UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sašo Skube

BREZŽIČNI DOSTOP DO NAPRAV Z VMESNIKOM RS-232 PREKO OMREŽJA ZIGBEE

UNIVERZITETNA DIPLOMSKA NALOGA

Mentor: izr. Prof. Dr. Uroš Lotrič

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01709/2010
Datum: 01.10.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **SAŠO SKUBE**

Naslov: **BREZŽIČNI DOSTOP DO NAPRAV Z VMESNIKOM RS-232 PREKO OMREŽJA ZIGBEE**

WIRELESS ACCESS TO DEVICES WITH RS-232 INTERFACE OVER ZEGBEE NETWORK

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Na tržišču je še vedno mnogo naprav, ki za komunikacijo z drugimi napravami uporabljajo fizično povezavo preko vmesnika RS-232. V nalogi raziščite možnosti, ki jih za premostitev fizične povezave ponuja moderno brezžično omrežje Zigbee. Izdelajte tudi napravo z vso potrebno programsko opremo, ki v omrežju Zigbee ustvari brezžični most za dvosmerno komunikacijo med napravami z vmesnikom RS-232.

Mentor:
prof. dr. Uroš Lotrič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI 2

Dekan:
prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Sašo Skube,

z vpisno številko 63050107,

sem avtor diplomskega dela z naslovom:

BREZŽIČNI DOSTOP DO NAPRAV Z VMESNIKOM RS-232
PREKO OMREŽJA ZIGBEE

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Uroša Lotriča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja: _____

Zahvala

Za mentorstvo, nasvete in veliko pomoč pri zaključevanju diplomske naloge se zahvaljujem prof. dr. Urošu Lotriču. Poleg tega sta mi bila v veliko pomoč dr. Andrej Štrancar z nasveti za izbiro čipov in programatorjev ter Tomaž Strmec, ki mi je pomagal z risanjem posameznih strojnih elementov in svetoval pri izdelavi tiskanega vezja. Zahvaliti se moram tudi svojemu bratu, Petru Skubetu, ki mi je pomagal najti in odpraviti nekaj programskih hroščev.

Kazalo

Povzetek

Veliko naprav za komunikacijo še vedno uporablja dobro znani protokol RS-232 . Ker smo pri tem vezani na določeno dolžino kablov in ker so takšne naprave velikokrat na težko dostopnih mestih, smo potrebovali rešitev, ki bi omogočala prenos teh podatkov v brezžičnem omrežju.

V zadnjih letih se je pojavilo kar nekaj novih standardov za brezžične komunikacije. Eden izmed njih je tudi standard ZigBee, katerega prednosti sta predvsem majhna poraba energije in možnost varnega in zanesljivega povezovanja naprav v splošna omrežja.

V delu je predstavljena naprava, ki preko omrežja ZigBee lahko prenaša podatke med več napravami z vmesniki RS-232. Nanjo lahko priklopimo do štiri končne naprave z vmesnikom RS-232 ter štiri brezžične odjemalce, od katerih je vsak vezan na natanko eno končno napravo. Kot odjemalec je predviden osebni računalnik z ustrezno programsko opremo in vmesnikom ZigBee.

Izdelali smo vso potrebno programsko in strojno opremo za delovanje takšne naprave, skupaj s postopki in teoretičnimi opisi standarda. Prvi testni kažejo, da naprava deluje zanesljivo in je stabilna.

Ključne besede: ZigBee, brezžičen, RS-232

Summary

For communications among devices the well-known protocol RS-232 is still frequently used. Since the protocol requires physical connection to devices which can also stand in inaccessible places, a solution that enables data transmission over a wireless network would be greatly appreciated.

In the last few years some new wireless communication standards appeared. One of them is ZigBee, which has the advantages of small energy consumption and the possibility of safe and reliable connection of devices in mesh networks.

In this work a device is presented, that can transmit data through the ZigBee network between client computers with ZigBee transmitters and several end devices with RS-232 ports. The developed device allows us to plug-in up to four end devices with the RS-232 port, and four wireless clients.

Further on, hardware schemes and software modules of the developed device are described in detail together with basic procedures and theoretical descriptions of the standard. The first tests show that the operation of device is reliable and stable.

Keywords: ZigBee, wireless, RS-232

1. Uvod

V zadnjem času se precej sliši o novih brezžičnih tehnologijah. Te naj bi bile vse boljše in boljše in na koncu enakovredne povezavam, za katere danes uporabljamo običajne splete kablov. Seveda so različne brezžične tehnologije prirejene za različne namene. Na eni strani imamo energijsko potratne tehnologije in omrežja za pretok velikih količin podatkov med dvema točkama, na drugi strani pa, na primer, omrežja senzorjev in aktuatorjev, pri katerih hitrost, za razliko od energijske varčnosti, ni bistvenega pomena. Eden od standardov, namenjen za povezovanje senzorjev in aktuatorjev je ZigBee. Standard vzdržuje organizacija za standardizacijo ZigBee Alliance. Kot vse ostale organizacije tega tipa združuje različne partnerje, ki skupaj skrbijo za razvoj standarda. Njihov cilj je končnemu uporabniku ponuditi napravo, ki bo izredno energijsko učinkovita, prenosljiva, cenovno ugodna, fleksibilna in predvsem varna. S standardom ciljajo na avtomatizacije domov, poslovnih prostorov in celo industrije. Pri tem poudarjajo moč standarda za nadzorne in kontrolne aplikacije. Z moduli ZigBee lahko ustvarimo poljubno omrežje med seboj povezanih naprav. To so lahko računalniki, prenosne naprave, senzorji, procesne enote in ostali vmesniki.

V računalništvu in na sploh v industriji še vedno srečujemo naprave, ki za povezovanje uporabljajo vmesnik RS-232. Najbolj slavni med njimi so morda modemi ali pa mrežna stikala in usmerjevalniki. Za povezavo s temi napravami pa še vedno potrebujemo običajen komunikacijski kabel. Poleg tega so naprave tega tipa skoraj vedno v odmaknjenih lokacijah, do katerih je fizičen dostop omejen. Hitrosti prenosa po vodilu RS-232 pa so majhne v primerjavi z možnostmi, ki nam jih ponuja omrežje ZigBee.

Priročno je imeti napravo, na katero ob inštalaciji opreme na svojo končno mesto priključimo različne ciljne naprave z vmesnikom RS-232. Z računalnikom, ki ima ustrezen vmesnik ZigBee se nato lahko brezžično povežemo na takšno napravo in preko nje na ciljne naprave z vmesniki RS-232. Komunikacija med računalnikom in ciljno napravo nato teče na enak način kot v primeru, ko smo na ciljno napravo povezani neposredno s kablom. Zaželeno je, da ima naprava, na katero priključujemo ciljne naprave, več priključnih mest.

Sama naprava je realizirana s pomočjo mikroprocesorjev in nekaj dodatne logike. Ker ima štiri vhodno izhodna vrata ima tudi štiri medpomnilnike za podatke, ki prihajajo iz teh naprav. Za povezavo v brezžično omrežje skrbi modul ZigBee, vse skupaj pa koordinira centralni procesor. Ta omogoča priklop naprav, v svojo notranjo tabelo shranjuje podatkovne poti različnih priključenih naprav, posreduje sprejete podatke in nadzira modul ZigBee.

Poleg naprave smo izdelali tudi aplikacijo, ki nam omogoča iskanje in povezovanje s ciljnimi napravami z vmesnikom RS-232, ki so preko razvite komunikacijske naprave na voljo v omrežju.

V nadaljevanju bomo najprej primerjali tehnologijo ZigBee z ostalimi bolj poznanimi in razširjenimi tehnologijami kot so običajna brezžična komunikacija, standard Bluetooth, in s tehnologijo, ki jo uporabljajo mobilni telefoni. Poleg tega podrobno opišemo sam standard ZigBee, ki ga razčlenimo po posameznih komunikacijskih plasteh v luči arhitekture ISO/OSI.

Nato v poglavju 3 nadaljujemo z opisom delovanja naprav ZigBee. Pregledamo osnovne korake, ki jim mora slediti vsaka naprava, ki deluje po tem standardu. Ustavimo se pri zgradbi samih paketov, ki potujejo po prenosnem mediju in povemo nekaj več o algoritmih za preprečevanje šuma in varnosti prenosov. Ker obravnavamo prenose podatkov med vmesniki RS-232, na kratko razložimo potek in pravila za ta tip serijske komunikacije skupaj z navodili za upravljanje izbranih modulov ZigBee.

Naša ideja in vsi napisani programi pa niso dovolj, če nimamo ustrezne strojne opreme. Zato v poglavju 4 opišemo kako smo načrtovali strojno platformo. Nadaljujemo z opisom programske rešitve zadanega problema. Sestavljena je iz treh delov. Prvi del predstavlja aplikacija na računalniku, ki omogoča uporabniku priključitev na naprave in komunikacijo z njimi. V drugem delu predstavimo zunanji programski medpomnilnik, ki ščiti opremo pred prevelikimi količinami podatkov iz končnih naprav, tretji del pa predstavlja sam program v nadzornem procesorju, ki skrbi za koordinacijo sistema. Opišemo tudi težave, na katere smo naleteli med projektom.

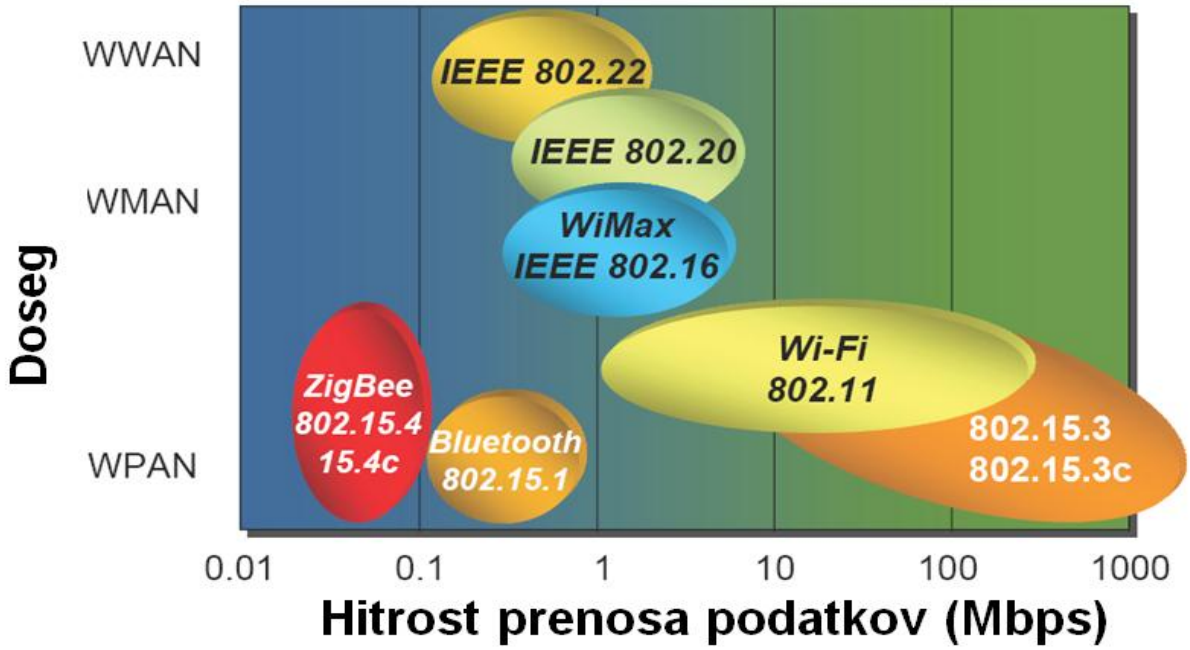
V poglavju 5 povzamemo glavne značilnosti projekta in se osredotočimo na izboljšave obstoječe naprave.

V dodatku A je priložena celotna izvorna koda za vse napisane programe.

2. Omrežje ZigBee

2.1. Osnovne lastnosti

Organizacija ZigBee Alliance je ustvarila standard za dvosmerno komunikacijo z napravami. Pri tem so poseben poudarek namenili čim manjši rabi energije in čim nižji ceni izdelkov. Kot je razvidno tudi iz slike 1, je namenjen predvsem senzorskim in kontrolnim omrežjem, domači avtomatizaciji, gradnji pametnih hiš, uporabi v industriji, medicini, igračah,…
Hitrost prenosa podatkov preko omrežja ZigBee doseže do 250kbit/s, v 2,4GHz območju ima 16 kanalov, v 915Mhz 12 in v 868Mhz območju en kanal.



Slika 1: Primerjava hitrosti in dosega različnih tehnologij [8]

V splošnem poznamo tri vrste naprav ZigBee. Prvi tip naprave je koordinator, ki je v vsakem omrežju samo eden. To je naprava, ki omrežje vzpostavi in na katerega se lahko priključijo druge naprave. Koordinator mora znati vzpostaviti omrežje, dodajati naprave v omrežje ter usmerjati podatke.
Drugi tip naprave je usmerjevalnik, ki se lahko poveže z drugim usmerjevalnikom ali pa s koordinatorjem. Nanj se lahko priklopi končna naprava ali pa drug usmerjevalnik. Tudi ta mora podpirati asociacijo z drugimi napravami ter usmerjanje prometa. Koordinato in usmerjevalnik se pogosto označujeta tudi kot napravi FFD (ang. Fully Functional Device).

Tretji tip naprave je končna naprava ali naprava RFD (ang. Reduced function Device), na katero je priključen, na primer, senzor, motor, rele in se lahko poveže na usmerjevalnik ali koordinator, če dopustimo pa tudi na drugo končno napravo.

Vse te naprave se lahko med seboj različno povezujejo. Poznamo tri tipične topologije, prikazane na sliki 2, ki jih omrežje ZigBee podpira:

- **Omrežja v obliki zvezde**

Prednost takšnega načina povezovanja je enostavnost, kajti usmerjanje v takšnem omrežju je trivialno. V osnovnem omrežju z eno centralno lokacijo je usmerjevalna pot med vsemi končnimi napravami dolga dva koraka.

Ozko grlo je preobremenjenost glavnega vozlišča ter nevarnost izpada le-tega, saj v takšnem primeru komunikacija med vsemi napravami izpade. Takšna topologija ni primerna za velika omrežja.

- **Omrežja v obliki drevesa**

Ta topologija je pravzaprav le večplastna zvezdna topologija. Algoritmi usmerjevanja so tudi v takšnem sistemu relativno preprosti, saj ima vsaka končna naprava eno samo možno povezovalno pot.

Izpad enega vozlišča tukaj ne ohromi celotnega omrežja, ampak le vse svoje sinove, prav tako pa je prepustnost omrežja večja, saj vse povezave ne potujejo skozi koren drevesa.

- **Splošna omrežja**

So poenostavitev polne topologije. Ta predvideva, da je vsaka naprava neposredno povezana z vsako drugo v omrežju. Splošna topologija vsebuje le poljubno podmnožico povezav popolne topologije, ki mora še vedno zagotavljati povezanost vseh naprav v omrežju, le da le-te niso nujno direktno povezane. Ta topologija je v praksi najbolj pogosto uporabljana. Kompleksnost omrežja, njegova cena in zahtevnost usmerjevalnih algoritmov je v končni fazi odvisna od same implementacije.

Slika 2: Tipi omrežij in naprav. S svetlo modro so označene končne naprave, z rdečo usmerjevalniki in s temno modro koordinatorji.

Kot je razvidno iz tabele 1, lastnosti kot so nizka cena, majhna porabe energije, nizka poraba sistemskih virov, ter možnost skoraj neomejenega števila naprav v omrežju dajejo temu protokolu veliko prednost pred drugimi podobnimi tehnologijami na področju nadzora, upravljanja z napravami in avtomatizacije. Količina prenesenih podatkov po takšnih omrežjih je majhna, zato pasovna širina ni zelo pomembna lastnost. Vsekakor je pomembna tudi cena kompletnega modula, ki nam omogoča vsaj osnovno komunikacijo, in se v času pisanja giblje okoli 14€ na enoto, v večjih količinah pa seveda ustrezno manj.

Tabela 1: Primerjava različnih tehnologij

Prodajno ime	**ZigBee**	**/**	**Wi-Fi**	**Bluetooth**
Standard	**802.15.4**	**GSM/GPRS CMDA/1xRTT**	**802.11b/g**	**802.15.1**
Namen	nadzor in kontrola	širokopasoven prenos glasu in podatkov	internet, pošta, multimedija	nadomestek žice
Zahtevani sistemski viri	4 KB-32 KB	16 MB+	1 MB	250 MB +
Vzdržljivost baterije(v dneh)	100-1000+	1-7	5	1-7
Število naprav v omrežju	skoraj neomejeno	1	32	7
Hitrost prenosa podatkov [Mbps]	20-250	64-128+	11000+	720
Doseg [m]	1-100+	1000+	1-100	1-10+
Prednosti	zanesljivost, moč, cena	doseg, kvaliteta	hitrost, prilagodljivost	cena, priročnost

2.2. Organizacija komunikacijskega sklada

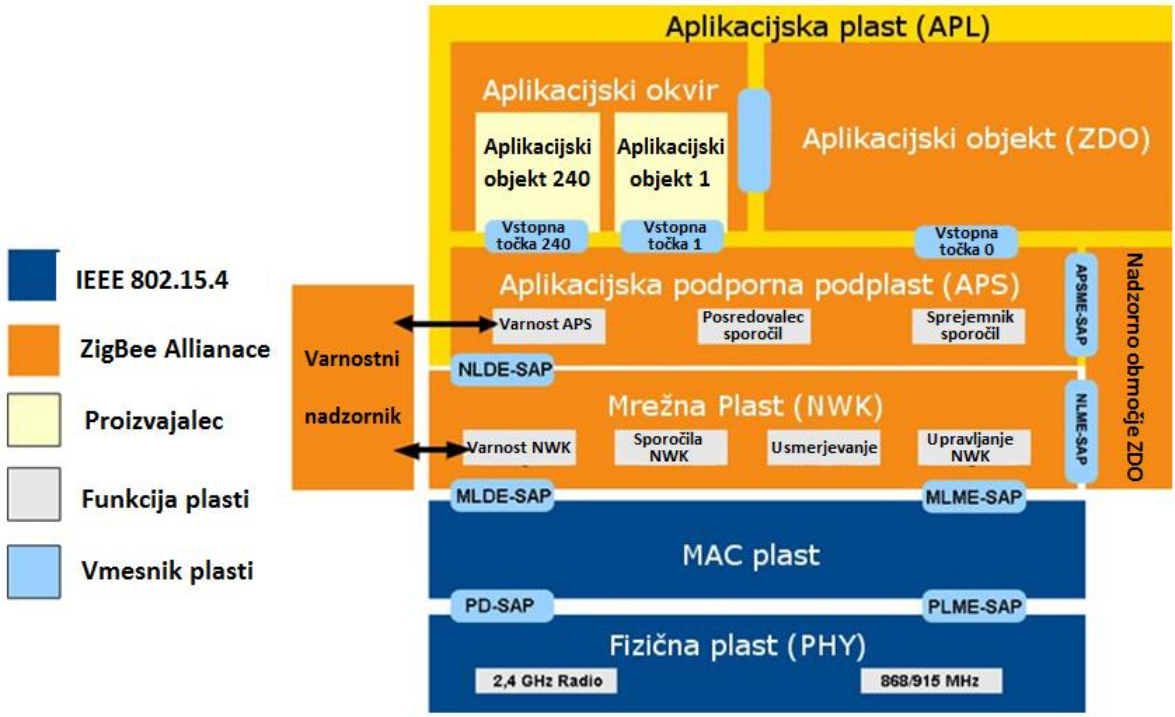
Glavna naloga opisanih naprav je izmenjevanje podatkov. Ti se prenašajo preko več plasti v isti napravi in se pri tem na različne načine spreminjajo. Te plasti lahko imenujemo komunikacijski sklad.

Sklad standarda ZigBee je sestavljen iz štirih plasti (ang. layers), ki so grafično prikazane na sliki 3. Vsaka plast je preko logične povezave povezana s plastjo istega nivoja na drugi strani

povezave. Tako si plasti prenašata logična sporočila, ki pa se v resnici pretakajo preko vseh nižjih nivojev na obeh straneh povezave. Vsaka plast lahko dejansko komunicira in si pošilja sporočila le s plastjo nad njo ali pod njo. Izjema je najnižja plast, ki komunicira preko prenosnega kanala z najnižjo plastjo druge naprave.

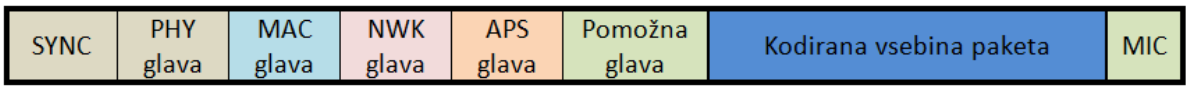
Spodnji dve plasti sta fizična plast (ang. physical layer, PHY) in plast MAC (ang. medium access control). Ti dve plasti sta definirani v standardu IEEE 802.15.4.

Naslednji dve plasti, definirani s strani ZigbBee Alliance, sta mrežna plast (ang. network layer, NWK) in aplikacijska plast, ki je sestavljena iz aplikacijske podporne podplasti (ang. application layer, APS), objektne podplasti (ang. ZDO) in aplikacijskega okvirja. Aplikacijski okvir vsebuje uporabniško definirane objekte in funkcije ter si deli varnostne nastavitve s podplastjo APS in podplastjo ZDO [4].



Slika 3: Organizacija sklada ZigBee [6]

Na najvišji plasti, aplikacijskem okviru v plasti APL, se pošilja zahteve plasti APL na drugi strani povezave, ki lahko odgovori ali pa le izvede predvideno funkcijo. To je logična povezava. V resnici začetno sporočilo potuje po skladu, kjer vsaka plast k obstoječemu sporočilu doda nekaj svojih podatkov potrebnih za uspešen prenos preko komunikacijskega kanala. Pravimo, da se sporočilo v vsaki plasti enkapsulira, na sprejemni strani pa dekapsulira. V sliki 4 je prikazan primer končno enkapsuliranega sporočila, ki je pripravljen za pošiljanje preko prenosnega medija.



Slika 4: Primer paketa, ki se pošlje preko prenosnega kanala z zaščiteno vsebino

2.2.1. Aplikacijska plast

Kot smo že povedali je ta plast sestavljena iz plasti ZDO, plasti APS in objektov, določenih s strani programerja ali proizvajalca. Povedano drugače, v tej plasti se nahaja aplikacija, ki teče na procesorju, operacijskem sistemu oziroma na napravi.

Aplikacijska podporna podplast (APS) deluje kot vmesnik med omrežno plastjo (NWK) in ostalo aplikacijsko plastjo (APL). Storitve, ki jih plast APS ponuja podplasti ZDO in uporabniško definiranim aplikacijskim objektom v aplikacijskem okvirju, so prenos podatkov med dvema ali več aplikacijskimi entitetama v istem omrežju, varnostne storitve in vezava naprav.

V **aplikacijskem okvirju** so objekti uporabniških aplikacij, shranjeni na napravi ZigBee. Definiramo lahko 240 različnih objektov z identifikacijskimi številkami od 1 do 240. Kot je razvidno iz Slika 3, poleg povezav vsakega od objektov iz aplikacijskega okvirja obstaja tudi povezava s podplastjo ZDO , ki ima identifikacijsko številko 0. Dodatna podatkovna povezava z identifikacijsko številko 255 pa je funkcija razpošiljanja podatkov vsem aplikacijskim objektom (ang. broadcast).
Znotraj aplikacijskega okvirja obstajajo tudi tako imenovani aplikacijski profili. To so dogovori za formate sporočil in procesiranje zahtev, ki omogočajo uporabniku ustvarjanje interoperabilnih, distribuiranih aplikacijskih entitet, ki živijo v ločenih napravah. Aplikacijski profili omogočajo aplikacijam pošiljanje ukazov, zahtevanje podatkov in procesiranje ukazov ter zahtev.
Primer takšnega profila je recimo profil domača avtomatizacija. Profil dopušča pošiljanje kontrolnih sporočil med množicami naprav. Naprave v takšnem profilu bi pošiljale ukaze kot so prižgi ali ugasni luč, preberi temperaturo senzorja, pošlji opozorilo, ker je detektor gibanja zaznal spremembo in podobne.
Ena sama ZigBee naprava lahko podpira več različnih profilov. To dosežemo z hierarhičnim naslavljanjem znotraj naprave na naslednji način:

- celotna naprava je naslovljena z enoličnim IEEE in NWK naslovom,
- vstopne točke (ang. endpoints) so 8 bitna števila, ki določajo različne aplikacije (oziroma aplikacijske objekte znotraj aplikacijskega okvirja) na isti napravi. Kot smo že omenili, je različnih aplikacij lahko 240, z nekaj rezerviranimi identifikacijskimi številkami.

Podplast ZDO predstavljajo glavni funkcionalni razred preko katerega so povezani aplikacijski objekti, profil naprave in APS. Naloga te podplasti je zadovoljevanje pogostih zahtev vseh aplikacij, ki delujejo na tem skladu. Podplast ZDO je odgovorna za:

- inicializacijo podplasti APS, omrežne plasti in varnostnega servisa,
- zbiranje informacij o konfiguraciji omrežja, odkrivanje naprav, upravljanje z varnostjo, omrežjem ter povezovanjem naprav.

Le ta predstavlja vmesnik aplikacijskim objektom za nadzor in nastavljanje naprave ter njenih mrežnih funkcionalnosti, kot so odkrivanje in povezovanje z drugimi napravami, odkrivanje servisov na napravi,…

Odkrivanje sosednjih naprav lahko poteka na dva načina. Prvi je preko naslova IEEE, pri katerem se pošlje sporočilo (ang. unicast) točno določeni napravi. Pri temu načinu moramo seveda poznati naslov naprave, kateri pošiljamo zahtevo. Drug način pa je naslavljanje preko plasti NWK, ki se razpošlje na vse možne naslove. V obeh primerih naslovljena naprava odgovori s svojim naslovom. Če gre za koordinator ali usmerjevalnik, ta pošlje tudi naslove vseh naprav, ki so priključene nanj.

Standardni podatki ki jih dobimo ko sosedi odgovorijo na našo poizvedbo o sosedih so 64 bitni stalni IEEE naslov, mrežni (NWK) naslov, tip naprave (koordinator, usmerjevalnik, končna naprava), podatek, ali ima naprava v času pripravljenosti prižgan sprejemnik (signal rxOnWhenIdle), sorodnost (je naprava naš sin, oče, neavtenticiran sin,…), podatek ali je prejšnji prenos do te naprave uspel ter približna kvaliteta povezave med napravama.

Odkrivanje servisov naprav je poizvedovanje po njihovih zmožnostih. Takšno poizvedbo se ponavadi pošlje točno določeni napravi, ki odgovori s sporočilom, ki vsebuje podatke o servisih, ki jih izvaja. Možno pa je poslati zahtevo po predstavitvi servisov tudi vsem napravam v omrežju (ang. broadcast). Zaradi količine podatkov v takšnih prenosih usmerjevalniki in koordinatorji tukaj pošljejo samo svoje podatke, ne pa tudi podatkov vseh svojih sinov, kot smo opisali pri odkrivanju sosednjih naprav. Včasih se takšne informacije

tudi shranijo v nekem začasnem pomnilniku na centralni lokaciji omrežja, ali pa vsak usmerjevalnik vsebuje kakšne zmožnosti imajo končne naprave priključene nanj.

2.2.2. Mrežna plast

Mrežna plast ali plast NWK je potrebna, da po standardu IEEE 802.15.4 zagotovimo pravilno delovanje plasti MAC in da zagotovimo ustrezni vmesnik za aplikacijsko plast.

Za povezavo z aplikacijsko plastjo plast NWK vsebuje dve glavni storitvi, to sta podatkovna entiteta (ang. Network Layer Data Entity, NLDE) in upravljalna entiteta (ang. Network Layer Management Entity, NLME).

Podatkovna entiteta pomaga aplikacijam prenašati podatkovne enote (ang. application protocol data unit, APDU) med več napravami v istem omrežju. To pomeni da mora podatkovna entiteta znati usmerjati pakete skozi omrežje glede na podano topologijo. Pakete mora usmeriti do naprave, ki je končna, ali pa do naprave, ki je naslednja stopnja v usmerjanju paketa do končne lokacije v komunikacijski verigi. Obenem mora zagotavljati tudi varnost prenesenih paketov.

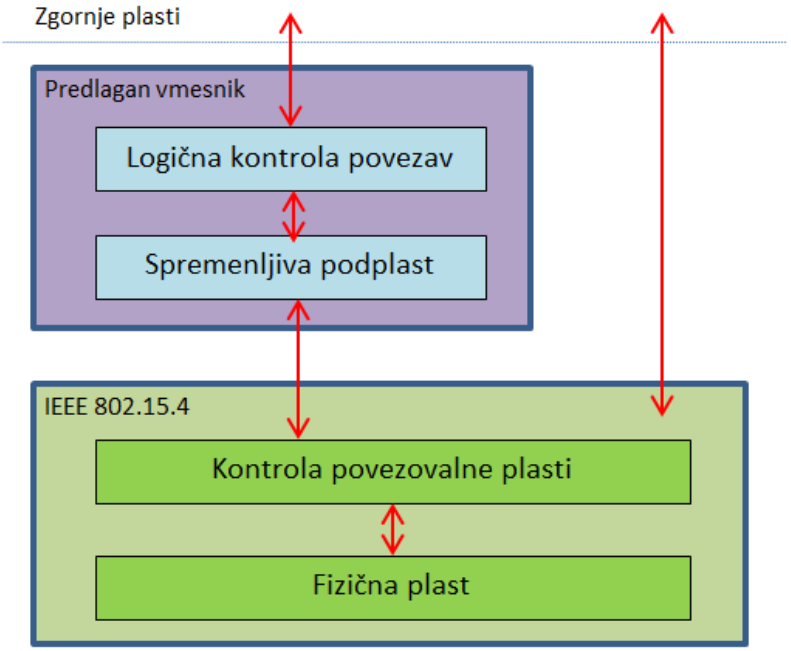
Upravljalna entiteta je odgovorna za to, da omogoči interakcijo aplikaciji s skladom. Ponuja storitve kot so:

- konfiguriranje nove naprave s katerimi sklad prilagodimo za potrebe končne aplikacije; takšna lastnost je recimo zagon operacij v funkciji koordinatorja ali pa pridružitev k že obstoječemu omrežju kot končna naprava,
- vzpostavitev novega omrežja,
- pridružitev obstoječemu omrežju ali zapustitev omrežja, ki jo po eni strani izvede končna naprava ali pa koodrinator in usmerjevalnik,
- naslavljanje s 16 bitnim naslovom, ki ga napravam ob priključitvi lahko dodeli koordinator ali usmerjevalniki,
- odkrivanje sosedov,
- kontrola sprejemanja signala (prižiganje in ugašanje sprejemnika) in sinhronizacija na plasti MAC,
- odkrivanje učinkovitih poti skozi omrežno topologijo in njihovo pomnjenje.

2.2.3. Plast MAC in plast PHY

Na sliki 5 sta prikazani obe plasti. Definirani sta v standardu IEEE 802.15.4, na katerem sloni ZigBee. Plast MAC skrbi predvsem za prenos podatkovnih okvirjev preko fizičnega kanala, detekcijo napak in zagotavljanje časovnih rež (ang. time slots). Protokol ZigBee dopušča, da se iz standarda izpusti nekatere funkcionalnosti, ki niso nujno potrebne za delovanje konkretnega aplikacijskega profila. Za to so se odločili, ker je protokol namenjen vgradnim sistemom, ki delujejo na majhnih procesorjih z malo prostora za programsko kodo. Plast MAC neposredno sodeluje tudi pri povezovanju naprav.

Fizična plast je opisana z frekvencami in lastnostmi oddajnika. Protokol ima dve fizični plasti, od katerih vsaka uporablja drugo frekvenčno območje. Tu se okvirji, ki nastanejo v prejšnjih stopnjah, spremenijo v bite in pošljejo po prenosnem mediju do naslednje naprave.



Slika 5: Osnovna struktura fizične in povezovalne(MAC) plasti

2.2.4. Kratka primerjava s standardom ISO/OSI

Standard ISO/OSI je, za razliko od opisanega, sestavljen iz 7 plasti [1].

1) **Fizična plast** skrbi za prenos bitov preko prenosnega medija in zagotavlja standardno aparaturno priključevanje sistemov na prenosni medij.
2) **Povezovalna plast** prenaša podatkovne okvire med dvema točkama, ki sta povezani s prenosnim medijem. Osnovna naloga te plasti je odkrivanje napak, ki se zgodijo med prenosom po fizičnem prenosnem mediju.

3) **Omrežna plast** skrbi za usmerjanje paketov skozi topologijo omrežja.
4) **Transportna plast** poskrbi za storitve, ki omogočajo prestop uporabniških informacijskih podatkovnih enot v transportni sistem in nazaj. Izvaja transport podatkov med dvema končnima računalniškima aplikacijama.
5) **Plast seje** je namenjena storitvam, ki podpirajo logično povezovanje oddaljenih procesov med seboj.
6) **Predstavitvena plast** skrbi za združljivost predstavitve podatkov v različnih računalniških okoljih in za zaščito podatkov.
7) **Aplikacijska plast** vsebuje vrsto standardnih aplikacij, za komunikacijo med napravami.

Fizična in povezovalna plast sta enaki pri obeh. Povezovalna plast modela ISO/OSI je pravzaprav plast MAC našega sklada.

Mrežna plast opisane arhitekture vsebuje naloge, ki jih v modelu ISO/OSI opravljata omrežna in transportna plast skupaj. To je predvsem usmerjanje paketov skozi omrežje ter rezanje prevelikih podatkov v več manjših in omogočanje pravilnega sprejema takšnih razrezanih paketov. Ta plast poskrbi tudi za zanesljiv prenos preko komunikacijskega kanala, kar je tudi glavna funkcija transportne plasti v ISO/OSI modelu.

V opisani arhitekturi je aplikacijska plast razdeljena na tri dele. Prvi del je dejanski vsebovalnik (ang. container) za aplikacijo, ki je enak aplikacijski plasti modela ISO/OSI. Drugi del predstavlja podplast ZDO, ki služi združljivosti aplikacije z zadnjim delom, podplastjo APS. Predstavitvena plast modela ISO/OSI skrbi za združljivost različnih binarnih tipov podatkov in kodnih strani. Podpira tudi storitve za stiskanje podatkov ter varnost. Zato je ta plast enakovredna podplasti ZDO in podplasti APS v plasti APL. Sejne plasti pa ne moremo direktno povezati s plastjo naše arhitekture, saj je njena funkcionalnost porazdeljena po celotni aplikacijski plasti.

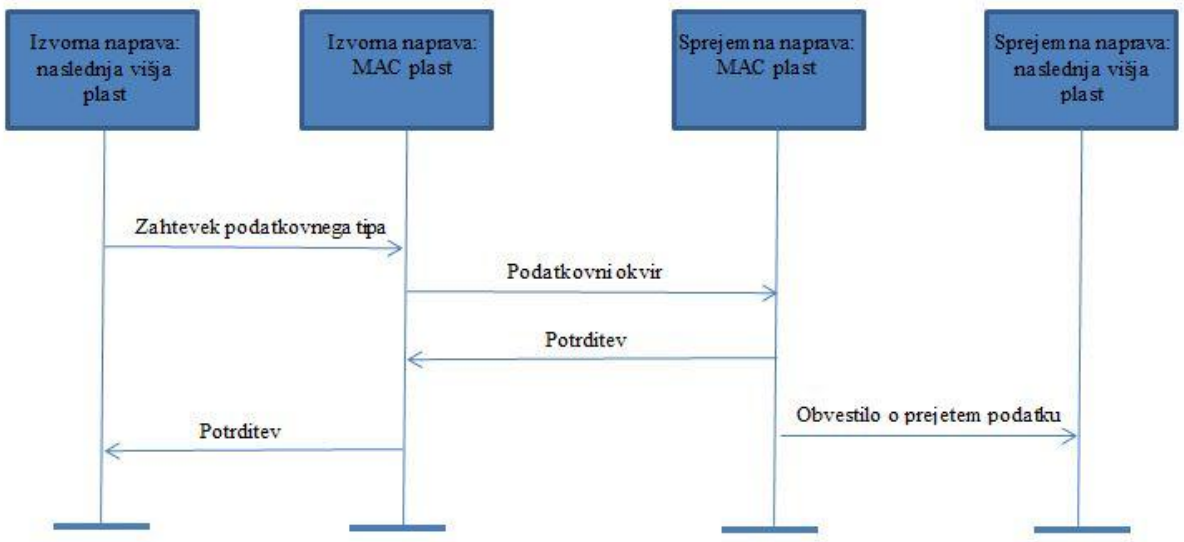
3. Osnovni principi delovanja

Kot smo omenili že v prejšnjih poglavjih komunikacija poteka s prenašanjem sporočil preko sklada do prenosnega medija in po skladu naslovne naprave nazaj do aplikacijske plasti. Implementacija konkretnega delovanja posameznih plasti je v končni fazi odvisna od posameznega razvijalca, v prejšnjem poglavju pa smo videli predlagano organizacijo, ki je kompatibilna s standardi. Primer pošiljanja sporočila iz ene naprave v omrežju na drugo napravo s potrditvijo prejema si lahko ogledamo na sliki 6.

Lahko rečemo, da si oddaljeni aplikaciji pošiljata sporočila naslednjih tipov:

- zahteva (ang. request),
- potrditev (ang. confirm) in
- in odziv (ang. indication).

Najprej izvorna naprava pošlje zahtevo. Ta potuje do naslovljene naprave, kjer se sproži odziv nanjo. Če je naslovljena naprava uspešno sprejela zahtevo, pošlje izvorni napravi sporočilo o uspehu (potrditev).



Slika 6: Enostaven prenos podatka med dvema napravama

V omrežju moramo imeti vsaj dve vrsti naprav. Prva je koordinator, ki vzpostavi in vzdržuje omrežje. Ta je sposoben prejemati ukaze preko vmesnika RS-232 in na njih reagirati. Prav tako lahko sprejema ukaze ali podatke iz vseh ostalih naprav v omrežju. Nanj se lahko

priklopijo tudi naprave izven omrežja, če zadoščajo določenim, predvsem varnostnim, pogojem.

Druga naprava je končna naprava, ki se lahko poveže s koordinatorjem. Tudi ta pozna nekaj ukazov, ki jih sprejema preko vmesnika RS-232 ali preko oddaljene naprave (koordinatorja). Če se hoče naprava povezati v omrežje, mora za dovoljenje prositi koordinatorja. Ta nato glede na pogoje omrežja napravi dovoli ali pa zavrne dostop.

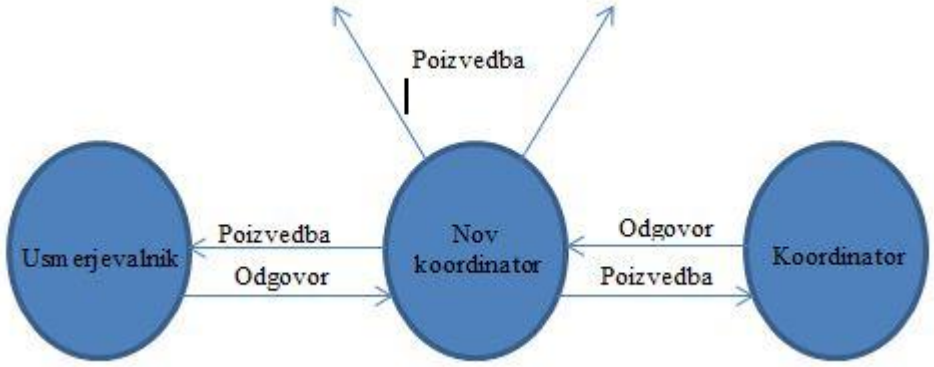
Ko se napravi povežeta (ang. association) si izmenjata podatke druga o drugi. Pomemben je predvsem podatek o naslovu obeh naprav, saj drugače ne moreta komunicirati. Pri asociaciji se lahko vzpostavi tudi varna povezava med napravama, naknadno pa lahko nastavimo in sprogramiramo dodatno varnost tudi na aplikacijskem nivoju. Šele po izmenjavi varnostnih ključev sta dve napravi povezani v pravem smislu besede in si lahko izmenjujeta podatke. Če se odločimo, da aplikaciji komunicirata direktno, lahko vse ukaze za nastavljanje naprav skrijemo pred končnim uporabnikom, kar je tudi pravilna odločitev za končen produkt.

3.1.1. Vzpostavitev omrežja

Kot smo že omenili poznamo tri tipe naprav. Končne naprave, usmerjevalnike in koordinatorje, ki so odgovorni za vzpostavitev novega omrežja. Pri tem koordinator izbere kanal in identifikacijsko oznako, ki skupaj določata omrežje. Ko se neka naprava v tako omrežje priključi, te podatke podeduje. Vsaka naprava v omrežju ima tudi dva naslova. Prvi, 16 bitni, je določen ob vstopu v omrežje. Po navadi ga določi kar koordinator ali usmerjevalnik. Koordinatorjev 16 bitni naslov je vedno 0. Drugi naslov, dolg 64 bitov, je enoličen za vsako napravo in se ne spreminja.

Da lahko koordinator vzpostavi omrežje, mora najprej pregledati različne radijske kanale in na njih določiti aktivnosti (ang. energy scan). Naprava na kanalih na tak način določi energijske nivoje in tiste z veliko energije izloči iz seznama potencialnih kanalov, na katerih se lahko vzpostavi omrežje. Na kanalih z visoko energijo, najverjetneje deluje neka druga naprava, morda drug koordinator, ki bi pravilno delovanje omrežja motila ali pa onemogočala. Po energijskem pregledu koordinator opravi še aktiven pregled tihih kanalov za že obstoječimi omrežji (ang. PAN scan, active scan ali beacon scan). To stori tako, da pošlje poizvedbo na vse možne naslove (ang. one-hop beacon broadcast), kot je prikazano na sliki 7. Ta poizvedba je lahko tudi zahtevek za povezavo v omrežje. Vsi okoliški koordinatorji ali usmerjevalniki mu bodo poslali povratno sporočilo. V povratnem sporočilu najdemo

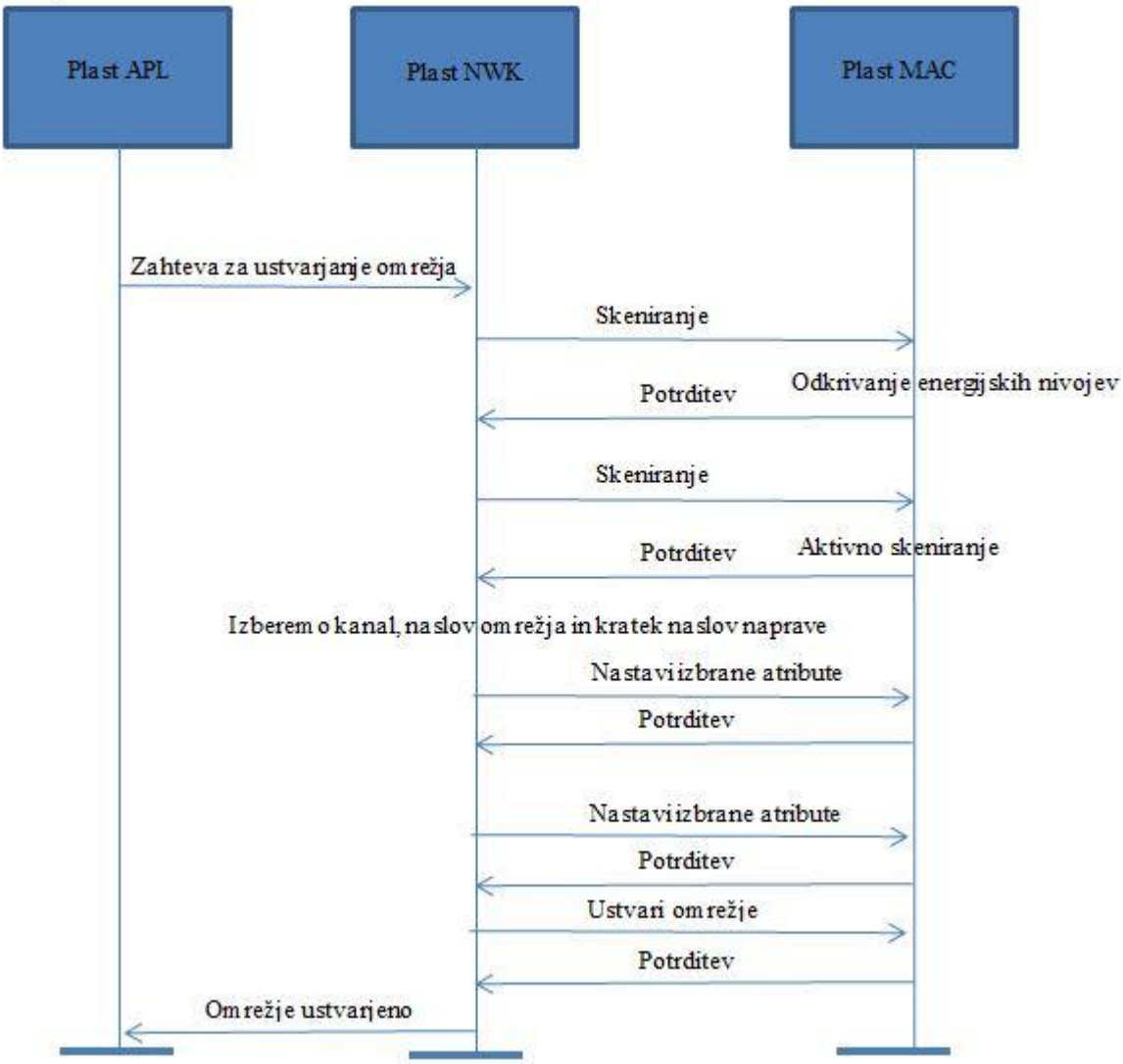
informacije o obstoječem omrežju, kot sta identifikacijska številka omrežja in ali omrežje sprejema nove povezave ali ne.



Slika 7: Po končanem energijskem skeniranju koordinator pošlje poizvedbo po svojih sosedih, da se izogne duplikaciji identifikacijske številke z drugim omrežjem

Po opravljenem energijskem in aktivnem pregledovanju koordinator pregleda zbrane podatke in izbere neuporabljeno identifikacijsko številko (PAN ID) ter kanal za novo ustvarjeno omrežje. Sedaj lahko koordinator dovoli povezavo usmerjevalnikov ali končnih naprav v pravkar ustvarjeno omrežje z vsemi osnovnimi podatki.

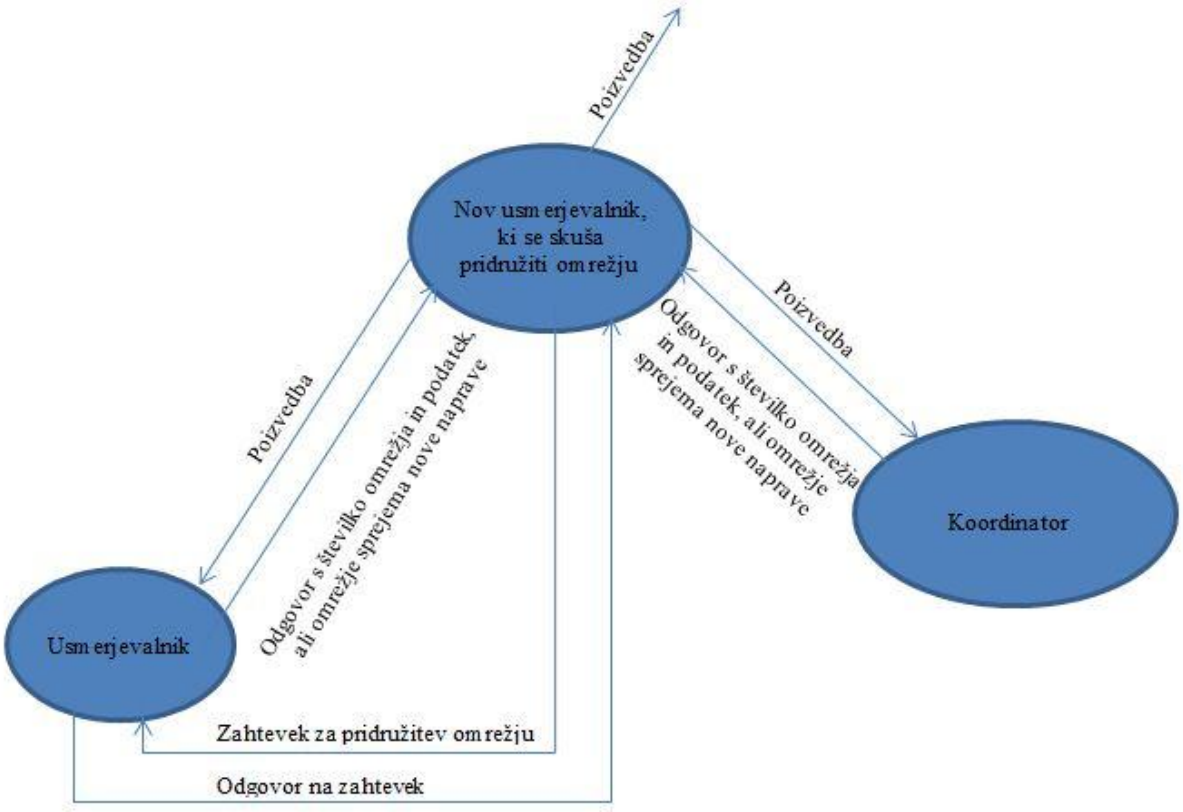
Vsak od opisanih korakov je v končni fazi sporočilo v različnih plasteh sklada komunikacijskega protokola. Opisani koraki so za lažjo predstavo prikazani na sliki 8.



Slika 8: Potek sporočil po skladu med ustvarjanjem novega omrežja

3.1.2. Pridruževanje obstoječemu omrežju

Usmerjevalnik ali končna naprava, ki se priključujeta v obstoječe omrežje, morata le-to najprej odkriti. Za to mora naprava najprej zagnati aktivno pregledovanje, na podoben način kot to počne koordinator, da iz obstoječih koordinatorjev in usmerjevalnikov dobi informacije o aktivnih omrežjih. Po pregledu podatkov naprava izbere v katero omrežje se bo vključila. Če to omrežje dovoljuje vključevanje novih naprav, lahko naprava pošlje zahtevo za povezovanje (ang. associatinon request). Po navadi po tem koraku sledi še varnostna avtentikacija vsaj na enem nivoju. Vse skupaj si lahko predstavljamo kot avtomat s tremi stanji in sporočili kot pogoji za prehode med njimi (slika 9).

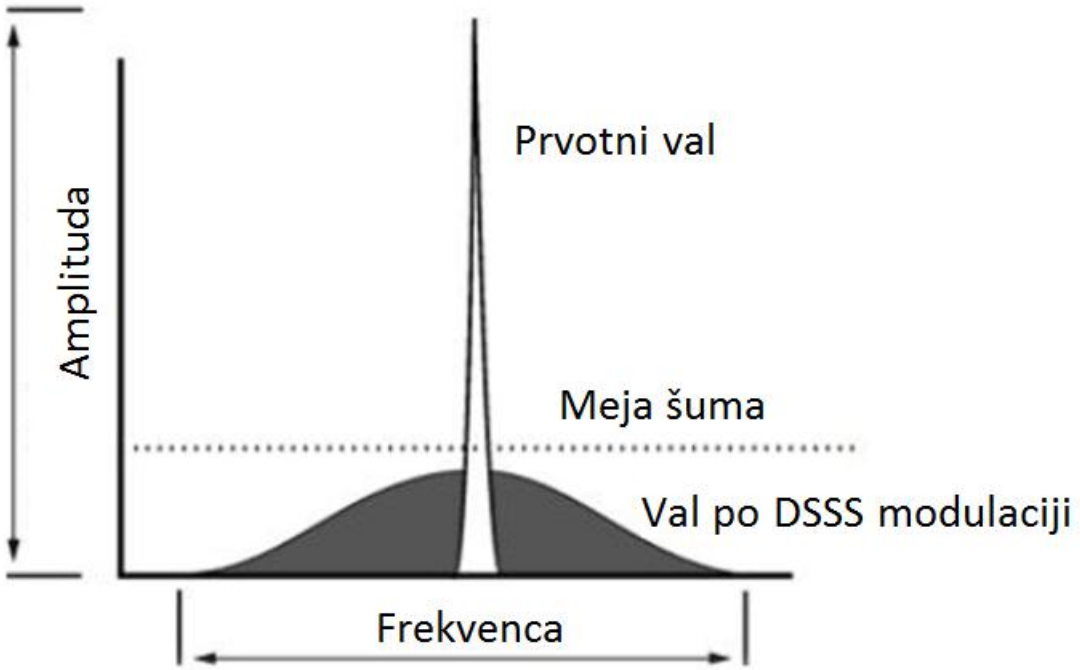


Slika 9: Pridruževanje obstoječemu omrežju

3.1.3. Zanesljivost dostavljanja paketov preko omrežja

Najprej naj nekaj besed povemo o načinu delovanja standarda 802.15.4 in njegovih prednosti pri zanesljivem prenosu podatkov. Standard predvideva uporabo podatkovne modulacije DSSS (ang. Direct Sequence Spread Spectrum) informacij še predenj se posredujejo na fizično plast v komunikacijskem skladu [2]. Praktično to pomeni, da se vsak poslani bit informacije modulira v štiri različne signale. S tem dosežemo nižjo spektralno moč, vendar izgubimo na pasovni širini prenosnega medija, oziroma na hitrosti prenosa.

Učinek izbrane modulacije si lahko ogledamo na sliki 10.



Slika 10: Modulacija signala DSSS

Ta trik ne povzroči veliko interference v frekvenčnih pasovih, poleg tega pa je sprejemniku lažje sprejeti modulirano informacijo, saj se razmerje med signalom in šumom (ang. Signal To Noise, STN) zmanjša. Seveda je tip izbrane modulacije odvisen od razpoložljive strojne opreme. Prednosti modulacije DSSS pred ostalimi tipi so predvsem:

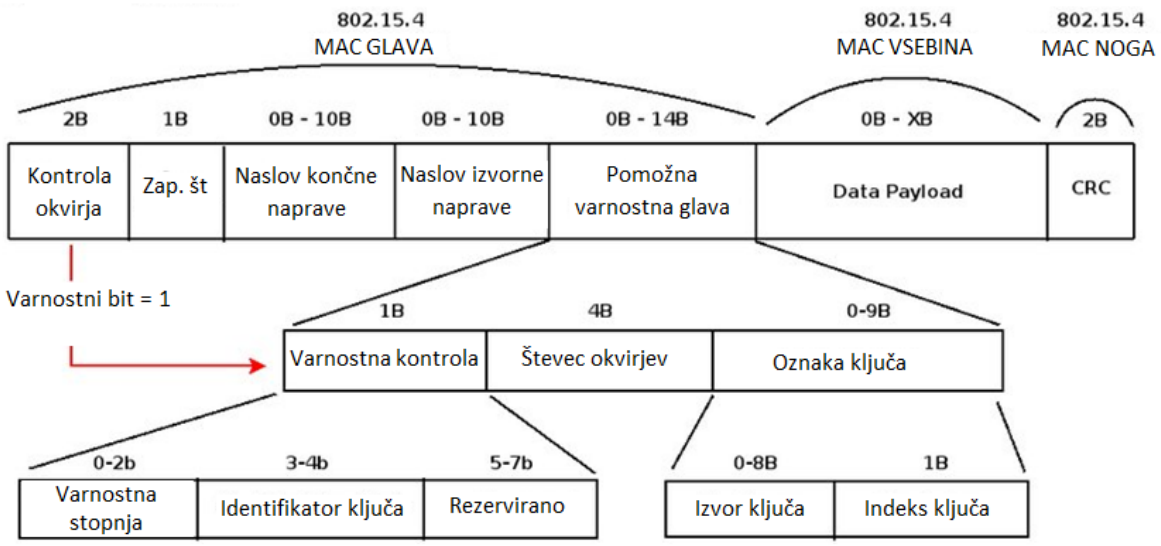
- odpornost proti namernemu in nenamernemu motenju signala,
- deljenje istega kanala med več uporabniki in
- zmanjšano razmerje STN.

Standard uporablja tudi dva načina izogibanja trkom v omrežju. To sta CSMA/CA (ang. Carrier Sense Multiple Access Collision Avoidance) in GTS (ang. Guarantee Time Slots). Ta dva protokola preprečujeta, da bi več naprav v omrežju začelo oddajati istočasno. Delujeta dokaj preprosto. Pri CSMA-CA naprava »posluša« medij pred oddajanjem. Če zazna energijski nivo, ki je večji od določene meje počaka naključno časa in poskusi znova. Pri drugem načinu si sistem pomaga s centralno napravo - koordinatorjem. Naprava pošlje zahtevo za časovno okno koordinatorju, ta pa ji v povratnem sporočilu pove kdaj in koliko oken ji je namenil. Tako naprave točno vedo, kdaj lahko oddajajo informacije v prenosni kanal.

3.1.4. Varnost prenosa podatkov po omrežju

Kot smo omenili že v prejšnjih odstavkih je standard IEEE 802.15.4 podlaga za večino storitev, ki jih uporablja na njem sloneč protokol ZigBee. Standard narekuje šifrirno metodo, ki se uporablja v komunikaciji skozi omrežje, ne pa tudi kako upravljati s ključi in kakšne avtentikacijske politike moramo izvesti. Slednje je prepuščeno ZigBee aplikacijam.

Šifrirni algoritem, ki se uporablja v omrežjih ZigBee je AES (ang. Advanced Encryption Standard) s 128 bitno dolžino ključa. Večino šifriranja je implementirano na strojnem nivoju, saj tako ne obremenjuje procesorja in je dovolj hitra. Algoritem se uporablja tako za šifriranje podatkov, ki se pošiljajo, kot tudi za preverjanje integritete in pravilnosti poslanih paketov (ang. Data Integrity Check). To dosežemo s posebnim ključem MIC (ang. Message Integrity Code), ki ga pripremo paketu. Ključ zagotavlja integriteto glave paketa in podatkov. Ključ MIC je sestavljen iz šifriranih delov paketa s pomočjo ključa omrežja. Če prejmemo sporočilo iz neznane naprave, takoj vidimo, da se ključ MIC, generiran za poslano sporočilo, ne ujema s tistim, ki bi ga dobili, če bi sporočilo kodirali z našim skrivnim ključem. Takšno sporočilo zato lahko preprosto izpustimo. Primer opisanega okvirja si lahko ogledamo na sliki 11.



Slika 11: Varnost v IEEE 802.15.4 MAC okvirju

V specifikacijah ZigBee lahko preberemo tudi, da poleg varnosti na plasti MAC obstajata še dodatni dve varnostni plasti na omrežnem in aplikacijskem nivoju. Obe seveda nadgrajujeta strojno kriptiranje AES. V ta nemen obstajajo trije različni ključi [3].

- **Glavni ključ (ang. Master Key)**, ki je fiksno nameščen v vsaki napravi, uporabimo za varno izmenjavo povezavnih ključev med proceduro izmenjave le teh med napravami.
- **Povezavni ključi (ang. Link Key)** so unikatni za vsak par naprav. Nadzoruje jih aplikacijski plast, namenjeni pa so kriptiranju informacij med izbranima napravama. Ker za to potrebujemo veliko več spominskega prostora v vsaki napravi, se uporaba teh ključev odsvetuje.
- **Mrežni ključ (ang. Network Key)** je unikaten 128 bitni ključ, ki ga poznajo vse naprave v omrežju. Ustvari ga centralni upravljalnik (ang. Trust Center) in ga obnavlja v določenih intervalih. Naprava se brez tega ključa ne more povezati v omrežje. Centralni upravljalnik je lahko posebna naprava, v večini primerov pa to vlogo prevzame koordinator. Tako je odgovoren za validiranje in avtentikacijo novih naprav, ki bi se rade vključile v omrežje, in za distribucijo novih mrežnih ključev zaupanja vrednim napravam v omrežju. Nov ključ, ki se razpošlje napravam, se kodira s starim mrežnim ključem, tako da ni nevarnosti napada.

3.1.5. Načina delovanja beacon in non beacon

Če je naprava nastavljena za delovanje v načinu **non beacon,** bo za komunikacijo uporabljala protokol CSMA/CA (ang. Carrier Sense Multiple Access With Collision Avoidance) brez žetonov. V takšnem omrežju so usmerjevalniki vedno aktivni. To pomeni da potrebujejo več energije in v praksi ne delujejo na baterijo. Ker nikoli ne spijo, lahko vedno sprejemajo in oddajajo podatke. Če uporabimo primer iz prakse, je takšna naprava lahko brezžično stikalo za luč. Naprava na luči je priključena direktno na napajanje iz omrežja. Kadarkoli dobi signal za vklop ali izklop ga obdela in v trenutku izvede. Na steni pa imamo lahko stikalo, ki se ob preklopu zbudi. Takrat pošlje ukaz luči, počaka na potrditev in znova zaspi. Ker bo stikalo večino časa v spanju bo porabilo zelo malo energije in je lahko priključeno na baterijsko napajanje. V takšnem omrežju je naprava na luči usmerjevalnik, stikalo na steni pa končna naprava.

Če naprave v načinu non beacon nikoli ne spijo, je v načinu **beacon** ravno obratno. Usmerjevalniki periodično pošiljajo signal, ki zbuja končne naprave. Ko signal ni več aktiven, naprave znova zaspijo, pred tem pa opravijo svoje delo. Ker je zbujanje usklajeno, se takrat lahko brez problemov pošiljajo podatki. Interval pošiljanja signalov pa je v končni fazi

odvisen od želene hitrosti prenosa podatkov po omrežju. Prednost načina beacon je občutno manjša poraba energije na vseh napravah v omrežju. Seveda pa naprave v načinu beacon ne komunicirajo po protokolu CSMA/CA in ne potrjujejo sprejetih sporočil.

3.1.6. Avtomat stanj

Naprava ZigBee je v osnovi avtomat, ki deluje v več stanjih. Osnovna stanja naprav so stanje mirovanja, sprejemanje podatkov, pošiljanje podatkov, speče stanje in stanje za obdelovanje ukazov.

Če naprava ne sprejema niti ne oddaja podatkov, je v stanju mirovanja. Od tu se lahko neposredno premakne v vsa ostala stanja.

V speče stanje preide naprava takrat, ko je pravilno nastavljena in so izpolnjeni pogoji za stanje mirovanja. Možnih nastavitev je več. Spanje lahko prekinemo s signalom na vhodu naprave (ang. interrupt signal), ali pa nastavimo ciklično spanje, pri katerem naprava v določenih časovnih intervalih preverja, ali so na voljo novi podatki.

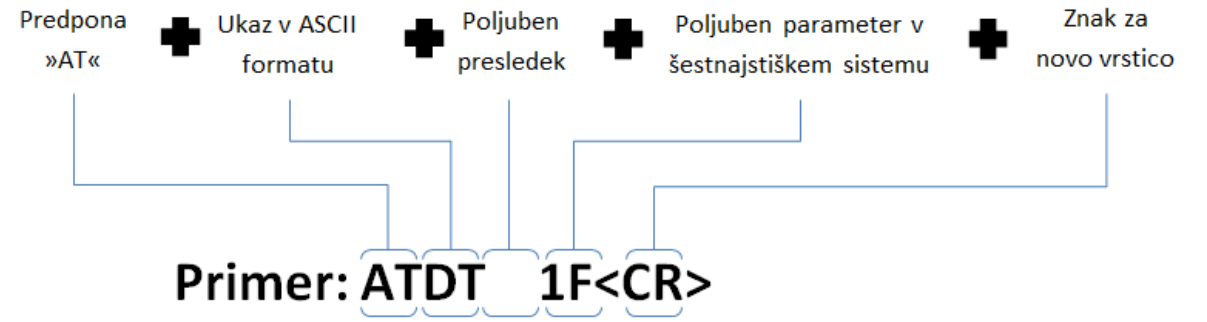
Iz stanja mirovanja lahko neposredno preidemo tudi v stanje za obdelovanje ukazov. V tem načinu napravi določamo lastnosti in ji nastavljamo parametre.

3.1.7. Opis ukazov

Moduli, ki smo jih izbrali za izgradnjo naprave za brezžično serijsko komunikacijo, lahko sprejemajo dva tipa ukazov. To so tako imenovani ukazi AT in ukazi API [5].

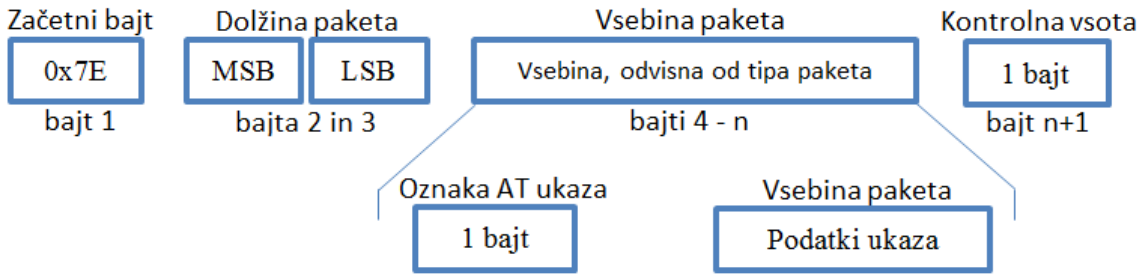
Ukaze AT lahko modulom, na katere smo serijsko povezani, vpišemo kar preko ukazne vrstice. Uporabljajo se za nastavljanje vseh pomembnih lastnosti modulov kot so ime, naslov, status priključkov in še kaj.

Z ukazi AT tako lahko izvedemo osnovno nastavljanje modulov. Slabost takšnega načina nastavljanja je, da smo omejeni na neposredno fizično serijsko povezavo z modulom. Ukazov je seveda več vrst. Prepoznamo jih po predponi AT, ki ji sledi identifikacija ukaza in parametri, če jih ukaz ima. Sintakso ukazov AT si lahko ogledamo na sliki 12.



Slika 12: Sintaksa AT ukazov

Ker z ukazi AT ne moremo vplivati na konfiguracijo oddaljenih modulov, imamo na voljo še **ukaze API** (ang. Application Programming Interface). Ukazi API so zasnovani v obliki paketkov, prikazanih na sliki 13. V grobem so sestavljeni iz glave, ukaza AT s podatki in repa.



Slika 13: Sintaksa API ukaza

Ukazi API so namenjeni prenosu ukazov AT do oddaljenih naprav. Poznamo jih več vrst. Z njimi lahko pošiljamo podatke na poljubno napravo v omrežju, lahko poizvedujemo po senzorskih vrednosti na oddaljenih napravah, ali pa module nastavljamo.

Primer: Spodnji ukaz API naredi prenos, ki modulu z naslovom `0x0013A20040014011` pošlje niz znakov `TxData0A`.

```
0x7E 0x00 0x16 0x10 0x01 0x00 0x13 0xA2 0x00 0x40 0x0A 0x01 0x27 0xFF 0xFE
0x00 0x00 0x54 0x78 0x44 0x61 0x74 0x61 0x30 0x41 0x13
```

Pri tem: **0x7E** označuje začetek paketa,
0x0016 določata dolžino paketa (22 bajtov brez preverjanja parnosti),

0x10 je tip okvirja API za prenos podatkov (ukaz ZigBee Transmit Request),
0x01 je oznaka okvira (Frame ID), nastavljena na neničelno vrednost,
0x0013A200400A0127 je 64-biti naslov modula, ki mu pošiljamo podatke,
0xFFFE je 16-bitni naslov modula, ki mu pošiljamo podatke,
0x00 je doseg naprav, ki jih doseže ukaz broadcast (ang. broadcast radius),
0x00 je namenjen za dodatne nastavitve,
0x5478446174613041 predstavlja podatke (niz TxData0A), ki jih prenašamo,
0x13 je kontrolna vsota za preverjanje sodosti.

Kontrolno vsoto izračunamo tako, da seštejemo vse bajte od vključno tipa okvirja API do konca sporočila in jih odštejemo od vrednosti 0xFF. V navedenem primeru torej:

```
0xFF −  (0x10 + 0x01 + 0x00 + 0x13 + 0xA2 + 0x00 + 0x40 + 0x0A + 0x01 +
0x27 + 0xFF + 0xFE + 0x00 + 0x00 + 0x54 + 0x78 + 0x44 + 0x61 + 0x74 + 0x61
+ 0x30 + 0x41).
```

3.2. Serijska komunikacija po protokolu RS-232

V standardnih osebnih računalniki smo praviloma vedno našli nameščena dva serijska vmesnika RS-232, ki sta bila namenjena prav komunikaciji med napravami. Pri serijski komunikaciji se biti pošiljajo zaporedno, eden za drugim, po eni žici. Prednost serijskega prenosa je v tem, da za hkratni dvosmerni (ang. full duplex) način prenosa potrebujemo le tri žice: eno za sprejemanje podatkov (ang. receive data, RD), eno za oddajanje podatkov (ang. trasnmit data, TD) in skupno ničlo (ang. common ground, GND).

V modernih časih je komunikacijo po standardu RS-232 zamenjal prenos podatkov po standardu USB. Kljub temu vmesnike RS-232 (tabela 2) še vedno najdemo na mnogih starejših (in tudi dragih) napravah, industrijskih napravah, mrežnih stikalih, modemih, baterijah za neprekinjeno napajanje in podobno.

Tabela 2: Opis standardnega priključka RS-232

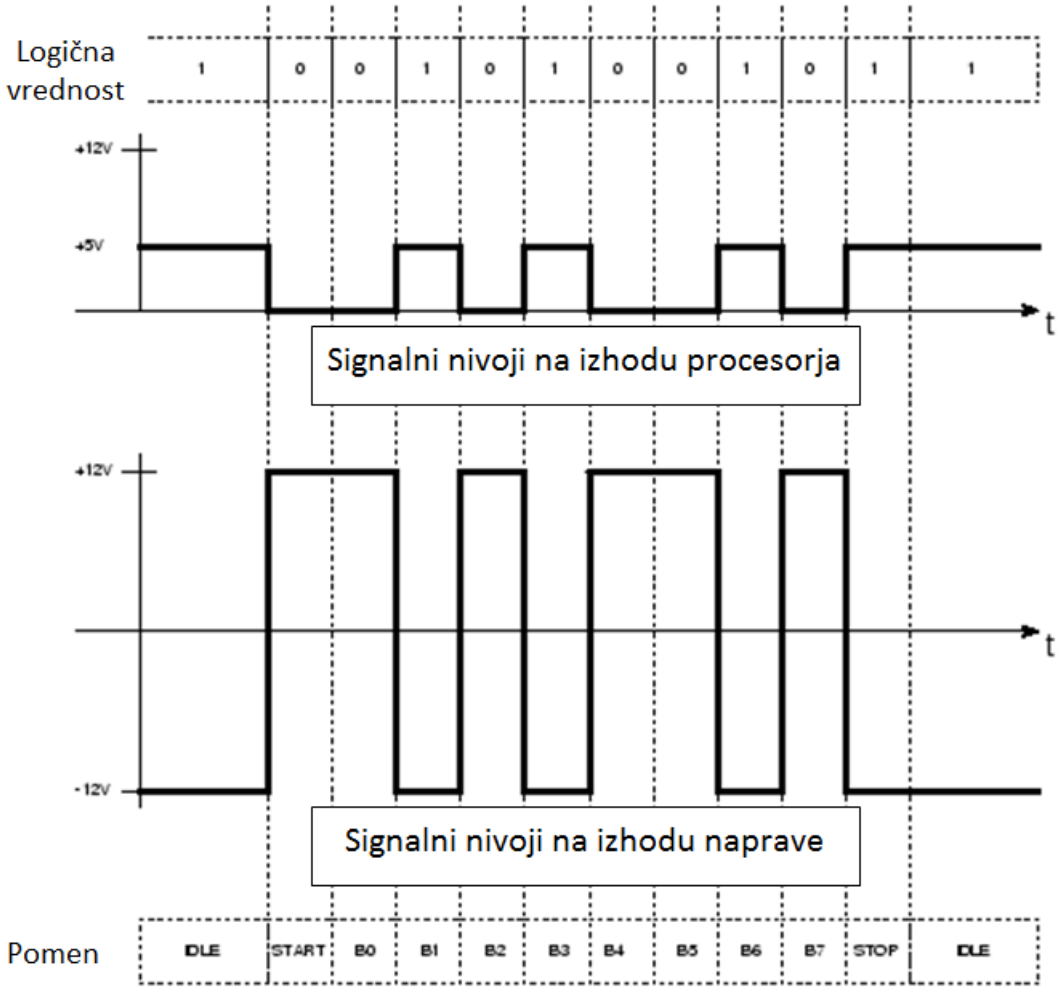
Moški RS-232
DB9 konektor

Številka pina	Smer in pomen signala
1	Carrier Detect **(CD)** – indikacija prihajajočega signala iz modema (za komunikacijsko opremo)
2	Received Data **(RD)** – vhodni podatki iz komunikacijske opreme
3	Transmitted Data **(TD)** – izhodni podatki za komunikacijsko opremo
4	Data Terminal Ready **(DTR)** – izhodni »handshake« signal
5	**Masa**
6	Data Set Ready **(DSR)** – vhodni »handshake« signal
7	Request To Send **(RTS)** – izhodni signal za kontrolo pretoka
8	Clear To Send **(CTS)** – vhodni signal za kontrolo pretoka
9	Ring Indicator **(RI)** – indikacija prihajajočega signala iz modema

Standard navaja, da so veljavni signali med ±3 in ±15 volti. Zato predvsem v integriranih vezjih uporabljamo posebne pretvornike, ki iz običajnih logičnih nivojev signale pretvorijo na nivoje, ki ustrezajo standardu RS-232. Problem lahko predstavlja tudi razlika v ničlah na oddajnem in sprejemnem delu. Če so razdalje med njima prevelike, lahko pride do razlike potenciala na masi, kar pomeni, da se masa spreminja zaradi česar pride pri komunikaciji do napak.

Ker se pošilja le po en bit na enkrat po eni žici in nimamo posebne ure, oddajni del začetek prenosa sporoči z **začetnim bitom** (start bit). Ta bit signalizira sprejemniku, naj si nastavi svojo notranjo uro tako, da bo lahko sprejel poslane bite. Zato je pomembno da sprejemni in oddajni del uporabljata enako hitrost prenosa (ang. baud rate). Začetnemu bitu sledi 8, 7, 6 ali 5 **podatkovnih bitov**. Ko so poslani vsi podatkovni biti, lahko pošljemo še dodaten **paritetni bit** (ang. parity bit). Izbiramo lahko med sodo in liho pariteto. Problem v takšnem preverjanju integritete podatkov je, kadar se v prenosu zgodi sodo število napak. Takrat se paritetna vsota izniči in napaka je spregledana. Na srečo pri dovolj kratkih razdaljah do napak praktično ne prihaja. Prenos se zaključi z **zaključnim bitom** (ang. stop bit). Ta signalizira konec prenosa in predstavlja manjšo zakasnitev pred novim pošiljanjem začetnega bita. Končni bit je nasproten začetnemu bitu, zato da ju sprejemnik zna ločiti. Primer prenosa črke »J« po

standarda RS-232 z osmimi podatkovnimi biti, brez paritete in brez kontrole pretoka si lahko ogledamo na sliki 14.



Slika 14: Primer električnih nivojev pri pošiljanju črke "J" po standardu RS-232 [7]

4. Izdelava naprave

Cilj naloge je izdelava komunikacijske naprave imenovane nadzornik, s pomočjo katere se lahko brezžično povežemo do poljubnih končnih naprav z vmesnikom RS-232. Osnovna ideja sistema je shematično predstavljena na sliki 16. Za brezžično povezavo med nadzornikom in različnimi odjemalci (računalniki) skrbijo moduli ZigBee na nadzorniku in odjemalcih. Naloga nadzornika je, da podatke, ki po fizičnih povezavah prihajajo iz naprav z vmesniki RS-232, pretvori v obliko primerno za prenos po omrežju Zigbee in jih posreduje odjemalcem in obratno. Podatke, ki jih odjemalci posredujejo preko omrežja Zigbee do nadzornika, pretvori v obliko primerno za prenos po fizičnih povezavah do končnih naprav.

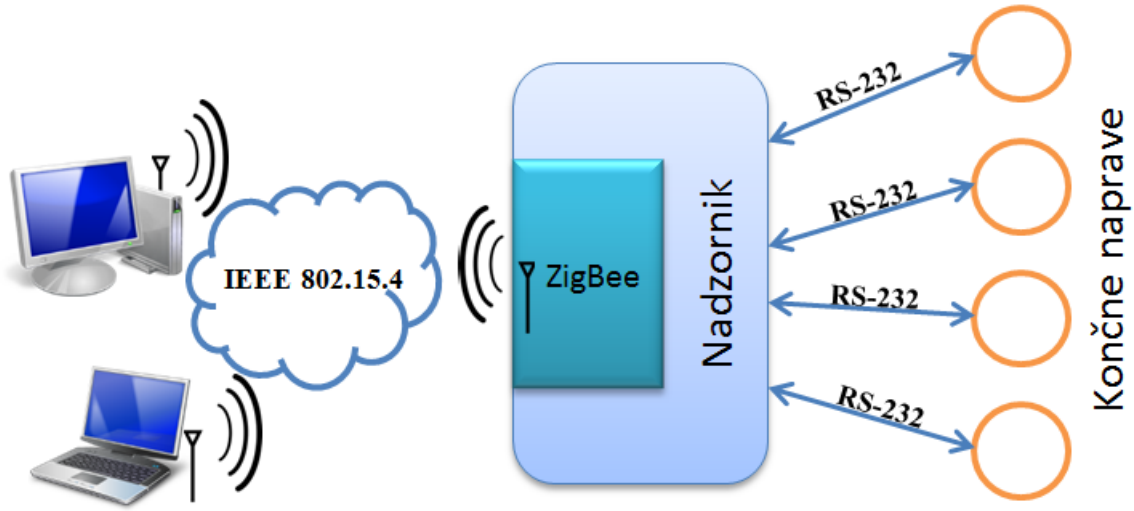
4.1. Idejna zasnova in zahteve projekta

Nadzornik je komunikacijska naprava, ki omogoča prijavljenim odjemalcem vzpostavitev povezave do neke končne naprave. Odjemalec v omrežju ZigBee vzpostavi brezžično povezavo do nadzornika, na katerega so lahko priklopljene do štiri končne naprave preko standardnih konektorjev DB-9 ali RJ45. Te končne naprave komunicirajo z nadzornikom po protokolu RS-232.

Standard ZigBee omogoča varno povezavo med odjemalci in nadzornikom, saj podpira šifriranje prenesenih podatkov in overjanje naprav v omrežju. Odjemalec lahko po vzpostavitvi varne brezžične povezave z nadzornikom vzpostavi direktno sejo s posamezno končno napravo. Za vzpostavitev povezave potrebuje poseben program, ki skrbi za osnovno komunikacijo z nadzornikom.

Naloga nadzornika je, da po eni strani vzdržuje brezžične povezave z vsemi povezanimi odjemalci, po drugi strani pa povezave RS-232 z vsemi povezanimi končnimi napravami. Skrbi za to, da je vsak odjemalec lahko povezan na poljubno napravo, ampak le na eno hkrati. Prav tako lahko do posamezne končne naprave dostopa le en odjemalec naenkrat. Prekinitveni način procesiranja podatkov in zunanji medpomnilniki omogočajo, da nadzornik deluje v večopravilnem načinu in zagotavlja nemoteno komunikacijo med odjemalci in končnimi napravami. Opisana shema je prikazana na sliki 15.

Projekt izdelave nadzornika vsebuje izdelavo vse potrebne strojne in programske opreme.



Slika 15: Logična shema sistema

4.2. Strojna oprema

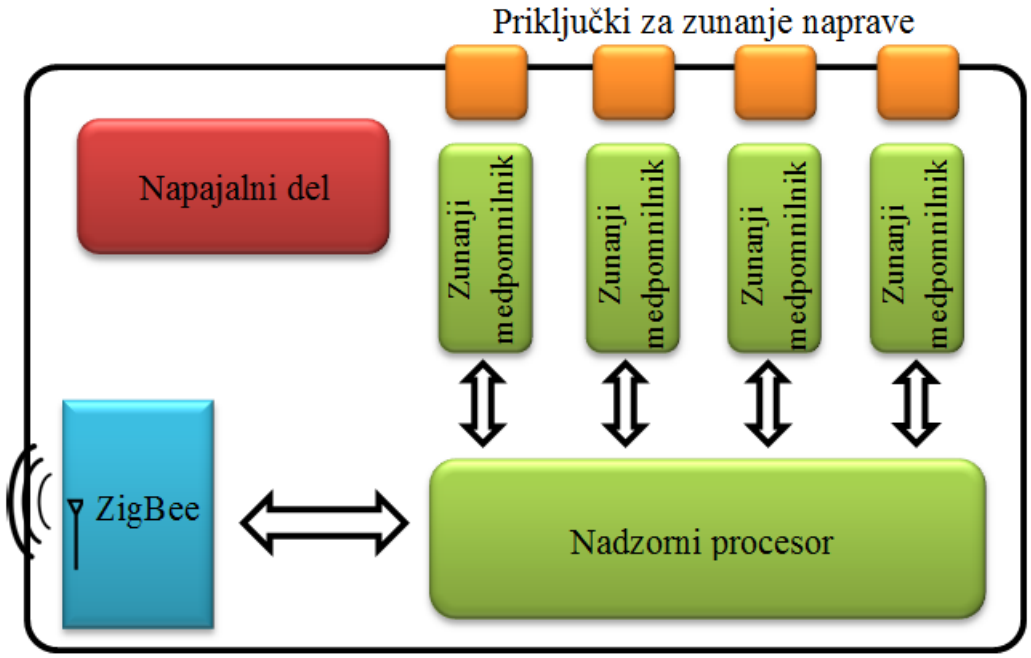
Odjemalci za komunikacijo z nadzornikom uporabljajo modul ZigBee, ki je preko vmesnika USB povezan na računalnik. Na računalniku gonilnik ustvari navidezna serijska vrata. Uporabili smo modul xBee proizvajalca Digi skupaj s podnožjem, ki pretvarja signale podane po specifikaciji USB v obliko, ki jo zahteva standard RS-232. Zato uporabnik na računalniku komunicira z modulom xBee preko navadnih komunikacijskih vrat po protokolu RS-232. Enak modul xBee smo uporabili na nadzorniku.

Nadzornik je namensko izdelano vezje, ki povezuje vse sestavne dele potrebne za delovanje sistema. Vezje je sestavljeno iz:

- napajalnega dela,
- štirih zunanjih medpomnilnikov,
- preklopnika signalov iz končnih naprav v zunanje medpomnilnike in iz nadzornega procesorja v končne naprave,
- štirih priključkov za končne naprave,
- modula ZigBee in
- priključkov za programiranje čipov.

Priključki za programiranje čipov so le dodatek za hitrejši razvoj in popravilo programske kode na procesorjih, zato jih v logično shemo na sliki 16 nismo vključili. Ker gre za prototip, je plošča narejena tako, da so vsi procesorji hitro zamenljivi. Poleg tega je večina elementov

in povezav na zgornji strani dvoplastne plošče. Na metalizirani spodnji strani plošče najdemo pravzaprav le nekaj povezav. Elementi so logično razporejeni po površini tako, kot so to najbolje dovoljevale povezave med njimi.



Slika 16: Logična shema nadzornika

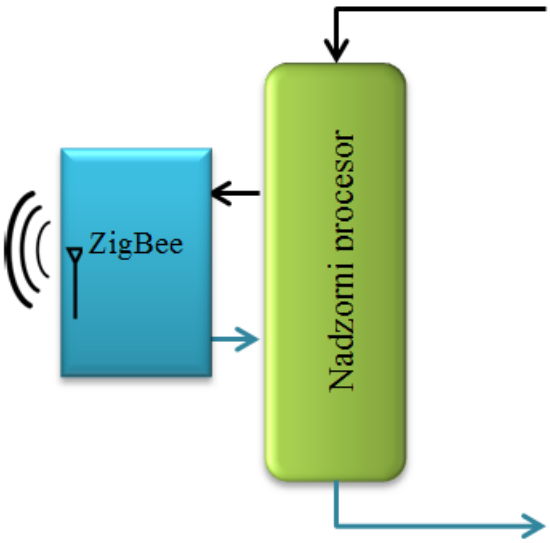
Za nadzorni procesor smo uporabili zmogljiv mikrokrmilnik AtMega2560. Ta RISC mikrokrmilnik dela s frekvenco 8 MHz, povečamo pa mu jo lahko tudi na 16 MHz. Uporabili smo dva od štirih nastavljivih vmesnikov USART (ang. Universal Asynchronous Receiver/Transmitter). Za nastavljanje preklopnikov in upravljanje signalov smo porabili 21 od skupno 86 nastavljivih vhodno izhodnih linij mikrokrmilnika. Pri tem so štiri linije namenjene komunikaciji USART, tri pa služijo za programiranje mikroprocesorja.

Za zunanje medpomnilnike smo uporabili podoben, vendar enostavnejši, mikrokrmilnik AtTiny88a. Ker nima nobenega vmesnika USART smo na njem porabili dve vhodno izhodni liniji za programsko komunikacijo USART in dve liniji za signala CTS in RTS.

4.2.1. Delovanje

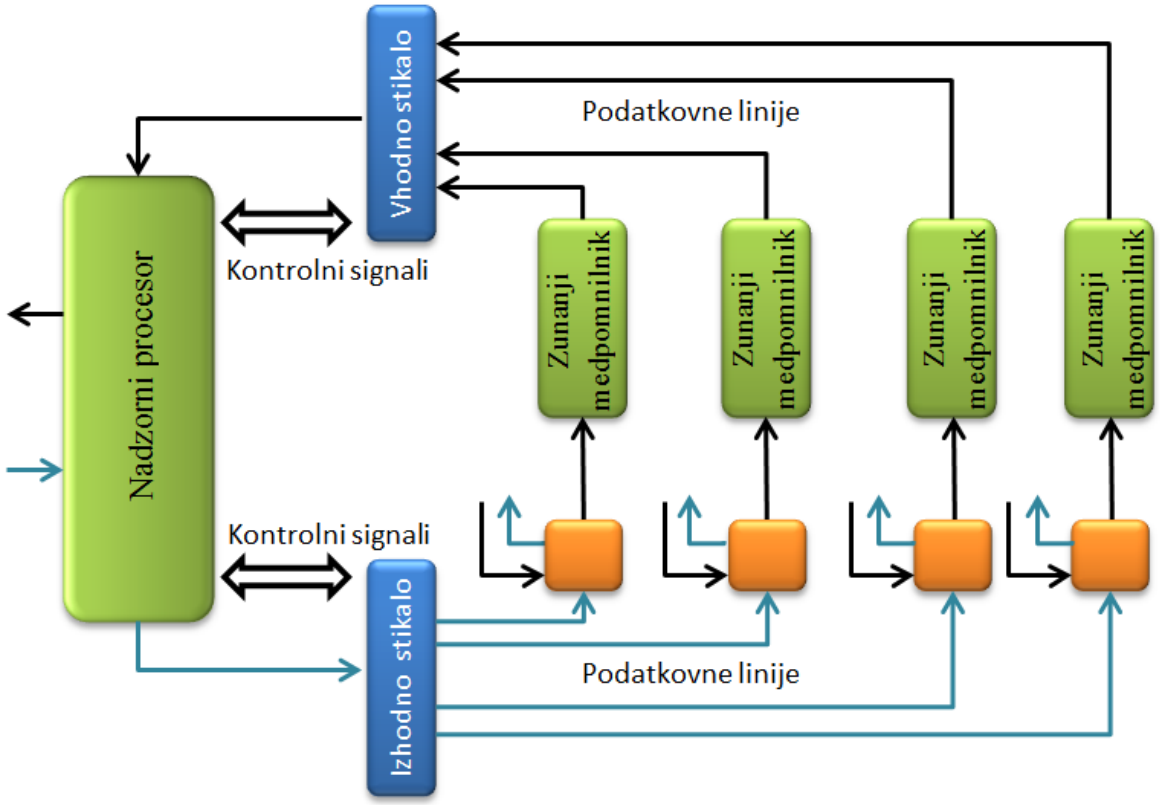
Nadzorni procesor na eni strani komunicira s končnimi napravami, na drugi pa z modulom ZigBee. Z modulom ZigBee je povezan direktno (slika 17), zato je komunikacija med njima

enostavna. Naloga nadzornega procesorja je, da pripravi ustrezne podatkovne pakete za pošiljanje skozi omrežje ZigBee in da prejete pakete ustrezno interpretira in posreduje naprej.



Slika 17:Prikaz toka podatkov med nadzornikom in modulom ZigBee

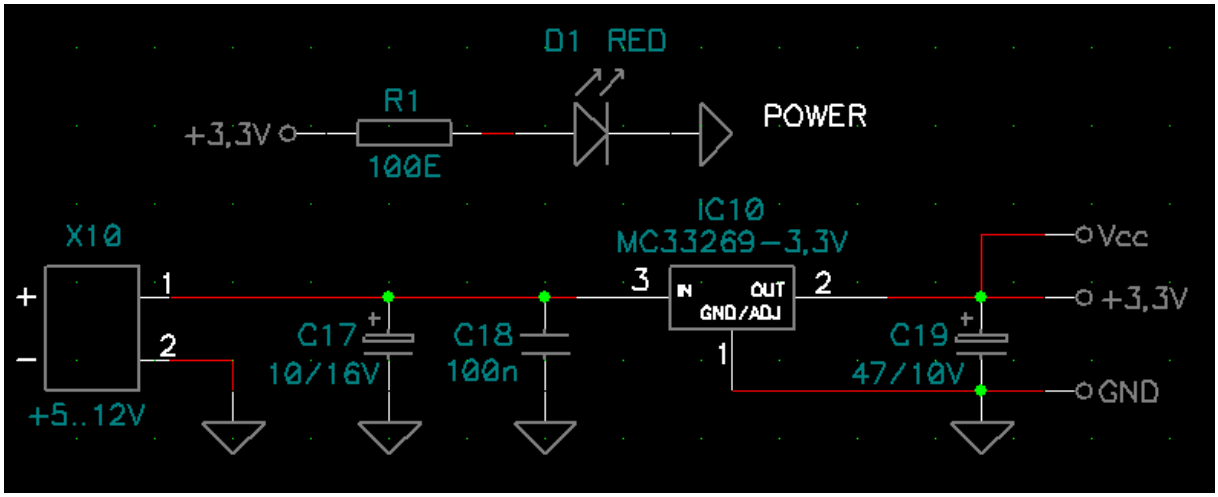
Komunikacija s končnimi napravami ni tako enostavna. Na sliki 18 so prikazane štiri končne naprave, nadzorni procesor pa ima le en vhod namenjen za komunikacijo z njimi. Zato imamo v vezju preklopnike. Vhodni preklopnik skrbi za signale, ki pridejo iz zunanje naprave preko zunanjega medpomnilnika in nadzornega procesorja do odjemalca. Zunanji pomnilnik mora te podatke hraniti dokler niso nadzorni signali nadzornega procesorja ustrezno preklopili vhodno stikalo. Nato zunanji pomnilnik podatke posreduje direktno nadzornemu procesorju, ki jih preoblikuje in pošlje modulu ZigBee. Od tu podatki pridejo do sprejemnega dela (odjemalca), ki je povezan na pošiljajočo končno napravo. Odjemalec prav tako lahko pošlje podatke v omrežje. Ko ti pridejo do nadzornega procesorja, mora ta s kontrolnimi signali najprej poskrbeti, da bo na izhodnem stikalu izbrana pravilna pot do končne naprave. V nadaljevanju nato interpretiran paket pošlje neposredno končni napravi.



Slika 18: Prikaz toka podatkov med nadzornikom in končnimi napravami

4.2.2. Napajanje

Glavni del napajalnega vezja, ki je predstavljeno na sliki 19, je napetostni regulator, ki napaja vse elemente z napetostjo 3,3V.



Slika 19: Napajalni del vezja

Glavni porabniki v vezju so procesorji. Kot je razvidno iz tabele 3 celotno vezje v nobenem trenutku ne porabi več kot 0,2 W. Pri tem je pomembno, da so v tabeli 3 prikazane le največje porabe ustreznih elementov v vezju. Ker gre v našem primeru za prenos podatkov po standardu RS-232, bodo procesorji večino časa neobremenjeni, torej bo njihova poraba še manjša. Naj samo omenimo, da modul xBee ob izključenem oddajno-sprejemnem delu porabi le 50 mW. Prav tako pa gredo lahko vse omenjene naprave v stanje spanja ob neaktivnosti, kar še občutneje zmanjša celotno porabo.

Tabela 3: Ocenjena največja poraba procesorjev v vezju pri hitrosti 8 MHz in napajanju 3,3 V

Element	Število	Poraba [mW]	Poraba skupaj [mW]
AtMega2560	1	23	23
AtTiny88a	4	8,25	33
xBee XB24-B	1	132	132
Skupaj	6	/	**188**

4.2.3. Vhodni in izhodni preklopnik

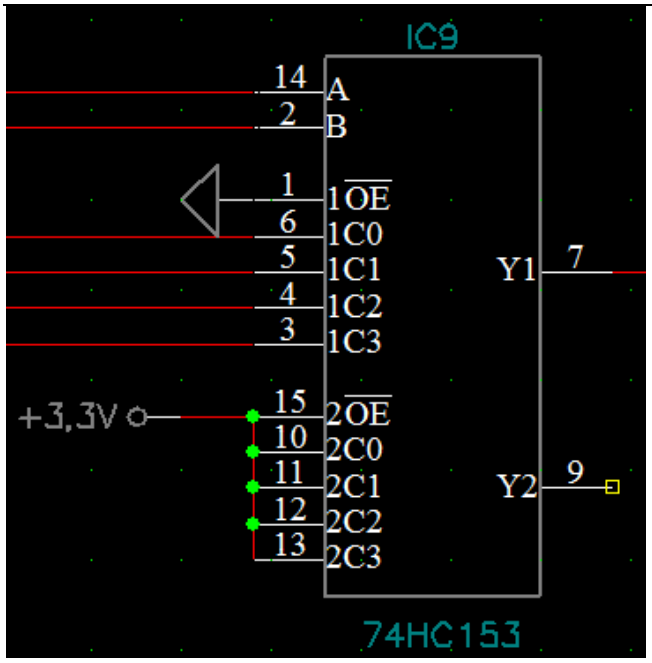
V vezju imamo dva preklopnika signala, ki skrbita za pravilno izbiro poti med odjemalcem in končno napravo. Nastavlja ju nadzorni procesor.

Iz oznake preklopnikov na slikah 20Slika 20 in 21 lahko razberemo da čipa spadata v družino integriranih vezij 74HC. Gre za hitre CMOS čipe, ki so nazaj združljivi z družino čipov 7400. Preklopnik 74HC153 je sestavljen iz dveh multipleksorjev 4/1. Njuno logično funkcija lahko zapišemo v obliki

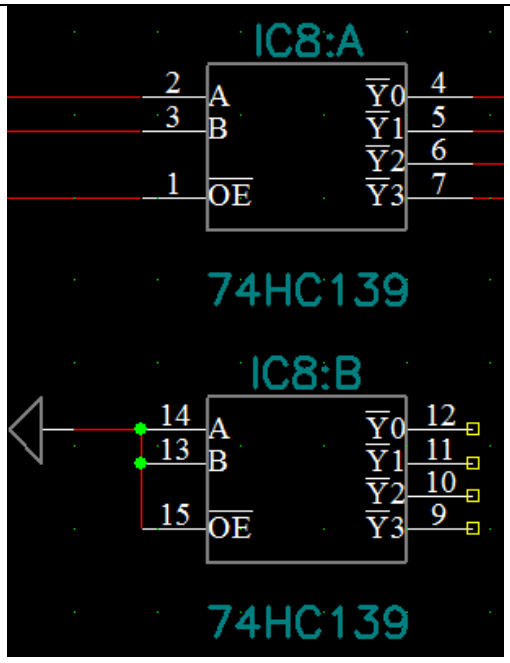
$$Y = \overline{OE} \cdot (C0 \cdot \bar{A} \cdot \bar{B} + C1 \cdot A \cdot \bar{B} + C3 \cdot \bar{A} \cdot B + C4 \cdot A \cdot B,$$

kjer je signal *OE* namenjen postavitev izhoda *Y* v visoko impedančno stanje, *A* in *B* sta selektorska vhoda, *C*0, *C*1, *C*2 in *C*3 pa podatkovni vhodi.

Preklopnik 74HC139 je sestavljen iz dveh demultipleksor 2/4, ki opravljata nalogo, nasprotno prej opisanima multipleksorjema. Z nastavitvijo selektorskih vhodov *A* in *B* namreč pripeljemo signal *OE* na izbran izhod *Y*0, *Y*1, *Y*2 ali *Y*3.



Slika 20: Multipleksor



Slika 21: Demultipleksor

4.2.4. Priključki za končne naprave

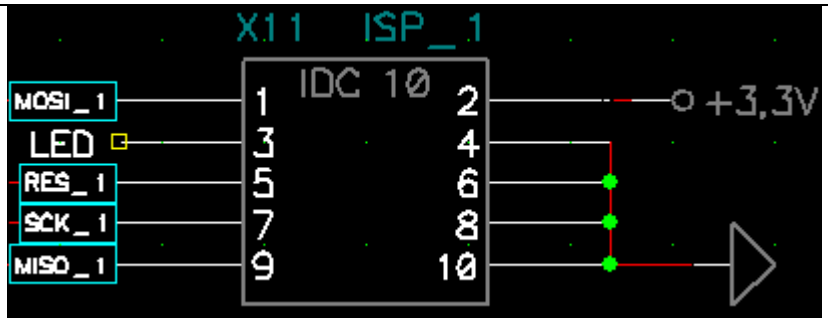
Kot smo že omenili na vezje lahko priklopimo štiri končne naprave, s katerimi bomo izmenjevali podatke preko standarda RS-232. Uporabili smo dva standardna ženska priključka tipa DB9 in dva priključka tipa RJ45. Slednje smo uporabili ker nekateri proizvajalci s svojimi napravami dostavljajo tudi posebne kable s takšnim priključkom.

Pri dizajniranju in naročanju priključkov je še posebej pomembno, da smo natančni pri priključitvi pravilnih nožic na signale, ki jih potrebujemo. Hitro lahko pride do zmede, saj so lahko sheme moških in ženskih priključkov skoraj enake.

4.2.5. Ostali elementi

Za mikrokrmilnike smo naredili ustrezne vmesnike s pomočjo katerih jih lahko poljubno odstranimo ali pripnemo v tiskano vezje. To velja tako za procesorja AtTiny88 in AtMega2560 kot tudi za modul xBee.

Na tiskanem vezju so tudi priključki ISP (ang. In - System Programming) za programiranje procesorjev neposredno na tiskanem vezju. Uporabili smo običajne 10-nožične priključke s standardnimi signali, kot je prikazano na sliki 22.

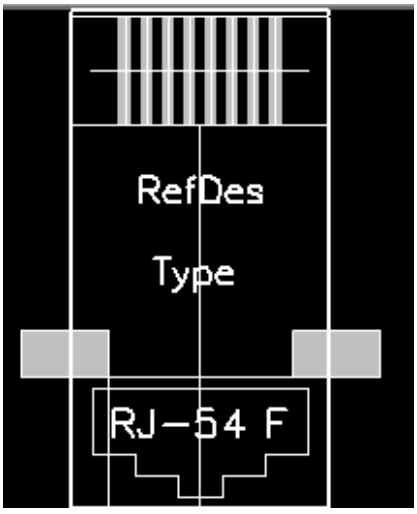


Slika 22: Shema priključka za programiranje čipov

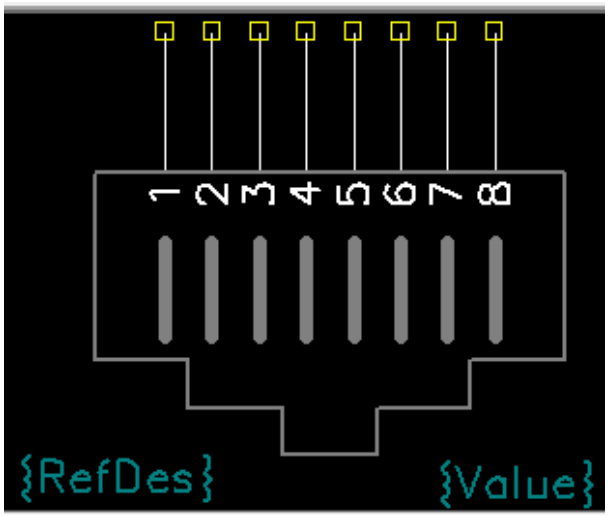
Vsi ostali elementi so tipa SMD (ang. Surface Mount Device). To pomeni, da so priključeni neposredno na površino tiskanine in so zelo majhni.

4.2.6. Izdelava tiskanega vezja

Najprej smo za vse elemente zbrali dokumentacijo. Nato je sledilo risanje vzorcev. To pomeni, da smo za vsak element narisali vse nožice in ohišja. Tukaj je pomembno, da uporabimo prave širine in razmike nožic ter pazimo na ustrezne dimenzije elementov. Sledi risanje logičnih simbolov elementov. Simboli nam kasneje pomagajo pri povezovanju elementov na shemi. Pri tem pazimo, da je simbol logičen in prepoznaven saj nam to olajša risanje sheme. Vemo, da imajo elementi lahko različna ohišja. Vsi enaki elementi bodo torej imeli enak simbol, lahko pa imajo različen vzorec. S kombinacijo vzorca in simbola dobimo komponento. Povežemo ju tako, da enačimo ustrezne nožice vzorca s tistimi na simbolu.
Primer vzorca in pripadajočega simbola lahko vidimo na slikah 23 in 24.



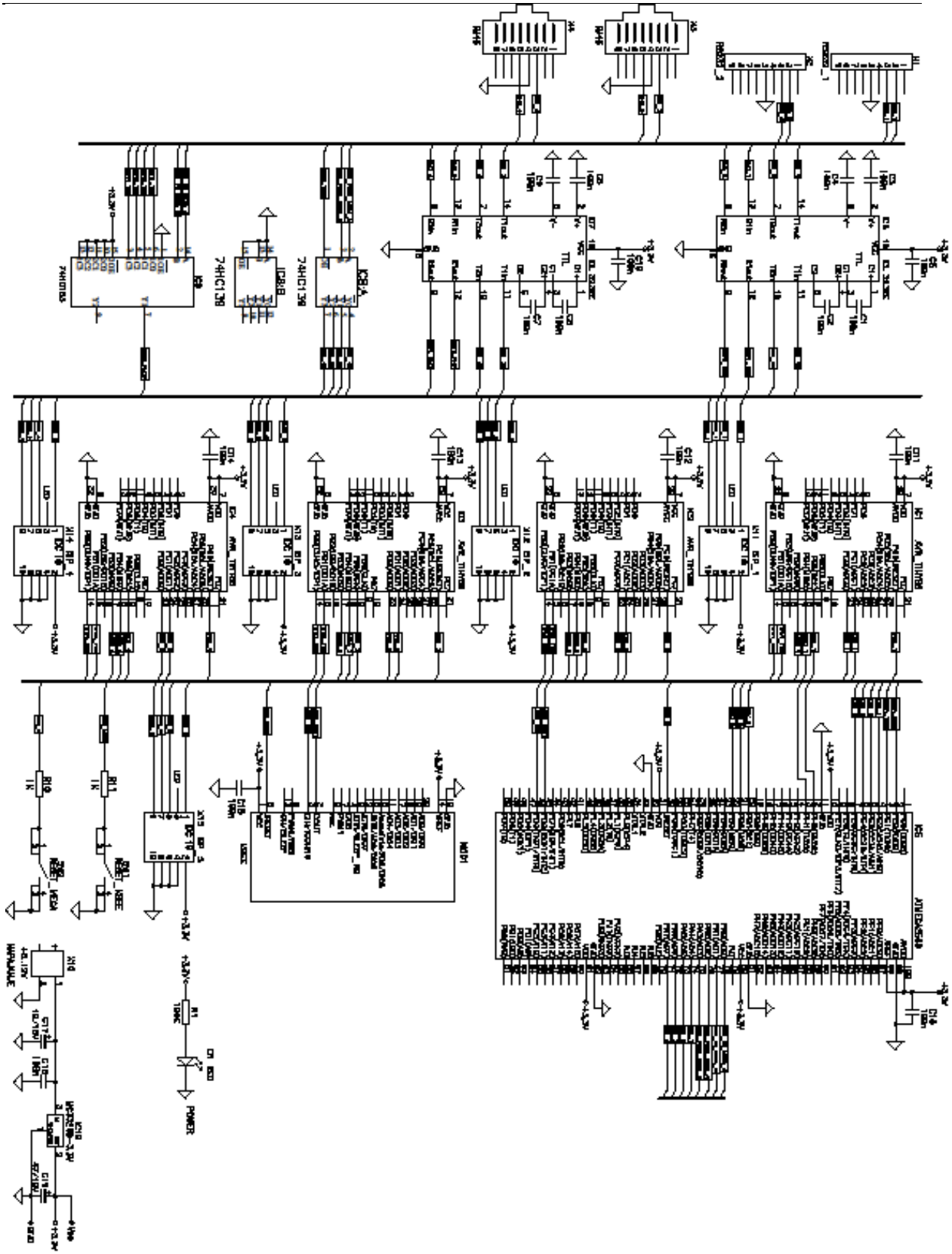
Slika 23: Primer vzorca



Slika 24: Primer simbola

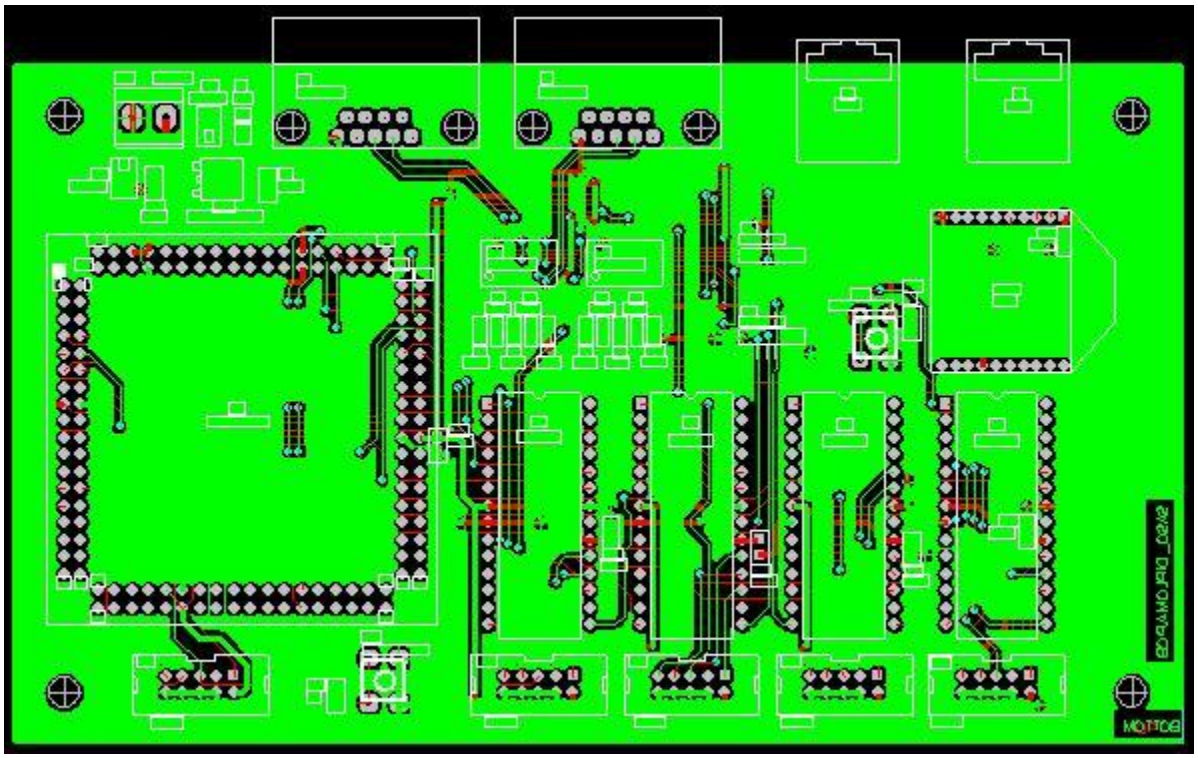
Ko smo pripravili knjižnice z vsemi potrebnimi elementi, se lahko lotimo sestavljanja sheme vezja. Pri tem moramo upoštevati električne in fizikalne zakonitosti. Da se izognemo motnjam moramo povezati vse vhode in poskrbeti da ni nedefiniranih vrednosti na linijah.

Opisani elementi so smiselno povezani med seboj v logični shemi, ki je prikazana na sliki 25.



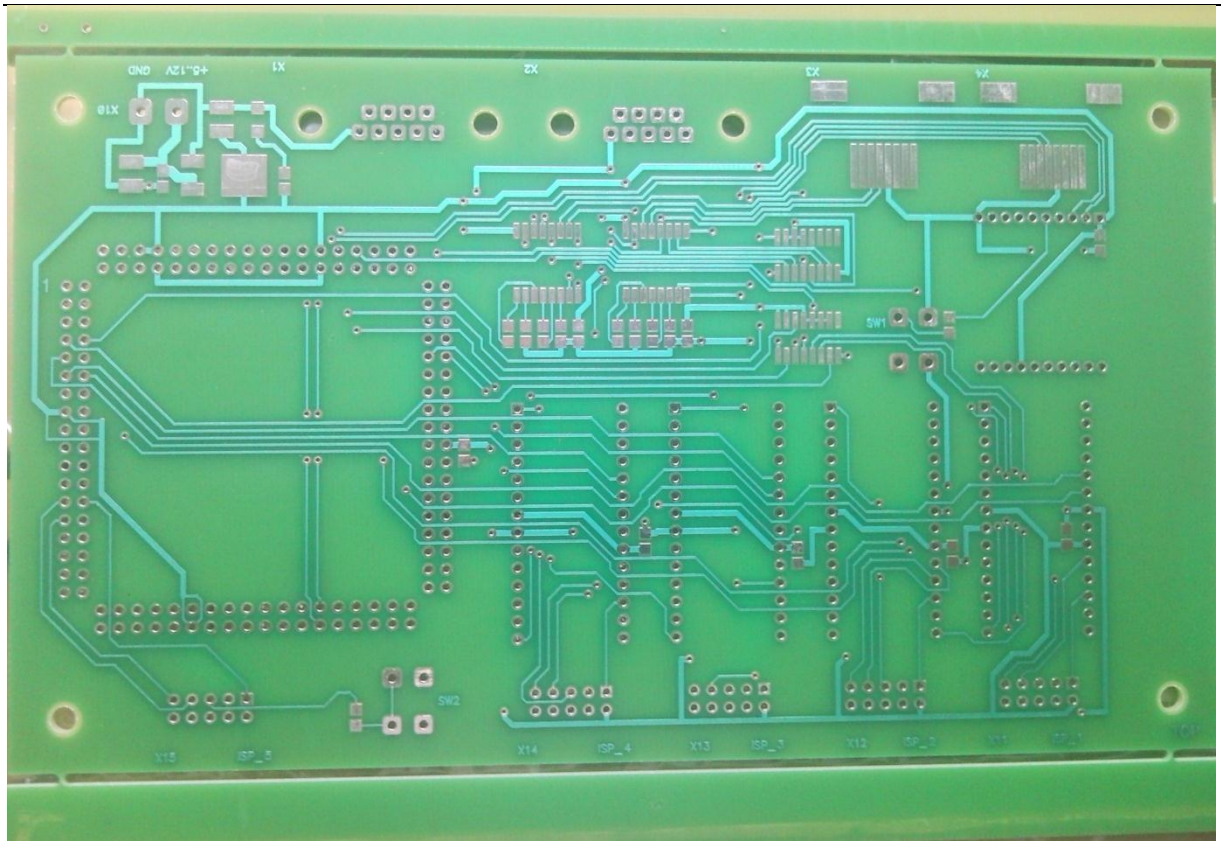
Slika 25: Logična shema vezja

Naslednja faza izdelave tiskanega vezja je fizična postavitev elementov in linij na površino. Tudi tukaj se moramo izogibati morebitnim presluhom in negativnim vplivom sosednjih komponent ena na drugo. Pravilno je tudi da skušamo komponente logično razporediti. Na sliki 26 je prikazana razporeditev elementov na tiskanem vezju projekta. V spodnjem delu najdemo štiri priključke ISP za programiranje mikrokrmilnikov. Nad njimi je prostor za mikrokrmilnike. Zgornjo stran plošče smo rezervirali za priključke zunanjih naprav. Celoten napajalni del je v zgornjem levem kotu, v preostal prostor pa smo umestili vhodni in izhodni preklopnik, pretvornik signalov za protokol RS-232, gumbe, modul ZigBee in pasivne elemente.



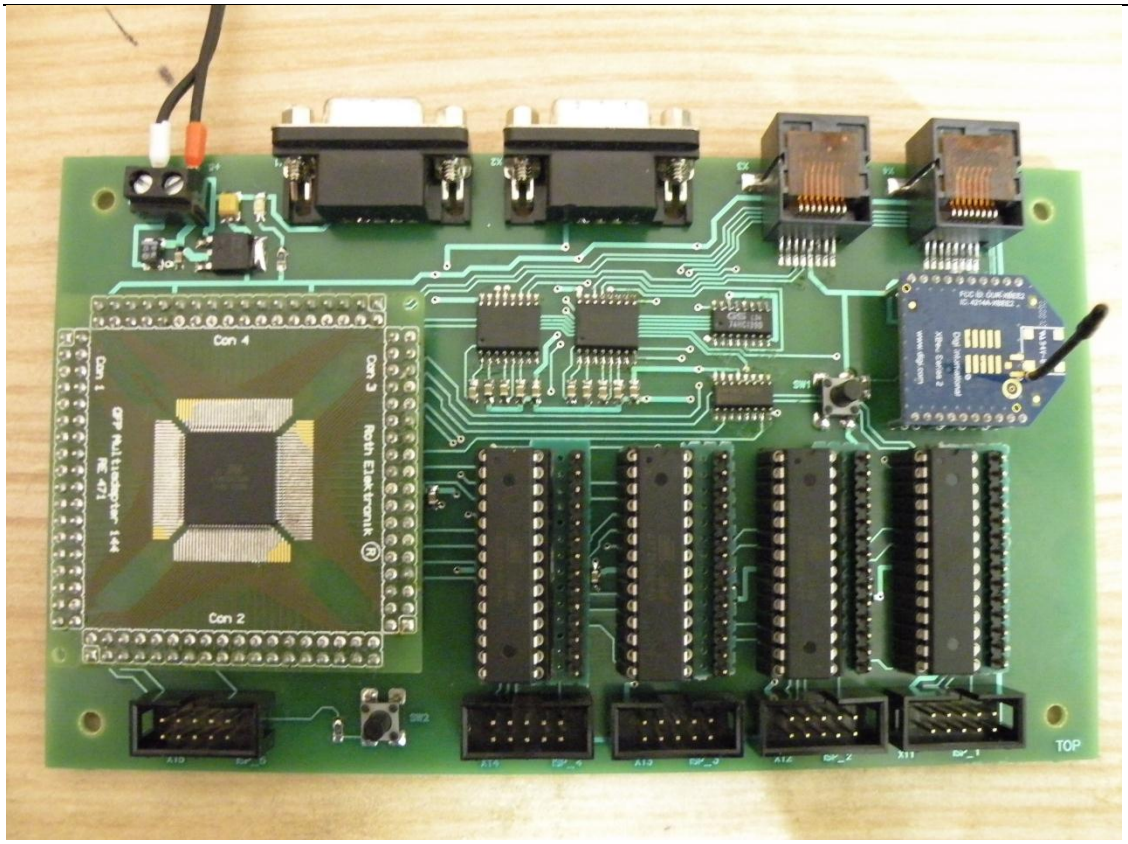
Slika 26: Programski pogled na postavitev elementov

Tako ustvarjeno shemo elementov uporabimo za izdelavo fizičnega tiskanega vezja. Program vsako plast narisanega vezja shrani v posebne datoteke, ki jih razumejo stroji za izdelavo tiskanih vezij. Iz programske postavitve elementov tako dobimo fizično ploščico prikazano na sliki 27.



Slika 27: Tiskano vezje brez elementov

Na takšno tiskano vezje moramo vstaviti nastavke za procesorje in vse ostale elemente. S posebnim postopkom to lahko naredijo tudi stroji. Pri lotanju elementov na površino moramo paziti, da jih ne pregrejemo in, da ne naredimo kratkih stikov med nožicami elementov ali linijami na tiskanem vezju. Na sliki 28 je prikazano tiskano vezje z vsemi elementi, ki je pripravljeno na programiranje in testiranje.



Slika 28: Končano tiskano vezje pripravljeno na uporabo

4.3. Programska oprema

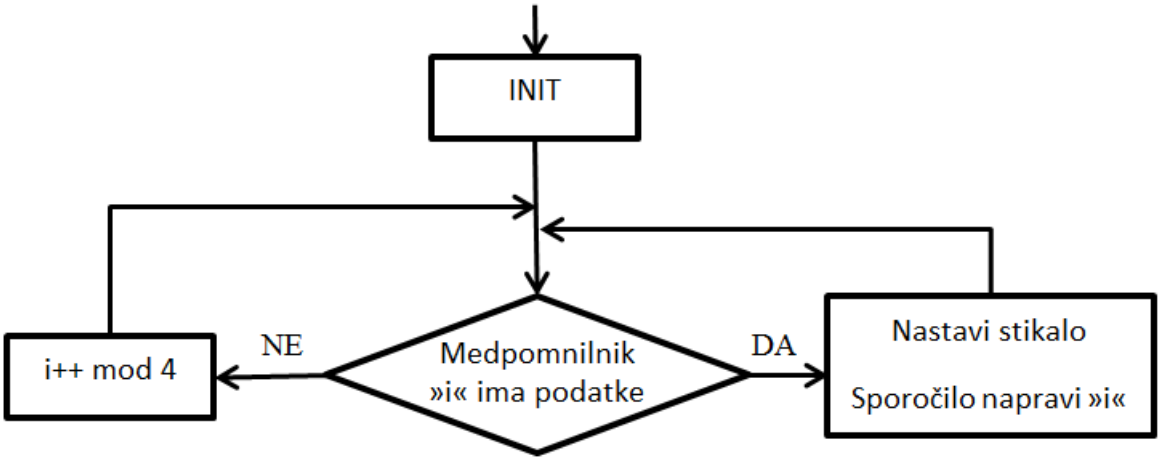
Programski sklop je sestavljen iz treh delov. Prvi del je koda nadzornega procesorja, ki skrbi za usmerjanje prometa od ustreznega odjemalca do ustrezne končne naprave. Drug del je koda zunanjega medpomnilnika, tretji del pa računalniška aplikacija, ki nam pomaga pri iskanju nadzornikov in povezovanju nanje.

4.3.1. Nadzornik

Programiranje mikroprocesorjev se nekoliko razlikuje od običajnega programiranja za osebne računalnike. Današnji računalniki so zelo hitri in imajo veliko količino delovnega spomina, zato ni zelo pomembno kakšne spremenljivke definiramo v programu. Poleg tega programiramo objektno in nas zaporedje ukazov, ki jih procesor obdeluje ne skrbi preveč. Vse to pa je zelo pomembno pri pisanju programov za mikroprocesorje. Poskrbeti moramo, da se določene spremenljivke vedno nalagajo iz programskega spomina v registre procesorja, poleg glavnega toka ukazov poznamo še prekinitve, pravilna definicija spremenljivk nam lahko prihrani prostor v pomnilnikih, poudarek je na vhodno – izhodnih povezavah.

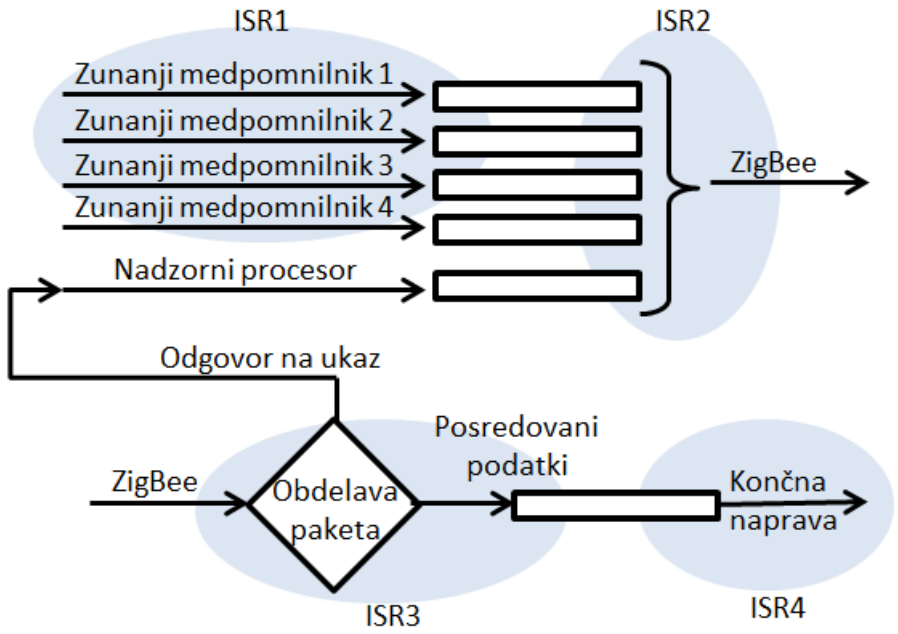
Glavni del programske kode je namenjene za nadzornik, ki skrbi za več opravil.

V zunanjih pomnilnikih se začasno shranjujejo podatki, ki prihajajo iz končnih naprav. Ko ima zunanji medpomnilnik na voljo podatke, to sporoči nadzorniku s signalom RTS (ang. Ready To Send). Glavni program na nadzorniku se zato začne z nastavitvijo vhodno – izhodnih naprav in nastavitvijo začetnih vrednosti spremenljivk. Nato nadaljuje v neskončno zanko, kot je prikazano na sliki 29. V njej neprekinjeno preverja signal RTS zunanjih medpomnilnikov. Če je signal aktiven, preklopi vhodno stikalo na ustrezen zunanji medpomnilnik in mu s signalom CTS (ang Clear To Send) sporoči naj se medpomnilnik začne prazniti. Ko signal RTS zunanjega medpomnilnika ni več aktiven, se preveri signal RTS naslednjega zunanjega medpomnilnika v čakalni vrsti. Zunanje medpomnilnike izbiramo po vrsti enega za drugim. Ko pridemo do zadnjega spet nadaljujemo s prvim v čakalni vrsti. Takšen način izbiranja elementov v čakalni vrsti imenujemo round robin.



Slika 29: Glavni program na nadzornem procesorju

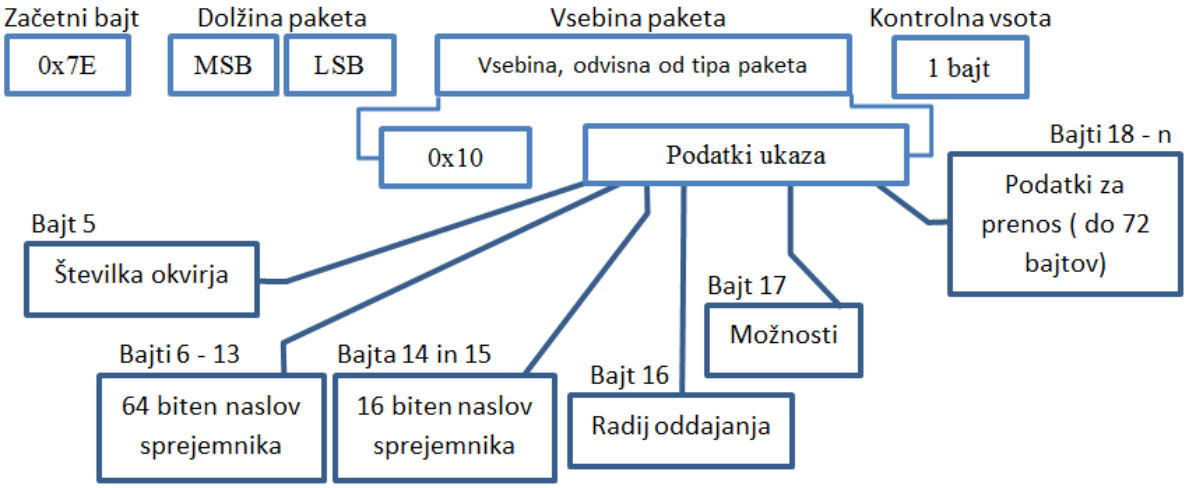
Ostala koda na nadzorniku skrbi za pravilno usmerjanje in obdelavo sprejetih in oddanih podatkov, zato se nahaja v prekinitvenih rutinah procesorja. Ob prekinitvi procesor prekine izvajanja glavnega programskega cikla in skoči na začetek prekinitveno servisne rutine (ang. Interrupt Service Routine, ISR). Ko se izvede vsa koda prekinitveno servisni rutine, se nadaljuje izvajanje glavnega programskega cikla od mesta prekinitve dalje. Pri tem se stanje vseh registrov v procesorju povrne na stanje pred prekinitvijo.



Slika 30: Glavne prekinitveno servisne rutine

Nadzorni procesor je nastavljen tako, da se prekinitve prožijo ob petih pogojih. Za vsak pogoj imamo svoj prekinitveno servisno rutino. Za lažjo predstavo delovanja so na sliki 30 shematično predstavljene najpomembnejše štiri.

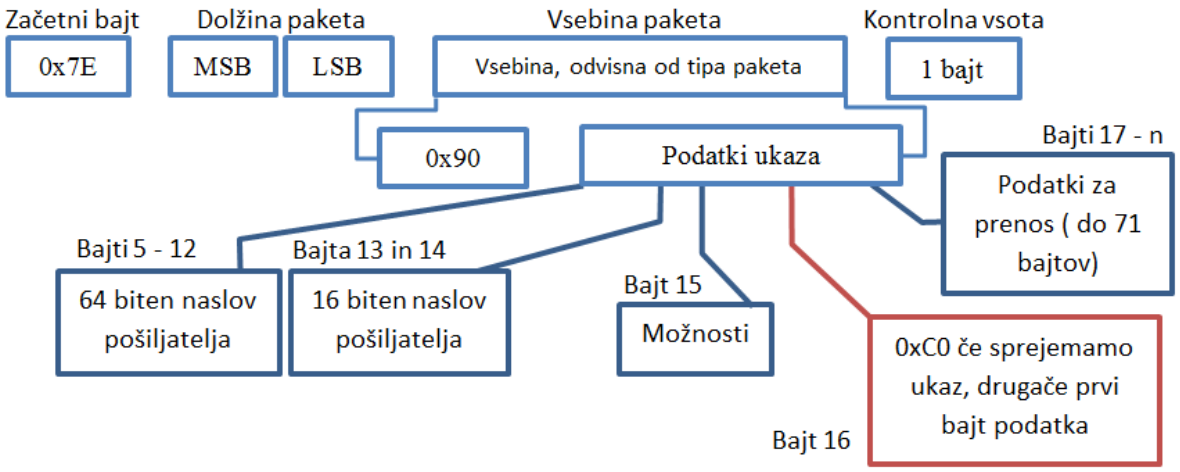
Prekinitveno servisna rutina ISR1 se proži, ko nadzornik na svoj serijski vhod sprejme podatke. Na ta vhod so povezani zunanji medpomnilniki. ISR1 poskrbi, da se tako prejeti podatki shranijo v ustrezen medpomnilnik nadzornega procesorja za kasnejšo obdelavo. Poleg tega rutina ISR1 preveri tudi dolžino uporabnih podatkov v medpomnilniku, ki ga polni. Ko prebrana dolžina preseže določeno vrednost, se vklopi proženje prekinitveno servisne rutine 2. Prekinitveno servisna rutina ISR2 začne s praznjenjem nastavljenega medpomnilnika. Za pošiljanje podatkov po omrežju ZigBee jih moramo naprej preoblikovati v ustrezne pakete. Oblika paketa je predstavljena na sliki 31. Prekinitveno servisna rutina ISR2 podatkom doda glavo. Ta vsebuje dolžino paketa, naslov naprave, ki ji podatke posredujemo, in tip paketa, ki ga pošiljamo. Na koncu doda še nekaj dodatnih lastnosti paketa in paritetno vsoto ter tako pripravljen paket pošlje.



Slika 31: Struktura paketa za prenos podatkov v omrežju ZigBee

Glavna naloga prekinitveno servisne rutine ISR3 je sprejemanje podatkov iz omrežja ZigBee in njihovo interpretiranje. Podatki so sestavljeni v obliki paketa prikazanega na sliki 32, rutina pa jih dobiva bajt po bajt. V vsakem koraku izlušči del podatka, ki ga potrebuje za nadaljevanje. Na podlagi interpretiranja zbranih podatkov se rutina odloči ali je sprejet paket ukaz ali pa podatek. To ji pove dodaten bajt na začetku podatkovnega dela sprejetega paketa. Prekinitveno servisna rutina ISR3 ukaze dodatno obdela, podatke pa shrani za nadaljnjo obdelavo v vhodni medpomnilnik in vklopi proženje prekinitveno servisne rutine 4.

Ukaz za povezavo odjemalca s končno napravo, ustrezno nastavi naslove v usmerjevalni tabeli. Ko je ukaz obdelan, prekinitveno servisna rutina ISR3 zapiše tudi status izvedbe ukaza v ustrezen izhodni medpomnilnik in vklopi proženje prekinitveno servisne rutine ISR2. Ko pa prekinitveno servisna rutina ISR3 prepozna ukaz za odklop odjemalca iz nadzornika, bo v usmerjevalni tabeli sprostila končno napravo, na katero je bil odjemalec povezan.



Slika 32: Struktura prejetega paketa iz omrežja ZigBee

Prekinitveno servisna rutina ISR4 je najenostavnejša med vsemi. Iz izhodnega medpomnilnika prebere podatek, nastavi izhodno stikalo in pošlje podatek na serijski izhod procesorja.

Prekinitveno servisna rutina ISR5 se proži po poteku nastavljenega časovnega intervala. Ob vsakem proženju pregleda dolžino uporabnih podatkov v izhodnih medpomnilnikih nadzornega procesorja. Če v katerem od njih najde uporabne podatke, aktivira proženje prekinitveno servisne rutine ISR2. To se po navadi zgodi ob koncih prenosov, ko dolžina uporabnih podatkov v medpomnilniku ne prekorači minimalne zahtevane dolžine za proženje prekinitveno servisne rutine ISR2.

4.3.2. Zunanji medpomnilnik

Procesor ATtiny88 je primeren za medpomnilnik, ker je poceni in dovolj hiter. Takšen medpomnilnik potrebuje za delovanje dva signala. Prvi signal je RTS, ki ga bo poslal glavnemu procesorju, ko se bodo v medpomnilniku nahajali uporabni podatki. Drug signal je CTS, s katerim nadzorni procesor sporoči medpomnilniku, da je nanj preklopil podatkovne linije in se lahko začne prazniti. Signal RTS bo medpomnilnik spustil na logično vrednost 0, ko ne bo več vseboval uporabnih podatkov.

Medpomnilnik dobiva podatke iz priključenih naprav preko vmesnika RS-232. Zato jih na ta način tudi sprejema in pošilja dalje. Ker sam po sebi nima strojnega vmesnika USART (ang. universal serial asynchronous receiver/transmitter), mora biti komunikacija rešena

programsko. S pomočjo časovnikov in zunanjih primerjalnikov postane to ob poznavanju standarda RS-232 relativno enostavna naloga.

V glavnem programu zunanjega medpomnilnika nastavimo naprave in začetne vrednosti spremenljivk. Nato zaženemo neskončno zanko, vse naloge procesorja pa izvedemo v prekinitveno servisnih rutinah.

Prva prekinitveno servisna rutina v medpomnilniku (ang. buffer) ISRb1 je nastavljena tako, da se proži ob prehodu signala na sprejemni nožici procesorja iz logične vrednosti 1 na logično vrednost 0. Na takšen način zaznamo začetni bit pri komunikaciji po protokolu RS-232 (slika 15). Nato rutina nastavi pogoje za proženje prekinitveno servisne rutine ISRb2. Ti so odvisni od hitrosti prenosa podatkov po protokolu RS-232 in morajo biti zelo natančno nastavljeni. Čas med dvema prenesenima bitoma je

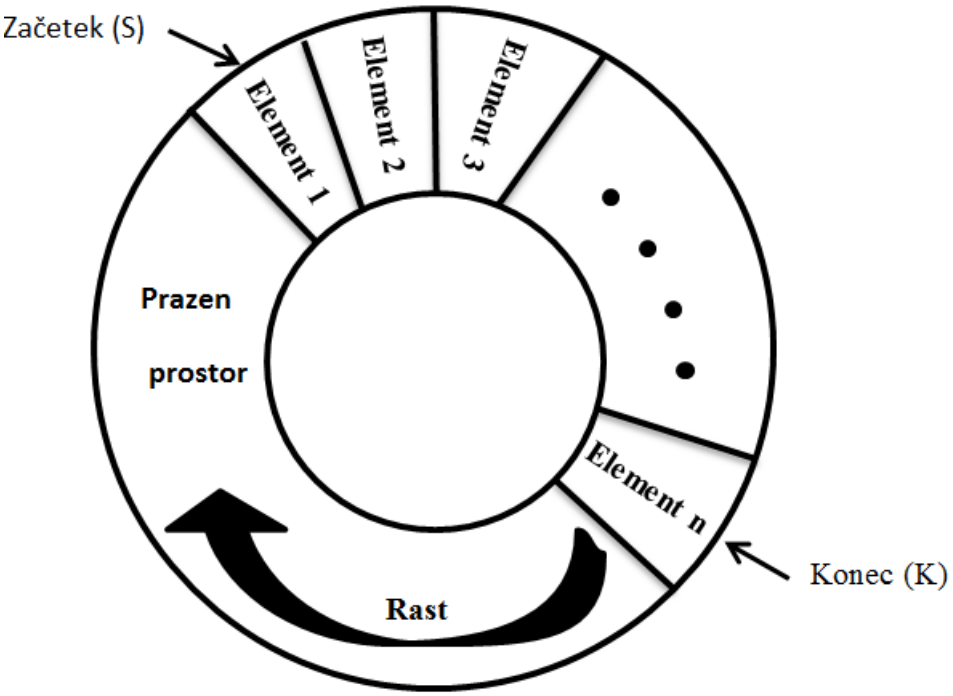
$$\Delta t = \left\lceil \frac{fCpu}{rate} \right\rceil$$

pri čemer je *fCpu* hitrost procesorja v Hz, *rate* pa hitrost prenosa podatkov v bitih na sekundo. Tako nastavljena prekinitveno servisna rutina ISRb2 se proži na sredini bitnega intervala in shranjuje vrednosti prebrane na sprejemni nožici procesorja v spremenljivko. Ko sprejme vseh osem bitov, preveri še pravilnost zaključnega bita in sprejet bajt shrani na ustrezno mesto v krožni pomnilnik procesorja.

Za pošiljanje podatkov iz zunanjega medpomnilnika skrbi prekinitvena servisna rutina ISRb3. Ker podatke pošiljamo z enako hitrostjo kot jih sprejemamo, prekinitveno servisno rutino ISRb3 prožimo z enakim časovnim intervalom kot prekinitveno servisno rutino ISRb2. Prekinitvena servisna rutina ISRb3 upravlja s signalom RTS glede na vsebino medpomnilnika. Ob prisotnosti signala CTS, ki ga nastavi nadzorni procesor, iz medpomnilnika prekopira prvi uporaben bajt v spremenljivko in začne s prenašanjem bit po bit, kot to narekuje protokol RS-232.

Krožni medpomnilnik (slika 33) je neprekinjena struktura dolžine n bajtov v pomnilniku procesorja s pripadajočimi kazalci, ki se spreminjajo po spodaj opisanem postopku. Programsko ga lahko določimo s pomočjo enostavnega seznama (ang. array). Pomembno je, da poleg kazalca na začetek seznama nanj vseskozi kažeta dva dodatna kazalca. Prvi predstavlja začetek uporabnih podatkov (S) drugi pa konec uporabnih podatkov (K). Ko pride v pomnilnik nov podatek, se ta zapiše na mesto K v seznamu, kazalec pa se za eno mesto poveča. Podobno se ob branju podatka iz pomnilnika kazalec S poveča za eno mesto. Ko

katerikoli od kazalcev doseže maksimalno dolžino seznama, se njegova vrednost vrne na ena in postopek se nadaljuje. Pomnilnik je poln takrat, ko kazalca S in K kažeta na isto mesto v seznamu. Zdi se, da je ob istem pogoju pomnilnik lahko tudi prazen. Ta problem je rešljiv na več načinov. Primerjamo lahko na primer število branj in pisanj, v pomnilniku lahko vedno pustimo eno mesto prazno, v spremenljivko lahko shranjujemo število preostalih uporabnih podatkov ali pa namesto relativnega uporabljamo absolutno naslavljanje lokacije v pomnilniku. Zadnje je zelo učinkovito, če uporabljamo pomnilnike dolžine, ki so potence števila 2 (64, 128, 256,…).



Slika 33: Prikaz krožnega pomnilnika

4.3.3. Aplikacija na odjemalcu

Na računalniku teče aplikacija, ki komunicira direktno s priključenim modulom ZigBee. Modul ZigBee je lahko nastavljen v AT ali API načinu. Ta nastavitev ne vpliva na možnost priključevanja in komuniciranja v omrežju, vsaka pa ima svoje prednosti in slabosti opisane v poglavju 3.1.7. Zaradi prednosti načina API in lažje razširitve programske opreme smo se odločili za slednji način. Aplikacija je uporabnikovo okno v omrežje, v katerega se povezuje, zato mu mora nuditi možnost:

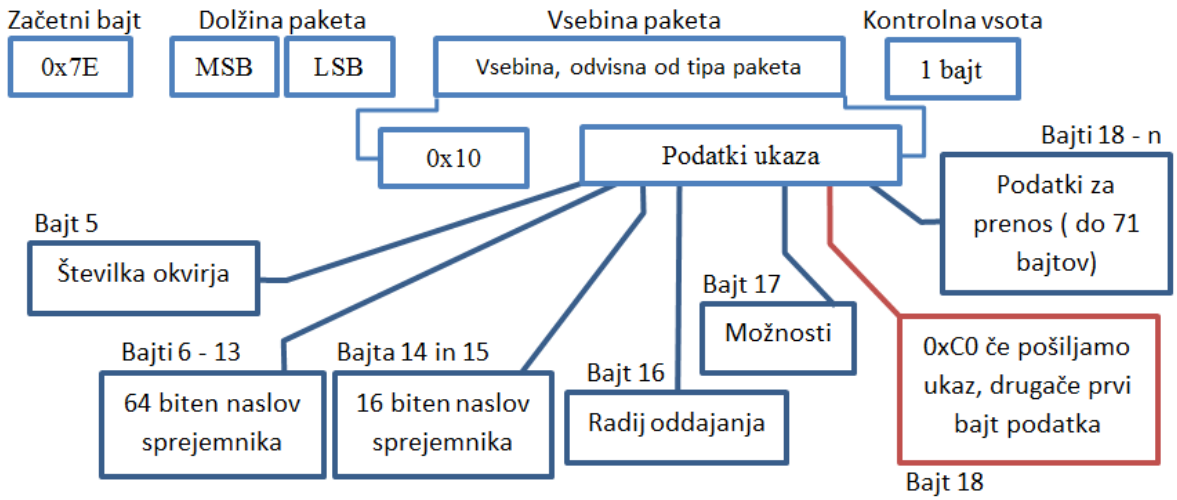
- odkrivanja sosednjih naprav,

- povezovanja na sosednje naprave,
- asociacije z izbrano napravo, pri čemer si napravi izmenjata podatke, ki jih potrebujeta za komuniciranje, na primer, naslov naprav za pošiljanje in oddajanje, verzija sistema, dodatne varnostne informacije,
- izbiranja vhodno-izhodnih poti na povezani napravi,
- pošiljanja in sprejemanje podatkov v uporabniku znanem (terminalskem) načinu in
- odklop iz izbrane naprave.

Aplikacija je napisana v programskem jeziku C#. Preko komunikacijskih vrat COM se povezuje z napravo ZigBee in ji pošilja ukaze v obliki paketov. Vsak odjemalec z modulom ZigBee, ki se hoče povezati z nadzornikom, mora znati pošiljati in sprejemati vsaj štiri različne tipe ukazov:

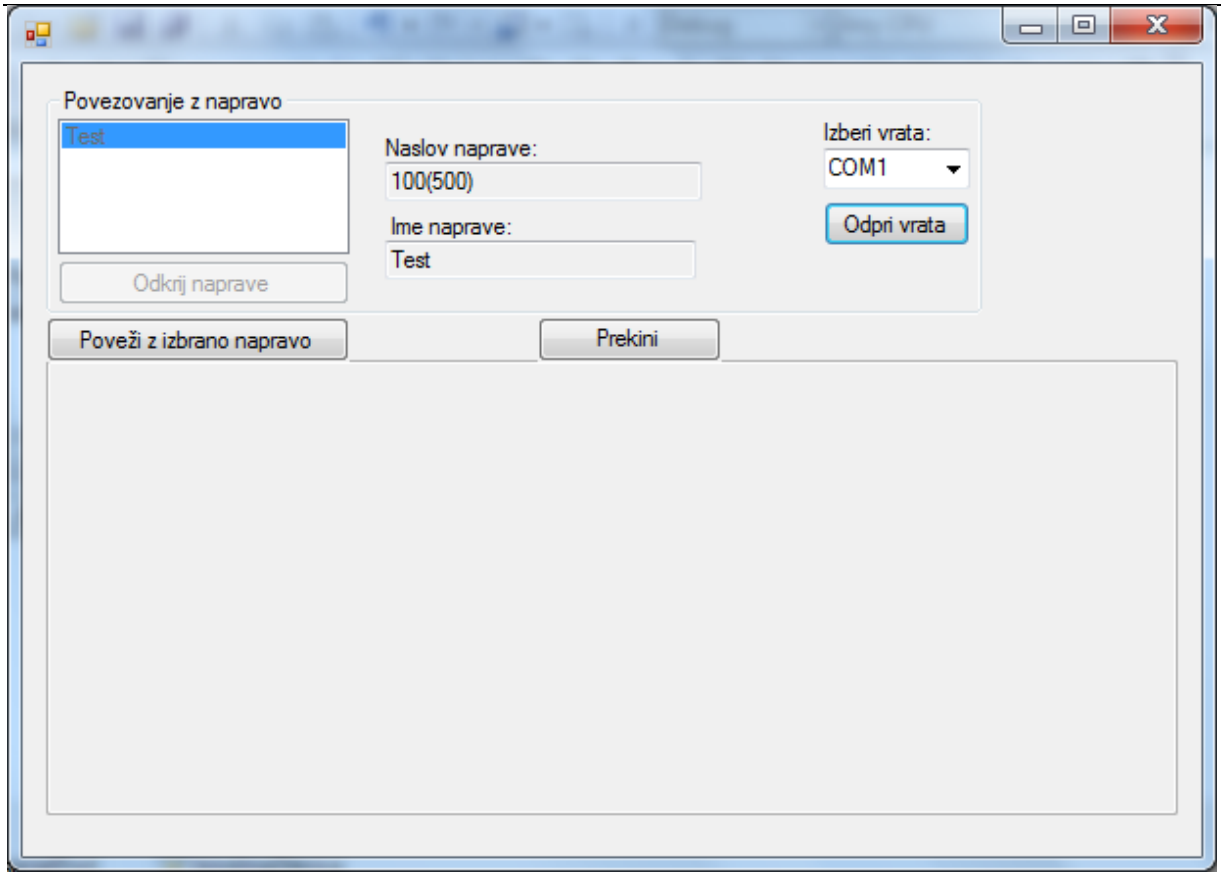
- ukaz, ki sproži proces prijave na nadzorniku,
- ukaz, ki pošlje podatke na izbrano končno napravo,
- ukaz, s katerim se odklopimo od izbrane končne naprave na povezanem nadzorniku in
- ukaz za odkrivanje sosednjih naprav, ki nam olajša pregled nad dostopnimi nadzorniki v omrežju.

Prvi trije ukazi uporabljajo tip okvirja ZigBee Transmit Request za prenos podatkov, zadnji pa poseben tip okvirja ZigBee Node Discovery.
Ker se mora aplikacija pogovarjati tudi nadzornikom, smo strukturo podatkovnega paketa razširili z dodatnim bajtom, ki pove ali gre za normalen podatkovni prenos ali pa je vsebina paketa ukaz nadzorniku. Sprememba je prikazana na sliki 34. Nadzornik seveda upošteva to spremembo protokola.



Slika 34: Spremenjena struktura paketa za prenos podatkov v omrežju ZigBee

Aplikacija na odjemalcu deluje tako, da v posebni niti sprejema podatke iz priključenega modula ZigBee. Ko sprejme celoten paket, ga posreduje funkciji za prepoznavanje ukazov (ang. parser). Ta se nato odloči ali je bila sprejeta informacija ukaz ali podatek, ki ga prikaže v tekstovnem polju. Komunikacija v nasprotno smer je enostavnejša. Ob pritisku na vnosno tipko se ustvari ustrezen paket, ki ukaz ali podatke pošlje preko modula ZigBee po komunikacijskem kanalu do nadzornika. Maska aplikacije je prikazana na sliki 35 in je sestavljena iz gumbov za proženje akcij in terminalskega vnosnega polja.



Slika 35: Maska odjemalca

4.4. Težave pri razvoju sistema

Izgradnja nadzornika je potekala od idejnega načrta do programske lupine in nato dejanske strojne rešitve, na kateri se je programska oprema dodelala do končne verzije.

Na srečo med razvojem sistema ni prihajalo do večjih napak. Ena izmed njih je bila napaka v shemi tiskanine. Prišlo je do zmešnjave v orientaciji DB9 priključka. Zato sprejemni del na priključkih ni delal, oddajni pa zelo slabo. Napaka je bila dokončno ugotovljena šele, ko smo z osciloskopom opazili nepravilno popačene signale.

Če ne štejemo običajnih problemov z računalniki, je imel projekt poleg manjših standardnih programskih napak tudi zanimivo napako, povezano s prenosom podatkov. Za medpomnilnik smo namreč uporabili procesor, na katerem smo ročno sprogramirali protokol USART. Pri tem je zelo pomembna pravilna nastavitev hitrosti oddajanja in sprejemanja bitov. Ker pa procesorji uporabljajo notranjo uro, je njihova frekvenca odvisna od več dejavnikov. Pri pošiljanju in oddajanju preko procesorja se je večkrat zgodilo, da se je zadnji bit v sporočilu spremenil. Tako smo dobili ob prehodu skozi zunanji medpomnilnik popolnoma spremenjen

podatek. Problem smo rešili z natančno kalibracijo časovnikov glede na zahtevano hitrost prenosa in frekvenco procesorja.

V celotnem projektu je najdlje trajal razvoj tiskanine. Pri tem je bilo potrebno najti ustrezne elemente, pregledati njihovo dokumentacijo in jih prenesti v program za risanje vezij. V programu smo narisali vsak element posebej z upoštevanjem fizičnih in električnih lastnosti. To je zelo pomembno in natančno delo pri katerem hitro pride do napak. Ko je tiskanina narejena, se kaj rado zgodi, da elementa fizično ne moremo vstaviti nanjo, da so nožice na čipih pomešane in podobno.

5. Zaključek

V deluj je predstavljena rešitev za brezžično povezovanje naprav, ki komunicirajo preko vmesnikov RS-232. Kot brezžični komunikacijski protokol smo uporabili standard ZigBee, okoli katerega smo razvili napravo, nadzornik, z več priključki RS-232 za fizični priklop končnih naprav. Nadzornik s programsko opremo in programsko opremo na odjemalcu je zasnovan tako, da za končnega uporabnika predstavlja skoraj enako uporabniško izkušnjo, kot da bi bil s končno napravo povezan preko kabla. Uporabnik se preko posebne terminalske aplikacije poveže na nadzornik, nato pa direktno komunicira s priklopljenimi končnimi napravami.

V samo programsko opremo in način komunikacije je možno vnesti še kar nekaj izboljšav, ki bi napravo naredile še bolj uporabno. Tako bi lahko na primer celoten oddajni del iz računalnika (odjemalca) preselili v dodatno napravo, ki bi na eni strani preko ustreznega vmesnika komunicirala z računalnikom in na drugi strani preko modula ZigBee z opisanim nadzornikom. Prednost take rešitve bi bila v tem, da sistem ne bi bil več odvisen od operacijskega sistema. Na računalniku bi tako lahko uporabljali svoj najbolj priljubljen terminalski program.

Prav tako bi bilo ob predpostavkah, da vse priključene naprave z vmesnikom RS-232 uporabljajo signala CTS in RTS in da je prenos podatkov dovolj počasen, mogoče močno poenostaviti zgradbo celotne naprave, saj bi glavni procesor lahko sam skrbel za prenos podatkov. Ta je namreč dovolj hiter, da lahko z njim preko enega samega vmesnika USART z multipleksiranjem upravljamo več priključenih končnih naprav.

Na glavnem procesorju bi lahko dodali tudi dodatne nivoje varne avtentikacije ali pa avtomatsko zbiranje podatkov o tipu, namenu in zmogljivostih priključenih naprav.

Z nekaj dela bi na enak način lahko brezžično prenašali skorajda vsak drug standard, ki je zasnovan na fizičnih povezavah. Poleg priključkov bi morali zamenjati le sprejemno-oddajni del programske opreme na čipih nadzornika in že bi imeli ustrezen vmesnik za nek drug standard.

Kljub vsem pastem smo projekt uspešno pripeljali do zaključka. Ker je narejen na modularen način, omogoča enostavno razširitev funkcionalnosti z dodajanjem novih funkcij. Lahko bi ga, na primer, razširili v nadzorni sistem senzorskih omrežij. Nadzornik ima več kot eno uporabno vrednost in prestavlja primer, kako lahko s pridom izkoristimo moderno brezžično omrežje ZigBee. Ker je standard ZigBee še mlad, na tržišču še ni velikega zaupanja vanj. Kljub temu nove naprave, prilagojene za omrežje ZigBee, vedno pogosteje srečujemo na

najrazličnejših področjih. Poleg tega s standardom raste tudi izbira in kakovost modulov ZigBee. Standard je perspektiven, enostaven za uporabo in zelo prilagodljiv, zato ga bomo v prihodnosti zagotovo še srečali!

6. Priloge

6.1. Dodatek A

6.1.1. Izvorna koda računalniške aplikacije

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace VajrlesKabl
{
    public partial class Form1 : Form
    {

        private delegate void SetTextDeleg(byte[] text);

        public Form1()
        {
            InitializeComponent();
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if ((e.KeyChar == '\r')||(e.KeyChar == '\n'))
            {
                //MessageBox.Show("ENTER key detected");
                device                        naprava                        =
(device)bindingDevice[bindingDevice.Position];
                System.Text.ASCIIEncoding         enc         =         new
System.Text.ASCIIEncoding();

                byte[] addrLong = BitConverter.GetBytes(naprava.addrLong);
```

```
                byte[]                          addrShort                          =
BitConverter.GetBytes(naprava.addrShort);
                byte[]                            text                             =
enc.GetBytes(textBox.Lines[textBox.GetLineFromCharIndex(textBox.SelectionLe
ngth)]+Environment.NewLine);

                byte[]               ukaz               =               new
byte[4+addrLong.Length+addrShort.Length+text.Length];
                ukaz[0] = 0x10;//ZigBee transmit request API frame type
                ukaz[1] = 0x01;//FrameID seto to non-zero value
                System.Buffer.BlockCopy(addrLong,      0,      ukaz,      2,
addrLong.Length);
                System.Buffer.BlockCopy(addrShort, 0, ukaz, addrLong.Length
+ 2, addrShort.Length);
                ukaz[addrLong.Length    +    2    +    addrShort.Length]    =
0x00;//broadcast radius
                ukaz[addrLong.Length    +    2    +    addrShort.Length+1]    =
0x00;//options
                System.Buffer.BlockCopy(text, 0, ukaz, 2 + addrLong.Length
+ addrShort.Length + 1 + 1, text.Length);

                commit(ukaz);
            }
        }

        private    void    lstNaprave_SelectedIndexChanged(object    sender,
EventArgs e)
        {
            if (lstNaprave.SelectedIndex != -1)
            {
                device prebran = (device)lstNaprave.SelectedItem;

                txtNaslov.Text = prebran.name;
                txtIme.Text                                                    =
prebran.addrShort.ToString()+"("+prebran.addrLong.ToString()+")";
            }
        }

        private void btnOdkrij_Click(object sender, EventArgs e)
        {
            odkrijNaprave();
```

```
            textBox.Enabled = true;
        }

        private void odkrijNaprave()
        {
            Byte[] ukaz = { 0x7E, 0x00, 0x04, 0x08, 0x01, 0x4E, 0x44, 0x64
};
            send(ukaz);
        }

        private void send(byte[] data)
        {
            try
            {
                byte checksum = 0xFF;
                for (int i = 3; i < data.Length - 1; i++) checksum -=
data[i];
                data[data.Length-1] = checksum;
                serialPort.Write(data, 0, data.Length);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "napaka........");
            }
        }

        private void commit(byte[] data)
        {
            try
            {
                //wrapping data with start byte, length and checksum
                byte[] length = new byte[2];
                length = BitConverter.GetBytes((UInt16)data.Length);
                if (BitConverter.IsLittleEndian)
                {
                    byte tmp = length[0];
                    length[0] = length[1];
                    length[1] = tmp;
                }
                byte[] send = new byte[data.Length+1+length.Length+1];
                send[0] = 0x7E;
```

```
                System.Buffer.BlockCopy(length, 0, send, 1, length.Length);
                System.Buffer.BlockCopy(data, 0, send, length.Length + 1,
data.Length);
                byte checksum=0xFF;
                for (int i = 0; i < data.Length; i++) checksum -= data[i];
                send[1 + length.Length + data.Length] = checksum;

                serialPort.Write(send, 0, send.Length);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "napaka........");
            }
        }

        private void btnOpenPort_Click(object sender, EventArgs e)
        {
            if ((cmbVrata.SelectedIndex == -1)&&(cmbVrata.Text==""))
            {
                MessageBox.Show("Izberite vrata, na katera je povezana
naprava!", "Neveljavna vrata");
                cmbVrata.Focus();
            }
            else
            {
                try
                {
                    serialPort.Close();
                    if (cmbVrata.SelectedIndex == -1) serialPort.PortName =
cmbVrata.Text;
                    else                serialPort.PortName                =
cmbVrata.SelectedItem.ToString();
                    serialPort.Open();
                    lstNaprave.Enabled = true;
                    btnOdkrij.Enabled = true;
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.ToString(),"Izbrana so neveljavna
vrata!");
                }
```

```
            }
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs
e)
        {
            if (serialPort.IsOpen) serialPort.Close();
            lstNaprave.Enabled = false;
            btnOdkrij.Enabled = false;
        }

        private      void      serialPort_DataReceived(object      sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            int length;
            byte[] data={0x00,0x00,0x00,0x00};

            if (serialPort.ReadByte() == 0x7E)
            {
                if (serialPort.BytesToRead >= 2)
                {
                    length     =     256     *     serialPort.ReadByte()     +
serialPort.ReadByte();
                    data = new byte[length + 1];
                    for (int i = 0; i <= length; i++)
                    {
                        while (serialPort.BytesToRead == 0) ;
                        data[i] = (byte)serialPort.ReadByte();
                    }
                    this.BeginInvoke(new SetTextDeleg(si_DataReceived), new
object[] { data });
                }

            }
        }

        private void si_DataReceived(byte[] data)
        {
            System.Text.ASCIIEncoding           enc           =           new
System.Text.ASCIIEncoding();
```

```
            switch (data[0]){
                case 0x88: //AT command response
                    if (data[2] == 'N' && data[3] == 'D')
                    {
                        device tmp = new device();

                        tmp.addrShort = BitConverter.ToUInt16(data, 5);
                        tmp.addrLong = BitConverter.ToUInt64(data, 7);
                        tmp.name = enc.GetString(data, 15, data.Length - 9
- 15);

                        ((ArrayList)bindingDevice.DataSource).Add(tmp);
                        lstNaprave.DataSource = null;
                        lstNaprave.DataSource = bindingDevice;
                    }
                    break;

                case 0x97: //Remote AT Command Response
                    string text = Environment.NewLine; ;
                    text += BitConverter.ToString(data, 1, 1);//frame ID
                    text += "   " + enc.GetString(data, 12, 2);//command
name
                    text += "      " + BitConverter.ToString(data, 14,
1);//status
                    text   +=   ""   +   BitConverter.ToString(data,   15,
data.Length - 15) + Environment.NewLine;
                    textBox.Text += text;
                    break;

                case 0x90: //Dobim podatek iz remote naprave
                    textBox.Text += BitConverter.ToString(data) + ": " +
enc.GetString(data,data.Length-2,1) + Environment.NewLine;
                    break;

                case 0x95: //Node Identification Indicator

                    break;

                default:
                    break;
            }
```

```
        }

        private void Form1_Load(object sender, EventArgs e)
        {

cmbVrata.Items.AddRange(System.IO.Ports.SerialPort.GetPortNames());
            if (cmbVrata.Items.Count > 0) cmbVrata.SelectedIndex = 0;
            ArrayList dev = new ArrayList();
            dev.Add(new device(100, 500, "Test"));
            bindingDevice.DataSource = dev;
            lstNaprave.DataSource = bindingDevice;
        }

        private void btnConnect_Click(object sender, EventArgs e)//nastavi
DH in DL
        {
            device naprava = (device)bindingDevice[bindingDevice.Position];
            System.Text.ASCIIEncoding          enc          =          new
System.Text.ASCIIEncoding();

            byte[] addrLong = BitConverter.GetBytes(naprava.addrLong);
            byte[] addrShort = BitConverter.GetBytes(naprava.addrShort);
            byte[] text = {0xC0, 0xC0};
            //string                        lineText                        =
textBox.Lines[textBox.GetLineFromCharIndex(textBox.SelectionLength)];

            byte[] ukaz = new byte[4 + addrLong.Length + addrShort.Length +
text.Length];
            ukaz[0] = 0x10;//ZigBee transmit request API frame type
            ukaz[1] = 0x01;//FrameID seto to non-zero value
            System.Buffer.BlockCopy(addrLong, 0, ukaz, 2, addrLong.Length);
            System.Buffer.BlockCopy(addrShort,  0,  ukaz,  addrLong.Length  +
2, addrShort.Length);
            ukaz[addrLong.Length + 2 + addrShort.Length] = 0x00;//broadcast
radius
            ukaz[addrLong.Length   +   2   +   addrShort.Length   +   1]   =
0x00;//options
            System.Buffer.BlockCopy(text,  0,  ukaz,  2  +  addrLong.Length  +
addrShort.Length + 1 + 1, text.Length);

            commit(ukaz);
```

```
        }

        private void btnDisconnect_Click(object sender, EventArgs e)
        {
            device naprava = (device)bindingDevice[bindingDevice.Position];
            System.Text.ASCIIEncoding enc = new
System.Text.ASCIIEncoding();

            byte[] addrLong = BitConverter.GetBytes(naprava.addrLong);
            byte[] addrShort = BitConverter.GetBytes(naprava.addrShort);
            byte[] text = { 0xC0, 0xD0 };

            byte[] ukaz = new byte[4 + addrLong.Length + addrShort.Length +
text.Length];
            ukaz[0] = 0x10;//ZigBee transmit request API frame type
            ukaz[1] = 0x01;//FrameID seto to non-zero value
            System.Buffer.BlockCopy(addrLong, 0, ukaz, 2, addrLong.Length);
            System.Buffer.BlockCopy(addrShort, 0, ukaz, addrLong.Length +
2, addrShort.Length);
            ukaz[addrLong.Length + 2 + addrShort.Length] = 0x00;//broadcast
radius
            ukaz[addrLong.Length + 2 + addrShort.Length + 1] =
0x00;//options
            System.Buffer.BlockCopy(text, 0, ukaz, 2 + addrLong.Length +
addrShort.Length + 1 + 1, text.Length);

            commit(ukaz);
        }
    }

public class device{
   public UInt16 addrShort;
   public UInt64 addrLong;
   public string name;
   public UInt16 parentNetworkAddress;
   public char deviceType;//0=coord, 1=router, 2=end device
   public char status;
   public UInt16 profileId;
   public UInt16 manufactorerId;
```

```
public device()
{
    addrShort = 0;
    addrLong = 0;
    name = "";
    parentNetworkAddress = 0;
    deviceType = (char)0;
    status = (char)0;
    profileId = 0;
    manufactorerId = 0;
}

public device(UInt16 addrShort, UInt32 addrLong, string name,
              UInt16 parentNetworkAddress, char deviceType,
              char status, UInt16 profileId, UInt16 manufactorerId)
{
    this.addrShort = addrShort;
    this.addrLong = addrLong;
    this.name = name;
    this.parentNetworkAddress = parentNetworkAddress;
    this.deviceType = deviceType;
    this.status = status;
    this.profileId = profileId;
    this.manufactorerId = manufactorerId;
}

public device(UInt16 addrShort, UInt32 addrLong, string name)
{
    this.addrShort = addrShort;
    this.addrLong = addrLong;
    this.name = name;
    parentNetworkAddress = 0;
    deviceType = (char)0;
    status = (char)0;
    profileId = 0;
    manufactorerId = 0;
}

public override string ToString()
{
    return this.name;
```

```
    }

    public void setShortAddr(UInt16 addr){
        addrShort = addr;
    }

    public void setLongAddr(UInt16 addr)
    {
        addrLong = addr;
    }

    public void setName(string name)
    {
        this.name = name;
    }
    }
}
```

6.1.2. Izvorna koda zunanjega medpomnilnika

```
//                      Target: ATtiny88
#include <avr\pgmspace.h>
#include <avr\io.h>
#include <avr\interrupt.h>
#include "mydefs.h"


#define     BAUD  9600
#define F_CPU 8050000uL
#define BIT_TIME  (u16)(F_CPU * 1.0 / BAUD + 0.5)

#include <util/delay.h>

#define     TX_HIGH           (1<<COM1A1^1<<COM1A0)
#define     TX_LOW            (TX_HIGH^1<<COM1A0)
#define     TX_OUT            TCCR1A            // use compare output mode

#define ROLLOVER( x, max )    x = ++x >= max ? 0 : x
                              // count up and wrap around

#define STX_SIZE  32

#define     STXD        SBIT( PORTB, PB1 )      // = OC1A
#define     STXD_DDR    SBIT( DDRB,  PB1 )

#define     SRXD_PIN    SBIT( PINB,  PB0 )      // = ICP

static u8 stx_buff[STX_SIZE];
static u8 stx_in;
static u8 stx_out;
static u8 stx_data;
static u8 srx_data;
static u8 stx_state;
static u8 srx_state;


void suart_init( void )
{
  DDRC = 0;
```

```
  DDRC |= 0x01;//DDRC |=(1<<DDA0);
  DDRB = 0x00;
  PORTB = 0x03;

  OCR1A = BIT_TIME - 1;
  TCCR1A = TX_HIGH;                 // set OC1A high, T1 mode 4
  TCCR1B = 1<<ICNC1^1<<WGM12^1<<CS10;     // noise canceler, 1-0 transition,
                              // CLK/1, T1 mode 4 (CTC)
  TCCR1C = 1<<FOC1A;
  stx_state = 0;
  stx_in = 0;
  stx_out = 0;
  srx_state = 0;
  STXD_DDR = 1;                     // output enable
  TIFR1 = 1<<ICF1;                  // clear pending interrupt
  TIMSK1 = 1<<ICIE1^1<<OCIE1A;            // enable tx and wait for start
}

int main(){
      suart_init();

      sei();
      while(1);
      return 0;
}

/*****************************     Interrupts
*****************************/

ISR( TIMER1_CAPT_vect )                   // start detection
{
  s16 i = ICR1 - BIT_TIME / 2;                 // scan at 0.5 bit time

  OPTR18                            // avoid disoptimization
  if( i < 0 )
    i += BIT_TIME;                        // wrap around
  OCR1B = i;
  srx_state = 10;
  TIFR1 = 1<<OCF1B;                       // clear pending interrupt
  if( SRXD_PIN == 0 )                     // still low
    TIMSK1 = 1<<OCIE1A^1<<OCIE1B;         // wait for first bit
```

```
}


ISR( TIMER1_COMPB_vect )                        // receive data bits
{
  u8 i;

  switch( --srx_state ){

    case 9:if( SRXD_PIN == 0 )      // start bit valid
             return;
           break;

    default:i = srx_data >> 1;      // LSB first
           if( SRXD_PIN == 1 )
             i |= 0x80;             // data bit = 1
           srx_data = i;
           return;

    case 0:if( SRXD_PIN == 1 ){           // stop bit valid
                 i = stx_in;
                 ROLLOVER( i, STX_SIZE );
                 if( i != stx_out ){           // no buffer overflow
                       stx_buff[stx_in] = srx_data;
                       stx_in = i;             // advance in pointer
                 }
           }
           TIFR1 = 1<<ICF1;              // clear pending interrupt
  }
  TIMSK1 = 1<<ICIE1^1<<OCIE1A;                  // enable next start
}


ISR( TIMER1_COMPA_vect )                        // transmit data bits
{
  if( stx_state ){
    stx_state--;
    TX_OUT = TX_HIGH;
    if( stx_data & 1 )                  // lsb first
      TX_OUT = TX_LOW;
    stx_data >>= 1;
```

```
    return;
  }
  if( stx_in != stx_out ){                // next byte to sent
    PORTC |= 0x01;      //data to send signal to AtMega
      if(bit_is_set(PINC, 1)){      //can send data-signal from AtMega
          stx_data = ~stx_buff[stx_out];
          ROLLOVER( stx_out, STX_SIZE );
          stx_state = 9;                        // 8 data bits + stop
          TX_OUT = TX_LOW;                      // start bit
      }
      return;
  }
  PORTC &=0xFE;   //clear data to send signal to AtMega
}
```

Datoteka »MyDefs.h«:

```
#ifndef _mydefs_h_
#define _mydefs_h_


//                Easier type writing:

typedef unsigned char  u8;
typedef   signed char  s8;
typedef unsigned short u16;
typedef   signed short s16;
typedef unsigned long  u32;
typedef   signed long  s32;


//                Access bits like variables:

struct bits {
  u8 b0:1;
  u8 b1:1;
  u8 b2:1;
  u8 b3:1;
  u8 b4:1;
  u8 b5:1;
  u8 b6:1;
  u8 b7:1;
```

```
} __attribute__((__packed__));

#define SBIT_(port,pin) ((*(volatile struct bits*)&port).b##pin)
#define     SBIT(x,y)   SBIT_(x,y)


//                Optimization improvements


#define     OPTR18 __asm__ volatile ("":::);          // it helps, but why ?
                                    // remove useless R18/R19

// always inline function x:

#define AIL(x)   static x __attribute__ ((always_inline)); static x


// never inline function x:

#define NIL(x)   x __attribute__ ((noinline)); x


// volatile access (reject unwanted removing access):

#define vu8(x)  (*(volatile u8*)&(x))
#define vs8(x)  (*(volatile s8*)&(x))
#define vu16(x) (*(volatile u16*)&(x))
#define vs16(x) (*(volatile s16*)&(x))
#define vu32(x) (*(volatile u32*)&(x))
#define vs32(x) (*(volatile s32*)&(x))


#endif
```

6.1.3. Izvorna koda glavnega procesorja

```
/*****************************************
* Chip type           : ATmega2560
* Clock frequency     : 8,000000 MHz
*****************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>


#define BAUDRATE 9600
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)

#define STOP_TIMER    TCCR0B &= 0xF8
#define START_TIMER   TCCR0B |= 0x05

#define setMux(pin,mask) ((PORTA) |= (mask)<<(pin))              //pin=
location(0,2), mask=mask to set
#define clearMux(pin,mask) ((PORTA) &= ~((mask)<<(pin)))   //pin=
location(0,2), mask=mask to clear

#define txBuffSize      512
#define rxBuffSize      512
#define maxBeePacketSize      8

#define txMask    (txBuffSize-1)
#define rxMask    (rxBuffSize-1)


volatile char currentOutputMux=0;
volatile int currentInputMux=0;
volatile int sendToDevice = 0;
volatile int wait = 128;

volatile unsigned char txBuff[5][txBuffSize];  // TX buffer
volatile unsigned char rxBuff[rxBuffSize];     // RX buffer

volatile unsigned char tx_in[5];         //buffer0 head/tail indicies
volatile unsigned char tx_out[5];
```

```
volatile unsigned char rx_in;
volatile unsigned char rx_out;

volatile unsigned char checksum[5]={0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
volatile    unsigned    char    currentPacketSize[5]={maxBeePacketSize+17,
maxBeePacketSize+17,        maxBeePacketSize+17,        maxBeePacketSize+17,
maxBeePacketSize+17} ;  //transmitting packet size used to determine header
and footer for xBee packet
volatile unsigned char lastPacketSize=0;
volatile unsigned char length;
volatile unsigned char addrLong=0x08;
volatile unsigned char addrH,addrL;
volatile unsigned char commandTmp;
volatile unsigned char adrLong[8];


volatile struct devices{
      unsigned char id;
      unsigned char addrH;
      unsigned char addrL;
      unsigned char addrLong[8];
} device[5];

typedef enum{
    START,
    LENGTH1,
    LENGTH2,
      API_IDENTIFIER,
    ADDR_LONG,
    ADDR_SHORTH,
    ADDR_SHORTL,
    OPTIONS,
    IS_DATA,
    DATA,
    COMMAND,
    CHECKSUM,
      CON_REQ,
      DCON_REQ,
} reciveStateT;

reciveStateT reciveState;
```

```
void timerInit(){
	TCNT0 = 0x00;
	TIFR0 |= (1<<TOV0);
	TIMSK0 |= (1<<TOIE0);
}

void init(void){
	DDRA=0;
	DDRA   =(1<<DDA0)|(1<<DDA1)|(1<<DDA2)|(1<<DDA3);   //Pins   0-3   are
outputs, 4-7 inputs
                                                        //  Pins  0,1  control
output mux; 2,3 control input mux
                                                        //  Pins  4-7  are  data
ready requests from buffers
	PORTA = 0xF0;     //initalize  pull-up  resistors  for  input  pins  and
sets input and output mux to 0
	currentOutputMux=0;
	currentInputMux=0;

	DDRE &= 0b11000011;
	DDRE |= 0b00111100;

	timerInit();

	int i;
	for(i=0;i<4;i++){
		device[i].id=0xFF;
		device[i].addrL=0xFF;
		device[i].addrH=0xFF;
		for(int j=0;j<8;j++)device[i].addrLong[j]=0xFF;
	}
}

void usartInit(){

	//////////////////////////
	/////// USART 0 ////////////
	//////////////////////////
	//Set baud rate
	UBRR0=UBRRVAL;
```

```
	//Set data frame format: asynchronous mode,no parity, 1 stop bit, 8
bit size
	UCSR0C=(0<<UMSEL01)|(0<<UMSEL00)|(0<<UPM01)|(0<<UPM00)|(0<<USBS0)|(0<
<UCSZ02)|(1<<UCSZ01)|(1<<UCSZ00);

	//Enable Transmitter and Receiver and Interrupt on receive complete

	UCSR0B=(1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);

	////////////////////////////
	/////// USART 1 ////////////
	////////////////////////////
	//Set baud rate
	UBRR1=UBRRVAL;

	//Set data frame format: asynchronous mode,no parity, 1 stop bit, 8
bit size
	UCSR1C=(0<<UMSEL11)|(0<<UMSEL10)|(0<<UPM11)|(0<<UPM10)|(0<<USBS1)|(0<
<UCSZ12)|(1<<UCSZ11)|(1<<UCSZ10);

	//Enable Transmitter and Receiver and Interrupt on receive complete

	UCSR1B=(1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1);
}

int main(void)
{
	init();
	usartInit();
	sei();
	START_TIMER;	// starts timer counter 0
	while(1){
		while (bit_is_set(PINA, 4)){ //data request from buffer0
			currentInputMux=0;
			PORTA=0xF0;
			PORTE |= 0b00000100;
		}
		PORTE &= 0b11000011;

		while (bit_is_set(PINA, 5)){ //data request from buffer1
```

```
                currentInputMux=1;
                PORTA=0xF1;
                PORTE |= 0b00001000;
            }
            PORTE &= 0b11000011;

            /*while (bit_is_set(PINA, 6)){ //data request from buffer2
                currentInputMux=2;
                PORTA=0xF2;
                PORTE |= 0b00010000;
            }
            PORTE &= 0b11000011;

            while (bit_is_set(PINA, 7)){ //data request from buffer3
                currentInputMux=3;
                PORTA=0xF3;
                PORTE |= 0b00100000;
            }
            PORTE &= 0b11000011;*/
        }
        return 0;
}

ISR(USART0_RX_vect){    // USART0 recive
        txBuff[currentInputMux][tx_in[currentInputMux]   &   txMask]   =   UDR0;
        //adds received data to send buffer 1
        tx_in[currentInputMux]++;           // increment head index

        if(tx_in[currentInputMux]       -       tx_out[currentInputMux]       >=
maxBeePacketSize && currentPacketSize[sendToDevice]>=maxBeePacketSize+17){
            sendToDevice=currentInputMux;
            UCSR1B |= (1<<UDRIE1);        // Enable UDR empty interrupt
        }
}

ISR(USART1_UDRE_vect ) {  // USART1 transmit Data Register Empty
        // proccesses send buffer x.
        if(currentPacketSize[sendToDevice]==0){   //we   reached   max   packet
size, so we finalize it
            UDR1=checksum[sendToDevice];
            checksum[sendToDevice]=0xFF;
```

```
            currentPacketSize[sendToDevice]=maxBeePacketSize+17;
            UCSR1B &= ~(1<<UDRIE1);
      }
      else{
            if(currentPacketSize[sendToDevice]>maxBeePacketSize           &&
sendToDevice!=4){ //we are starting a new packet
                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+17){
                        UDR1 = 0x7E;
                  }else
                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+16){
                        UDR1 = 0x00;
                  }else
                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+15){
                        if(tx_in[sendToDevice]   -   tx_out[sendToDevice]   <=
maxBeePacketSize){
                              lastPacketSize=tx_in[sendToDevice]           -
tx_out[sendToDevice];
                        }else{
                              lastPacketSize=maxBeePacketSize;
                        }
                        UDR1 = lastPacketSize+4+8+2;
                  }else
                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+14){
                        UDR1 = 0x10;      //ZigBee   transmit   request   API
frame type
                        checksum[sendToDevice]-=0x10;
                  }else
                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+13){
                        UDR1 = 0x02;      //FrameID set to to non-zero value
                        checksum[sendToDevice]-=0x02;
                  }else


                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+12){
      //addrLong
                        UDR1 = device[sendToDevice].addrLong[7];
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[7];
                  }else
if(currentPacketSize[sendToDevice]==maxBeePacketSize+11){
                        UDR1 = device[sendToDevice].addrLong[6];
```

```
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[6];
                  }else
if(currentPacketSize[sendToDevice]==maxBeePacketSize+10){
                        UDR1 = device[sendToDevice].addrLong[5];
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[5];
                  }else
if(currentPacketSize[sendToDevice]==maxBeePacketSize+9){
                        UDR1 = device[sendToDevice].addrLong[4];
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[4];
                  }else
if(currentPacketSize[sendToDevice]==maxBeePacketSize+8){
                        UDR1 = device[sendToDevice].addrLong[3];
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[3];
                  }else
if(currentPacketSize[sendToDevice]==maxBeePacketSize+7){
                        UDR1 = device[sendToDevice].addrLong[2];
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[2];
                  }else
if(currentPacketSize[sendToDevice]==maxBeePacketSize+6){
                        UDR1 = device[sendToDevice].addrLong[1];
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[1];
                  }else
if(currentPacketSize[sendToDevice]==maxBeePacketSize+5){
                        UDR1 = device[sendToDevice].addrLong[0];
                        checksum[sendToDevice]-
=device[sendToDevice].addrLong[0];
                  }else

                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+4){
                        UDR1 = device[sendToDevice].addrH;
                        checksum[sendToDevice]-=device[sendToDevice].addrH;
                  }else
                  if(currentPacketSize[sendToDevice]==maxBeePacketSize+3){
                        UDR1 = device[sendToDevice].addrL;
                        checksum[sendToDevice]-=device[sendToDevice].addrL;
```

```
                }else

                if(currentPacketSize[sendToDevice]==maxBeePacketSize+2){
      //brodcast radius
                      UDR1 = 0x00;
                }else{
                                  // options
                      UDR1 = 0x00;
                      currentPacketSize[sendToDevice]=lastPacketSize+1;
                }
                currentPacketSize[sendToDevice]--;
          }
          else{ //filling the data part of the package
                UDR1    =    txBuff[sendToDevice][tx_out[sendToDevice]    &
txMask];
                checksum[sendToDevice]-
=txBuff[sendToDevice][tx_out[sendToDevice] & txMask];
                tx_out[sendToDevice]++;  // txMask makes it unnecessary to
range limit this
                if(sendToDevice==4 && tx_out[4]==tx_in[4]){
                      currentPacketSize[4]=0;
                      checksum[sendToDevice]+=0x8E;
                }
                else currentPacketSize[sendToDevice]--;
          }
      }
      TCNT0 = 0x00;     //resets timed checking for data*/
}

ISR(TIMER0_OVF_vect){   // checks for pending data in TX buffer for xBee
      if(wait==0){
          for(int i=0;i<=3;i++){
                if(tx_in[i] != tx_out[i]){
                      sendToDevice=i;
                      UCSR1B |= (1<<UDRIE1); //    Enable    UDR    empty
interrupt
                      return;
                }
          }
          wait = 128;
      }else{
```

```
            wait--;
      }
}


ISR(USART1_RX_vect){    //recive xBee data/command
      if(reciveState==START){
            unsigned char tmp=UDR1;

            if(tmp == 0x7E){
                  reciveState=LENGTH1;
            }
      }
      else
      if(reciveState==LENGTH1){
            length = UDR1;
            reciveState=LENGTH2;
      }
      else
      if(reciveState==LENGTH2){
            length = UDR1;
            reciveState=API_IDENTIFIER;
      }
      else
      if(reciveState==API_IDENTIFIER){
            unsigned char apiIdentifier=UDR1;

            if(apiIdentifier==0x90){
                  reciveState=ADDR_LONG;
            }else{
                  reciveState=START;
            }
            length--;
      }
      else
      if(reciveState==ADDR_LONG){
            unsigned char tmp=UDR1;

            addrLong--;
            adrLong[addrLong]=tmp;
            if(addrLong==0){
```

```
                reciveState=ADDR_SHORTH;
                addrLong=8;
            }
            length--;
      }
      else
      if(reciveState==ADDR_SHORTH){
            addrH=UDR1;
            reciveState=ADDR_SHORTL;
            length--;
      }
      else
      if(reciveState==ADDR_SHORTL){
            addrL=UDR1;
            reciveState=OPTIONS;
            length--;
      }
      else
      if(reciveState==OPTIONS){
            unsigned char tmp=UDR1;

            //set the sending device
            int i=0;
            while(i<4          &&          (addrL!=device[i].addrL          ||
addrH!=device[i].addrH)){
                  i++;
            }
            currentOutputMux=i;
            reciveState=IS_DATA;
            length--;
      }
      else
      if(reciveState==IS_DATA){     //is incomming data or command?
            volatile unsigned char tmp=UDR1;

            length--;
            if(tmp==0xC0){    //if it's command
                  reciveState=COMMAND;
            }
            else if(currentOutputMux!=4){ //we have data incomming
                  rxBuff[rx_in & rxMask] = currentOutputMux;
```

```
                  rx_in++;
                  rxBuff[rx_in & rxMask] = tmp; //adds   received   data   to
recive buffer 1
                  rx_in++;
                  if(length==0){
                        reciveState=CHECKSUM;
                  }
                  UCSR0B |= (1<<UDRIE0);  // Enable UDR empty interrupt
            }else{      //data request was send from unpaired device
                  reciveState=START;
            }
      }
      else
      if(reciveState==COMMAND){
            //parse command
            unsigned char tmp=UDR1;

            if(tmp==0xC0){    //connect request
                  int i=0;
                  while(i<4  &&  (device[i].addrL  !=  0xFF  ||  device[i].addrH
!= 0xFF)){
                        i++;
                  }

                  txBuff[4][tx_in[4] & txMask]=0x7E;  //start
                  tx_in[4]++;
                  txBuff[4][tx_in[4] & txMask]=0x00;  //length msb
                  tx_in[4]++;
                  txBuff[4][tx_in[4] & txMask]=16;    //length lsb
                  tx_in[4]++;
                  txBuff[4][tx_in[4] & txMask]=0x10;  //api identifier
                  tx_in[4]++;
                  txBuff[4][tx_in[4] & txMask]=0x03;  //frame id
                  tx_in[4]++;
                  for(int j=7;j>=0;j--){                    //long address
                        txBuff[4][tx_in[4] & txMask]=adrLong[j];
                        tx_in[4]++;
                  }
                  txBuff[4][tx_in[4] & txMask]=addrH; //short address msb
                  tx_in[4]++;
                  txBuff[4][tx_in[4] & txMask]=addrL; //short address lsb
```

```
                  tx_in[4]++;
                  txBuff[4][tx_in[4] & txMask]=0x00;  //broadcast radius
                  tx_in[4]++;
                  txBuff[4][tx_in[4] & txMask]=0x00;  //options
                  tx_in[4]++;

                  if(i!=4){
                        for(int
j=0;j<8;j++)device[i].addrLong[j]=adrLong[j];
                        device[i].addrL=addrL;
                        device[i].addrH=addrH;
                        txBuff[4][tx_in[4] & txMask]=0;//'O';
                        tx_in[4]++;
                        txBuff[4][tx_in[4] & txMask]=i;
                        tx_in[4]++;
                  }else{
                        txBuff[4][tx_in[4] & txMask]='N';
                        tx_in[4]++;
                        txBuff[4][tx_in[4] & txMask]='O';
                        tx_in[4]++;
                  }
                  sendToDevice=4;
                  UCSR1B |= (1<<UDRIE1);
                  TCNT0 = 0x00;
            }else if(tmp==0xD0){    //disconnect request
                  int i;
                  for(i=0;i<4;i++){
                        if(addrL==device[i].addrL                              &&
addrH==device[i].addrH){
                              device[i].addrL=0xFF;
                              device[i].addrH=0xFF;
                              for(int
j=0;j<8;j++)device[i].addrLong[j]=0xFF;
                              break;
                        }
                  }
            }
            reciveState=START;
      }
      else{ //if nothing else, it must be checksum
            unsigned char tmp=UDR1;
```

```
            reciveState=START;
      }
}

ISR(USART0_UDRE_vect ) {      //sends data to specified end device
      if(rx_in != rx_out){
            clearMux(2,3);
            setMux(2,rxBuff[rx_out & txMask]);
            rx_out++;
            UDR0 = rxBuff[rx_out & rxMask];
            rx_out++;
      }else{
            UCSR0B &= ~(1<<UDRIE0);
      }
}
```

Literatura

1. Tone Vidmar, Informacijski komunikacijski sistemi, Ljubljana, Pasadena, 2002, stran 144
2. (2008) Wireless sensor networks 802.15.4. vs ZigBee. Dostopno na: http://sensor-networks.org/index.php?page=0823123150
3. (2008) Wireless sensor networks Security in 802.15.4 and ZigBee networks. Dostopno na: http://sensor-networks.org/index.php?page=0903503549
4. (2005) ZigBee Technical Documents. Dostopno na: http://www.zigbee.org/Products/DownloadZigBeeTechnicalDocuments.aspx
5. (2010) xBee Product Literature. Dostopno na: http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-module.jsp?pub=CircuitCellar_xbee_prog&utm_source=CIrcuitCellar_2010&utm_medum=print&utm_campaign=CircuitCellar_print_xbee_prog_0510#docs
6. (2009) ZigBee Faq to learn about the ZigBee standard, ZigBee stack, topologies, devices and more. Dostopno na: http://www.meshnetics.com/zigbee-faq/#22
7. (2010) How RS232 works. Dostopno na: http://www.best-microcontroller-projects.com/how-rs232-works.html
8. (2008) ZIGBEE specification. Dostopno na: http://specifications.nl/zigbee/zigbee_UK.ph