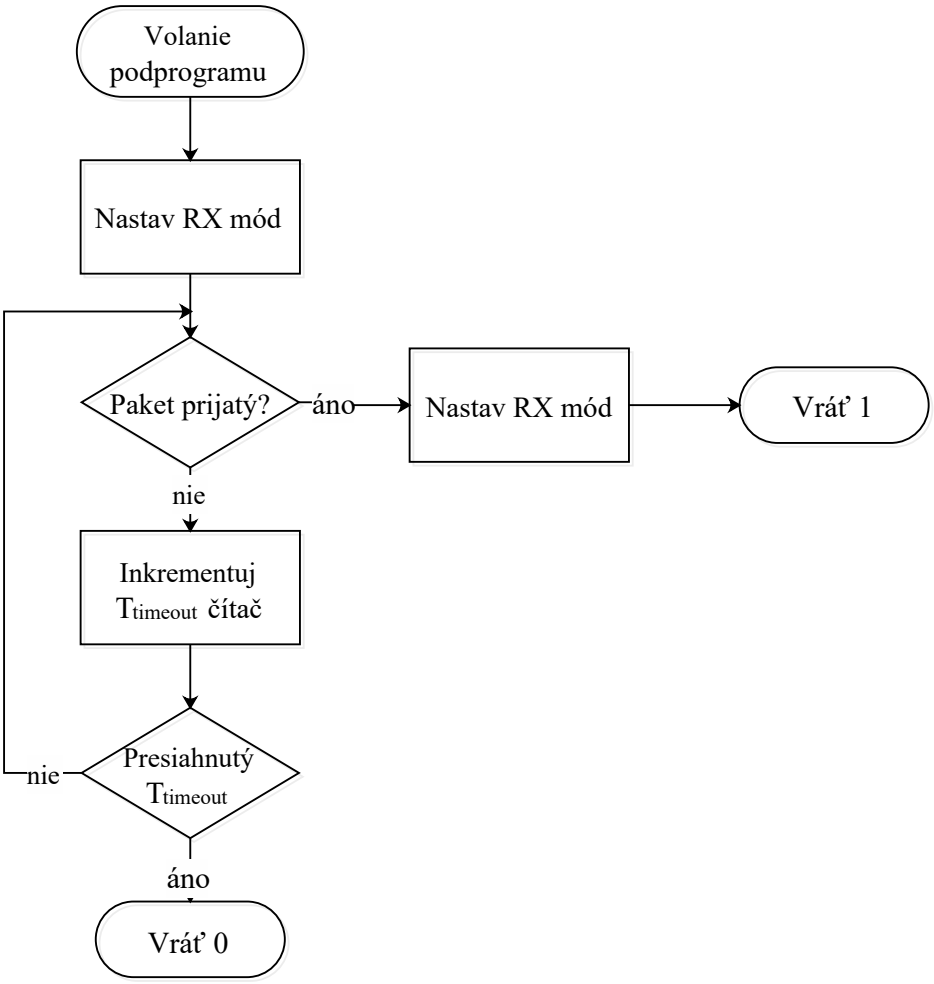
Pakety v centrálnom uzle sú zhromažďované až pokým neskončí retransmisia stratených paketov. Tieto dáta je nutné ukladať do pamäte, ktorá je organizovaná podľa obrázku 4.2 ako kruhová pamäť, kde do jednej časti sú prijaté dáta zapisované a z druhej sú vysielané cez UART do hosťovského zariadenia za pomoci DMA kanálu, čo nezaťažuje procesor.

Hlavný program zobrazený na obrázku 4.3 je principiálne jednoduchý. Niektoré zložitejšie funkcie sú taktiež zobrazené na nasledujúcich obrázkoch.

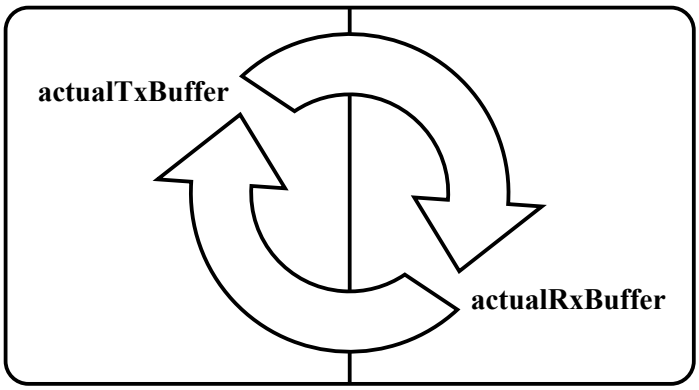
Dôležitou funkciou pri fungovaní programu je funkcia **receivePackets** zobrazená na obrázku 4.4 zodpovedá za prijatie ohodnotenie a uloženie paketu do pamäte 4.2.

Pre dosiahnutie spoľahlivosti prenosu je implementovaná funkcia **ifMissPktGet** zobrazená na obrázku 4.5 zodpovedná za požiadanie a prijatie stratených paketov, paketov ktoré boli prijaté s nesprávnym CRC.

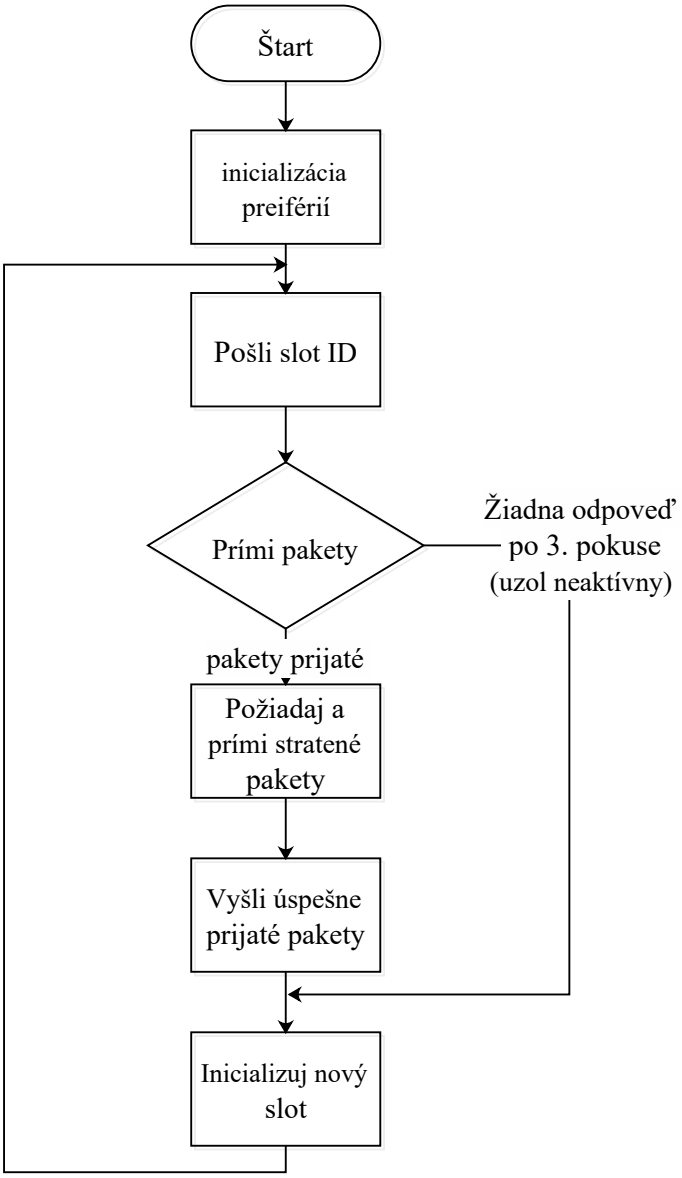
Pre pochopenie fungovania celého programu je potrebné taktiež poznať čo sa vykonáva v procesore počas prerušení, čo je opísané na obrázku 4.6.



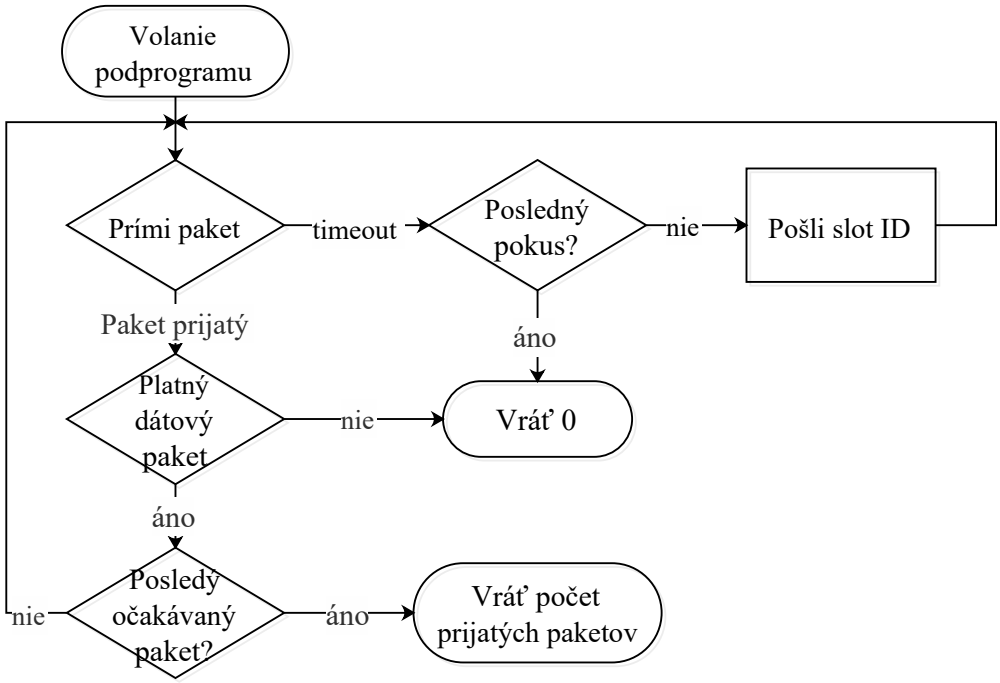
Obr. 4.1: Blokový diagram prijímania jedného paketu



Obr. 4.2: Organizácia pamäte



Obr. 4.3: Blokový diagram hlavného programu centrálneho uzla



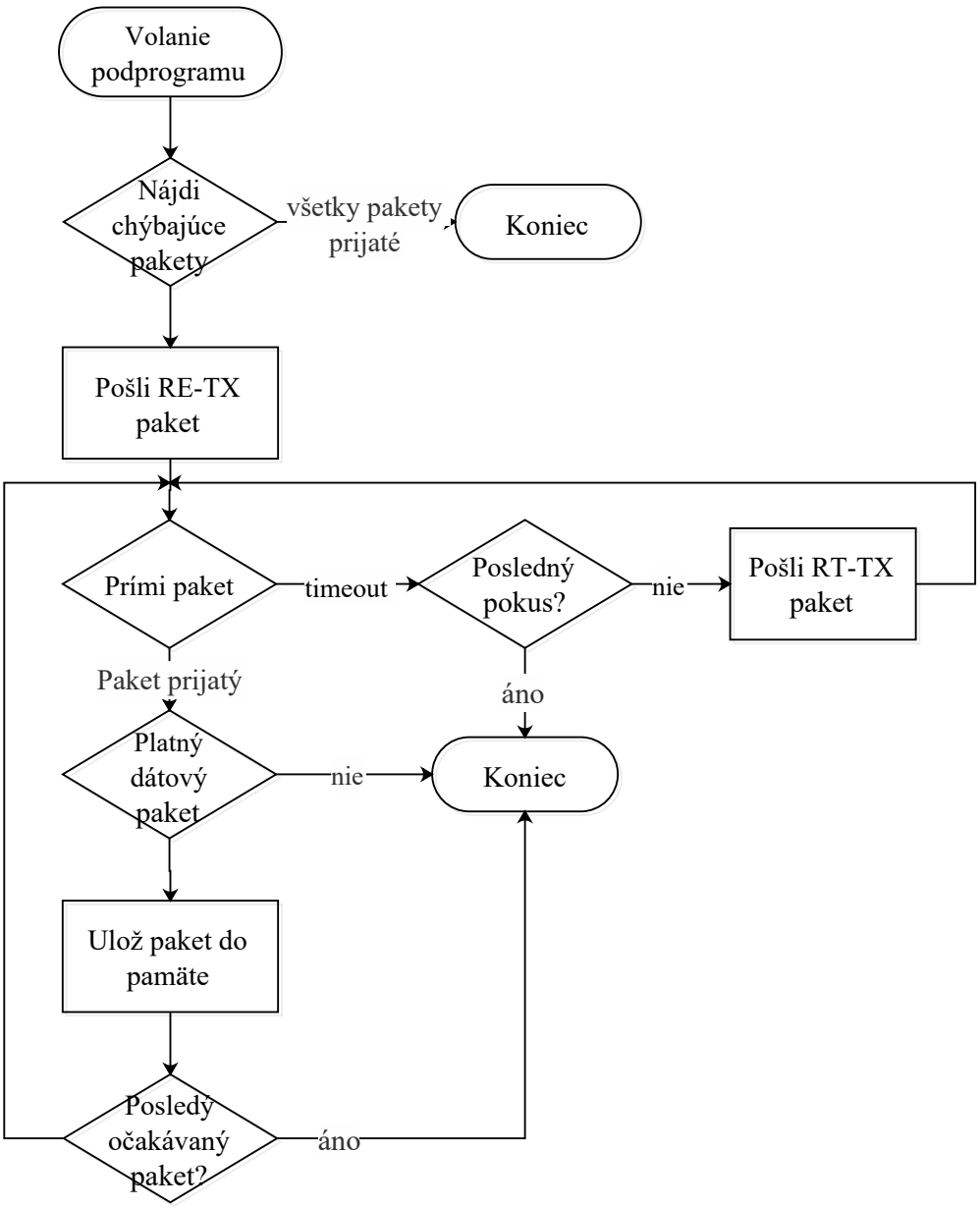
Obr. 4.4: Blokový diagram prijímania paketov v slote

4.2 Program pre jednotlivé uzly

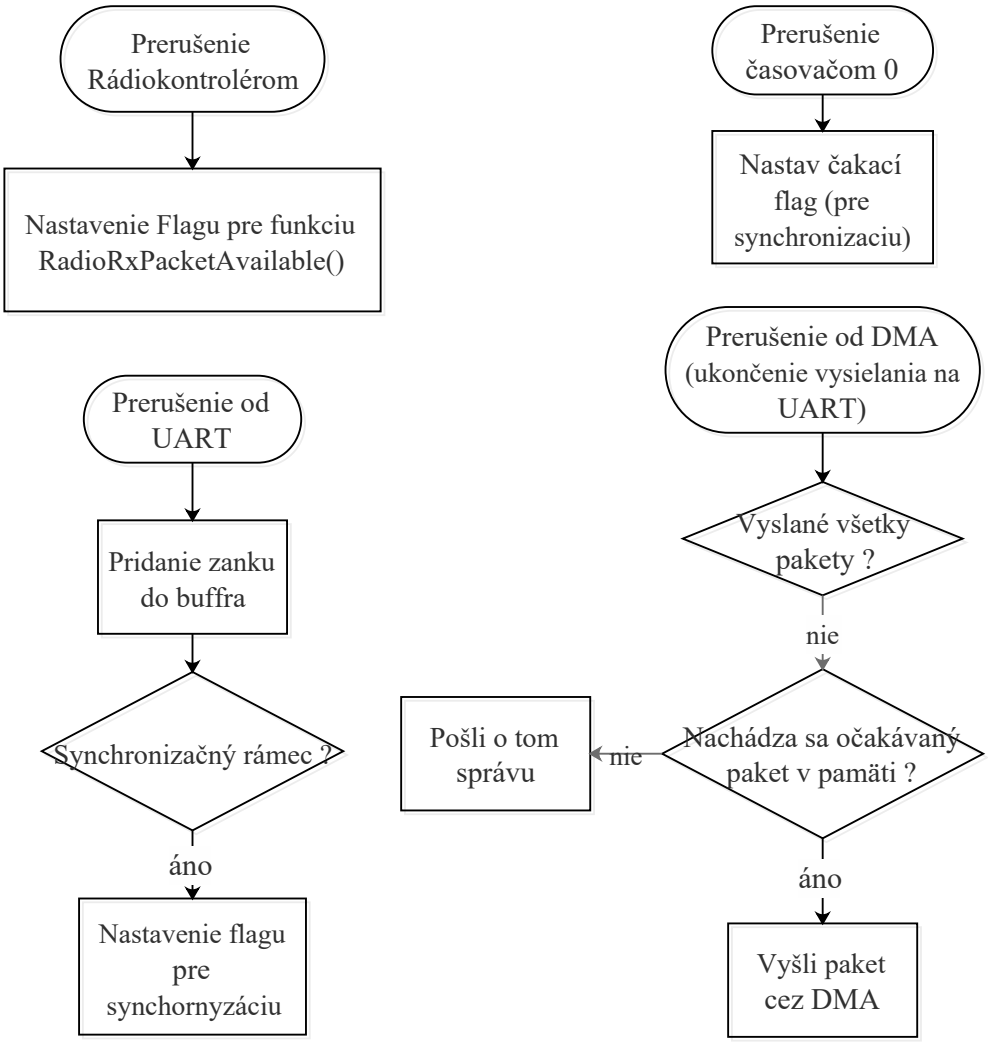
Program pre jednotlivé uzly zhromažďuje pakety v pamäti až pokým príme riadiaci paket s príslušným identifikátorom ktorý požaduje o poslanie nahromadených dát. Preto má rovnakú kruhovú organizáciu pamäte ako centrálny uzol rozdelenú na dve časti ako zobrazuje obrázok 4.2. Takáto organizácia umožňuje uzlu zhromažďovať pakety a zároveň pristupovať k už poslaným paketom, o ktoré môže centrálny uzol požiadať.

V hlavnom programe pre uzly nezohráva žiadny z podprogramov výraznú úlohu alebo je ľahko pochopiteľný zo zdrojového kódu, preto nebude žiadny z nich bližšie opísaný.

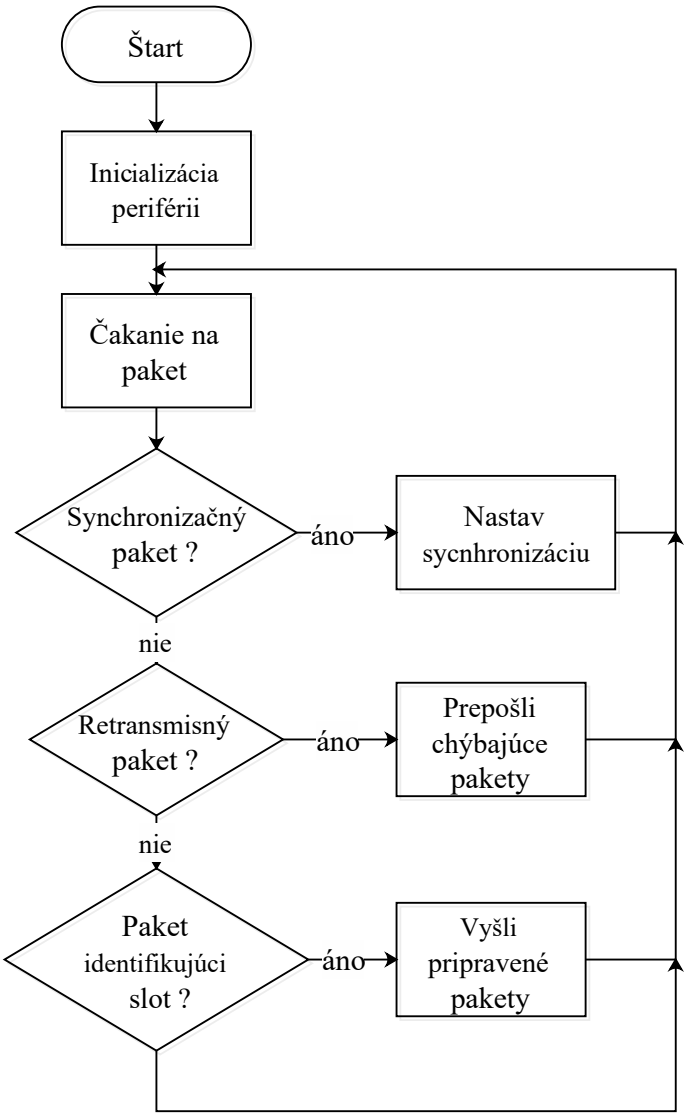
Pre pochopenie fungovania celého programu je potrebné taktiež poznať čo sa vykonáva v procesore počas prerušení, čo je opísané na obrázku 4.8.



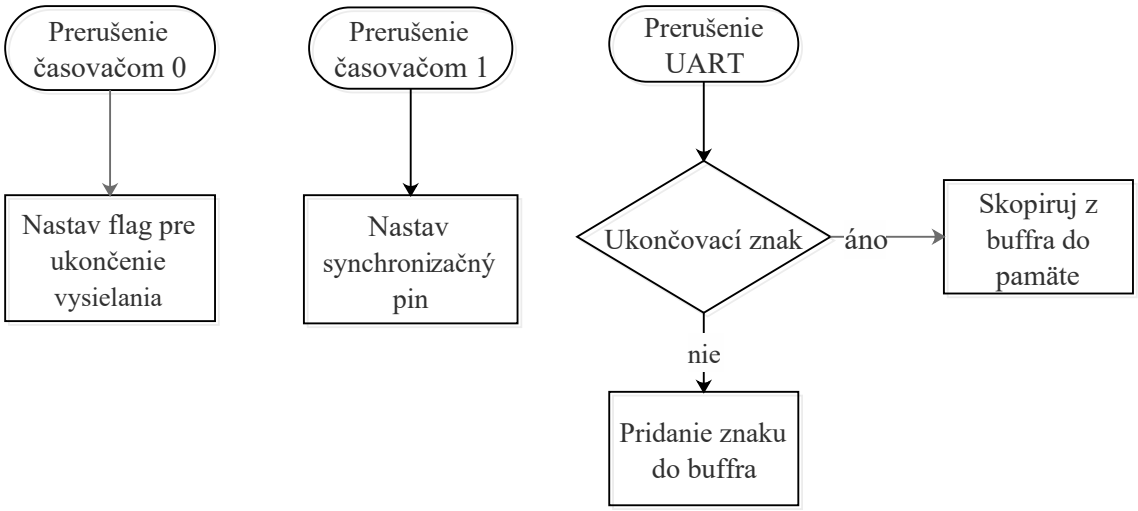
Obr. 4.5: Blokový diagram funkcie opätovného posielania stratených paketov



Obr. 4.6: Blokový diagram prerušení centrálneho uzla



Obr. 4.7: Blokový diagram hlavného programu uzla



Obr. 4.8: Blokový diagram prerušení uzla

5 Výsledky meraní

Zatiaľ táto časť nie je veľmi dobre spracovaná ale dostupné sú predbežné výsledky

Výsledky pre testovanú verziu 1

- dosiahnutá prenosová rýchlosť 30 kbit/s na uzol pri štyroch uvažovaných
- stratovosť paketov 1/110
- potreba re-transmisie paketu 1/10

Výsledky pre testovanú verziu 2

Je obtiažne merať prenosovú rýchlosť prenosu nakoľko je testovaný iba model s jedným aktívnym uzlom, pričom podľa 3.3 sa centrálny uzol stále pokúša o odpoveď od ostatných uzlov, čo má za následok veľké oneskorenie medzi ID slotom pre aktívny uzol. Stratovosť paketov sa znížila vplyvom použitia novej funkcie na prijatie jedného paketu pre oba programy, ktorá má vylepšenú konštrukciu, kde sa príkaz na prijatie paketu posiela len jeden krát, pokým paket nie je prijatý,a to aj v prípade opakovaného prijímania paketu. Prenosová rýchlosť sa môže taktiež zvýšiť používaním novo implementovaného binárneho módu, ktorý ukladá prijaté dáta v binárnej forme nie ako reťazec, a vysiela pakety o maximálnej možnej dĺžke čím sa zväčšuje efektivita prenášania dát, narozdiel od reťazcového módu kde každý reťazec je uložený v jednom pakete s zakončovacím znakom.

5.1 PRNG (pseudo random number generator)

Pre otestovanie a stabilitu programu z dlhodobého pohľadu bola zvolená forma testovania náhodnými dátami. Za týmto účelom bol použitý **lineárny kongruentný generátor** produkujúci pseudonáhodnú postupnosť čísel $x_1, x_2, x_3, ...$ s využitím lineárnej rekurentnej rovnice

$$x_n = (ax_{n-1} + b) mod\ m \tag{5.1}$$

pričom a, b, m sú parametre charakterizujúce generátor a x_0 je (tajná) počiatočná hodnota (angl. **seed**).

Parameter m je v prípade realizácie zvyčajne volený tak, aby operácia bola realizovaná automaticky bez nutnosti realizovať dodatočné matematické operácie. Typickým príkladom sú hodnoty $m = 2^{16}$, resp. $m = 2^{32}$ automaticky pri výpočtoch s presnosťou 16 resp. 32 bitov. Parametre m, a, b je možné zvoliť tak, aby perióda generovanej pseudonáhodnej postupnosti bola m.

Pri testovaní boli zvolené hodnoty $a = 214013$, $b = 2531011$, $m = 2^{32}$

Pri tvorbe referenčného skriptu pre program Matlab som narazil na problém presnej reprodukovateľnosti výpočtu pseudonáhodnej postupnosti, nakoľko program Matlab pracuje s väčšou presnosťou ako jadro procesora ADuC-RF101, čoho výsledkom boli zaokrúhľovacie chyby, čo sa ukázalo ako geometricky sa zväčšujúca chyba vo výpočte takejto postupnosti. Pri použití rovnakých dátových typoch ako v jazyku C nastal problém, že matlab používa saturačnú aritmetiku, čoho dôsledok bola konštantná saturovaná hodnota čísla 2^{32}. Z tohto dôvodu bol na odkontrolovanie a správne generovanie pseudo náhodných postupnosti použitý jazyk C++ ktorý je pre oba architektúry (cortex - M3, intel x86) používa rovnaké dátové typy definované ako štandard.

Pri vytváraní dátovej štruktúry "náhodného"paketu obsahujúceho jednotlivé premenné bolo nutné použiť direktívu **pragma pack(1)** čo zaručilo usporiadanie bajtov v poradí za sebou bez vynechania jediného, bez ochranného bytu, čo sa v praxi často využíva.

6 Zhodnotenie

Na základe porovnania z predchádzajúcou verziou je takto navrhnutý model výhodnejší pre spoľahlivosť prenosu ako aj pre dosiahnuteľnú prenosovú rýchlosť na uzol. Prípadne by v tomto modele mohol byť nastavený časový interval medzi opätovným vysielaním časových slotov, pre uvoľnenie frekvenčného pásma, keďže v použitej aplikácií nieje časovo kritická doba prijatia dát.

Najnovšiu verziu programu je možné nájsť na internetovom úložisku github https://github.com/petersoltys/

Literatúra

[1] Analog Devices. *ADuCRF101 User Guide*, 2013. 4

[2] Analog Devices. *Precision Analog Microcontroller with RF Transceiver, ARM Cortex-M3*, 2013. ii, 2