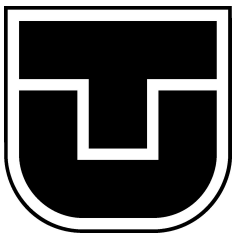
Program pre ADuC RF101, časový multiplex v2



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

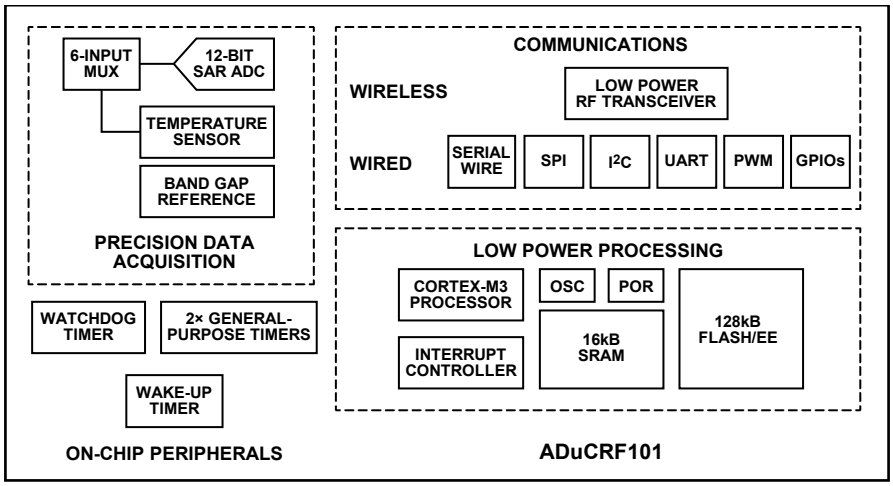
Zoznam obrázkov

1 Zadanie

Vytvorte pomocou komunikačného procesora **ADuC RF101** komunikačnú sieť s centrálnym zberom dát, z jednotlivých uzlov siete. Pre komunikáciu použite voľné komunikačné pásmo 890 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, Manchester kódovanie, premenlivú dĺžku paketov až do 240 bajtov, pri použití FSK alebo GFSK modulácie. Jeho bloková schéma je na obrázku 2.1.



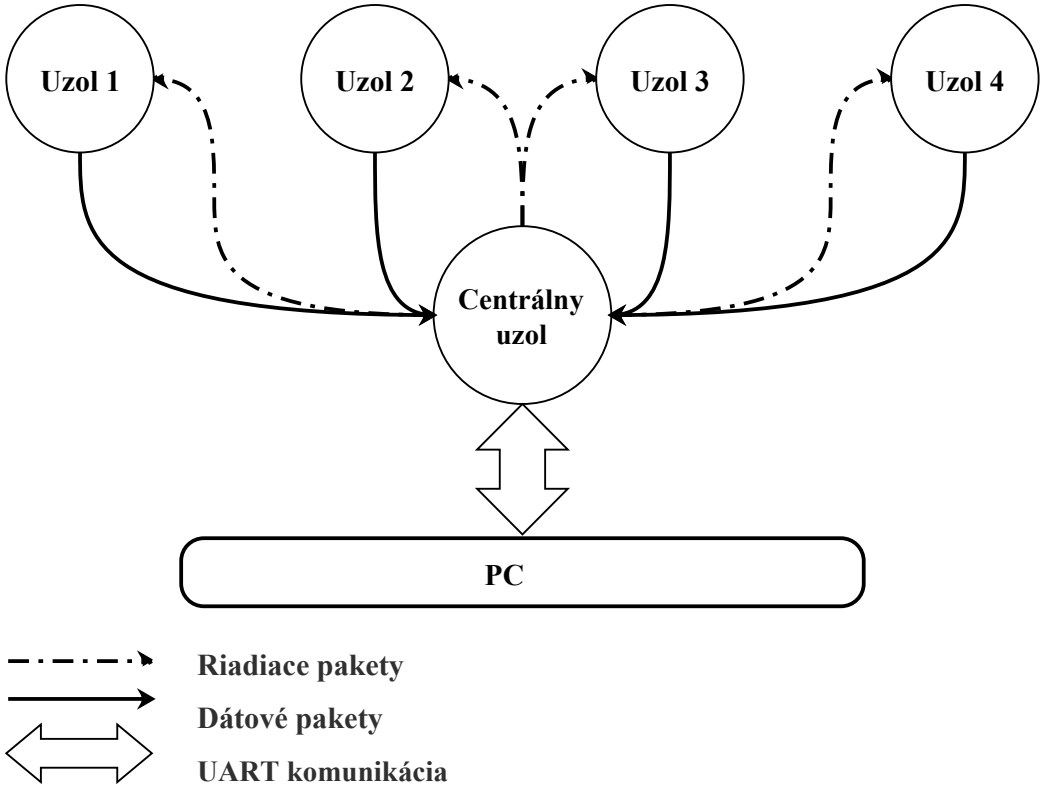
Obr. 2.1: Bloková schéma ADucRF101 [2]

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

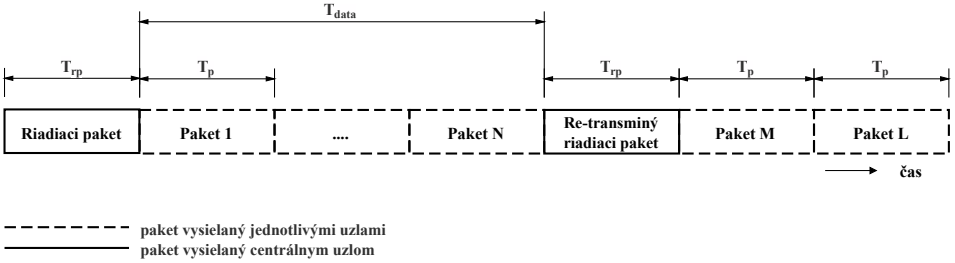
V princípe by celá komunikácia mala mať tvar zobrazený na obrázku 3.1.



Obr. 3.1: Základná schéma komunikácie

Pre dodržanie požadovaných parametrov bolo potrebné navrhnúť schému fungovania prenosu. Pre dôkladný návrh riešenia je nutné poznať vlastnosti rádiofrekvenčného 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 ako je zobrazené na obrázku 3.2, kde riadiace pakety vysiela centrálny uzol s požiadavkou, aby požadovaný uzol s priradeným **ID** (identifikátorom) vysielal nahromadené dáta, čo vykonáva bez čakania na potvrdenie prijatia. Keďže pakety majú svoje poradové čísla, tak centrálny uzol dokáže zistiť či došli všetky nahromadené dáta, ak nie tak pošle riadiaci paket na retransmisiu chýbajúcich paketov **M, L, ...**

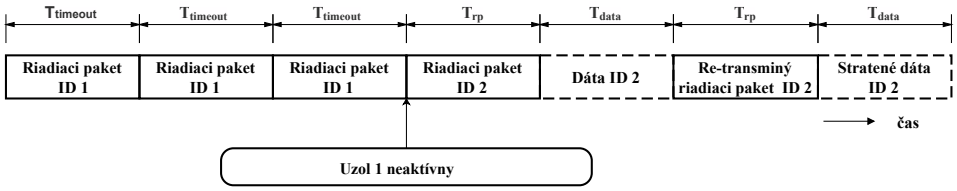


Obr. 3.2: Časová schéma multiplexu

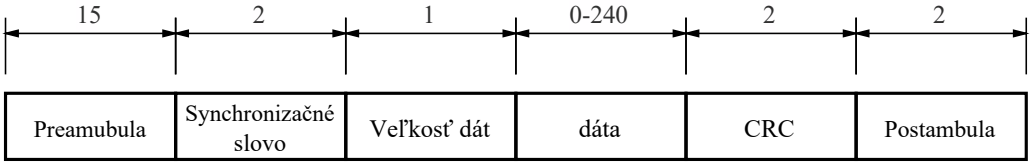
Pred začatím vysielania má uzol v pamäti nazhromaždené pakety, maximálne v tomto prípade 20, z dôvodu obmedzenia RAM pamäte ktorá má len 16 kbajt, teda $2 * 20 * 240 = 9600$ bajt (pozri obrázky 3.4 a 4.2). Uzol vysiela pakety s hlavičkou ktorá obsahuje počet nahromadených paketov ako aj poradové číslo 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} sú premenlivé, keďže pakety môžu nadobúdať rôznu dĺžku.

Pretože vždy existuje možnosť že sa budú strácať aj riadiace pakety, centrálny uzol tieto správy posiela viacnásobne, kým požiadaný uzol nepošle požadované dáta, alebo nulový paket, 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.

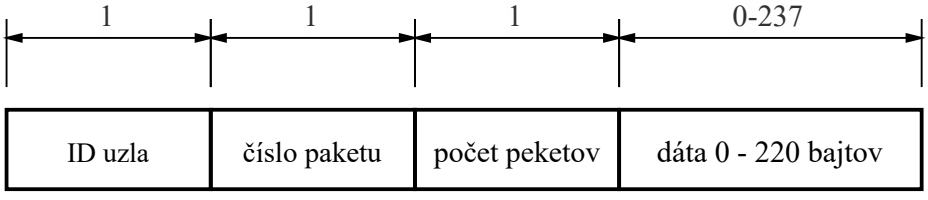
Vo všeobecnosti každý posielaný paket má tvar 3.4 a v jednotlivých paketoch je rozdielna iba dátová (ďalej označovaná ako paket) oblasť o dĺžke 240 bajtov ale pri dátovom pakete kvôli pridávanej hlavičke ostáva pre dáta 237 bajtov ako je vidno na obrázku 3.5. Ak uzol nemá nahromadené žiadne dáta vo svojej pamäti, tak vysiela nulový paket, ktorý má nulové oba polia označujúce počet a poradové číslo.



Obr. 3.3: Časová schéma viacnásobného posielania

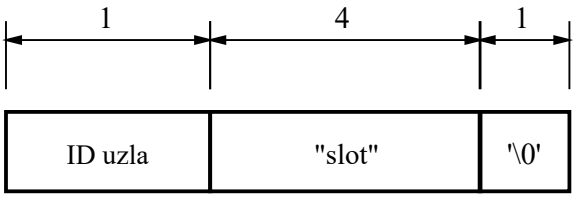


Obr. 3.4: Všeobecný formát paketu



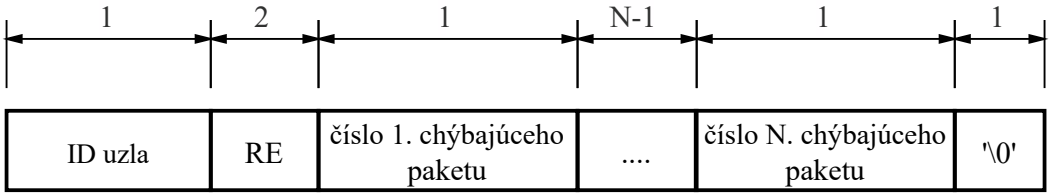
Obr. 3.5: Formát dátového paketu

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



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.

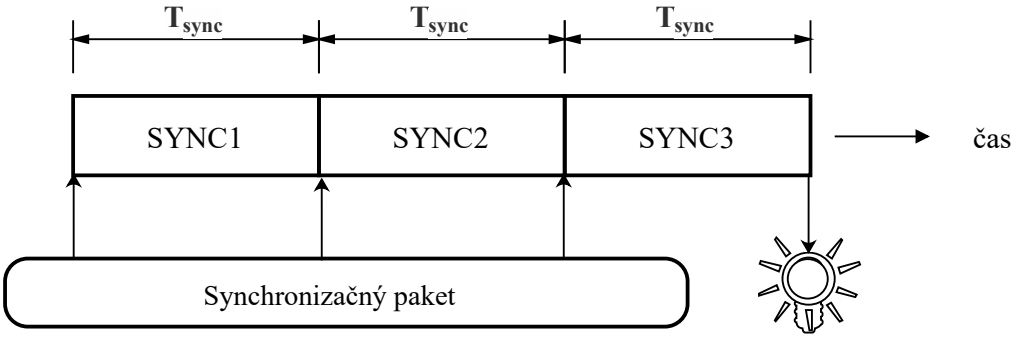


Obr. 3.7: Formát re-transmisného paketu

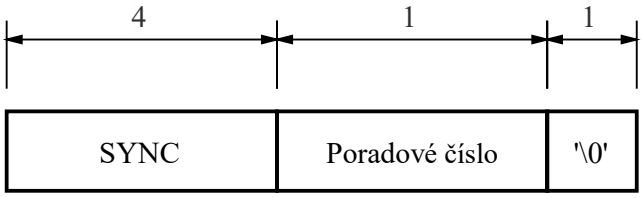
3.1 Synchronizácia

V konkrétnej aplikácií kde majú byť nasadené tieto procesory je dôležité zabezpečiť synchronizáciu merania. Spúšťanie merania bude uskutočňované zmenou logickej úrovne na výstupnom pine. 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 $T_{sync} = 1.2ms$. Teda postačuje ak je prijatý jediný synchronizačný paket, ktorý v poradovom čísle nesie aj informáciu o zostávajúcom čase do synchronizácie. Čas po synchronizáciu ($T_{synchronize} = T_{sync} * N$, kde N =poradové číslo), z tejto informácie si každý uzol nastaví čas spustenia. Táto schéma synchronizácie sa spúšťa až po ukončení posielania dát.

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



Obr. 3.8: Schéma synchronizácie



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 správnej funkčnosti softvéru je kontrolovaná integrita pomocou softvérového CRC (Cyclic Redundancy Check) súčtu, ktorý postupne spravý súčet z celej programovej pamäti, kde na konci je externým nástrojom **Srecord** [5] zapísaný výsledok CRC, ktorý zabezpečí výsledný súčet kontroly je rovný nule, v prípade že výsledok nieje rovný nule, znamená to že je poškodený riadiaci softvér a program sa nespustí.

4 Programová realizácia

Pri vytváraní programu pre **ADuC RF101** bola použitá oficiálna knižnica pre komunikačný modul, čo značne zjednodušovalo prácu s interným rádio-frekvenčným modulom ovládaným vnútornou SPI zbernicou. Počas vytvárania programu, 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 prístupu bolo vykonanie rutiny v nevhodný okamih, čo malo častokrát za následok nestabilitu programu.

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 prípadoch 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 paketu a pod. Pričom nastavenia všetkých periférnych obvodov je možné zmeniť v spoločnom hlavičkovom súbore **settings.h**

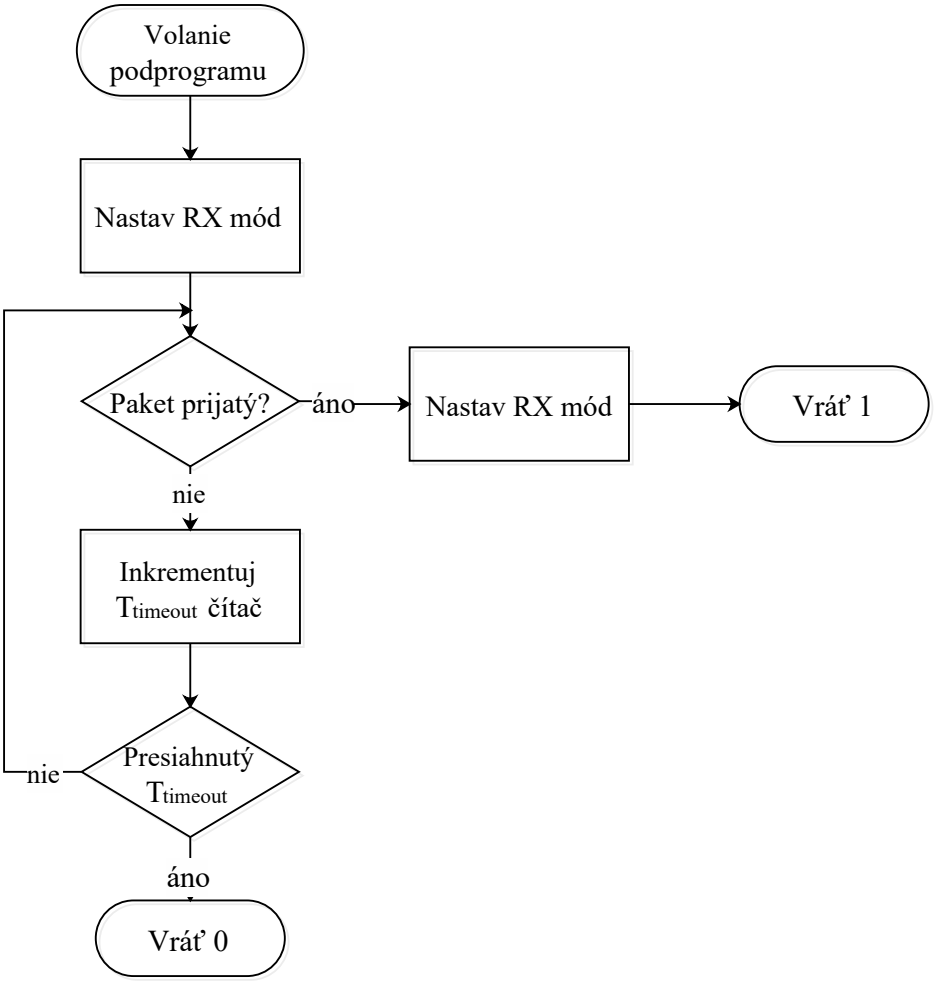
V ďalších častiach budú opísané len tie riešené problémy, ktoré tvorili veľké obtiažnosti pri ladení a je vhodné vedieť o ich existencii.

4.1 Sériová komunikácia

Na zabezpečenie prenosu dát v hexadecimálnych znakoch je potrebné spraviť prevod z binárnych dát do ASCII znakov. no testovaním bolo zistené že tento prevod sa nemôže uskutočňovať v prerušení nakoľko spôsoboval nestabilitu knižníc komunikujúcich s interným rádiovým modulom. Z tohto dôvodu sa prevod vykonáva počas čakania na prijatie paketu.

4.2 Kontrola integrity firmvéru

Za týmto účelom bola použitá už pripravená knižnica spočívajúca CRC [4] v ktorej bol nastavený štandard CCITT [3] používajúci polynóm v tvare $x^{12} + x^5 + 1$ [3]. Táto funkcia bola použitá v kombinácií s nástrojom Srecord [5] pomocou ktorého je súbor *.hex predĺžený na veľkosť pamäte a na miesto posledných dvoch bajtov vloží ïnverzný"výsledok CRC, čo pri kontrole vnútorným programom zabezpečí výsledok rovný nule. Následne tento vygenerovaný *.hex súbor musí byť



Obr. 4.1: Blokový diagram prijímania jedného paketu

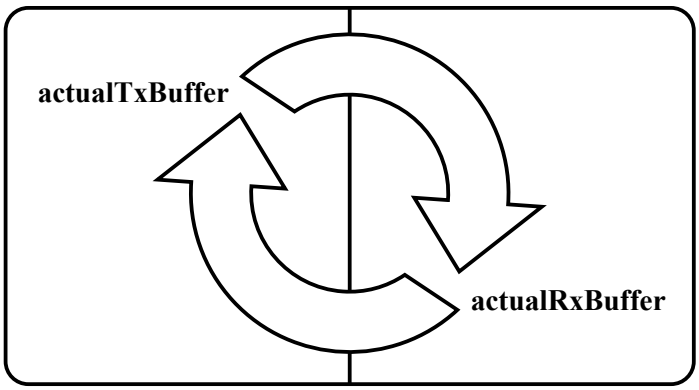
nahraný do mikroprocesora pomocou zavádzača (bootloader) na čo slúži program **CM3WSD** od **Analog Devices** .

4.3 Reštart rádiokontroléra

Počas práce s cieľovými procesormi bol nájdený nedostatok interného rádiokomunikačného modulu. Počas svojej štandardnej prevádzky fungoval spoľahlivo, do chvíle kým výkon prijímaného signálu nepoklesol pod minimálnu požadovanú úroveň -107.5 dBm (podľa [2] pri 38.4 kbps FSK), alebo nedošlo k reštarte niektorého z uzlov, počas vysielania paketu. Pri takejto udalostí nastalo zastavenie fungovania rádiokomunikačného modulu. Na takto vzniknutý stav nepostačovalo znovu inicializovanie, preto to bolo nutné riešiť hardvérovým reštartom. Na to bola vytvorená nová funkcia, ktorá hardvérovo reštartuje len modul, bez procesora. Za týmto účelom bola rozšírená knižnica ovládajúca rádiokomunikačný modul o funkciu **RadioHWreset**. Samotný rádiokontrolér podľa [2] podporuje hardvérový reštart príkazom cez SPI zbernicu, čo bolo využité v tejto funkcii.

4.4 Program pre centrálny uzol

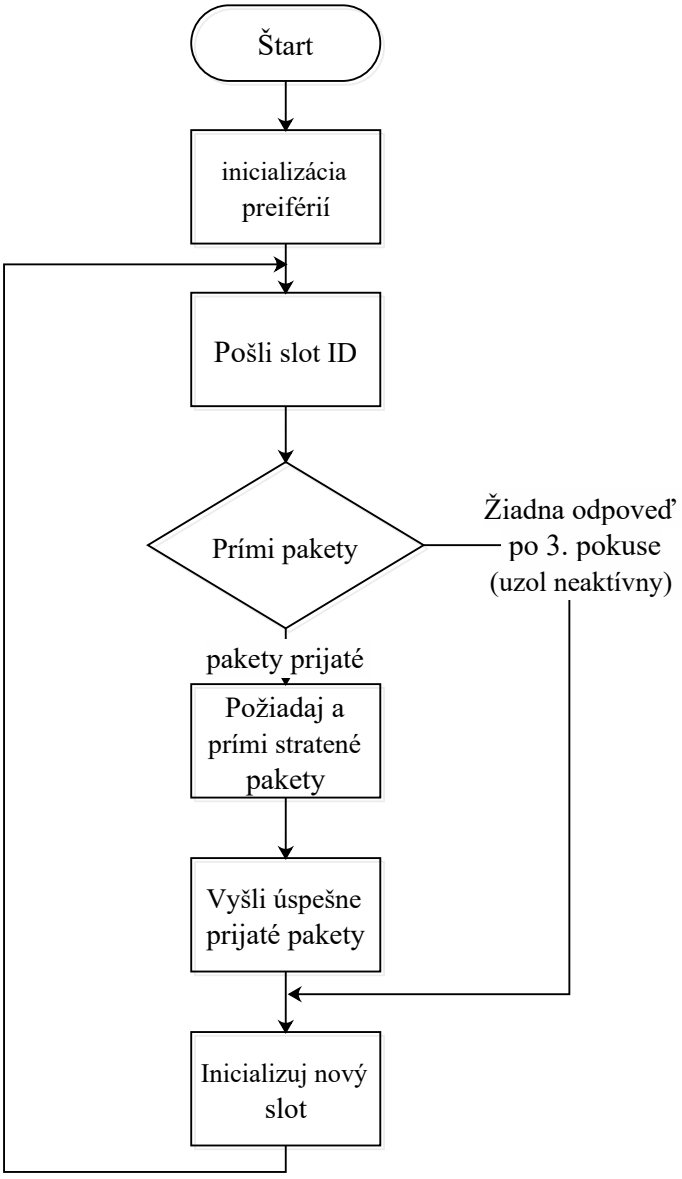
Pakety v centrálnom uzle sú zhromažďované až pokým neskončí retransmisia stratených paketov. Tieto dáta je nutné ukladať do pamäte, ktorá je organizovaná podľa obrázku 4.2 ako kruhová pamäť, kde do jednej časti sú prijaté dáta zapisované a z druhej sú vysielané cez UART do hosťovského zariadenia za pomoci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



Obr. 4.2: Organizácia pamäte

Hlavný program zobrazený na obrázku 4.3 je principiálne jednoduchý. Nie-

ktoré zložitejšie funkcie sú taktiež zobrazené na nasledujúcich obrázkoch.

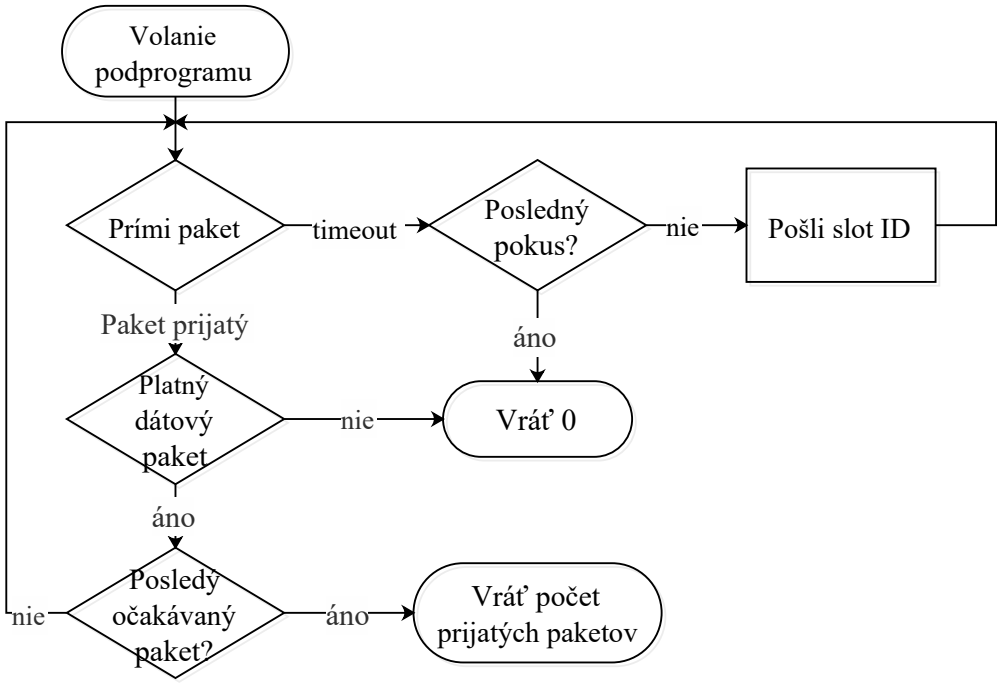


Obr. 4.3: Blokový diagram hlavného programu centrálneho uzla

Dôležitou funkciou pri fungovaní programu je funkcia **receivePackets** zobrazená na obrázku 4.4 zodpovedná za prijatie, vyhodnotenie 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, ktoré neboli prijaté alebo nesedel ich CRC súčet.

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



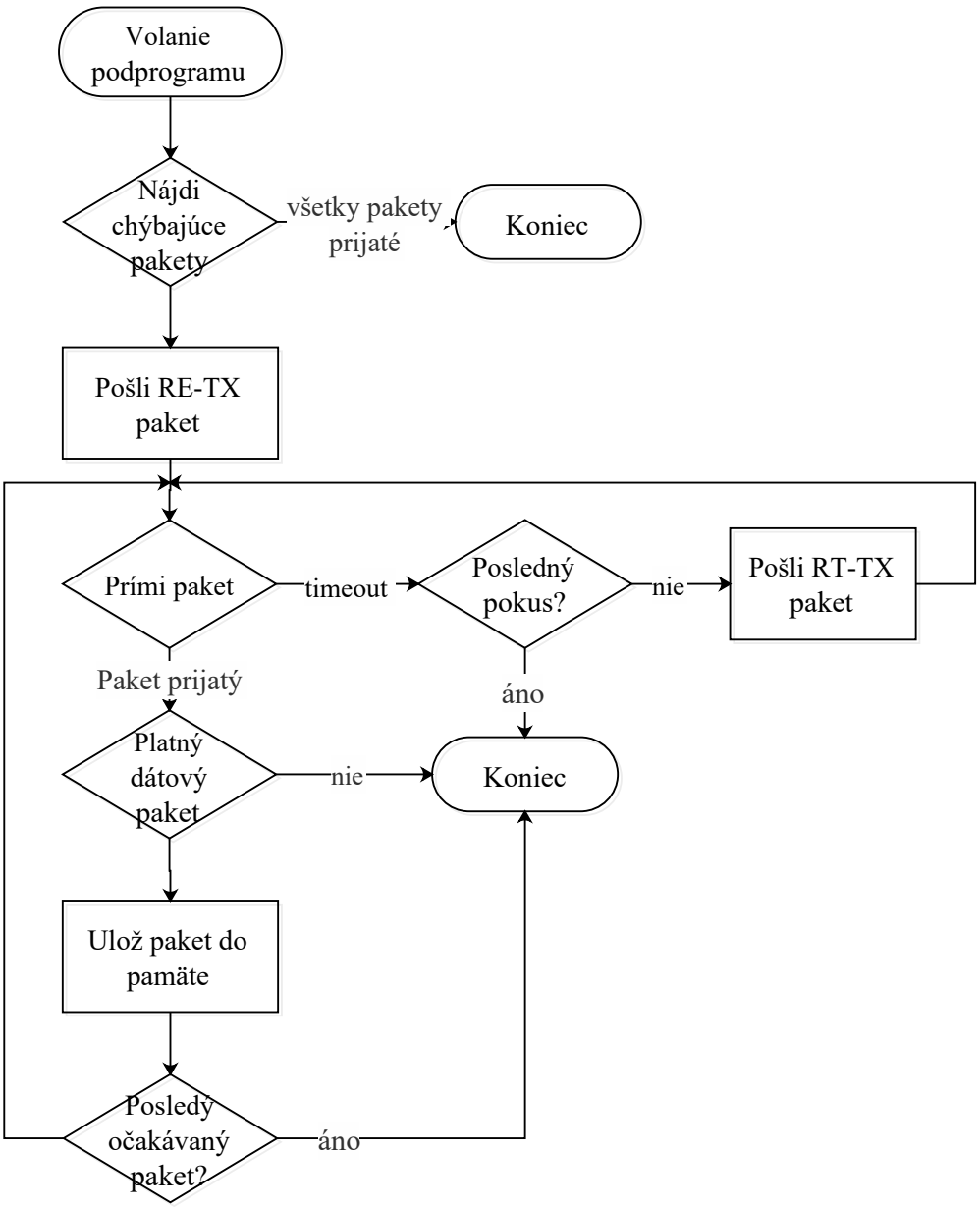
Obr. 4.4: Blokový diagram prijímania paketov

4.5 Program pre jednotlivé uzly

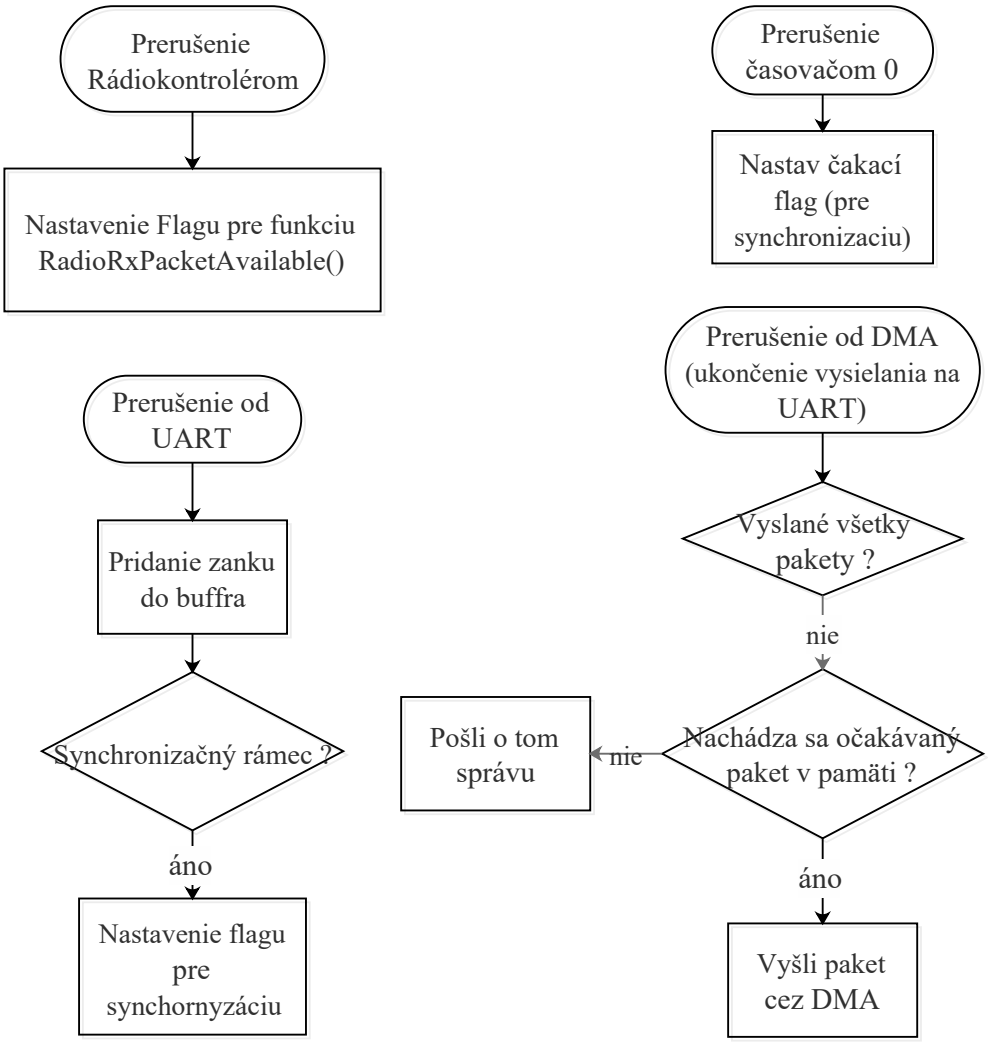
Program pre jednotlivé uzly zhromažďuje pakety v pamäti až pokým príme riadiaci paket s príslušným identifikátorom ktorý požaduje o poslanie nahromadených dát. Preto má rovnakú kruhovú organizáciu pamäte ako centrálny uzol rozdelenú na dve časti ako zobrazuje obrázok 4.2. Takáto organizácia umožňuje 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.

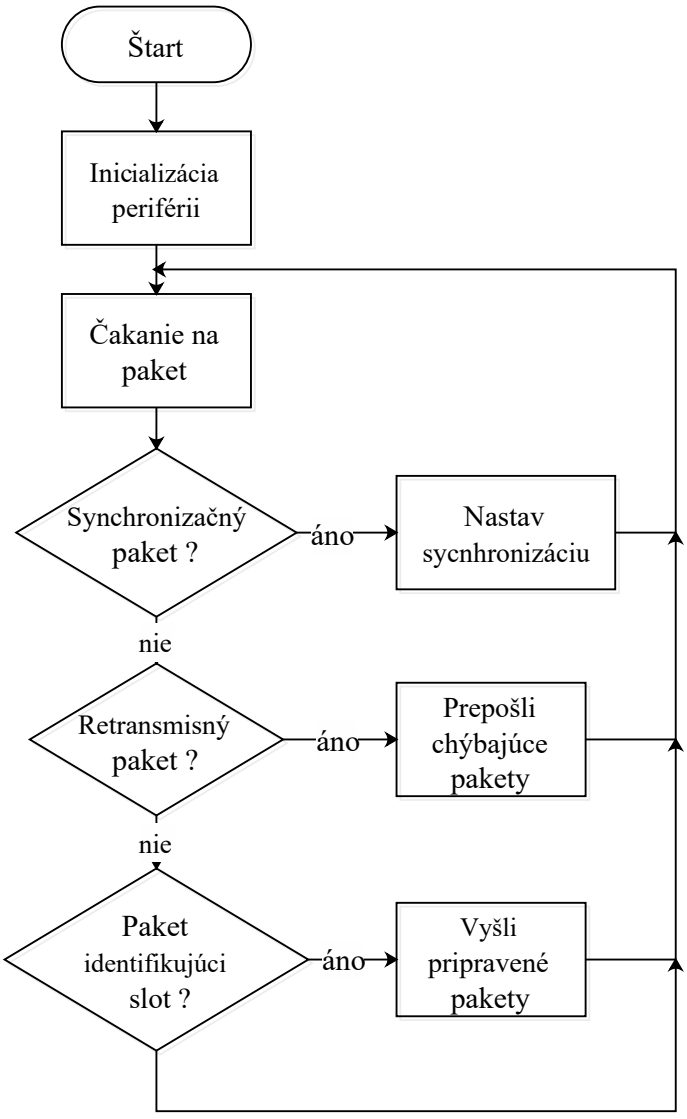
Pre pochopenie fungovania celého programu je potrebné taktiež poznať čo sa vykonáva v procesore počas prerušení, čo je opísané na obrázku 4.8.



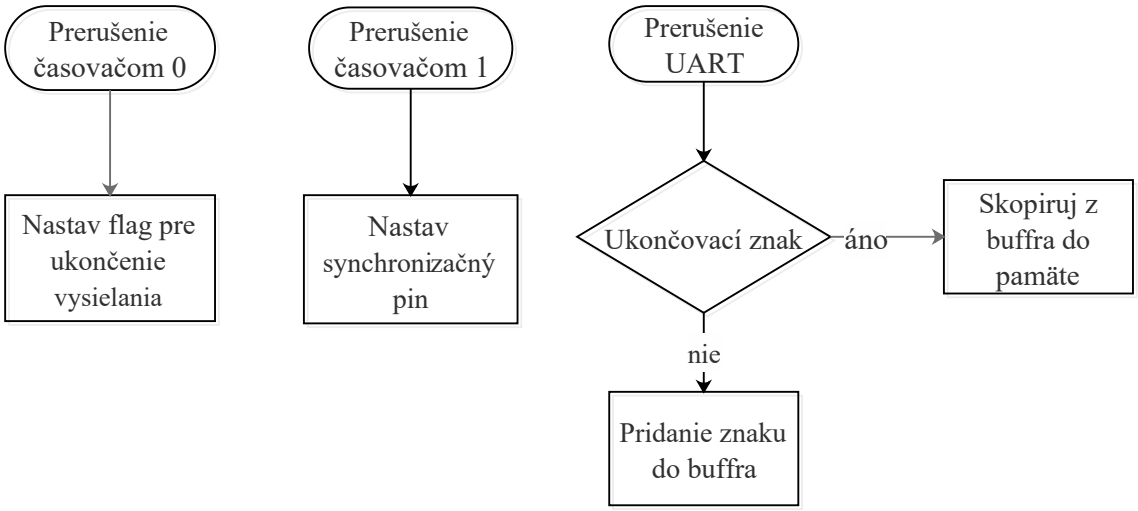
Obr. 4.5: Blokový diagram funkcie opätovného posielania stratených paketov



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

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 zvýšila použitím binárneho módu, ktorý ukladá prijaté dáta v binárnej forme nie ako reťazec. Vysielané pakety o maximálnej možnej dĺžke zvyšujú efektivitu prenášania dát, narozdiel od reťazcového módu kde každý reťazec je uložený v jednom pakete s ukončovacím znakom.

5.1 PRNG (pseudo random number generator)

Pre otestovanie priepustnosti a stability 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**).

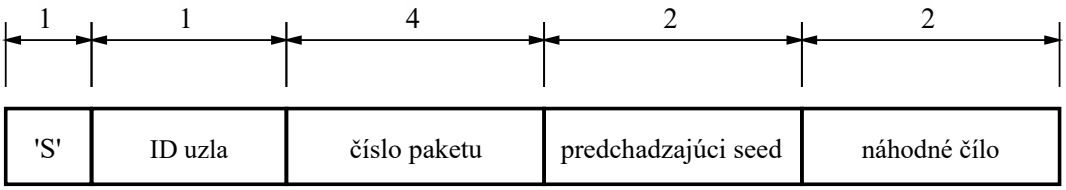
Parameter m je v prípade realizácie zvyčajne volený tak, aby operácia bola realizovaná automaticky bez nutnosti realizovať dodatočné matematické operácie. Typickým príkladom sú hodnoty $m = 2^{16}$, resp. $m = 2^{32}$ automaticky pri výpočtoch s presnosťou 16 resp. 32 bitov. Parametre m, a, b je možné 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 ol zistený problém presnej reprodukovateľnosti výpočtu pseudonáhodnej postupnosti.Keďže program Mat-

lab 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* a *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 vkladania ochranného bajtu, čo sa v praxi často využíva. Samotný formát takéhoto paketu je zobrazený na obrázku 5.1 Takto formátované 12 bajtové slová zaplnili celú dátovú oblasť paketu.



Obr. 5.1: Formát pseudonáhodného paketu

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, 10

[3] Joe Geluso. Crc16-ccitt, 2001-2007. http://srecord.sourceforge.net/crc16-ccitt.html. 8

[4] Michael Barr. *Slow and fast implementations of the CRC standards.*, 2000. 8

[5] Peter Miller. Srecord 1.64, Jun 2014. http://srecord.sourceforge.net/. 7, 8