LUNDS UNIVERSITET

Lunds Tekniska Högskola

Department of Information Technology

Digitala Projekt SK

Trådlöst nätverk med AVR

Stefan Skoog E04

Joakim Arnsby E04

Handledare: Bertil Lindvall

29 maj 2007

Abstract

The use of small intelligent radio units is growing big in the measurment area. The requirements of both long range and low power consumption is somewhat hard to achieve. The use of small short-range units in a wireless mesh network could be one solution. In this project, a wireless ad-hoc network with small nodes is developed and implemented with Atmel AVR microcontrollers and nRF2401 radio modules. Three nodes where built and tested as a network during this project.

Innehållsförteckning

1 Inledning

Små intelligenta och självständiga trådlösa enheter börjar bli riktigt populära inom mät- och datainsamlingstillämpningar. Ett problem med dessa är räckvidden, som begränsas både enligt lag och kompromisser i design för att få lägre ström-förbrukning. När antalet enheter inom en begränsad yta ökar finns det en möjlighet för enheterna att utnyttja varandra och skapa ett nätverk för att överföra information från punkt A till punkt B. Två enheter måste inte nödvändigtvis vara i direktkontakt med varandra för att överföra information mellan varandra. Med denna metod finns det potential att begränsa sändeffekt och på så vis öka livslängden för varje enhet.

Vi har i detta arbete konstruerat och implementerat ett protokoll för att kunna studsa radiotrafik mellan enheter. Implementeringen ser till att varje enhet vet hur nätverket ser ut och på så vis har en chans att välja den bästa vägen till målet redan vid tidpunkten för avsändning.

Projektet genomfördes i kursen Digitala projekt större kurs (ETI022) vid Lunds Tekniska Högskola vårterminen 2007.

1.2 Problem

Protokollet ska implemeteras med små enheter med begränsat minne och liten bandbredd. Ett radioprotokoll kräver mycket arbetsminne för att fungera bra, vilket är begränsat i de mikrokontrollers vi valt att använda.

Protokoll består av många mjukvarulager och det kommer därför dels bli mycket kod att hålla reda på, och dels kommer det behövas mycket programminne hos målenheten.

1.3 Avgräsning

Vi använder oss av färdiga radiomoduler med seriellt gränssnitt och kommer därför inte gå djupare in i hur dessa fungerar rent radiomässigt. Genomströmnings-hastigheten eller bandbredden är inget vi kommer fokusera på. Vi har i protokollet begränsat maximalt antal noder i nätverket till 16 st för att spara minne. De största datamängd man kan överföra per paket är 25 byte.

2 Teori

Principen bakom vårt nätverksprotokoll går i linje med de befintliga topologierna ”mesh networking” och ”wireless ad-hoc network”. Den stora skillnaden är att varje enhet i förväg ska veta ungefär vilken väg den ska skicka sin data, för att undvika att översvämma nätverket med redundant data varje gång ett paket behöver studsas. Genom att kontinuerligt sända ut en speciell *broadcast* har varje enhet en möjlighet att erhålla en bild av hur nätverket ser ut. På detta vis vet den alltid vilka andra enheter den kan nå, både direkt och indirekt.

2.1 Paketen

Vi har försökt göra protokollens overlay så liten som möjligt, eftersom radiomodulens paketlängd är 29 byte. Efter en hel del funderande fastställde vi paketens header till 3 byte, eller 6 st 4-bits ord. Anledningen till indelningen i 4-bits ord är att alla noders adress kan representeras av just 4 bitar (16 kombinationer). Uppställningen för ett normalt datapaket är följande:

```
Byte        Bit
     7 6 5 4  3 2 1 0
 0  [SEND_ID][ REC_ID]
 1  [ VIA_ID][ DEEP  ]
 2  [A][ Res][LAST_ID]
 3  [   data byte0   ]
 4  [   data byte1   ]
 5  [   data byte2   ]
 ...
28  [   data byte26  ]
```

Send_ID:	Avsändarens ID.
Rec_ID:	Mottagarens ID.
Via_ID:	Mellanhandens ID.
Deep:	Räknare för antalet gånger paketet studsat.
A:	Flagga för ack eller ny data.
Res:	Reserverade bitar.
Last_ID:	ID för den enhet som sist hanterat paketet.

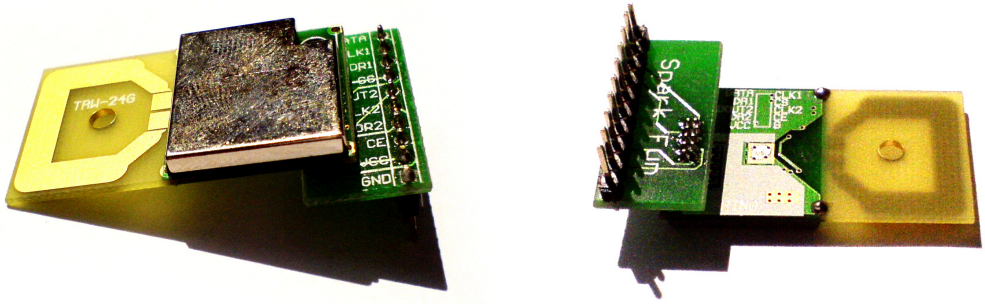
Utöver datapaketen finns även speciella paket för *broadcast.* Dessa identifieras genom att sändaridentiteten är samma som mottagaridentiteten i paketets huvud. Ett broadcastpaket hanteras olikt datapaketen eftersom det innehåller en routingtabellen från sändaren som data. Denna infromation kan extraheras av mottagaren för att uppdatera sin egen routingtabell. En förutsättning för att protokollet ska fungera är att *broadcast*-paket skickas kontinuerligt oavsett annan trafik.

3 Genomförande

3.1 Hårdvara

3.1.1 Radio

Huvudhårdvaran i projektet är radiomodulerna TRW-24G. Dessa jobbar i 2.4 GHz bandet och varje modul kan både sända och ta emot data (trasceiver). Varje enhet har en inbyggd antenn vilket ger små och kompakta moduler. Kommunikation med modulerna görs via 3-tråds gränssnittet. Mottagen data meddelas via en separat pinne som kan användas för att trigga t.ex ett avbrott. Vid uppstart av modulerna skickas konfigurationsdata till var och en. Denna ställer in vilken frekvens modulen ska använda, modulens adress, CRC-check, uteffekt och datamängd. Modulerna kan jobba på två olika sätt, *Direct Mode Transmit and Receive* eller *Shockburst Mode*.



Figur 1: Radiomodulerna TRW-24G från sparkfun.com, här med fastlödd laborationsadapter.

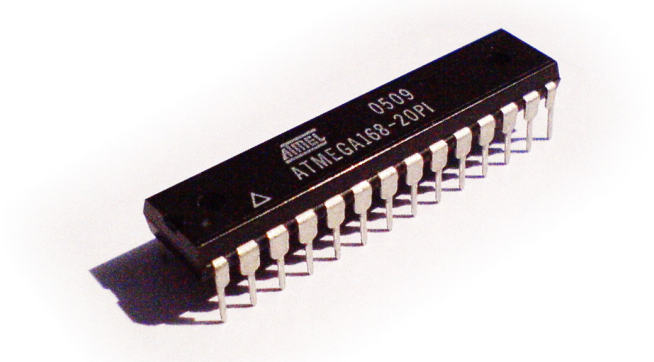
Vid användning av *Direct Mode* kommer datan ut ur mottagaren direkt i samma hastighet som den skickades in till sändaren. Nackdelen här är att man måste ha en snabb mikrokontroller för att kunna utnyttja full bandbredd och där finns ingen inbyggd kontroll mot skräpdata.

I *Shockburst Mode* mellanlagras datan som ska sändas i ett FIFO-minne som kan ta emot data i hastigheter upp till 1 Mbps. När modulen fått in tillräckligt med data för att fylla bufferten skickas datan iväg i högsta möjliga hastighet. Fördelen med detta är lägre energiåtgång och minskad risk för datakollision under sändning. För att försäkra att bara giltig data tas emot av mottagaren räknas CRC-summor ut av den mottagna datan och jämförs med mottagna summor.

3.1.2 Mikrokontroller

I projektet används Atmels 8-bit mikrokontroller AVR, dessa finns i ett stort utbud av storlekar och innehåller många funktioner. Det finns tre olika familjer av AVRer. TinyAVR, MegaAVR och applikationspecifika AVRer. Tiny-familjen är relativt små: 1-8 kB programminne i 8-20 pins kapslar. Mega-serien är större med minnesalternativ mellan 4-256 kb i 28-100 pins kapslar. Nästa serie, applikationsspecifika AVRer, är snarlika Mega-serien men innehåller en del specialfunktioner såsom LCD-kontroller, USB-controller mm.

I projektet används främst Atmega168 då denna har tillräckligt med minne för programkoden och lagom fysisk storlek för att göra små radiomoduler.



Figur 2: En AVR Mega168 från Atmel, den mikrokontroller vi använder i projektet.

3.1.3 Spänningsreglering

Då många delar i projektet är byggda för 3.3 V matning måste det garanteras att de matas med en stabil källa som aldrig överstiger detta värde. Detta ordnas genom att använda en spänningsregulator. För att få hög flexibilitet i systemet används spänningsregulatorer med lågt spänningsfall över sig. Detta för att kunna mata modulerna med så låg spänning som möjligt då de drivs med batteri. Högre spänning på batteriet ger ju högre förluster, högre vikt och större volym.

3.1.4 Accelerometer

Som ett sensorelemet på en av modulerna användes en 3-axlig accelerometer (MMA7260 från Freescale). Denna kan ställas in i 4 olika känslighetsgrader 1.5 g, 2 g, 4 g eller 6 g via två konfigurationspinnar. Accelerationen i de olika axlarna representeras med tre analoga spänning, som lätt läses av med mikrokontrollerns integrerade AD-omvandlare.

3.1.5 Seriell kommunikation

I Atmels AVR finns det en hårdvarumodul som tar hand om seriell kommunikation, USART (Universal Synchronous and Asynchronous Serial Receiver and Transmitter). Modulen kan ställas in för att matcha de flesta system vad gäller datalängd, paritet och hastighet (baudrate). Efter att USARTen är konfigurerad sköter den sig själv i bakgrunden med lite stöd av avbrottshantering i mjukvaran, huvudprogrammet kan då exekveras nästintill ostört.

Normala nivåer på en RS-232 signal är +3 till +12 V för en nolla och -3 till -12 V för en etta. En avr ger 5 V för en etta och 0 V för nolla. Detta är både i fel storlek och inverterat jämför med RS-232. För att kunna koppla in sig till datorns COM-port används kretsen MAX202. Denna inverterar signalen och höjer amplituden.

Eftersom det behövs seriekommunikation mellan PC och AVR för åtminstone två enheter i vårt projekt byggde vi USB till seriell konverterare av några FT232-chip.

3.1.6 USB

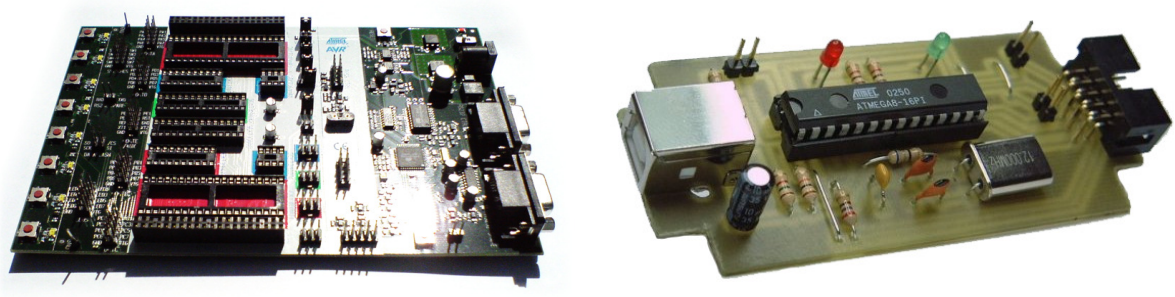
För att koppla projektet till datorns USB-port används FTDIs FT232BM. Detta ger en simulerad COM-port i datorn vilket förenklar mjukvaruutvecklingen. Den ger även ut rätt nivåer för att direkt anslutas till AVRens USART.

3.1.7 Programmerare

Som gränssnitt mellan PC och AVR för programmering användes främst STK500 och USBasp.

STK500 (se figur 3) är ett utvecklingskort gjort av Atmel som tillåter användaren att utveckla projekt antingen direkt i socklar på utvecklingskortet eller på externa kort via ICSP. Vi använde oss endast av ICSP, d.v.s. programmering av AVR direkt i målsystemet.

USBasp är en USB-kopplad ICSP-programmerare med öppen källkod vilket ger möjlighet till optimering av användandet och bra stöd för arbete i Linuxmiljö.



Figur 3: Vänster: Atmel STK500. Höger: USBasp (www.fischl.de/usbasp).

3.1.8 Lysdiod

För att ge ett enkelt gränssnitt mot användaren används lysdioder av flera färger. Olika färger lyser eller blinkar beroende på vad modulen gör för tillfället, exempelvis grön för sändning, blå för mottagning och röd för fel.

3.1.9 Joystick

Som givare till en av modulerna användes en Wingman Extreme Digital-joystick. För att underlätta avläsningen av denna byttes orginalelektroniken ut mot en AVR Mega88 så det gick att upprätta en SPI-överföring mellan denna och huvudenheten. Via Mega88ans inbyggda AD-omvandlare läses de tre potentiometrarna av som ger X-, Y- och gasspakens kordinater. De tio knapparna läses av via I/O pinnar.

3.1.10 Moduler

Tre stycken radioenheter byggdes för att laborera med. Två av dem är byggda på batterimatade kort, den sista på experimentbräda. Varje enhet består av en Mega168 som styr radiokommunikationen via radiomodulen, samt läser av sensorer och andra indataenheter som är kopplade till modulerna. En RGB-diod är monterad på varje

modul för att indikera att den är igång och visar vad som görs för tillfället. Kontakter för seriell kommunikation och ICSP-programmering används för snabb utveckling och felsökning av de olika noderna. Ett enkelt schema för uppkopplingn finns i appendix.

3.2 Mjukvara

3.2.1 Programmeringsmiljö

Det är viktigt med en ordentlig integrerad programmeringsmiljö (IDE) när man ska ge sig på större programmeringsuppgifter, därför valde vi att koda i språket C och att använda det fria paketet WinAVR version 20060421 (avr-gcc 3.4.6, avr-libc 1.4.4, Programmers Notepad 2.0.6) och AVR Studio 4.12 för arbetet i Windows. För Linux finns samma kompilatorverktyg, men som IDE användes K Develop 3.3.5 och AVR dude 5.2.

3.2.2 Debugging

I ett stort system där flera separata mikrokontrollers ska jobba tillsammans är det viktigt att ha ett bra debuggsystem för att snabbt kunna lokalisera fel. I högnivåspråk såsom java och C++ finns det bra inbyggda debuggers som ger felkoder som kan härledas till var problemet uppstått. När man kommer ner i lägre nivåer som C och Assembler syns oftast inte felen direkt. De leder oftast till att programmet stannar eller ger felaktiga värden.

Det finns ett par olika debuggsystem som ger programmeraren en översikt av vad som händer i processorn. Till Atmels 8-bitars mikrokontroller AVR som använts i projektet finns ett vanligt debuggsystem som heter JtagIce MkII. Denna ger användaren möjligheter att rad för rad exekvera programkoden i processorn samtidigt som både minnesvärden och I/O-status kan granskas. En av nackdelarna med detta är att mjukvaran för PC-sidan har begränsad kompatiblitet med olika operativsystem. Vid tillfället projektet utfördes fanns endast stöd för Windows vilket ställde till en del besvär. Därför valde vi att använda ett lite enklare debuggsystem som inte tillåter rad-för-rad-exekvering utan istället rena textutskrifter till PC via UART. Dock är detta ett väldigt kraftfullt system då det fungerar under alla operativsystem utan dyr program- och hårdvara. Der går år en hel del programminne hos mikrokontrollern för att lagra dessa utskrifterna, därför har vi tilldelat en eller flera debug-nivåer för varje mjukvarumodul som går att stänga av då man inte vill veta något om dess exekveringsstatus, på så sätt lagras endast de utskrifter som används i programminnet.

3.2.3 Struktur på mjukvaran

Det är väldigt viktigt att ha god struktur på sitt arbete då man försöker implementera ett nytt protokoll. Vi tittade på lite befintliga modeller, exempelvis OSI-modellen, och konstaterade att det blev svår att följa eftersom radiomodulens integrerade funktioner redan strecker sig över de två nedersta lagren. Däremot har vi behållt idén om att

bygga protokollet i moduler för att få en större överskådlighet och göra det lättare att bygga ut systemet vid behov.

3.2.4 Radioprotokollet

Radioprotokollet består av fyra moduler med bestämd hierarki; alla anrop sker från lagret ovanför eller från samma nivå. Endast information utbytes uppåt via buffrar eller returvärden. Vidare finns det sju moduler som hanterar övriga funktioner i projektet. Ett flödesschema över mjukvarumodulerna finns i appendix. Här följer en förklaring för varje modul:

- **RF_SPI** 117 + 20 rader

  Emulerar en SPI-port mot radiomodulen. Möjliggör sändning och mottagning av godtyckligt stora datablock.

- **RF0** 424 + 118 rader

  Abstraherar gränssnittet mot radiomodulen till få och enkla metoder. Radiomodulen kan liknas vid en tillståndsmaskin med ganska invecklade metoder för att byta tillstånd, därför är detta lager viktigt för att göra det vidare arbetet lättare.

- **RF1** 672 + 168 rader

  Här finns metoder för att hantera in- och utgående data från radiomodulen via RF0-lagret. Detta lager sköter ömsesidig kommunikation med routingtabellen och hanterar även all overlay för datapaketen.

- **RF2** 538 + 68 rader

  Detta lager samordnar buffertar och sköter återsändning av paket som inte nått mottagaren korrekt. Den viktigaste metoden här är pollande och måste köras kontinuerligt. Vill man använda en realtidskärna så är det här man ska ändra.

- **buffer** 384 + 83 rader

  Innehåller tre buffrar, inbuffert 1 & 2 samt utbuffert. Inbuffert 1 används för att lagra rådata från radiomodulen som matas in via en avbrottsrutin. Inbuffert 2 lagrar bearbetad data som är klar att användas för ett enventuellt nästa lager av protokollet. Utbuffert är den största, som lagrar bearbetad utdata *och* paket som väntar på att bekräftas (ack) från mottagaren.

- **LEDs** & **timer** 117 + 19 rader

  Timern är tidsreferensen för hela protokollet. Via timerns avbrottshanterare uppdateras alla tidsberoende variabler, såsom timeout för ack och timeout i routingtabellen. Timern styr även lysdioderna. En så enkel sak som att blinka en lysdiod har vi gjort helt avbrottstyrt eftersom det inte finns någon tid att använda en fördröjningsloop i huvudprogrammet.

- **commons** 162 + 4 rader

Här defineras alla modulers gemensamma funtioner som inte har med radioprotokollet att göra, exempelvis hur inkommande UART-data ska hanteras och vad som händer vid odefinerade avbrottsanrop (felhantering).

- **joystick** 56 + 14 + 123 rader

  En enkel samling metoder på radiomodulsidan som gör det möjligt att kommunicera med vår något modifierade joystick.

- **IO** 114 rader

  Abstrahering av mikrokontrollerns I/O-portar och definiering av alla använda pinnar.

- **main** 175 rader

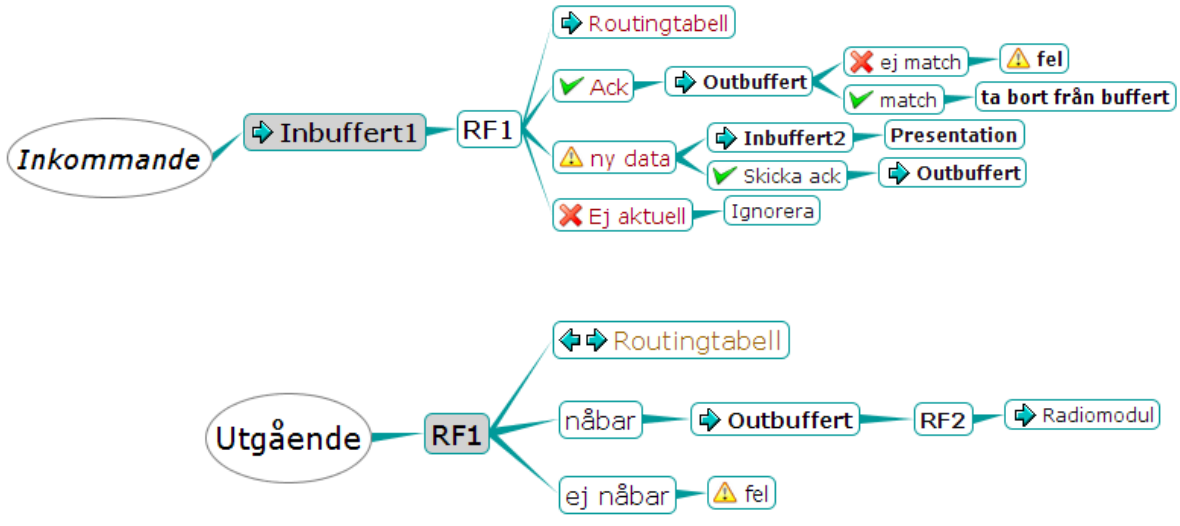
  Huvudmetoden som initierar alla andra moduler och sedan kontinuerligt utför alla nödvändiga anrop, binder ihop radioprotokollet med övriga funktioner (UART, sensoravläsning och joystick).

- **ADC** 303 + 26 rade

  Ett större bibliotek för att hantera flera analog till digital-kanaler i hög hastighet och full upplösning (10 bitar). Avbrottshanterade alternativt pollande metoder.

- **UART** 621 + 198 rader

  Helt avbrotsstyrd UART för både in- och utgående data. Ringbuffert med valbar storlek och stöd för ett brett utbud av AVR-modeller.



Figur 4: Flödesschema för inkommande respektive utgående datapeket.

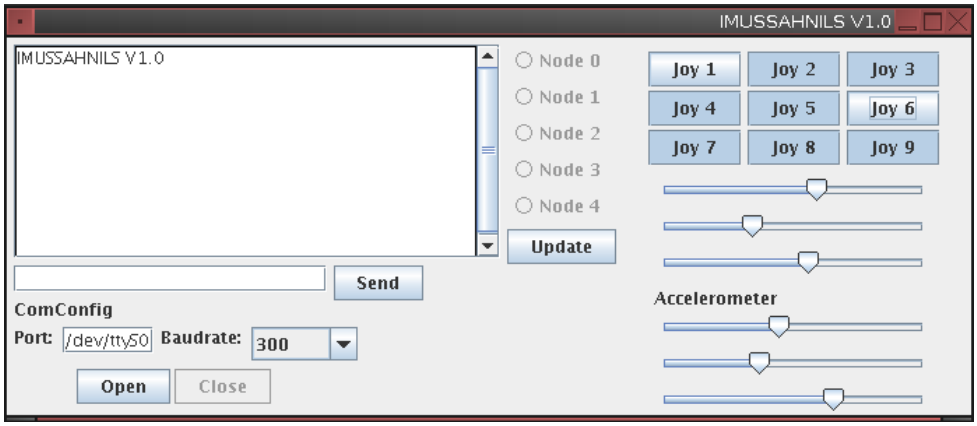
Det viktigaste i protokollets uppbyggnad är att all trafik passerar metoderna i lagret RF1. För inkommande trafik, oavsett datans relevans, kan alltid informationen i huvudet (overhead) plockas ut och användas för att uppdatera routingtabellen.

I figur 4 syns övergripande hur inkommande respektive utgående paket hanteras. Specielfallet broadcast-paket har inte tagits med för att hålla illustrationerna enkla.

Ett bekräftande paket (ack) skickas alltid ut när en enhet får in ett nytt datapaket. Avsändaren sparar sitt utgående meddelande tills det förfallit enligt tidsstämpeln eller det fått ett bekräftande ack. Innan paketet förfaller omsänds det många gånger (ställbart med parameter för period och max antal omsädningar).

3.2.5 PC-mjukvara

För att på ett smidigt sätt testa modulernas funktioner och nodernas kontakt utvecklades ett Javaprogram som kommunicerar med en av modulerna via en COM-port. Efter att rätt hastighet har valts och en port öppnats kan man skicka meddelande till de olika modulerna via en textruta och läsa ut vilka noder som är aktiva.



Figur 5: PC-mjukvaran för kommunikation mot enheterna.

4 Resultat

Tre separata noder har byggts och fungerar bra, noderna hittar varandra och kan skicka data från olika sensorer mellan varandra och ut till en PC. Tester visar att modulerna tyvärr inte gärna väljer att studsa trafiken via andra enheter om direktkontakten är dålig. Ett problem som kräver mer arbete är att skapa en algoritm för att hitta bästa vägen till målet i avseende på antal mottagna paket.

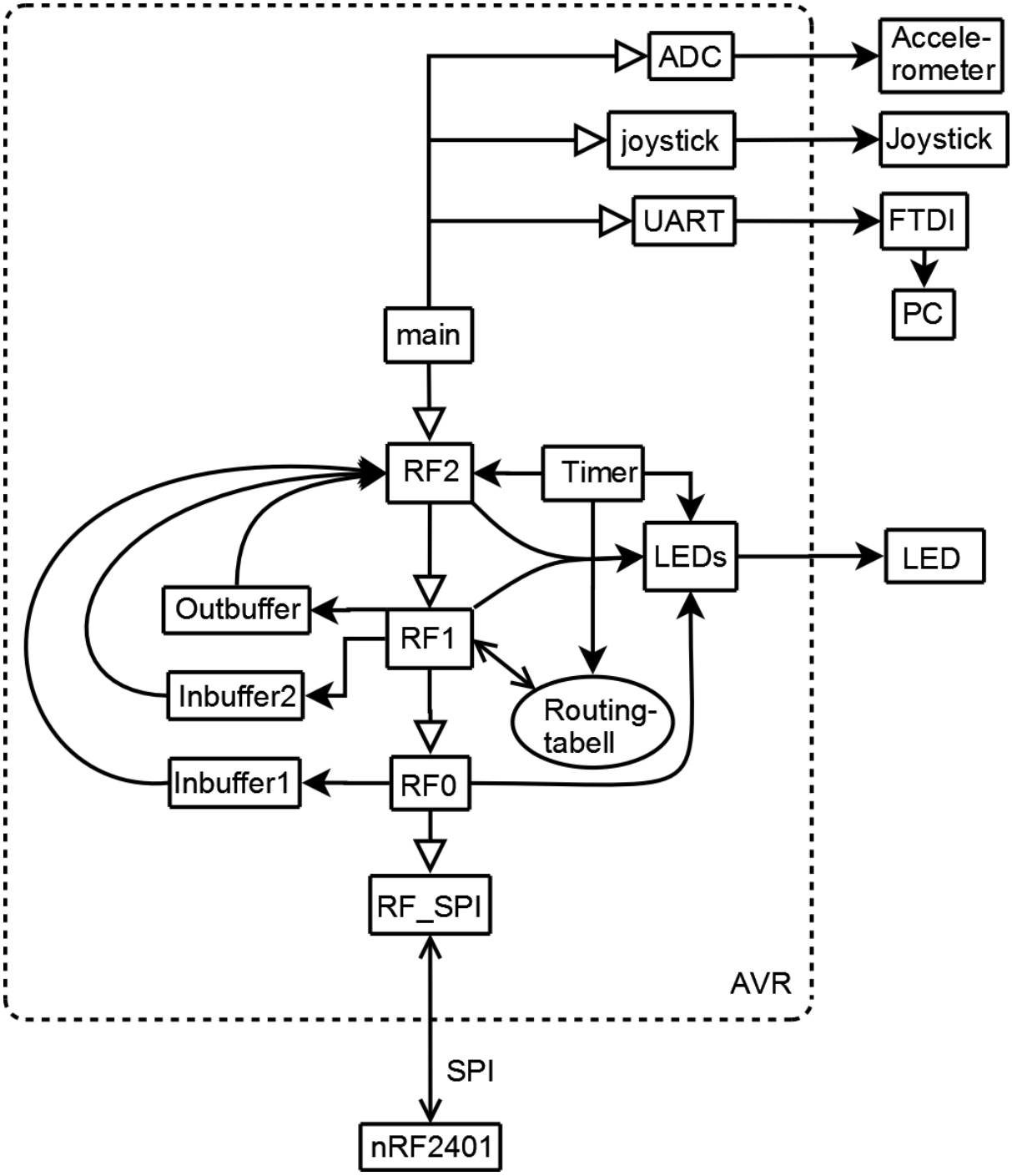
Målet med lång batteridrift har också uppfyllts, mätningar ger att varje enhet drar ca 25 mA. Med ett antagande att laddningsbara NiMh-batterier används ger det en drifttid på ca 100 h. Gränssnitt till dator har skapats både med USB och RS-232 som presenteras i en grafisk applikation.

Som indataenheter till modulerna har accelerometer och joystick använts och fungerar enligt specifikationerna.

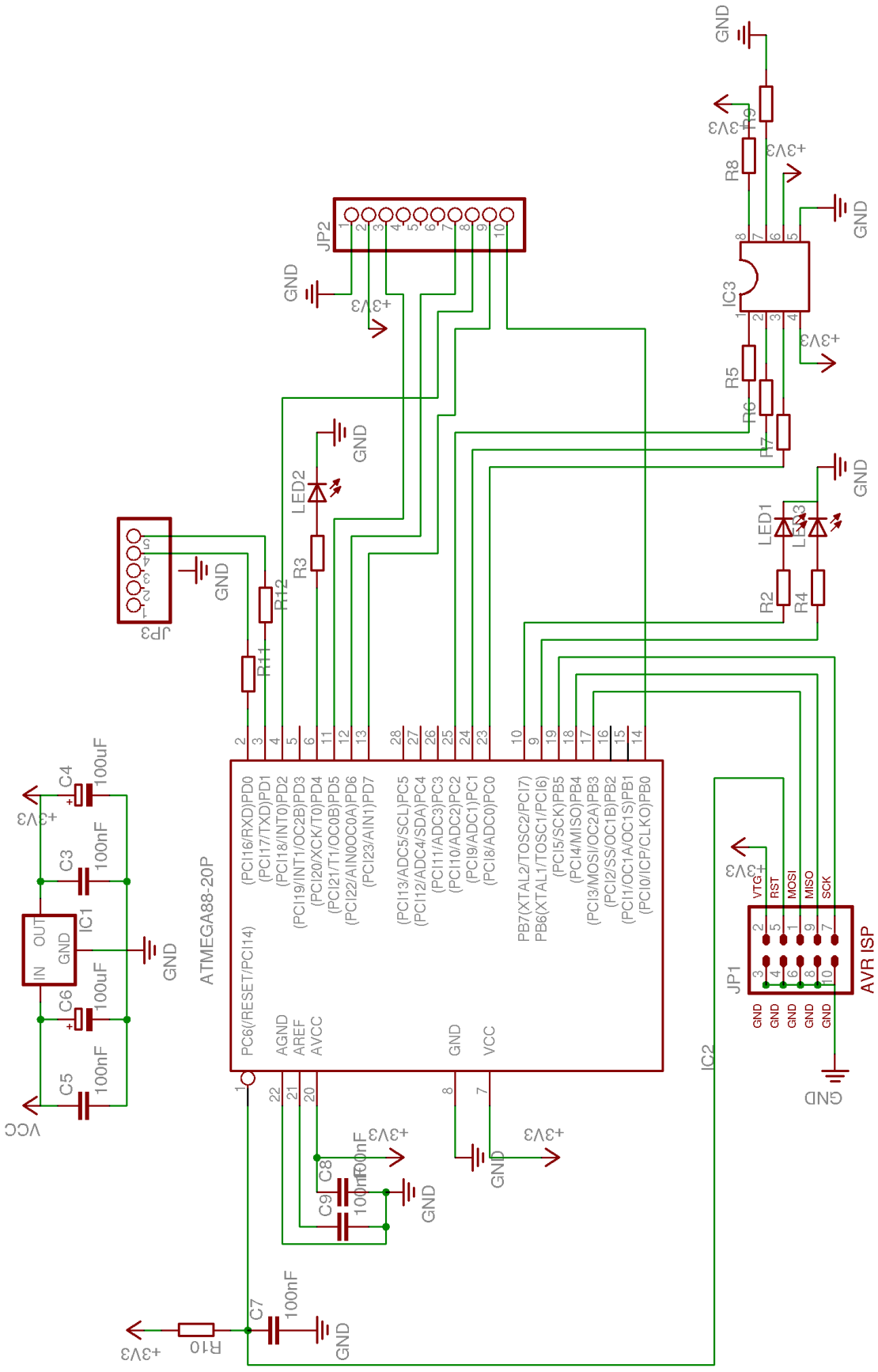
5 Slutsats

Den generella uppbyggnaden av protokollet fungerar bra, en stor nackdel är dock att överföringshastigheten minskar snabbt när datan måste hoppa via noderna. Många paket når aldrig sin slutdestination utan försvinner på vägen av brus eller krock med annat paket. Möjliga lösningar är att utveckla algoritmer som kan bestämma kvaliten på de olika vägarna mellan noderna på ett bättre sätt och även kunna hoppa mellan de olika möjliga frekvenserna för att få bästa möjliga signal. Större mikrokontrollers med mer arbetsminne skulle ge utrymme för fler paket i in- och utbuffrarna vilket öppnar för mer optimerad överföringslösningar. En utveckling av tillfällig peer-2-peer kommunikation mellan två noder skulle kunna implementeras för att få högre överföringshastighet vid kritiska moment.

6 Appendix



Översikt för de olika mjukvarumodulerna vi implementerat. Innanför AVRen representerar fyllda pilar informationsflöde och ihåliga pilar är anrop till funktioner.



Kretsschema för den enhet som är utrustad med accelerometer.

Kodstorlek

Rader kod för varje mjukvarumodul. Märk väl att koden är välkommenderad och luftig, därför kan radantalet vara missvisande.

```
Radiomjukvaran:
RF_SPI:          117 + 20
RF0:             424 + 118
RF1:             672 + 168
RF2:             538 + 68
buffer:          384 + 83
Totalt RF:       2592

Övrigt omkring, egenskrivet:
LEDs & timer:    117 + 19
commons:         162 + 4
joystick:        56 + 14 + 123
I/O:             114
main:            175
ADC:             303 + 26
Totalt övr eget:1113

Övrigt importerat:
UART:            621 + 198
Totalt övr:      819

PC-mjukvara:     543

Subtot:          5067
```