Nettverksprotokoll for et kortholdsradiokommunikasjonssystem basert på CC1010

av

Tuan M. Nguyen
Andrei V. Ouglov

Hovedoppgave til mastergraden i informasjons- og kommunikasjonsteknologi

Høgskolen i Agder
Grimstad, Mai 2003

Sammendrag

Chipcon AS har utviklet et CC1010 integrert kretskort. I den forbindelsen har Chipcon AS et ønske om å designe en nettverksprotokoll som vil basere seg på CC1010. En typisk applikasjon hvor en slik protokoll skal kunne benyttes er hjemmeautomatisering.

I et slik hjemmeautomatiseringssysstem er det ønskelig å opprette en slags Master–Slave nettverk (MS nettverk), hvor av det bare er én Master som er koplet opp mot mange slaver. Et slik nettverk vil være enklere å implementere enn et Peer–to–Peer nettverk (PTP nettverk), siden intelligensen og beslutningstaking i systemet ikke vil bli spredt over alle noder men samlet hos Master. Dessuten i systemer som krever en viss grad av sikkerhet vil det også være enklere å sørge for sikre transaksjoner.

Litteraturstudium var det første vi satte oss i, da fant vi ut at protokollen ”*Scaleable Node Address Protocol”* (SNAP) passer ganske godt til vårt formål. Det var ut ifra denne protokollen og ”*Simple Packet Protocol”* (SPP), som allerede brukes i CC1010, at vi designet vår ”*Master Slave Protocol”* (MSP). Teorier som er knyttet til designing av MSP protokollen ble gjennomgått, og ulike løsninger ble vurdert ut ifra krav og spesifikasjoner, som ble gitt etter samtalen med Chipcon AS.

Det viktigste kriteriet for MSP protokollen var at det er Master som initierer en kommunikasjon og at kommunikasjonen kun skal skje mellom Master og en slave til enhver tid. Slavene skal ut ifra meldingene de mottar fra Master, kunne avgjøre om de kan kommunisere med Master. TDMA ble valgt for å tildele slaver retten til snakke med Master. Det ble gjort i form av tidsluker tildelt til hver slave som slaver da bruker for kommunikasjon med Master.

Det ble gjort et forsøk på å regne ut tiden Master ville bruke på å utføre en hel runde og resultatene viste seg at den tiden var uakseptabel høy for applikasjoner som vil kreve rask responstid. Som løsning på dette problemet ble det lagt inn en *interrupt–funksjonalitet* som vil bli brukt for applikasjoner som krever rask responstid.

Det ble også gjort en undersøkelse om muligheter for å øke rekevidden til noder som vil kjøre MSP. Løsningen var å bruke noen noder som repeatere som vil videretransmittere meldinger adressert til slaver som har blitt registrert hos dem.

MSP protokollen vil også ha innebygd mulighet til å sjekke om meldinger er blitt transmittert uforandret i form av CRC-feilkontrollmekanismen og til kryptering ved hjelp av DES-mekanismen.

Det som mangler i protokollen nå er verifisering av tidslukene, for å komme fram til best mulig størrelse på tidsluker. Vi har gjort et forsøkt på å regne ut denne tiden, men det ville være best å teste ut systemet og dermed kunne justere tiden til den mest passende.

Forord

I siste semesteret ved sivilingeniør utdanning i Informasjon Kommunikasjon Teknologi (IKT) ved Høgskolen i Agder (HiA), skal studentene skrive en diplomoppgave. Denne oppgaven, som er utført i vårsemesteret 2003, har en varighet på 20 uker og er verdsatt til 10 vekttall. I den forbindelsen har Chipcon AS formulert og gitt oppgaven.

Veiledere for oppgaven har vært Karl Helmer Torvmark fra Chipcon AS og Arild Haglund fra HiA. En takk til dem for gode råd og veiledning.

Oslo, mai 2003

Tuan M. Nguyen Andrei V. Ouglov

Innholdsfortegnelse

Figurliste

Tabelliste

1 Innledning

Det er ikke mangel på visjoner om morgendagens digitale tjenester. Lenge har science fiction-genren beskrevet fullautomatiserte hus, styrt av intelligente systemer. Med enheter tilknyttet et felles nettverk, åpner det seg en rekke muligheter for nye tjenester. I de siste årene har vi vært vitner til en enorm utvikling innenfor trådløse enheter og nettverk. Fra kommunikasjon via infrarøde signaler, og kabler med kontakter, ser det nå ut til at trenden beveger seg mot trådløs kommunikasjon over kortholdsdistanseradionett.

1.1 Bakgrunn

Chipcon AS har utviklet et CC1010 integrert kretskort som er den første komplette Radio Frequency (RF) "*System-on-Chip*" løsning. I den forbindelsen har Chipcon AS et ønske om å designe en nettverksprotokoll som vil basere seg på "*Simple Packet Protocol*" (SPP) som CC1010 kretskortene for tiden bruker. En typisk applikasjon hvor en slik protokoll skal kunne benyttes er hjemme- og industriautomatisering. Slike trådløse kommunikasjonssystemer har i den siste tida blitt veldig populære. CC1010's lav strømforbruk og allsidig funksjonalitet gjør den meget godt egnet til trådløse applikasjoner, og attraktiv på markedet på grunn av sin lav pris.

1.2 Fokus og Mål

SPP mangler funksjonaliteten til å kunne brukes i et hjemmeautomatiserings- eller industristyringsnettverk. Vårt mål er å definere en ny protokoll som kan bli brukt til de nevnte formål. Den nye protokollen skal bygge seg på tilgjengelige protokollfunksjoner i SPP.

For å kunne designe og utvikle denne protokollen må vi gjennomgå og senere drøfte følgende problemstillingene:

- Litteraturstudium. Det skal undersøkes om det finnes standard nettverksprotokoller og teoretiske metoder som egner seg.
- Det skal gjøres et valg mellom Master-slave nettverk med mulighet for multihopp og peer-to-peer nettverk med dynamiske rutingstabeller.
- Det skal velges en multiple aksess metode.
- Påloggingsalgoritme: vurdering for muligheter og nødvendighet for autentisering.
- Kryptering og nøkkelutveksling: vurdering om dette er nødvendig og eventuelt vurdering av ulike metoder.

1.3 Generell beskrivelse

En protokoll kan sammenliknes med et språk. To eller flere kommunikasjonsparter som skal utveksle informasjon må snakke samme språk. Noder i nettverket som skal utveksle data, må altså bruke samme protokoll. Strukturen på pakkene og reglene for håndtering av forskjellige situasjoner inkludert feilsituasjoner er noe som er innbefattet i det vi kaller protokoll. Sagt med andre ord, en nettverksprotokoll er et sett rutiner og tjenester som applikasjonene bruker for kommunikasjon og overføring av pakker.
Disse reglene må si noe blant annet om følgende:

- Hvordan meldinger skal adresseres?
- Hvilket format skal meldingene være på?
- Hvordan bekreftes mottatt melding?
- Hvordan er retten til å sende meldinger gis til nodene i et nettverk?

1.4 Rapportens oppbygging

- Kapitel 1 er innledning hvor vi forteller om bakgrunn for dette prosjektet.
- Kapitel 2 beskriver forskjellige utstyr og programvare som har blitt brukt i oppgaven.
- Kapittel 3 gir innføring i forskjellige teknologier som finnes og på grunnlag av denne beskrivelsen velger vi teknologier som vil bli brukt i den nye protokollen.
- Kapitel 4 er protokolldesign. Her skal vi beskrive noen eksisterende protokoller som er relevante for oss, og vårt synspunkt om disse protokollene. Vi forteller her også litt om OSI modell, som er det sentrale i designing av en protokoll.
- Kapitel 5 presenterer den nye protokollen ut ifra valgene som ble tatt. Det beskriver i detalj pakkeformat og funskjonaliteten til protokollen.
- Kapitel 6 beskriver resultatprodukt. Det beskriver hvordan vår nye protokoll fungerer.
- Kapitel 7 er drøfting av valgene som ble tatt og fordeler og ulemper ved disse.
- Kapitel 8 er konklusjon.

2 Hardware og software

For å kunne gjennomføre prosjektet og designe en radiokommunikasjonsprotokoll er vi avhengig av forskjellige hardware og software.

2.1 Hardware

2.1.1 CC1010 og CC1010EM, Evaluation Module

***Figur 2.1**: CC1010 integrert i CC1010EM*

CC1010 er den første komplette *Radio Frequency* (RF) *System-on-Chip* løsning utviklet av Chipcon. CC1010's transceiver opererer på frekvensbånd mellom 300 og 1000 MHz, sammen med CC1010 er det integrert en 8051 mikrokontroller med 32 kB av programmerbare *FLASH* minne, som gjør det enkelt å utvikle og teste programmer for enheten. CC1010 er også beregnet for en veldig lav strømforbruk for trådløse applikasjoner, dessuten finnes det en rekke former for strømsparing som *powerdown-, idle- og sleep-modes*. Det finnes mulighet for DES-kryptering av data og 3-kanal 10 bits ADC i hardware. Det betyr at det trenges lite eksternt utstyr for å lage et kraftig integrert trådløst system med godt utviklet grensesnitt mot omverden.

De viktigste karakteristikkene er som følge:

RF Transceiver

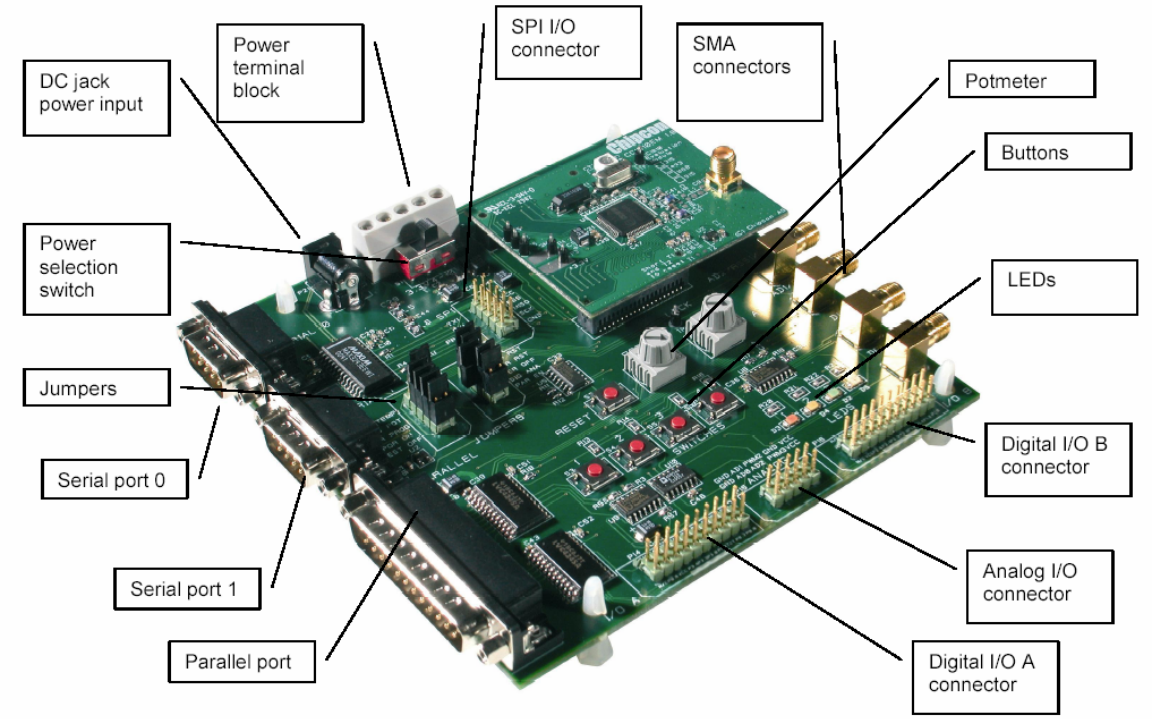
- Veldig lav strømforbruk
- Frekvensbånd mellom 300 MHz og 1000 MHz
- Høyt følsomhet (typisk – 107 dBm)

- Programmerbar avgitt energieffekt opp til +10 dBm
- Datarate opp til 76.8 kBit/s
- Begrenset antall av eksternt utstyr

8051-kompatibel mikrokontroller

- 2.5 ganger bedre ytelse enn standard 8051
- 32 kB Flash, 2048 + 128 Byte SRAM
- 3 kanal 10 bit ADC, 4 timers /2 PWMs, 2 UARTs, RTC, Watchdog, SPI, DES kryptering, 26 I/O pinner.
- 2.7 - 3.6 V strømforsyning
- 64-lead TQFP

2.1.2 CC1010EB Evaluation Board



Figur 2.2: *CC1010EB Evaluation Board*

CC1010EB brukes til å overføre hex-kodede applikasjoner til CC1010EM. EB'et med CC1010EM på kan kobles til en PC via en parallell kabel, sånn at overføringen vil skje forholdsvis ubemerket for brukeren. CC1010EB kan mates direkte fra vanlig strømnett via stikkontakt.

2.2 Software

2.2.1 Keil uVision 2 development

Keil uVision 2 er et Integrert Utviklingsmiljøverktøy (*IDE – Integrated Development Environment*). Dette integrerte utviklingsverktøy brukes til utvikling, debugging, kompilering og simulering av software applikasjoner.

Keil består av:

- Prosjekt manager GUI (Grafic User Interface)
- Programmeringsverktøy
- Verktøy konfigurasjonsmuligheter
- Tekst editor
- Kraftig debugger GUI
- Kompilator/assembler/linker

Keil uVision2 støtter de fleste 8051 mikrokontroller plattformer. Det inneholder også en prosjekt manager GUI, tekst editor, simulator og debugger GUIs, og kompilator/assembler/linker. CC1010s IDE er basert på uVision 2, et software utviklingsverktøy fra Keil Elektronikk. Editor brukes for editering av kilde- og assembly filer. Editoren utfører også syntakssjekk. Kompilator konverterer en eller flere C kildefiler til assembly kode, som legges til i assembler, eventuelt sammen med håndskrevne assembly filer. Assembler lager så objektfiler (maskinkode) som i sin tur legges til i linker, sammen med på forhånd kompilerte biblioteker. Til slutt lager linkeren en hex-fil som kan bli overført til FLASH minne på CC1010-chipen.

2.2.2 Flash programming utility

Dette brukes til å overføre hex-filer som ble produsert av Keil uVision til FLASH minne på CC1010-chipen via vanlig PC parallell kabel. I vår oppgave ble det brukt et program laget av Chipcon – ”Chipcon CC1010 Flash Programmer” til dette formål. Programmet er utrolig enkelt å bruke og krever minst mulig forkunnskaper.

3 Bakgrunnsteorier og valg

3.1 Master-Slave nettverk eller peer-to-peer nettverk

Nettverkstype har uten tvil stor innvirkning på hvordan hele nettverket vil fungere, både med tanke på måten kommunikasjonen skal foregå på og ikke minst effektiviteten. Med dette i tankene vil vi gjennomgå to prinsipielt forskjellige nettverkstyper, vurdere deres sterke og svake sider fra applikasjonsområdets innblikk og foreta valg til fordel av den ene nettverkstype.

3.1.1 Master-Slave nettverk

Denne nettverksmodellen gir muligheten til pålitelig styring av nettverksressurser. MS nettverk kan bestå av alt fra titalls til tusentalls noder, og styres av en Master. Master har i sin funksjon å styre driften av nettverket, fordele nettverkets ressurser og kontrollere at sikkerheten opprettholdes.

De aller fleste industrielle nettverk er basert på MS prinsippet, der Master sender sekvensielle meldinger til slaver som så reagerer på meldingene. Denne sekvensen kalles for en *polling*. Siden systemet er transparent, er det en forutsetning for prosedyren at hver slave har sin egen adresse.

En Master sender en melding som er adressert til en bestemt slave. Slaven gjenkjenner adressen og utfører kommandoer som følger med meldingen. Når dette er gjort, returneres en kvittering til Master som fortsetter til neste slave osv.

Noen karakteristiske trekk ved MS nettverk:

- Funksjonelle moduler: Master og slaver
- Definerte grensesnitt: Master styrer, slaver tjener
- Master vil forespørre en tjeneste fra slaver
- Faste roller for en gitt forespørsel
- Informasjonsutveksling kun gjennom meldinger

Fordeler:

- Skalerbarhet
- Sentralisert (tilgang, sikkerhet, ressurser)
- Fleksibilitet (ny teknologi kan bytte ut gammel)

Ulemper:

- Systemet er sårbart for nedkobling av Master – hvis Master er nede, er hele nettverket nede.
- Forsinkelser for nettverk med mange slaver

3.1.2 Peer-to-peer (PTP) nettverk

Et PTP nettverk brukes vanligvis hjemme eller i små firmaer. I denne modellen er det en direkte kommunikasjon mellom noder, og det er ikke nødvendig å bruke en spesiell node (Master) til å styre nettverksressursene. Vanligvis passer denne modellen best i miljøer der det er færre enn ti noder innenfor samme område. Det er mindre kostbart og lettere å vedlikeholde, men nettverk av denne typen har generelt ikke like god sikkerhet og færre funksjonaliteter enn MS modellen. Nodene er likestilte, og ressursene deles mellom dem uten at det er behov for en Master som styrer hele nettverket sentralt. Hver enkelt node bestemmer hvilke data som skal deles på nettverket.

Noen karakteristiske trekk ved PTP nettverk:

- Lite antall noder i nettverk
- Prosesser som vekker hverandre mellom noder
- Ingen sentral kontrollrolle
- Enhver node kan sende eller motta fra enhver annen node

Fordeler:

- Lav oppstartskostnad
- Lett å kontrollere i mindre omfang

Ulemper:

- Sårbar for nedkobling av enhet
- Blir for kompleks og forvirrende hvis antall noder skal øke (>10-20 noder)

3.1.3 Hva bestemmer type nettverk? Vurdering og valg.

1. Nettverksstørrelse
2. Grad av sikkerhet
3. Forventet størrelse på nett-trafikken
4. Behovene til nettverksbrukerne

Vi skal nå gå punkt for punkt og finne ut hvilket nettverkstype som passer oss best.

1. Nettverksstørrelse:
   Vi går ut ifra at et hjemmeautomatiserings- og industriellstyringsnettverk kan bli mye større enn 20 noder. På grunn av begrensningen i antall noder i PTP nettverk ser MS nettverk ut til å være et bedre valg. Ut ifra dette er MS nettverk å bli foretrukket, siden PTP nettverk bør helst være begrenset i antall noder for å gi en tilfredsstillende service.

2. Grad av sikkerhet:
   MS nettverk generelt kan gi en høyere sikkerhetsgrad, siden ”ansvaret” i nettverket er konsentrert hos en Master som sørger for sikker og pålitelig dataoverføring. Derimot er ”ansvaret” spredd over alle noder i et PTP nettverk.

3. Forventet størrelse på nett-trafikk:
   PTP nettverk kan normalt håndtere større trafikkmengder enn MS fordi alle dataene ikke nødvendigvis må igjennom master.

4. Behovene til nettverksbrukerne
   I dette punktet kommer mer menneskelige aspekter inn i bildet. Hvis vi tenker oss et typisk hjemmeautomatiseringssystem hvor man ønsker å styre forskjellige enheter, da er det mye enklere for brukeren å gjøre det fra ett sted, ved hjelp av én styringsenhet (Master). Da slipper man å springe fra en node til en annen for å få ting gjort, slik det hadde vært i et PTP nettverk. MS modellen er altså å foretrekke her.

Nå som vi har gått gjennom disse punktene kan vi med sikkerhet konkludere at MS modellen passer best for vårt formål og rapporten videre skal dreie seg utelukkende om denne nettverkstypen.

3.2 MS nettverk og valg av forskjellige teknologier

Etter vurderingen i avsnittet over har vi kommet fram til at MS nettverk ville passe best til vårt formål. Nå går vi videre til andre aspekter som trenger å bli tatt hensyn til i protokolldesign.

3.2.1 Valg av nettverkstopologi

Noder i et nettverk kobles sammen i bestemte "mønstre" som man kaller nettverkstopologier. Her finnes det mange muligheter, men det er tre typer som er mer vanlig enn andre. Det er stjerne-, buss- og ring- topologi. Disse tre grunnstrukturene kan man sette sammen til mer komplekse løsninger, men som regel kan man fremdeles skille om topologien følger en av de tre grunnstrukturene. Nettverkets topologi påvirker dets kapasitet. Hvilken nettverkstopologi man velger vil innvirke på:

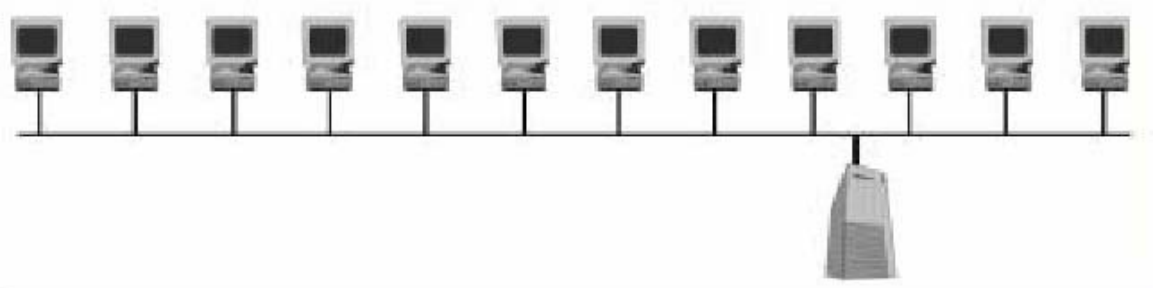
- Transmisjonsmediet
- Kommunikasjons egenskaper
- Hvilken type utstyr nettverket trenger
- Utstyrets kapasitet
- Nettverkets vekst
- Hvordan nettverket blir administrert

Det å velge en nettverkstopologi som passer til et viss formål er uten tvil avgjørende for at nettverket vil fungere bra.

3.2.1.1 Buss

Busstopologien er mest beregnet for konvensjonelle datanettverk hvor alle datamaskiner er knyttet til samme kabelen. I et bussnettverk er alle maskinene koblet mot en felles

kommunikasjonskanal, en buss. Nodene på nettet bruker felles kommunikasjonskanalen og derfor må dele på å bruke den. Alle nodene på kanalen lytter. Dette betyr at en pakke som sendes ut på kanalen kan avleses av alle nodene simultant. Slike nett har fellesbenevnelsen kringkastingsnett. Den mest utbredte standarden for denne typen nett kalles Ethernet.

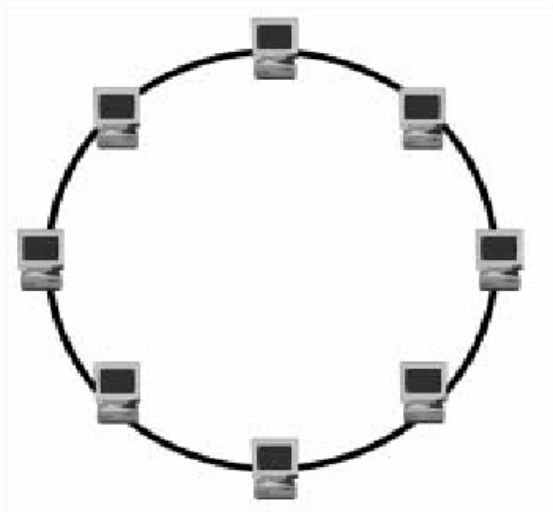


Figur 3.1: *Busstopologi*

Fordeler:
Ingen store konsekvenser om en maskin går ned
Høy kapasitet
Ulemper:
Sårbar for svikt i Master.

3.2.1.2 Ring

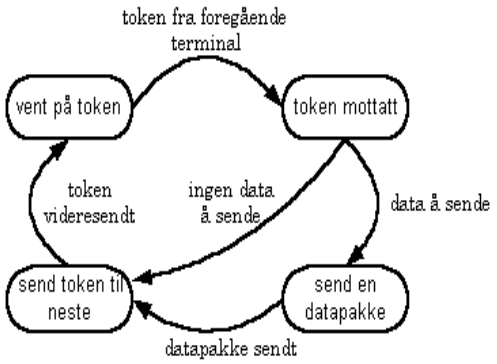
Ringtopologien er et spesielt tilfelle av busstopologi som bruker en annen teknikk for å styre adgang til mediet. Det karakteristiske ved denne topologien er at alle noder er koblet sammen i en ring.



Figur 3.2*: Ringtopologi*

Kommunikasjonen i ring-nettverk følger som regel *"token ring"*-prinsippet.

Token ring – prinsippet
Alle noder har definert en foregående og en etterfølgende node. En spesiell pakke, token, sendes rundt i nettet, mottas fra foregående og videresendes til neste node. Prinsippet kan beskrives med tilstandsdiagrammet.



Figur 3.3: *Token ring tilstandsdiagram*

En node som genererer og fjerner pakker fra transmisjonsmediet betraktes som Master, og de andre for slaver.

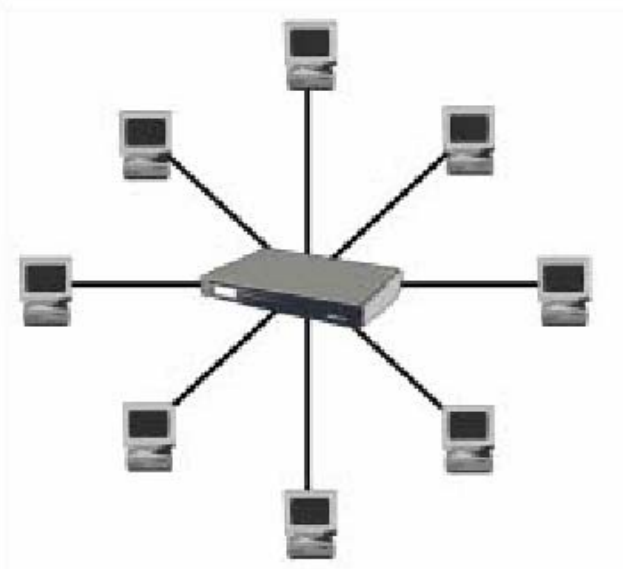
Master i token ring nettverk heter Monitor, og den har egentlig litt annerledes oppgaver enn de en "klassisk" Master har. Monitor kan bli valgt manuelt på forhånd av nettverksadministrator eller det kan gjøres ved hjelp av en av mange *election algoritmer*. Monitor har ansvaret for en rekke oppgaver, blant annet styre tidsforsinkelser i systemet, passe på at token ikke blir ødelagt eller mistet, lage nye token ved behov osv.

Fordel:
- Godt egnet for store nettverk, forutsatt at alle noder fungerer som de skal.

Ulemper:
- Alle pakker går en hel runde i ringen, innom hver node, en node med feil kan ødelegge ringen
- Tapt eller kopiert token ødelegger funksjonen
- Komplisert å flytte, legge til og fjerne en node
- Lavere kapasitet enn buss topologi
- Dårlig egnet for trådløs kommunikasjon og henger ikke sammen med trådløst konseptet: noder i trådløst nettverk har ikke fysiske forbindelser mellom seg og i tillegg har de til en viss grad bevegelsesfrihet. Med det menes at situasjoner når en pakke som er adressert til en nabonode må gå via en rekke andre noder før den kommer fram, vil bli nokså vanlige.

3.2.1.3 Stjerne

Et stjernenett har fått sitt navn for sin karakteristiske struktur. I et stjernenett står noder rundt i ring med en "supernode" i sentrum. Dette sentrumet fordeler signaler og pakker etter behov rundt til nodene i periferien.



Figur 3.4: *Stjernetopologi*

Karakteristiske trekk ved stjernetopologi:

- Hver stasjon er direkte tilkoplet en sentral node
- Kan sende broadkast
- Fysisk stjerne, logisk buss
- Kun en stasjon kan sende av gangen

Fordeler:

- Ingen store konsekvenser om en node, utenom sentral node, går ned
- Lett å utvide
- Lett å lokalisere feil
- Linjene kan bruke forskjellige teknologier

Ulemper:

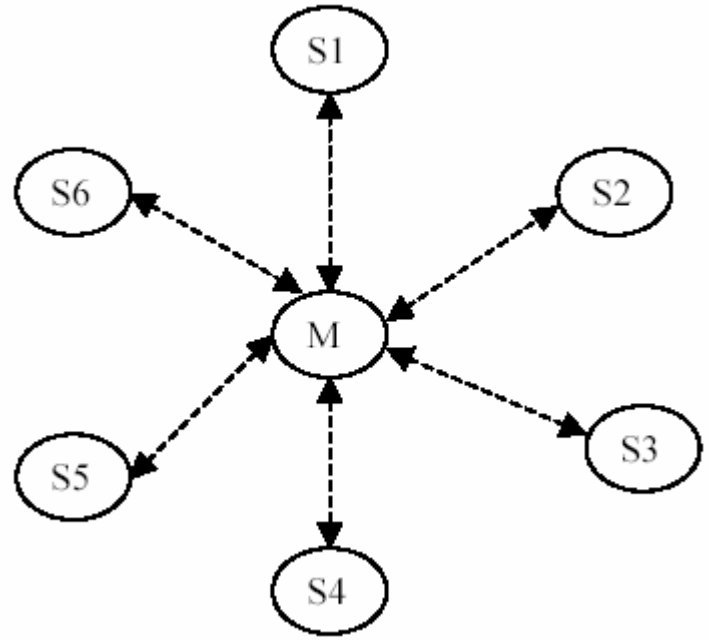
- "*Single-point-of-failure*" – Hvis sentral node faller ut, er hele nettverket nede.

3.2.2 Vurdering og valg av nettverkstopologi

Den logiske topologien forteller noe om hvordan nodene i nettverket henvender seg til felles mediet, og hvordan trafikken styres på nettverket. Etter diskusjon med Chipcon, fikk vi beskjed om at det er ønskelig med bare én Master i et MS nettverk. De ville også ha at kun kommunikasjon mellom Master og én Slave er mulig til enhver tid.

Master skal bruke aksessmetode for å distribuere retten til å kommunisere med forskjellige slaver. Denne sentraliserte aksessmetoden gjør at Master "*poller*" slaver en etter en for å se om de har noe å sende. Det betyr at slaver må vente på sin tur for å få sende. *Polling* er en aksessmetode hvor en master tildeler slaver rett til å kommunisere.

Etter ønskene fra Chipcon ser det ut til at stjernenettverkstopologi bli å foretrekke. Her befinner Master seg rent fysisk i midten av nettverket med slaver rund seg. I stjernenettverkstopologi er det Master som bestemmer all kommunikasjon.



Figur 3.5: *Den valgte nettverkstopologien*

Andre aspekter som taler til fordel av stjernenettverk er som følge:

1. Stjernenettverk er veldig enkelt å utvide.Det kommer godt med, med tanke på vårt anvendelsesområde. Hvis man for eksempel kjøper en ny enhet til sitt hjemmeautomatiseringssystem, da behøver man ikke å ta hensyn til alle andre slaveenheter som var på nettverket fra før, siden slaver kommuniserer kun med Master, og det eneste man da trenger er å registrere den nye enheten hos Master.
2. Det er enkelt å oppdage feil i stjernenettverk. Hvis feil oppstår i stjernenettverket, vil det være mye enklere å finne den enn hvis det hadde vært, for eksempel, i et ring-nettverk. Dette er på grunn av at slaver ikke er koblet med hverandre men kun til Master.
3. En annen fordel som egentlig er en følge av den forrige fordelen er at hvis en eller flere slaver går ned, så vil det ikke føre til store konsekvenser for resten av nettverket.

Mulig problem med den valgte topologien

Slaver i et *polling nettverk* får ”best mulig service”, dvs at servicekvalitet er direkte avhengig av antall slaver i nettverket. Jo flere slaver det er i nettverket, desto mindre tid får hver enkelte slave til å kommunisere med Master.

Dette problemet er imidlertid av en mindre betydning i vårt nettverk. Dette er grunnet til at meldingene som skal gå mellom Master og slaver er nokså kortvarige (Slå på/ned- meldinger). Men hvis det er behov for at viktige meldinger må komme umiddelbart er det ikke helt umulig å sette for eksempel noen interrupt-rutiner.

3.3 Multippel aksess

3.3.1 Metoder for multippel aksess

Et moderne telesystem representerer en felles ressurs i den forstand at flere brukere kan hente tjenester fra samme sted. Dette krever en tilgangskontroll som kalles multippel aksess.

Problemstilling:

- Mange brukere – en kanal
- Effektiv utnyttelse av kanalen
- Rettferdig fordeling av kanalressursen
- Må unngå at flere bruker kanalen samtidig (kollisjoner)

Prinsipielt for valg av metode av multippel aksess er hvor i nettverket man implementerer beslutningstaking. Her er det to muligheter:

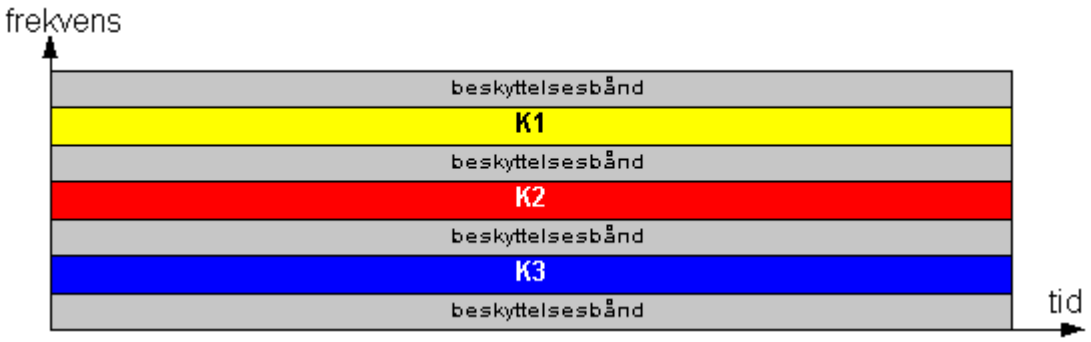
- Sentralisert aksesskontroll
  En organisator med total autoritet styrer tilgangen til kanalen. Enkelt system, men sårbart ved feil i kontrollpunktet. I trådløse systemer er ofte basestasjonen master og terminalene slaver. For pakkedatatrafikk kan *polling* være en enkel master-slave teknikk, master spør hver slave etter tur om de har noe å sende.
- Desentralisert aksesskontroll
  Alle er likestilte, sender når de har behov og eventuelt når de ut ifra visse kriterier antar at kanalen er ledig. Mer kompleks enn sentralisert aksesskontroll, men kan gi høyere pålitelighet og bedre kapasitetsutnyttelse. En enkel form for distribuert aksess er *polling* slik det forekommer i TDM linjesvitsjede systemer, her er det avsatt en fast tidsluke til hver kanal, dette konseptet forutsetter synkronisme.

Siden vi allerede har valgt å gå for MS nettverkstype og stjernetopologi vil sentralisert tilgangskontroll være et naturlig valg her. Dessuten er et desentralisert system mye vanskeligere å implementere siden det er flere viktige ting man bør ta hensyn til, og vårt mål er å holde ting enkelt for å definere en enkel men noenlunde sofistikert protokoll.

Løsninger på deling av kanalressurser kan være forskjellige, men det finnes tre hovedprinsipper for å oppnå multippel aksess på. Disse er som følge:

1. FDMA - Frequency Division Multiple Access
2. TDMA - Time Division Multiple Access
3. CDMA - Code Division Multiple Access

3.3.1.1 FDMA (Frequency Division Multiple Access)



Figur 3.6: *Bruk av tids- og frekvensresurser i FDMA*

FDMA deler hele kanalens båndbredden inn i M like subkanaler som er tilstrekkelig separat, via beskyttelses bånd, at de forhindrer forstyrrelse fra samme kanal. Hver node vil da få tildelt en eller flere av disse subkanalene til sin egen bruk. For å motta pakker fra en viss sendernode, må en destinasjonsnode høre på den samme subkanalen. Kapasiteten til hver enkel subkanal er lik C/M, hvor C er kapasiteten til hele kanal båndbredden, de små mengde av tap frekvens på grunn av beskyttelses bånd er ikke tatt med.

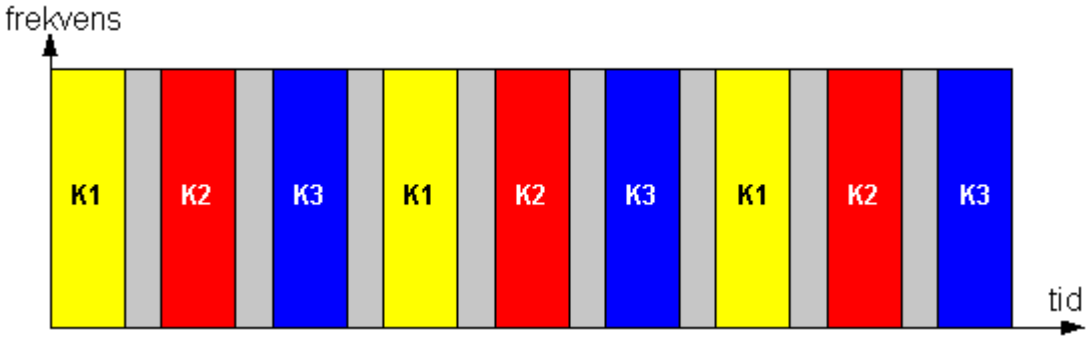
Hovedfordelen til FDMA er mulighet for å kunne sende mange pakker samtidig på de forskjellige subkanalene uten kollisjon. Men denne fordelen begrenser seg til systemer som har noe å sende meste parten av tiden. Hvis i et MS nettverk hvor noen slaver som har fått tildelt kanaler sender sjeldent eller ikke i det hele tatt, vil det resultere i sløsing med frekvenskapasitet.

En annen ulempe vil være at sendetiden (transmisjons tiden) til en pakke vil øke, siden hele båndbredden er delt inn i flere subkanaler, som ikke har like stor kapasitet som en stor ”super-kanal”. Dette vil dermed resultere i lengre pakkeforsinkelser. FDMA krever også bedre radiofiltrering og det vil øke systemets kostnad.

Den største ulempen er imidlertid metodens kompleksitet. CC1010 har ikke variabel kanalbredde. Hovedgrunnene for ikke å bruke FDMA er at få kanaler er tilgjengelig i 868 MHz båndet samt at det vil være unødig kompliserende å måtte skanne igjennom alle kanalene hele tiden.

3.3.1.2 TDMA (Time Division Multiple Access)

I TDMA brukes hele båndbredden i en del av tiden.

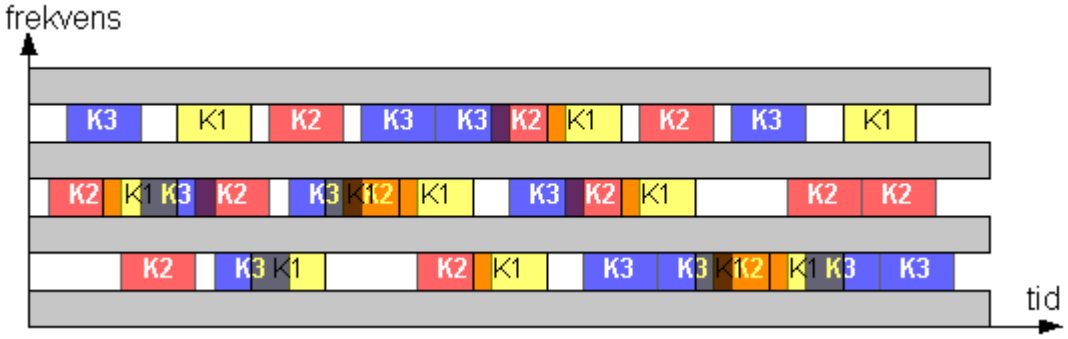


Figur 3.7: *Bruk av tids- og frekvensresurser TDMA*

TDMA deler hele kanal båndbredde inn i M like tidsluker som er igjen organisert inn i en synkronisert ramme (se figur 3.7). Hver node kan da få tildelt en eller flere tidsluker for sin egen bruk. Dette betyr at pakker transmisjon i TDMA skjer i seriell måte, hvor hver node tar sin tur å aksessere kanalen. Siden hver node kan aksessere hele kanal båndbredden i sin tidsluke, vil tiden å sende en L bit pakke være lik L/C. Men igjen vil det være sløsing med tidsluker hvis en sendernode ikke har pakker å sende ofte.

3.3.1.3 CDMA (Code Division Multiple Access)

I CDMA brukes hele båndbredden hele tiden. Dette er en pakkesvitsjet teknologi for digital overføring av signaler hvor hver samtale eller dataoverføring bruker hele frekvensspektret og identifiseres med en unik kode.



Figur 3.8: *Bruk av tids- og frekvensresurser i CDMA*

CDMA gir mulighet for at flere brukere kan benytte samme ressurs samtidig. Den mest karakteristiske egenskapen ved denne metoden er at alle brukerne bruker samme frekvensbånd samtidig. Dette gir en god utnyttelse av den tilgjengelige båndbredden, men stiller store krav til signalbehandling på mottaker for å skille en bruker fra de andre brukerne.

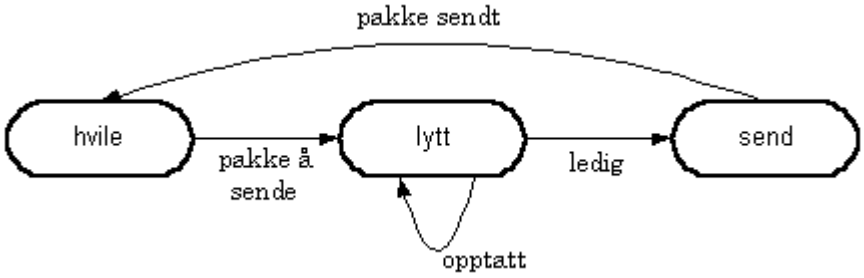
Utenom disse tre metodene beskrevet ovenfor finnes det en annen metode som ikke deler opp enten felles frekvens- eller tidsressurs.

3.3.1.4 CSMA (Carrier Sense Multiple Access)

CSMA er en av Media Access Controll (MAC) protokoller som benytter transportsensor for å unngå kollisjon med pågående transmisjoner. CSMA fungerer slik at den først hører på mediet for å avgjøre om det er ledig eller ikke, før sending av pakker.

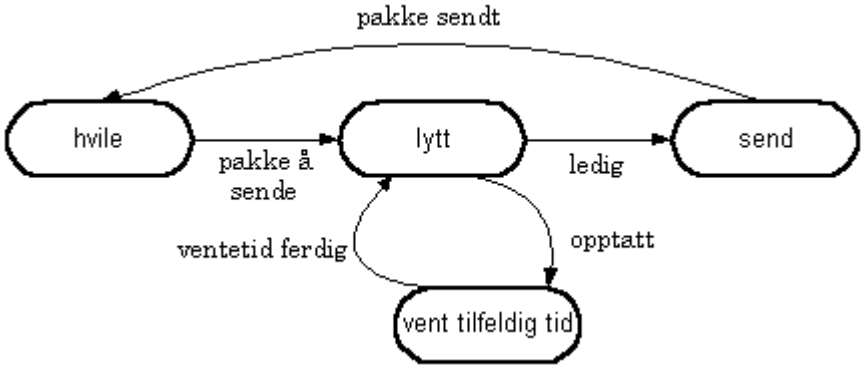
Det fins mange typer av CSMA, men de to viktigste er *Persistent CSMA* og *Nonpersistent CSMA*.

Persistent CSMA går ut på å kontinuerlig høre på kanalen for å avgjøre når aktiviteten vil slutte. Med en gang kanalen har vendt tilbake til *idle* tilstand, vil det transmitteres en pakke umiddelbart. Med denne metoden vil det lett forekomme kollisjoner, hvis flere venter samtidig og dermed sender samtidig når kanalen blir ledig.



Figur 3.9: *Persistent CSMA tilstandsdiagram*

Nonpersistent CSMA vil i sin tur redusere sannsynligheten for slike kollisjoner ved å legge inn tilfeldig, forskjellig ventetid for terminalene. Hver gang en opptatt kanal er detektert, vil sendernoden vente i en tilfeldig tid før den føler på kanalen igjen. Denne prosessen vil repetere seg med eksponentiell økning av tilfeldig tid, helt til kanalen blir detektert ledig.



Figur 3.10: *Nonpersistent CSMA tilstandsdiagram*

3.3.2 Vurdering og valg av aksess metode

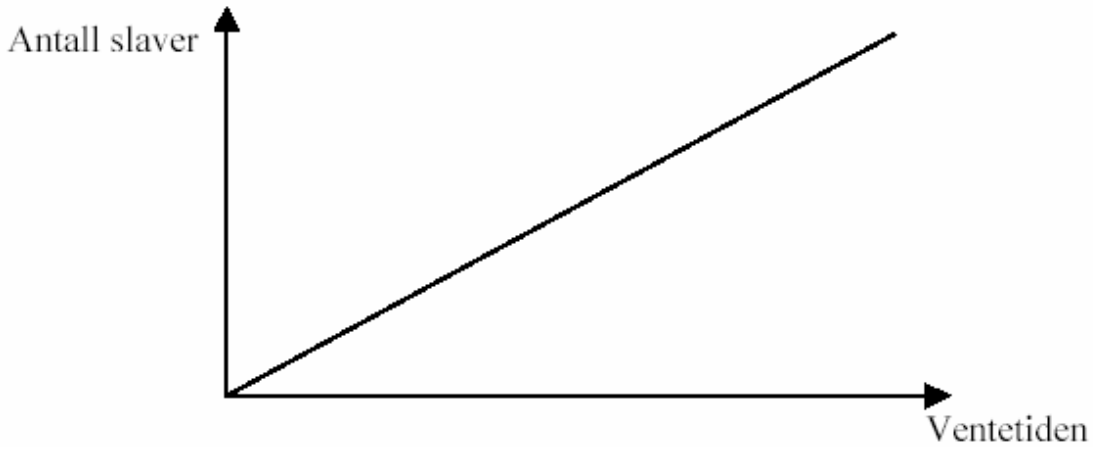
Valget står altså mellom TDMA, FDMA, CDMA og CSMA. Valget er avhengig av flere aspekter.

CDMA er uten tvil den vanskeligste teknikken blant de nevnte her og kan med sikkerhet fjernes fra listen på grunn av sin forholdsvis vanskelig implementering og høye kostnader i implementering og drift.

Bruk av FDMA forutsetter at man deler opp frekvensbånd i like mange frekvenskanaler som det er slaver. Dette vil gjøre nettverket ikke skalerbar, dvs at jo flere slaver man får på nettverket, desto smalere frekvenskanal får hver enkelte slave å sende og motta på. Dette vil resultere at tiden å sende ut en pakke vil øke dramatisk i nettverk med stort antall noder. Men for lite antall slaver kan likevel denne løsningen være hensiktsmessig. Dessverre må vi regne med at systemet vil bestå av stort antall noder. Den største ulempen av denne metoden ligger imidlertid i dens kompleksistet. Blant annet kan det nevnes at man må skanne igjennom alle kanalene hele tiden. I tillegg er det få kanaler er tilgjengelig i 868 MHz båndet. Økning i kompleksiteten betyr dessuten økning i kostnader.

Da ser TDMA til å være det beste valget i denne sammenheng. I tillegg til sin nokså enkel fysisk implementering og dermed lav pris, er TDMA et naturlig valg i systemer hvor noder ikke trenger å sende altfor ofte. Dette er tilfelle i vårt system. TDMA forhindrer også kollisjoner av pakker mellom Master og mange slaver, siden det bare er én kommunikasjon mellom Master og én slave som er mulig til enhver tid. Men TDMA forhindrer ikke kollisjoner av pakker som sendes i én og samme kommunikasjon. Dette betyr at vi må finne en annen måte å forhindre kollisjoner av pakker i én og samme kommunikasjon mellom Master og slave. En løsning her kan være CSMA eller at det ved behov retransmitteres nok ganger slik at pakken garantert vil komme fram.

TDMA har også den ulempe at hvis antall slaver som bruker nettverket er for stort, vil aksesstiden for hver enkelte slave øke dramatisk. Denne ventetiden vil være direkte avhengig av antall slaver på nettverket og vil øke med økning i antall slaver.



***Figur 3.11**: Sammenheng mellom antall slaver og ventetiden*

Dette vil ødelegge for applikasjoner som trenger en rask responstid, siden de da blir nødt til å vente på sin tur og når den turen kommer kan det allerede være for sent. En løsning på dette kan være å innføre en *interrupt-funksjonalitet* som kan bli brukt for viktige meldinger som trenger rask responstid. Siden det er godt mulig at det kan inntreffe to eller flere interrupt samtidig, vil det være hensiktsmessig for slaver å sjekke først om felles mediet er ledig før de sender. Den mest kjente og oftest brukte metoden for sjekking av transmisjonsmedia er CSMA. CSMA vil bli brukt i vårt system i sin nonpersistent variant for å redusere sannsynligheten for kollisjoner.

Som konklusjon vil vi gjenta de valgene som er blitt foretatt. Det vil bli brukt TDMA som hoved multippel media aksessmetode, dvs hver slave får tildelt sin tidsluke som kan brukes til å sende og ta imot meldinger. Master vil si ifra til slaver når de er inne i sin tidsluke. Men siden ved økt antall slaver på nettverket øker også responstiden, tiden fra en slave vil sende en melding til den egentlig får sende, vil det bli brukt *interrupt-rutiner* for applikasjoner som krever rask responstid. Nonpersistent CSMA vil i denne sammenheng bli brukt for å forhindre to eller flere slaver aktivisere sine *interrupt-rutiner* samtidig.

Multippel aksess kontroll vil altså bestå av kombinasjon av TDMA og nonpersistent CSMA, med TDMA som hovedmetode.

3.3.3 Opp- og nedkommunikasjon

Nå som vi har bestemt oss for multippel aksess metode, kan vi gå videre og bestemme hvilken teknikk vi skal bruke for å skille kommunikasjonen fra og til Master.

Datakommunikasjon kan være enveis (*simplex*), toveis men en vei av gangen (*halv duplex*) eller begge veier samtidig (*full duplex*). Toveis kommunikasjon over et felles transmisjonsmedium kan organiseres med atskilte frekvensbånd *FDD (frequency division duplex)*, eller atskilte tidsintervaller *TDD (time division duplex)*.

3.3.3.1 FDD

Frekvenser organiseres slik at det brukes forskjellige frekvensområder for opp- og ned-kommunikasjon. Dette gir mulighet til at både Master og slaver kan ”snakke” når de vil og til og med samtidig. Ulempen med denne metoden er dens relativ kompleksitet. Man må passe på å ikke la frekvensområdene overlappe hverandre, til dette brukes det vanligvis et lite frekvensområde mellom opp- og nedkommunikasjon frekvensområdene. En annen ulempe er at frekvensområde for Master og slave kommunikasjon skal være mindre enn det opprinnelige frekvensområdet og det vil redusere overføringshastighet. Det kan også nevnes at prisen på sånne systemer vil generelt være høyere siden de inneholder komplekse frekvensfiltre.

3.3.3.2 TDD

I TDD brukes det et felles frekvensområde som brukes for både opp- og nedkommunikasjon. Med denne metoden slipper man å dele opp frekvensområde i to, og får høyere overføringshastighet for hver enkel kommunikasjon. Ulempen er at man må

sørge for synkroniseringen med atskilte tidsintervaller, slik at det er enten Master eller en av slavene som får ”snakke” til enhver tid.

3.3.3.3 Vurdering og valg

Vi vil helst unngå å dele opp båndbredde, siden det vil føre til mer komplekst og dyrt system, dessuten vil dataraten minke siden det vil bli brukt mindre båndbredde. Av disse grunner skal vi se bort fra FDD. TDD passer best for oss men vi vil ikke bruke denne metoden i full duplex modus. I vårt tilfelle vil kommunikasjon skje på en halv-duplex måte, både Master og slaver får ”snakke”, men normalt vil det ikke forekomme situasjoner når begge to får ”snakke” samtidig. Dette er på grunn av at kommunikasjonen skal følge ”respons-reply” mønster. Sagt med andre ord slaver får ikke snakke før Master vil gi beskjed om det.

3.4 Feilkontrollmekanismer

3.4.1 Hvorfor å bruke feilkontrollmekanismer

Kontrollmekanismer sørger for dataintegritet. Med dataintegritet menes vanligvis at mottaker kan verifisere at innholdet i en melding ikke er blitt endret, bevisst eller feilaktig, fra det tidspunkt meldingen ble avsendt til den ble mottatt.

Et velkjent faktum er at det kan oppstå feil under transmisjon. Dette gjelder spesielt i trådløse applikasjoner. Så her har kontrollmekanismer en viktig rolle.

Feil som kan oppstå ved overføring av digital informasjon:

- På bitnivå: Forveksling av 0 og 1
- På pakkenivå: tap, duplisering og feil rekkefølge på mottatte pakker

Noen årsaker til bitfeil.

- Gaussfordelt støy.
- Ikke gaussisk støy: f.eks. lyn og elektriske gnister gir mange feil i korte perioder og lange feilfrie perioder imellom.
- Tap av synkronisme: kan forekomme ved lange perioder med lite skifting i bitmønster da takt hentes fra datasignalet.
- osv

3.4.2 Grunnleggende prinsipp

Grunnleggende prinsipp er å tilføre signalet redundans, dvs flere bit enn datainnholdet tilsier. Ved å innføre redundans på en systematisk måte kan mottakeren i mange tilfeller oppdage og i noen tilfeller rette feil som oppstår på datainnholdet.

3.4.3 Feilkorrigering eller feildetektering

3.4.3.1 Feilkorrigerende metoder

Hensikten: avsender vil garantere sin klient/bruker feilfri kommunikasjon

En pakke sendes med redundant bit som mottaker kan bruke til å rette feil. Relativt mye tilleggsinformasjon kan komme med og det er ikke alle typer feil som kan rettes. Det ofte vil være en avveining mellom antall typer feil og mengde tilleggsinformasjon. Jo flere feil kan metoden rette, desto mer tilleggsinformasjon det kreves. Det vil igjen kreve mye båndbredde. En generell regel er derfor å bruke feilkorrigering på større blokker av data.

3.4.3.2 Feildetekterende metoder

En pakke sendes med redundant bit som mottaker kan så bruke til å oppdage feil. Siden det er enklere å oppdage en feil enn å rette, vil bruk av feildetekterende mekanismer ikke kreve like mye tilleggsinformasjon som feilkorrigerende mekanismer. Dette vil igjen spare båndbredden.

Bruk av feildetektering framfor feilkorrigering bør velges når det er enklere å sende en pakke på nytt enn å rette den hos mottaker. Dette er tilfellet med mindre pakker, når tiden å retransmittere pakken er mindre enn tiden det hadde tatt å sende ekstra tilleggsinformasjon og evt. rette på pakken hos mottaker.

3.4.3.3 Konklusjon og valg

Ut ifra drøftingen over kan vi konkludere at det er mer hensiktsmessig å bruke en feildetekterende kontrollmekanisme i vår protokoll. Dette er fordi vi forventer meldingene å bli nokså korte. Det vil derfor ta mindre tid å retransmittere en melding enn å sende med store mengder av redundant informasjon for hver sending slik at det kan rettes feil hos mottaker.

3.4.4 Valg av en feildetekterende metode

Vi skal se på noen mest brukte metoder, deres fordeler og ulemper og velge en metode som vil passe best for vårt formål.

3.4.4.1 3 times re-transmission

Dette er en veldig enkel måte å oppdage feil i transmisjon på. Senderen sender en pakke tre ganger. Mottakeren tar imot alle tre og sammenligner disse med hverandre. Hvis disse er like, antas det at transmisjonen gikk feilfritt.

Metoden er utrolig enkel, trenger lite ekstra funksjonalitet hos noder men bruker dessverre unødvendig mye båndbredde.

3.4.4.2 Paritetsbit

Det sendes en ekstra bit sammen med datapakke som heter paritetsbit.

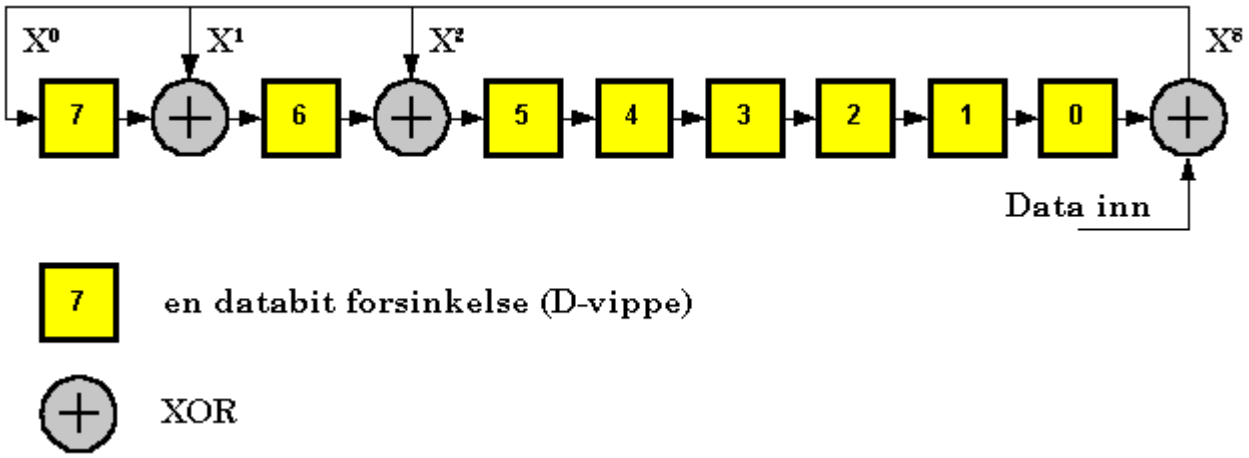
$D_7D_6D_5D_4D_3D_2D_1D_0P$ her betyr D_i databit nr i og P paritetsbit.

P settes til 1 hvis det er et odde antall enere i dataordet og 0 ellers. På mottaker siden kontrolleres antall enere slik at det antas å være en feil hvis summen av enere (inkludert P) er et odde tall. Metoden finner enkeltfeil, tre, fem eller syv feil, men ikke to, fire, seks eller 8 feil.

En variasjon av paritetsbit kontroll er rektangulær paritetskode som både kan detektere og rette enkeltfeil.

3.4.4.3 CRC (Cyclic Redundancy code)

Denne metoden baserer seg på at hele datainnholdet sees på som et binærkodet tall, dette divideres modulo-2 med et forhåndsdefinert tall, en generator og resten fra denne divisjonen legges til etter datainnholdet. Divisjonen kan realiseres enkelt med et tilbakekoblet skiftregister som vist på figur nede.



Figur 3.12: *CRC prinsippskisse*

Dette skiftregisteret realiserer divisjon med generatoren $X^8 + X^2 + X + 1$ (8-bit CRC). CRC kontrollordet ligger som bitkombinasjonen på utgangene av alle D-vippene, etter at siste databit er kommet inn i skiftregisteret. Denne resten legges til etter dataordet hos senderen og når det foretas en kontroll som inkluderer resten på mottaker siden, vil resultatet bli rest = 0. Denne metoden kan i praksis finne langt over 99% av alle feil. Den brukes stort sett i alle tilfeller der det er feildeteksjon i lange meldinger som er det viktigste. CRC kommer også 16- og 32-bits format. CRC metoder legger en, to eller fire (8-bits, 16-bits, 32-bits) bytes checksum til headeren.

3.4.4.4 Konklusjon og valg

Metodene som er beskrevet her har alle sine fordeler og ulemper. Men siden det allerede finnes støtte til CRC-metode i SPP, går vi for denne metoden. CRC er en veldig sikker metode og vi vil bruke 8-bits CRC på headeren og 16-bits CRC på data, på samme måte som det er gjort i SPP.

3.5 Autentisering

3.5.1 Hva er autentisering?

Autentisering er den første sikkerhetstjenesten, som skal kontrollere at brukeren som skal ha tilgang til et system, virkelig er den han/hun utgir seg for å være.

Ved autentisering inngår en part som skal autentiseres, en part som trenger forsikring om ektheten av den annen partis identitet. I tillegg kommer informasjonen som benyttes for autentiseringen.

Brukeren kan identifisere seg på flere måter. Den første måten å identifisere seg på er å bruke noe man kjenner til – autentiseringsinformasjon, og/eller noe man har (for eksempel et smartkort).

Figur 3.13: *Autentisering prinsippskisse*

Forskjellige metoder for autentisering gir forskjellig grad av sikkerhet. Det er også veldig viktig å være klar over hva det er som virkelig autentiseres. Noen ganger er det identiteten til et menneske, andre ganger er det identiteten til en mobiltelefon og ikke hvem som faktisk bruker den.

3.5.2 Hvorfor autentisere i vårt nett?

Generelt kan det sies at sender og mottaker ønsker å få bekreftet hverandres identitet. Hensikten er å skaffe bekreftelse på påstått identitet, i vårt tilfelle – utstyridentitet. Hovedtrussel er at en entitet kan gi seg ut for å være en annen entitet.

3.5.2.1 Trusselbilde i vårt system

I vårt system kan graden av den nødvendige sikkerheten diskuteres. Trusselgrad er direkte avhengig av applikasjonsområde til et system. Vi kan vurdere autentiseringsbehovet for forskjellige applikasjonsområder på eksempler av

hjemmeautomatiserings- og industrielle fjernmåling og styringssystemer. Disse systemene vil beskrive to diametralt forskjellige sikkerhetsbehov.

1 Hjemmeautomatiseringssystem

Hvis, for eksempel, systemet vil brukes i hjemme for å styre forskjellige brune og hvitevarer, så er det vel ikke så mange som vil prøve seg på å knekke det. Utbyttet vil nok ikke være stort.

Hjemmeautentiseringssystemer byr på et annet problem: Hva skjer hvis to eller for den saks skyld flere systemer vil ligge såpass tett innpå hverandre at slaver vil kunne registrere seg hos flere Mastere? Det kan selvfølgelig skape en god del problemer for brukeren. I den sammenheng kan autentisering eller pålogging være løsningen.

2 Fjernmåling og styring av sensitive applikasjoner

Hvis systemet vil brukes for fjernmåling og styring, vil trusselbilde være noe annerledes. I noen applikasjoner kan det være veldig viktig at uvedkommende ikke skal kunne få greie på transmitterende signal eller enda verre endre på det. I dette tilfelle bør mer avanserte metoder brukes som vil garantere sikrere tilgangskontroll.

Siden vi har kommet til den beslutningen at det vil finnes to helt forskjellige trusselbilder og dermed to ulike måter å møte dem på, vil vi drøfte de hver for seg.

3.5.3 Enveis- eller toveisautentisering?

Problemet som skal løses ved hjelp av autentiseringsmekanismer er at en maskin kan utgi seg for å være en annen maskin enn den, den er. Dette problemet kan være veldig viktig for både slaver og Master som kan være veldig sårbare mot falske Mastere og slaver.

For å løse dette problemet finnes det to grupper av autentiseringsmetoder: enveis- og toveisautentiseringsmetoder.

I vårt tilfelle:

- Enveisautentisering betyr at slaver autentiserer seg for Master.
- Toveisautentisering betyr at slaver autentiserer seg for Master, og Master for slaver.

Enveisautentisering er selvfølgelig enklere og raskere, mens toveisautentisering er sikrere i det den kan sikre mot trusler som enveisautentisering er sjanseløs mot (for eksempel falske Mastere). Det fører oss tilbake til vurderingen av trusselbildet for hvert enkelt applikasjonsområde. Som det er nevnt tidligere kan vi vurdere autentiseringsbehovet for forskjellige applikasjonsområder på eksempler av hjemmeautomatiserings- og industrielle fjernmåling og styringssystemer.

1 Hjemmeautomatiseringssystem

I disse systemer vil det være nok med enveisautentisering, da slaver vil autentisere seg for Master. Dette vil forhindre Master i å sende meldinger til slaver fra andre nettverk som tilfeldigvis ligger tett innpå hverandre. Når det gjelder toveisautentisering så hadde det vært overflødig hvis Master skulle også autentisere seg for slaver. Dette er på grunn av at

faren for falske Mastere vil være nokså begrenset i hjemmeautomatiseringssystemer, grunnet lite utbytte av slik bedrageri.

2 Fjernmåling og styring av sensitive applikasjoner

I disse systemer er utbytte av å bruke falske Mastere er en hakk høyere, siden man da får tilgang til informasjon som kan være av interesse for en god del uvedkommende. Fordeler kan også være store hvis man får tilgang til nettverksressurser og vil kunne sende falske kommandoer til slaver og få dem til å gjøre ting de ellers ikke skulle gjort.

I sånne applikasjoner vil man tro at det er hensiktsmessig å få autentisert både Master og slaver.

3.5.4 Autentiseringsmetoder

Felles for ulike autentiseringsmekanismer er at prosessen bør være helt usynlig for brukeren (nå snakker vi ikke om påloggingsmetoder hvor man bes om å taste inn brukernavn og kode). Vi skal nevne noen mest kjente metoder og så velge en for vårt system.

3.5.4.1 Fysisk autentisering

Dette er den mest enkleste autentiseringsmetoden. Brukerne gis tilgang til systemet ved ved bruk av, for eksempel, smartkort. Hvis en bruker har den nødvendige hardwaren, da får han/hun komme seg inn i systemet, ellers ikke. Denne metoden kan brukes alene, men oftest brukes den sammen med en annen mer avansert autentiseringsmetode for å tilby høyere sikkerhetsnivå.

3.5.4.2 Åpen autentisering

Definisjon: alle slaver med gyldige MAC-adresser autentiseres og assosieres med Master.

Med gyldige adresser menes at adresser skal ha samme format. I motsatt fall vil ikke slaver med adresser som har en annen format bli autorisert og assosiert med Master.

En utrolig enkel metode, men som dessverre ikke løser problemet med flere Mastere som ligger tett innpå hverandre. For da vil alle slaver ha MAC-adresser av samme format og skal kunne autentiseres hos flere Mastere samtidig. Det er heller ikke en sikker metode siden det kun skal til å kunne dette adresseformatet for å trenge seg på i systemet.

3.5.4.3 Autentisering med utvalgte MAC-adresser

Definisjon: kun slaver med utvalgte MAC-adresser autentiseres og assosieres med Master. MAC-adressene må på forhåndsprogrammeres i Master.

En enkel metode som garanterer en sikker autentisering og som ikke lar uvedkommende å trenge seg på i systemet, så lenge de ikke har vært i fysisk kontakt med Master og omprogrammert den. Den største ulempen er rart nok forårsaket av samme grunn som hovedfordelen – man må manuelt programmere inn MAC-adressene som gir

tilgang til systemet. Men når det er gjort kan metoden regnes som ganske sikker for applikasjoner som for eksempel hjemmeautomatiseringssystem.

Denne metoden er sårbar for avlytting (man-in-the-middle-attack) og hvis en uvedkommende avlytter kommunikasjonen lenge nok så vil han få greie på registrerte i systemet MAC-adresser og bruke dem for å komme seg inn på nettverket. Derfor i applikasjoner hvor kostnaden å få en uvedkommende i systemet er høyere (fjernmåling og styring) bør det brukes andre metoder.

3.5.4.4 Delt nøkkel autentisering

Definisjon:

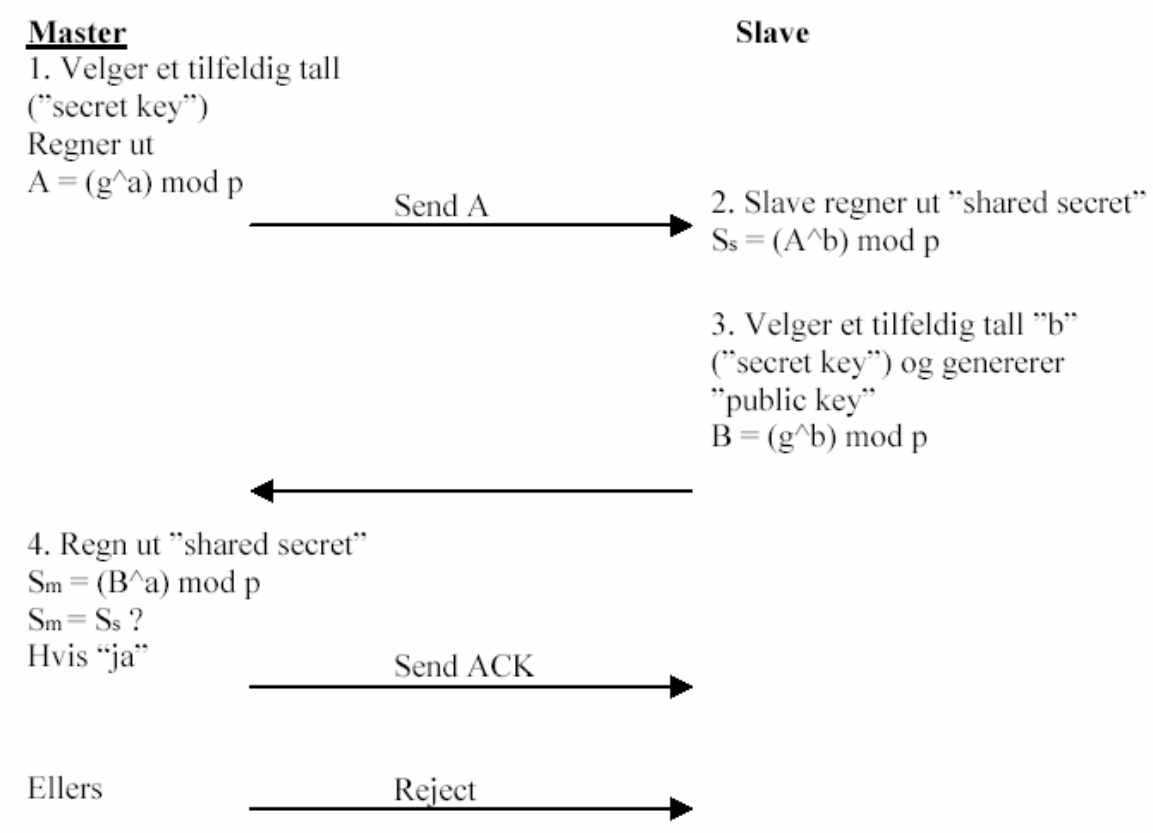
- kun slaver med utvalgte MAC-adresser autentiseres og assosieres med Master (ikke påkrevd men anbefalt).
- Identitet til slave verifiseres med en hemmelig nøkkel (delt nøkkel) som er lik for alle slaver hos en Master.
- Data krypteres

Dette er en veldig sikker mekanisme for autentisering så lenge uvedkommende ikke har adgang til den hemmelige nøkkelen. Til og med kjennskap til MAC-adresser vil ikke hjelpe her: man får adressere pakker til Master, men uten nøkkelen får man ikke kryptere dem riktig og Master rett og slett får ikke dekryptere dem og dermed lese. Ulempen med denne metoden er at man på en eller annen viss må distribuere hemmelige nøklene. Nøklene må skiftes fra tid til en annen for å forhindre at noen skal få greie på dem.
Diffi-Hellman nøkkeldistribusjon mekanisme er et godt valg her.

3.5.4.5 Diffi – Hellmann

Algoritmen autentiserer kun én vei, dvs kun gir støtte til Master-Slave autentisering.
Master og slaver får på forhånd oppgitt to parametrer ”g” og ”p” som skal brukes i autentiseringen.

Fremgangsmåte er vist i figur 3.14:



Figur 3.14: *Prinsippskisse av Diffi-Hellmann algoritmen*

En uvedkommende må satse ganske mye på for å få tak i disse hemmelige nøklene og lese eller styre kommunikasjonen i systemet.

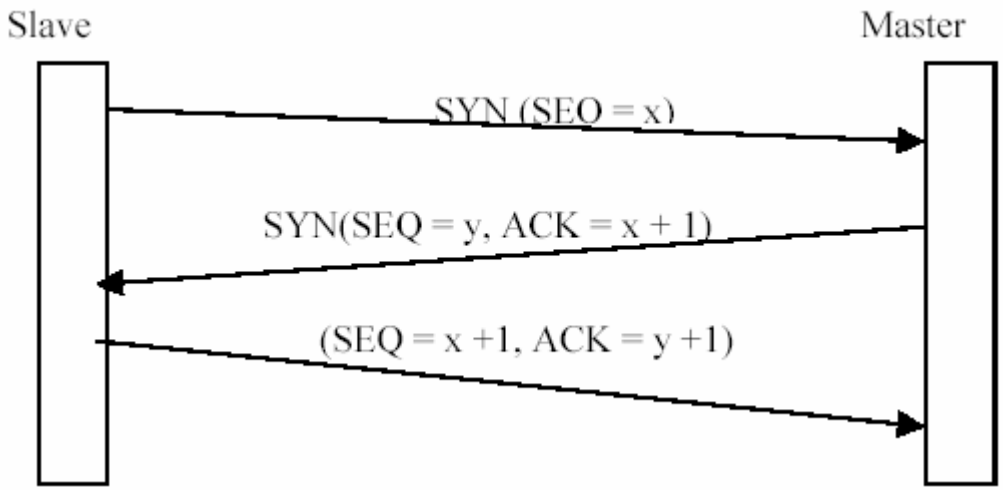
3.5.4.6 Handshake mekanisme

Denne metoden kombinerer enkelhet og sikkerhet. Den er ikke like sikker som *Delt nøkkel autentisering* (hvis brukes uten datakryptering), men mye sikrere enn *Åpen autentisering* og *Autentisering med utvalgte MAC-adresser*.

Virkemåte:

1. Noden som begynner kommunikasjon (i vårt tilfelle – slave) velger et tilfeldig tall (x) og sender det over til Master.
2. Master tar imot meldingen, legger 1 til x (x+1), velger et tilfeldig tall (y) og sender begge tallene over til slave.
3. Slaven tar imot og sammenligner (x+1)-tallet med det sin egen versjon som slaven selv har regnet ut. Hvis tallene stemmer overens – Master er autentisert. Videre legger slaven 1 til y (y+1) og sender over til Master.
4. Master tar imot meldingen og sammenligner (y+1)-tallet med sin egen versjon som Master har regnet ut selv. Hvis tallene stemmer overens – slaven er autentisert.

5 Nå har Master og slaven autentisert hverandre og fått bekreftet at den de snakker med er den, den gir seg ut for.



Figur 3.15: *Prinsippskisse av 3-veis håndshake algoritmen*

3.5.5 Valg og begrunnelse

Siden vår protokoll lages med tanke på spesiell hardware, er det ganske naturlig at vi ikke skal se bort fra fysisk autentisering. Har man Chipcon CC1010-kort, har man den nødvendige startbetingelse for å komme seg inn i systemet. Vi skriver startbetingelse fordi fysisk autentisering alene kan ikke bli betraktet som en sikker autentiseringsmekanisme. Men et bra utgangspunkt er den.

Ved valg av tilleggsmetode, hvis vi kan kalle det slik, er en grundig vurdering av trusselbildet veldig viktig. Som vi har nevnt, så ser vi for oss to hovedapplikasjonsområder: hjemmeautomatiseringssystemer og fjernmåling og styring i industriell sammenheng. Disse gruppene har forskjellige sikkerhetskrav og derfor bør de gjerne bruke forskjellige autentiseringsmetoder.

Hjemmeautomatiseringssystem
Dette applikasjonsområdet krever ikke altfor avanserte sikkerhetsmekanismer grunnet begrenset verdi av informasjon som blir transmittert. Da er det ingen grunn til å gjøre systemet mer komplisert enn det, det egentlig bør være, og vi kan med god samvittighet se bort fra mer avanserte metoder. På denne måten står valget mellom *Åpen autentisering*, *Autentisering med utvalgte MAC-adresser* og *Handshake* metoder. En av disse metoder kan bli brukt i tillegg til den fysiske autentiseringen.

Åpen autentisering løser ikke problemet med overlappende celler hvor slaver får anledning til å registrer seg hos flere Mastere samtidig eller registrere seg hos en feil Master som bare tilfeldigvis var først ute med sin registreringsanmodning.

Autentisering med utvalgte MAC-adresser ser ut til å passe bra til hjemmeautomatiseringssystem. Den største ulempen er at man må manuelt programmere inn MAC-adresser til slaver som skal assosieres med Master. Det samme gjelder hvis

man vil koble en slave til et eksisterende system. I små og mellomstore systemer kan denne metoden imidlertid være et rimelig bra valg.

Med *handshake-metoden* slipper man problemet med manuelt registreringen, men får et annet et. Det at slaver velger et tilfeldig tall i begynnelsen av registreringen og sender dette tallet til Master kan medfører at den blir registrert hos en annen Master som bare tilfeldigvis var raskere til å svare. For å unngå dette må man planlegge celler nøye på forhånd. Dette ser ut til å være vanskelig å implementere. I tillegg vil denne form av autentisering ta lengre tid enn de to foregående metoder og ha større sjanse for at meldingene blir mistet eller ødelagt under transmisjon siden her er det 3 meldinger som skal sendes.

Ut ifra drøftingen over blir det valgt *Autentisering med utvalgte MAC-adresser* - metoden for hjemmeautomatiseringssystem. Dette er egentlig et mellompunkt mellom mer komplekse men også mer sikre metoder og noen enkle metoder som gir lav sikkerhetsbeskyttelse.

Fjernmåling og styring i industriell sammenheng

For dette applikasjonsområdet som krever nokså høy grad av sikkerhet valget står mellom to metoder i tillegg til fysisk autentisering på grunnlag av CC1010 kretskort.

Handshake hvis den blir brukt sammen med kryptering av datainnholdet kan være en veldig sikker metode, men har en stor ulempe beskrevet i avsnittet over.

Delt nøkkel autentisering fra sin side er veldig sikker metode og på grunn av at den vanligvis brukes sammen med *Autentisering med utvalgte MAC-adresser* har ikke problemer med overlappende celler. Dette er det mest hensiktsmessige valget i industriell sammenheng.

3.6 Kryptering

3.6.1 Hva er kryptering?

Kryptering brukes for å garantere konfidensialitet. Konfidensialitet skal sikre at meningsinnholdet i meldingen som sendes over åpne nett (for eksempel Internett eller trådløse nett) ikke skal kunne leses eller forstås av uvedkommende. I datakommunikasjon kan kryptering benyttes for å oppnå sikker kommunikasjon over et usikkert medium. Dette er den mest effektive måten å oppnå datasikkerhet på. Krypteringen skjer normalt ved at data automatisk krypteres når den forlater senderen og dekrypteres når den kommer frem til mottakeren. For å lese en kryptert data må man ha tilgang til en hemmelig nøkkel som setter man i stand til å dekode dataen.

***Figur 3.16**: Kryptering – prinsipiell skisse*

3.6.2 Trusselbilde eller hvorfor kryptere i vårt nett?

Det er alltid viktig å ta standpunkt om man virkelig har behov for å benytte kryptering, og i så fall i hvilken grad. Det nødvendige sikkerhetsnivået vil variere fra nett til nett. Derfor er det viktig å fastlegge hva som er det nødvendige nivået for et nett. Som følge av pkt. 3.5 har vi kommet fram til at det vil finnes to hovedapplikasjonsområder for vår protokoll: hjemmeautomatiseringssystemer og fjernmåling og styring i industriell sammenheng. Forskjellen mellom disse to er i hvordan de tar seg av sikkerhetsproblematikken. I hjemmeautomatiseringssystemer som tar seg av styring av forskjellige brune- og hvitevarer eller sagt på en mer generell måte, som ikke har med sensitive data å gjøre som er verd å "stjele" eller "redigere", er kryptering vel ikke like viktig som i industrielle systemer hvor mye kan gå galt hvis uvedkommende får kontroll over eller tilgang til systemet. Spesielt data som overføres trådløst kan med riktig teknisk utstyr fanges opp av alle som befinner seg i dekningsområdet. Derfor er det spesielt viktig å beskytte disse dataene med kryptering.

Med dette kan vi konkludere at det ikke er nødvendig med krypteringen i hjemmeautomatiseringssystemer men i industrielle fjernmåling- og styringssystemer bør krypteringen brukes.

3.6.3 Valg av kryptering metode

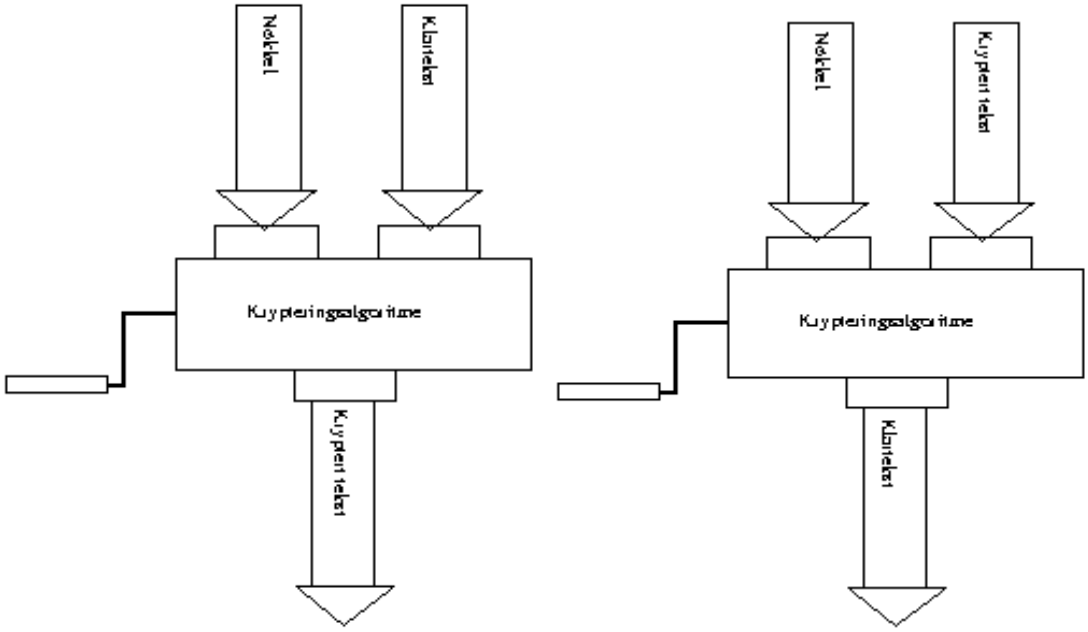
Det finnes en mengde krypteringsmetoder som kan bli brukt i vårt system. Men for å holde ting enkle, går vi for DES (Data Encryption Standard), metoden som det finnes støtte til i CC1010's hardware.

Mest hensiktsmessig hadde vært en løsning hvor brukeren kunne slå på eller av krypteringsmekanismer avhengig av om det er behov for det. Typiske eksempler for dette behovet, som vi alt har vært innom, er hjemmeautomatiseringssystem og fjernmåling og styringssystem i industrisammenheng. Siden trusselbildet i først nevnte system er ikke like stort som i det andre, vil vi foreslå at man kan la være å bruke kryptering i

hjemmeautomatiseringssystemer, men bør gjøre det i industrielle eller andre systemer som har med sensitive data å gjøre. I spesielle tilfeller kan 3DES-krypteringen brukes.

3.6.4 Litt om DES-kryptering

DES er basert på en symmetrisk kode, dvs en hemmelig nøkkel som er felles for begge parter i en kommunikasjon. Dvs at symmetrisk kryptering i prinsippet fungerer slik at krypteringsnøkkel blir brukt både til kryptering og dekryptering.



.

Figur 3.17: *Symmetrisk kryptering og dekryptering – prinsipiell skisse*

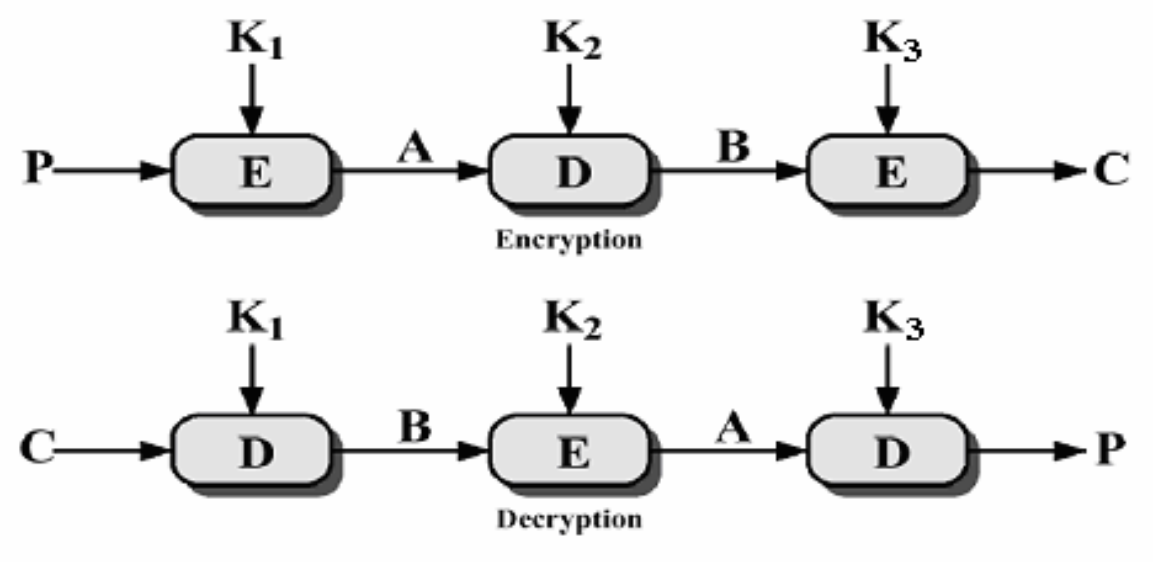
Følgende er en skisse av DES-virkemåten

DES bruker en nøkkel på 64 bit. Egentlig er nøkkelen bare på 56 bits fordi hver 8. bit er en paritetsbit. DES deler opp klarteksten i blokker på 64 bits, og krypterer hver blokk med nøkkelen. DES krypterer hver 64-bits blokk av klarteksten ifølge måten beskrevet i neste avsnittet.

Blokken utsettes for innledende transponeringer. I en løkke som gjennomføres 16 ganger kombineres blokken og nøkkelen med hjelp av en kombinasjon av XOR-operasjoner og transponeringer. Mellom hver runde i løkken roteres bitmønsteret til nøkkelen. Det hele avsluttes med enda en transponering. Resultatet er en chiffertekst bestående av 64 bits blokker. For å dekryptere chifferteksten trenger man den samme nøkkelen som ble brukt for krypteringen.

Vanlig DES regnes som svak kryptering i dag. Man kan oppnå høyere sikkerhetsgrad ved å utvide algoritmen til 3DES ved å kjøre DES-algoritmen tre ganger. Data krypteres med tre forskjellige 56 bit nøkler. Innkommende data krypteres med den første krypteringsnøkkelen (K1), den krypterte data krypteres igjen med den andre

krypteringsnøkkelen (K2) og det som fås ut krypteres enda en gang med den tredje nøkkelen (K3). For å dekryptere gjøres det samme, men i motsatt rekefølge.



Figur 3.18: *3DES krypteringsalgoritme – prinsipiell skisse*

Den store fordelen med *DES* er at både koding og dekoding er lite tidkrevende og at data kan krypteres uten store kostnader. I tillegg støttes både konfidensialitet og integritet under transport av data.

Ulempen er at både sender og mottaker må kjenne til krypteringsnøkkelen. Ettersom både avsender og mottaker bruker samme nøkkel må det være en sikker kanal som kan transportere nøkkelen fra avsender til mottaker. Dette problemet blir enda mer akutt hvis avsenderen vil sende en melding til flere mottakere.

Hvordan skal en få til sikker distribusjon av *DES*-nøkler? Enten kan begge parter møtes eller nøkkel må sendes med kurer. Innenfor en lukket mengde med deltakerer, som i vårt tilfelle, vil dette kunne gå bra, men generelt er det lite ønskelig at ens sikkerhet er avhengig av at alle motparter klarer å holde felles nøkkel hemmelig.

4 Noen teoretiske aspekter i protokolldefinisjon

4.1 Sjekkpunkter for protokolldefinisjon

En fullstendig protokollspesifikasjon bør inkludere følgende fem punkter:

1. Service, oppgaver, omfang protokoll sørger for (Servicespesifikasjon)
2. Antagelser om protokollens omgivelser
3. Kommunikasjonskanal
4. Protokollens vokabular
5. Prosedyreregler

4.1.1 Servicespesifikasjon

4.1.1.1 Anvendelsesområdet

Protokollen må kunne ha en troverdig ende til ende melding utveksling. Meldingsutveksling vil foregå mellom Master og slaver. Hjemmeautomatiseringssystem og industrielle fjernmåling og styringssystemer tenkes å være en typisk applikasjon for MSP som skal kunne kjøres på CC1010. Et brukscenario kan være at de fleste funksjoner og enheter i et hus og/eller i en industribedrift er koplet sammen i nettverk, og der det meste kan styres og overvåkes ved hjelp av en sentral styringsenhet. En sentralenhet skal være CC1010 med masterprogram og enheter (slaver) som sentralenheten skal styre vil bruke samme kretskort med slaveprogram. Funksjoner som skal kunne styres/kontrolleres er for eksempel alarm, forskjellige sensorer, lys, varme, TV, video, komfyr, brødrister etc. Noen av disse trenger en kort responstid (alarm, noen av sensorer, lysbrytere, TV osv), andre ikke (varme, komfyr osv). Aksess skal skje trådløst.

4.1.1.2 Krav til systemet

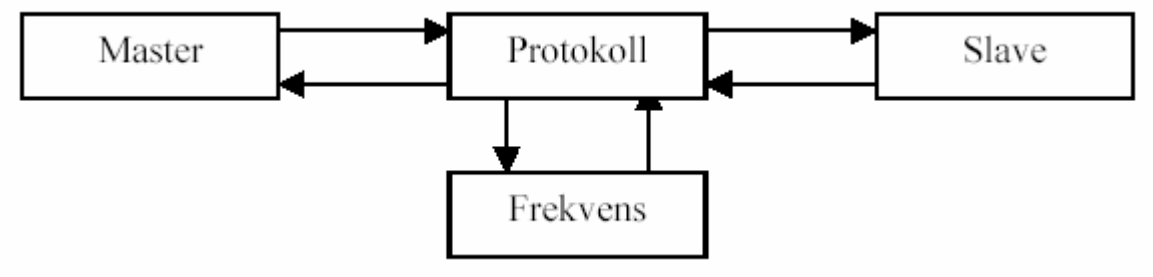
Med tanke på anvendelsesområdet kan det stilles følgende krav til systemet:

- Master styrer all kommunikasjon.
- Slaver og Master må kunne både sende og motta meldinger.
- Kun en frekvenskanal blir brukt. TDMA blir brukt for multiple aksess.
- Antall noder skal ikke være større enn 255, inkludert Master (8-bits adresser: 2^8 = 256 mulige adresser; ”0000 0000” – er reservert for broadkast adressering). Et typisk hjemmeautomatiseringssystem eller industrisystem kan neppe være større.
- Noen enheter vil kreve raskere responstid enn andre. Det må legges inn en mulighet for ”umiddelbar” aksess for applikasjoner som krever det.
- Alle noder har et unikt serienummer.
- Registrering av slaver blir gjort manuelt (serienummer blir tastet inn fra brukeren).
- Sikker transmisjon av meldingene.

4.1.2 Antagelser om protokollens omgivelser

Protokoll trenger å kommunisere med omgivelser for å utføre dens funksjoner.

Noder skal kunne kommunisere med hverandre ved hjelp av en protokoll. Protokoll definerer reglene for kommunikasjon mellom noder. Kommunikasjon vil foregå via såkalte kanaler, som er ingen fysiske kanaler men bare et teoretisk begrep for veien kommunikasjon vil foregå på.



Figur 4.1: *Protokollens omgivelser*

Frekvensproblematikken står veldig sentralt i vår protokoll, dette er grunnet det at kommunikasjon skal foregå trådløst. Dvs si at man må få til en sikker frekvenspolitikk, for å få til at protokoll skal fungere i det hele tatt.

4.1.3 Kommunikasjonskanal

Kanal er en frekvens som er delt opp i mange deler (*TDMA-tidsluker* i vårt system), som er fordelt på like mange slaver.

- Størrelsen på tidsluke er av fast, på forhånd bestemt verdi.
- Master skal distribuere retten til å benytte seg av kommunikasjonskanal til slaver etter rundtur, resten av tiden vil slaver ”tie”. Denne ventetiden vil være nokså kort, siden det forventes at en typisk melding vil være et signal om å slå på/av en enhet, nokså skal være kortvarig.
- I tillegg til tidsluker for de forskjellige slaver, vil det være en tidsluker for *interrupt*, for at de slavene som har interruptfunksjonalitet vil få sende utenom sin tur hvis applikasjon krever en rask respons.
- Meldingene vil være veldig korte og det vil til enhver tid være kommunikasjon mellom Master og én slave, derfor blir det ikke tatt hensyn til flytkontroll.

4.1.4 Protokollens vokabular

En mengde av meldinger som brukes til å implementere protokoll heter protokollens vokabular. Disse meldingene bør helst ha omtrent samme meldingsformat. På denne på måte kan samme regler bli anvendt til forskjellige meldinger.

4.1.5 Prosedyreregler

Reglene som beskriver alt som skjer i kommunikasjon. Alt som skjer utenom disse reglene blir betraktet av protokoll som feil.

4.2 Introduksjon til MSP protokolldesign

Før vi begynner med selve designet vil vi gå tilbake i rapporten for å friske opp de teknologiene som ble valgt til bruk i vår protokoll. Oppfriskningsrunde presenterer vi i form av tabell.

Tabell 4.1: Valgte metoder

Nettverkstype: Master-Slave vs Peer-to-Peer	**Nettverks-topologi**	**Multippel aksess**	**Kommunikasjons-modi**	**Kontrollmekanisme**
Master-Slave	Stjerne	TDMA + CSMA	Halv duplex	8-bits CRC på headeren 16-bits CRC på data

Autentiseringsmekanisme	
Hjemmeautomatiseringssystem	Fjernmåling og styring i industrisammenheng
Fysisk autentisering + Autentisering med utvalgte MAC-adresser	Fysisk autentisering + Delt nøkkel autentisering

Krypteringsmekanisme	
Hjemmeautomatiseringssystem	Fjernmåling og styring i industrisammenheng
Kan bli unnlatt	DES eller 3DES

Ut fra disse kriteriene skal vi ”komponere“ en protokoll for kortholdsradiokommunikasjonssystem. Disse valgte mekanismer og teknikker representerer på en måte små komponenter i et større system. Det kan også sammenlignes

med puslespill, og akkurat som i puslespill må vi sette smådeler sammen på riktig måte for å oppnå riktig resultat. Resultatet i vårt tilfelle er en protokoll som vil fungere for kortholdsradiokommunikasjonssystemer.

Man kunne begynne med protokolldesign på samme måte som man ofte gjør i puslespill og sette detaljene sammen på mer eller mindre tilfeldig måte. Denne måten kan selvfølgelig med en del prøving, testing og feiling fungere for småsystemer, men en stor ulempe vil den likevel ha, nemlig at det vil ta en god del tid for å komme fram til en riktig løsning etter noen runder med feilavgjørelser og valg.

Det man trenger for å unngå situasjoner som er preget av mistro og forvirring som kan dukke opp i designeprosess, er et regelverk eller en referansemodell som tilsier hvordan man bør sette systemet sammen og hvordan alt i systemet skal henge sammen. For grunnleggende protokolldesign er det vanlig å bruke referansemodeller for å få bedre oversikt og for å få gjort ting konsekvent.

Det finnes forskjellige teknikker og referansemodeller som kan bli brukt i protokolldesign. Den mest kjente og oftest brukte er OSI-referansemodellen for datakommunikasjon (Open Systems Interconnection). Vi skal benytte oss av teorien og fremgangsmåter som ligger bak OSI-modellen i designeprosessen.

4.3 Litt om OSI-referansemodell

OSI-modellen for datakommunikasjon er definert av den internasjonale standardiseringsorganisasjonen ISO og den blir derfor også omtalt som ISO-OSI.

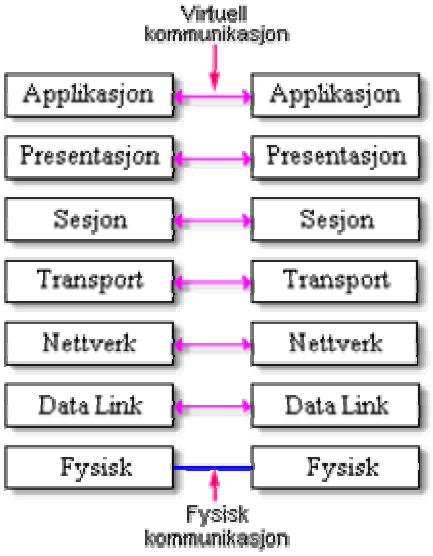
OSI referansemodell forsøker å definere hvor funksjonalitet skal plasseres i et nettverk, basert på fornuftige vurderinger når modellen ble laget. Det er et åpent spørsmål om disse er egnet for fremtiden. I praksis har det vist seg at nye systemer ofte implementerer funksjonalitet på tvers av anbefalingene i denne lagdelte modellen og at modellen modifiseres og utvides, eksempelvis ved at det defineres sublag innenfor de eksisterende lagene. Det eksisterer derfor for tiden ingen klare og dominerende referansemodeller for hvor funksjonalitet skal plasseres i et system. Ikke alle lag i OSI-modellen er i bruk i alle systemer. Faktisk er det få systemer som benytter OSI-modellen slik den er definert.

For å si det veldig kort er OSI-modellen en standard som gir en beskrivelse av hvordan nettverks hardware og software jobber sammen i en lagbasert form for å gjøre kommunikasjon mulig og definerer et rammeverk for implementering av protokoller i sju lag.

Disse er:

Figur 4.2: *Lagene i OSI-modellen*

Lagene spesifiserer forskjellige funksjoner og tjenester på forskjellige nivåer. Hvert lag yter tjenester eller handlinger som forbereder dataene for leveranse over nettverket. Lagene er atskilt og får kontakt gjennom grensesnittene mellom hverandre. Hvert lag har som oppgave å yte tjenester til det laget som ligger over, samtidig som det skjermer det høyereliggende laget for hvordan tjenestene faktisk utføres. Lagene er konstruert slik at de "tror" de kommuniserer direkte med tilsvarende lag hos mottaker – dette er virtuelt. Dette har den fordel at det er enklere å beskrive et system og hvert lag er ofte uavhengig av andre, dvs det gjør det mulig å bytte ut teknologi i ett lag uten å påvirke de andre lagene. Logisk sett kommuniserer de ulike lagene med hverandre, og de underliggende lagene er transparente for dem. Fysisk sett går dataene fra lag til lag inntil de når det fysiske laget, hvor data blir transportert til mottaker i nettet.



Figur 4.3: *Kommunikasjon i OSI-modellen*

4.4 Eksisterende protokoller for Master-Slave nettverk

Dette avsnittet vil ta for seg noen allerede eksisterende nettverksprotokoller. Disse kan skape fremverk og gi mer innblikk i hva det innebærer å definere en protokoll. Protokollene blir vurdert med tanke på vårt applikasjonsområde og noen av protokollenes funksjonelle detaljer kan bli gjenbrukt i vår nye protokoll.

4.4.1 Simple Packet Protocol – SPP

SPP er utviklet av Chipcon for bruk i sine kretskort. Det er en veldig enkel protokoll for sending og mottak av data. Data blir da sendt jevnlig med en enkel feilkontroll (CRC). Med sending i SPP menes det broadkast sending, da alle noder kan ta imot en og samme pakke.

4.4.1.1 Hovedtrekkene ved SSP

- 255 fysisk nettverks adresser og en spesielt broadkast adresse (SPP BROADCAST)
- Feildetekterende metoder av 8-bit CRC som brukes for overhead og 16-bit CRC som brukes for datadelen
- Automatisk retransmisjon (valgfritt)
- Mottakelse tidsavbrudd (valgfritt)

4.4.1.2 Pakkeformat

SYNC	DAB	SAB	DATA length	FLAG	CRC-8	DATA	CRC-16

Figur 4.4: *Pakkeformat i SPP*

SYNC: Synkroniserings byte
DAB: Destinasjons adresse byte
SAB: Source (Kilde) Adresse byte
DATA length: Lengden av data delen
FLAG: - SPP_ACK_REQ, ber om å få ACK
- SPP_ACK, ACK som brukes intern
- SPP_ENCRYPTED_DATA, om data som sendes er kryptert eller ikke
- SPP_SEQUENCE_BIT, brukes for å identifisere pakken
CRC-8: Dette brukes til å sjekke feil på overhead
CRC-16: Dette brukes til å sjekke feil på datadelen

4.4.1.3 SPP fra vårt synspunkt

SPP er en grei protokoll som kan brukes hvis nettverket ikke er for stort og komplisert. MS nettverket kan imidlertid vokse seg til å bli større og mer kompleks enn det som kan

håndteres av SPP. Siden SPP allerede er en ”standard” protokoll som brukes i de forskjellige kretsene hos Chipcon, har Chipcon et ønske om at vi skulle utvikle en ny protokoll med utgangspunktet i SPP.

4.4.2 S.N.A.P – Scaleable Node Address Protocol

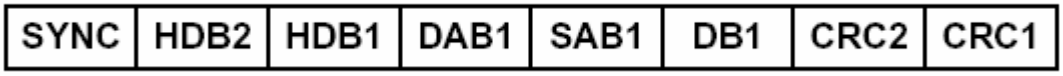
SNAP er en protokoll utviklet i 1998-2002 av et svensk firma ”High Tech Horizon” med tanke på å kunne brukes i første rekke i hjemmeautomatiserings- og kontrollsystemer. Dette er imidlertid ikke en absolutt begrensning siden SNAP er veldig lett å bruke og er en veldig fleksibel protokoll som kan brukes i ulike miljøer. Dette er ikke minst takket være faktumet at protokollens kompleksiteten kan styres og kan være forskjellig, alt etter brukerens ønsker og behov. SNAP kan brukes enten uten forskjellige finneser som flager og feilkontrollmekanismer, eller man kan velge hvilke flager av mulige 24 og feilkontrollmekanismer som vil bli brukt avhengig av krav som stilles av en konkret applikasjon.

4.4.2.1 Hovedtrekkene ved SNAP

- Lett å implementere på mange forskjellige plattformer og for mange programmeringsspråk
- Skalerbarhet
- 8 forskjellige feildetekterende metoder
- 24 forskjellige flagger
- Kan brukes både i MS og PTP nettverk
- Mediet uavhengig

4.4.2.2 Pakkeformat

Først sendes det en synkroniserings byte, etter det kommer to header definisjons bytes. Disse to definerer hva de kommende bytene betyr og hvilke byte burde eller ikke burde sendes. Dette gjør at protokollen er veldig skalerbar, akkurat som navnet tilsier.



Figur 4.5: *Pakkeformat i S.N.A.P*

SYNC Synchronization byte
Den første byten er en synkroniseringsbyte, som er satt til 01010100 som standard.

HDB2 The Header Definition Byte 2
Definerer hvor mange byte som vil bli brukt for destinasjons- og kildeadresser (Destination Address Byte, DAB, Source Address Byte, SAB) samt flagene for protokollen (Protocol specific Flag Bytes, PFB). HDB2 inneholder også ACK og NACK bit’er.

HDB1 The Header Definition Byte 1
Har samme funksjon som HDB2, men definerer antall data bytes (Number of Data Bytes, NDB) som skal sendes, og en feildetekterende metode (Error Detection Method, EDM) som definerer hvor mange Checksum bytes som vil bli sendt. HDB2 innerholder også en kommando modus bit (command mode bit, CMD), som indikerer om data delen er en kommando eller data.

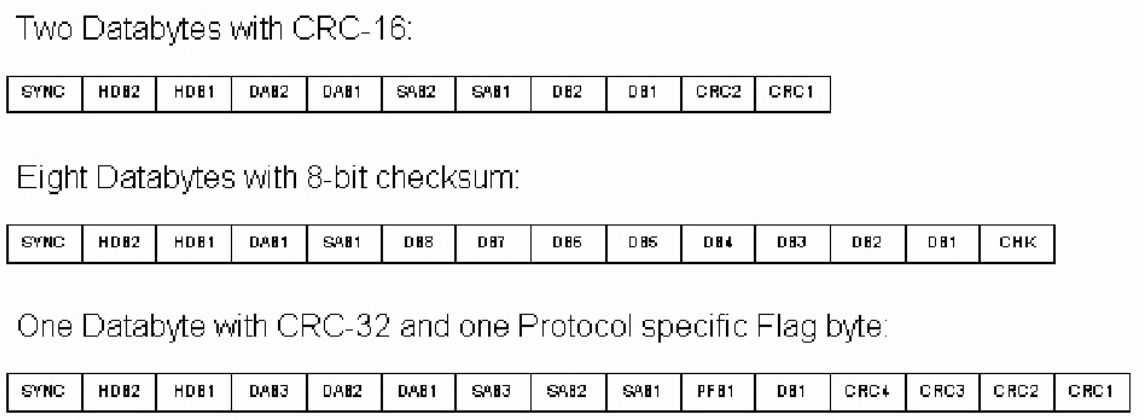
DAB Destination Address Bytes
Avhengig av hva som ble definert i de to DAB-bitene i HDB2, kan dette feltet enten utelates eller kan det bli fylt opp til tre byte lang adresse. Å utelate adresse er en god måte å minke protokollets header når nettverket består bare av to noder.

SAB Source Address Bytes
Samme som DAB

PFB Protocol specific Flag Bytes
Disse flagene er reservert for framtidig bruk. Disse kan for eksempel brukes til Fjern konfigurasjon, Pakke teller, Prioritering, osv.

DB Data Bytes
Dette feltet inneholder data som skal sendes. Protokollen støtter data som har størrelse innen 0,1,2,3,4,5,6,7,8,16,32,64,128,256,512 og den støtter også brukerens egen definert størrelse. Første byte kan være en kommando hvis CMD i HDB1 er satt.

CRC Checksum Bytes
Sjekksumen er beregnet avhengig av hva som er satt i den EDM-biter i HDB1. Disse feil detekterende metoder er støttet: 3 ganger retransmisjon, 8-bit checksum, 8-bit CRC, 16-bit CRC, 32-bit CRC FEC, støtter også egen definert feil detekterende metoder.



Figur 4.6: *S.N.A.P examples*

4.4.2.3 S.N.A.P fra vårt synspunkt

Mye av denne protokollen tilfredsstiller kravene som vi setter for MS nettverket vårt. Den er både skalerbar og har de fleste funksjoner som kunne trenges i et MS nettverk. Ulempen med denne protokollen er at den har for mye overhead, mye mer enn det vi trenger. Dermed kan vi også ut ifra denne protokollen ta noen viktige funksjoner og gjenbruke i vår protokoll.

5 Master-Slave Protocol – MSP

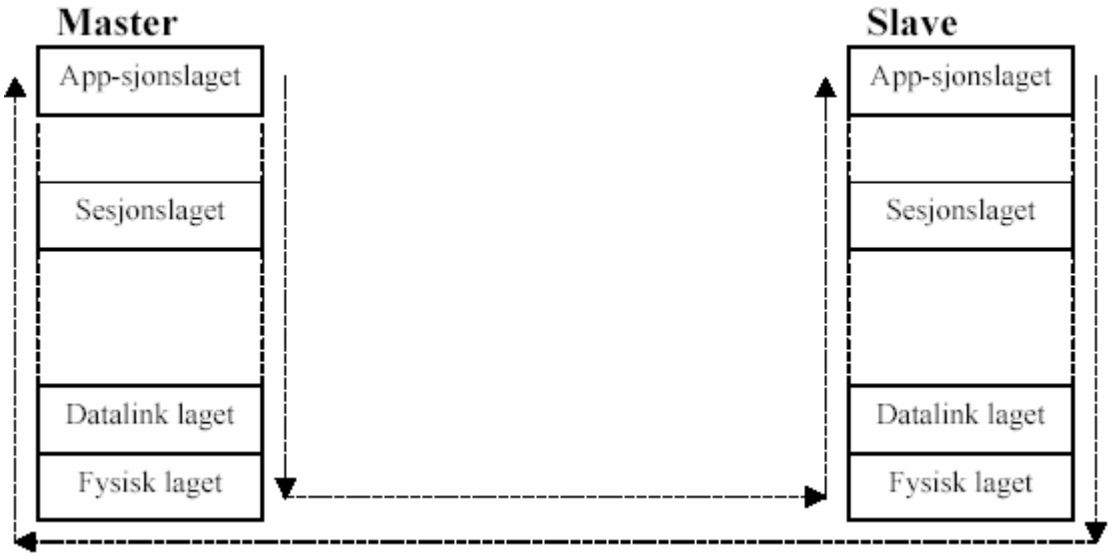
5.1 Introduksjon

Vi valgte å kalle vår protokoll for *Master Slave protocol* eller *MSP*. Som navnet tilsier er den beregnet for bruk i et MS nettverk. Protokollen vil definere et regelverk for kommunikasjon i et MS nettverk.

5.2 Lagoppdeling i MSP

I dette underkapittelet vil vi gå nærmere inn på strukturen funksjonaliteten i MSP er organisert på. Vi skal stadig benytte oss av OSI-referansemodellen.

Kommunikasjonsflyt vil skje som følge:
En melding fra en applikasjon vil gå via flere lag helt ned til fysiske laget hvor den blir transmittert til fysiske laget på mottakerside. Mottakeren i sin tur vil sende meldingen oppover i sin lagstruktur helt til applikasjonen. Hvert lag meldingen går via har sine egne funksjoner og er ansvarlig for en strengt definert funksjonalitet. For å si det ganske kort mottar fysiske laget "rå data" og etter at denne dataen ble sendt oppover og bearbeidet underveis, mottar applikasjonslaget en melding det kan lese og forstå. Ved sending skjer det samme men i motsatt rekefølge.



Figur 5.1: *OSI-lag og kommunikasjonsflyt i MSP*

I avsnittene som følger vil vi gå gjennom aktuelle for vår protokoll OSI-lag.

5.2.1 Det fysiske laget

Det fysiske laget transporterer ”rå” data (bits) over en kommunikasjonskanal. Fysiske laget steller med elektriske og mekaniske spesifikasjoner, blant annet:

- Metoder for oppkobling
- Fysisk topologi
- Digital signalering
- Analog signalering
- Bit synkronisering
- Båndbredde
- Kommunikasjonsmodi
- Multipleksing

Hovedpoenget er at når man har sendt en 1-bit, skal det komme frem en 1-bit og ikke en 0-bit. Det man spør seg om her er typisk hvor mange volt som skal brukes for å representere 1 og hvor mange som skal representere 0, hvor mange mikrosekunder en bit varer, om man kan sende samtidig i begge retninger, hvordan forbindelsen blir etablert og hvordan den tas ned, hvor mange pins nettverkskoplingen har og hva hver pin brukes til. Det er selve CC1010 som tar seg av det meste fra fysiske lags funksjonalitet i vårt system.

5.2.2 Datalinklaget

Datalinklaget tar den rå bitstrømmen fra det fysiske laget og gjør så den virker fri for uoppdagede transmisjonsfeil for nettverkslaget. Dette gjør den ved å be senderen bryte input-dataene opp i datapakker, sende pakkene sekvensielt, og prosessere ”acknowledgement-pakkene” som blir sendt tilbake fra mottakeren. For å gjenkjenne hvor en pakke starter og stopper, kan man for eksempel starte hver pakke med et spesielt bitmønster (preamble). Det er også datalinklaget som tar seg av feilkorrigeringen.

Felles delte mediet er ofte en stor utfordring for datalinklaget. Og et spesielt sublag av datalinklaget, medium-aksess-laget, tar seg av tilgangen til felles delte mediet. Dette sublaget er av spesiell interesse for oss.

5.2.2.1 Medium Access Control

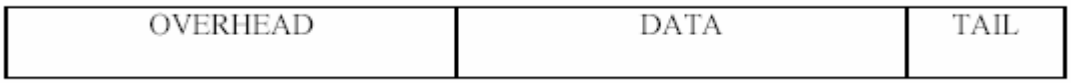
MSP vil bestå av kombinasjon av TDMA og CSMA. Ved TDMA får hver slave tildelt en tidsluke, når den kan kommunisere med Master. Når en slave får beskjed fra Master at tiden er inne for den å kommunisere med Master, kan dette gjøres med en gang. I tillegg til sine *dedikerte tidsluker* får slaver mulighet til selv å ta kontakt med Master, når det er høyst nødvendig. Det kan de gjøre i løpet av *interrupt-tidsluker*. Her får samtlige slaver *broadkast-beskjed* (fra pkt. 6.3) fra Master at de kan sende hvis det finnes noe viktig å sende. Men siden det er flere slaver som kan begynne å sende med en gang, skal de først utføre CSMA-sjekk (pkt 5.1.2).

5.2.2.2 Pakkeformat

Pakkeformat er et regelverk og form for hvordan meldinger i nettverket skal sendes på. Pakkeformat er noe av det helt grunnleggende en protokoll skal gjøre rede på. All kommunikasjon i nettverket er avhengig av måten et pakkeformat er bygd på. *Start-stopp asynkron kommunikasjon* som er tilfellet i vårt nettverk, kjennetegnes ved at hver dataenhet begynner med et startsymbol og avsluttes med et eller flere stoppsymboler.

Utenom selve data finnes det en mengde annen informasjon som vil bli sendt i hver eneste pakke. Denne tilleggsinformasjonen kalles for overhead og "erklærer" måten en pakke vil bli behandlet på. I tillegg vil en pakke også ha noe som heter "Tail" som på norsk betyr hale, og som er et bakerste felt i en pakke. Dette feltet er som regel feilsjekk for datafeltet, og det er det vi vil bruke det for.

Data pakken skal ha denne formen:



Figur 5.2: *Generell oppbygging av datapakken*

Overhead inneholder informasjon om sender, mottaker, forskjellige flagg, osv. Datafeltet er selve data som sendes til mottakeren, mens Tail er som nevnt feilsjekk for datafeltet.

Ut ifra de eksisterende protokoller som ble nevnt i forrige avsnitter bestemte vi oss for følgende pakkeformatet for MSP:

Preamble	Sync.	DAB	SAB	SQN
16 bit	8 bit	8 bit	8 bit	2 bit

FLAG	NDB	CRC-headersjekk.	DATA	CRC - datasjekk
4 bit	3 bit	8 bit	Maks. 56 bit (7 byte)	16 bit

Figur 5.3: *MSP pakkeformat*

Vi skal nå gå gjennom pakkeformatet og forklare hvert felt.

Preamble
Preamble bit kommer først i pakken. Brukes til å synkronisere radiomottaker med innkommende data. Mottak av preamble vil i første omgang indikere at noen prøver å kommunisere med mottaker. *Preamle* er et på forhånd bestemt bitmønster. Typisk lengde på dette er 16-32 bit. For å spare på nettverksressurser går vi for løsningen med 16 bits.

Sync – synchronization byte
Synk byte brukes til å indikere slutten på preamble og starten på datapakken. Den er også av fast bitmønster. *Synk* er også med på å filtrere bort falske datapakker. Hvis *Synk* ordet ikke er riktig mottatt, vil programmet gå tilbake å lette etter en gyldig preamble.

DAB – Destination Address Byte
DAB er mottakerens logiske adressen. Den 8 bit lang, med ”00000000” og ”00000001” reservert til henholdsvis broadkast- og masteradresse.

SAB – Source Address Byte
SAB er senderens logiske adressen. Akkurat som *DAB* er også *SAB* 8 bit lang.
Disse adressene er alias med serienumre til CC1010’ene. Vi skal senere forklare hvordan forholdet mellom serienumre og logiske adresser blir til.

SQN – Sequence number
For å hindre prosessering av en og samme pakke flere ganger skal hver pakke sjekkes for sekvensnummer (SQN) ved mottak. Pakker nummereres med *SQN* hos senderen og dette nummeret skal igjen sjekkes hos mottakeren. Hvis det viser seg at en pakke allerede er mottatt (hvis en pakke med dette *SQN* var allerede mottatt), skal denne pakken ignoreres. En *DAB*-feltet, *SQN* samt tidsluken pakken er mottatt i skal unikt identifisere en pakke. Første pakken sendt i hver tidsluke skal få *SQN* likt ”1”, påfølgende pakker skal økes med én. Siden antall meldinger som vil bli sendt i løpet av en tidsluke fra en sender ikke er mer enn 4 (Kap. 6), blir det brukt 2 bits for *SQN*-feltet.

Flag-feltet består av fire forskjellige flagg. Disse er *ACK*, *CMD*, *MTH* og *EC*.
Vi skal her forklare i detalj flaggenes betydning.

- *ACK – Acknowledge*
  Dette flagget viser om den mottatte meldingen trenger å bli kvittert for.
  Hvis ”PÅ” (1), skal kvittering sendes,
  hvis ”AV” (0), skal ikke kvittering sendes.
- *CMD – Command mode*
  Dette flagget har som formål å gjøre rede på hva slags melding er det som sendes: om det er Query- eller Reply-melding.
  ”0” er for Query.
  ”1” er for Reply.
  Dette flagget og kombinert med *DAB*-feltet og *ACK*-flagget sier hva slags melding det er. Dette skal vi se nærmere på senere i rapporten.
- *MTH – Multihopp*
  Dette flagget skal vise om meldingen har kommet fram til mottaker direkte eller via en repeater. Hvis slaven mottar en melding med multihopp flagget ”PÅ” (1), så vet slaven at meldingen har kommet fra en repeater. Hvis flagget er ”AV” (0), vet slaven at meldingen kommer rett fra Master. Dette flagget er spesielt viktig for repeatere (Kap. 6) siden de må sette *MTH-flagget* på for alle meldinger de videresender.
- *EC – Encrypted*
  Dette flagget skal informere mottakeren om data som sendes er kryptert eller ikke.
  ”1” er for kryptert.

"0" er for ikke kryptert.

NDB – Number of data bytes

Dette feltet viser hvor mange byte datafeltet består av. Vi regner med at informasjonen som sendes i pakker under initieringsfasen vil være av maksimal mulig størrelse som vil bli transmittert i vårt nettverk i én pakke. Datafeltet i disse initieringspakker vil inneholde 4 bytes: 24 bit lang serienummer og 8 bit lang logisk adresse (Kap. 6.2). Dette betyr at vi trenger et *NDB-felt* som er 3 bit lang. Et 2 bit lang felt hadde vært for lite fordi det trenges 4 bitmønstre for å indikere 1, 2, 3 og 4 bytes og i ett i tillegg for å indikere 0 byte.

Med tre bit langt *NDB* kan datafeltet teoretisk inneholde opp til 7 bytes av data og det vil være et av 8 forskjellige bitmønstre i *NDB*, avhengig av hvor mange bytes det er i datafeltet (tre siste bitmønstre kan bli brukt hvis protokollen blir videreutviklet til å transmittere pakker med opptil 7 bytes lang datainnhold):

000 - 0 byte i datafeltet
001 - 1 byte i datafeltet
010 - 2 byte i datafeltet
011 - 3 byte i datafeltet
100 - 4 byte i datafeltet
101 - 5 byte i datafeltet
110 - 6 byte i datafeltet
111 - 7 byte i datafeltet

Dette gjør det mye enklere for mottakere å lese data, mottakere bare setter på teller som er lik *NDB* og når teller er utgått betyr det at data er lest.

CRC headersjekk

Det utføres en 8-bits CRC-sjekk på overhead for å sjekke om transmisjon har gått feilfritt. Hvis det har forekommet feil så skal pakken kastes.

Data

Selve data som blir transmittert.

CRC datasjekk

Det utføres en 16-bits CRC-sjekk på datafeltet for å sjekke om datatransmisjon har gått feilfritt. Hvis det har forekommet feil så skal pakken kastes.

5.2.2.3 Retransmisjon og ACK

Det er opp til datalinklaget å løse problemer med ødelagte, forsvunne og duplikater rammer. Siden tidslukene er korte skal det kun retransmiteres én gang. Hvis ikke meldingen kommer frem etter to forsøk (det opprinnelige forsøket + retransmitterte) , vil den få mulighet til å bli sendt neste gang slaven får kommunisere med Master.

Alle meldingene skal kvitteres med ACK, det gjelder ikke selve ACK-meldingene. For å markere om det trenges kvittering på meldingen skal det brukes ACK-felt i pakken, hvis det er på – sendes det kvittering tilbake.

Det skal ventes ”en bestemt tid” før meldingen blir retransmittert for å unngå situasjoner når en melding er retransmittert i det kvitteringen på den opprinnelige meldingen kommer fram til senderen.

5.2.2.4 Adressegodkjenning

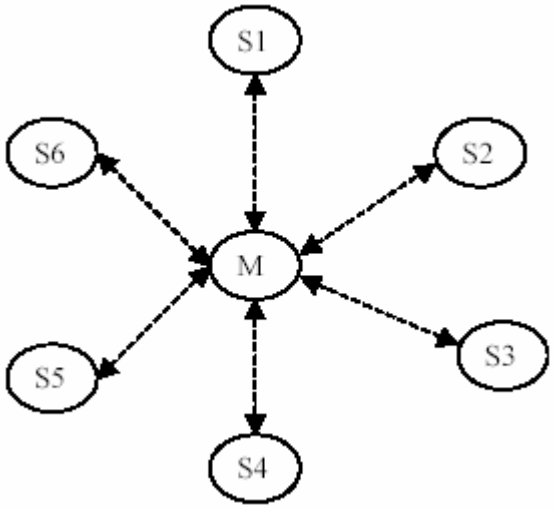
Logiske adressegodkjenning gjøres i Datalinklaget og ikke i Nettverkslaget, som det normalt gjøres ifølge OSI modellen, fordi serienummer (MAC-adresser) ikke vil bli sendt hver gang meldingene skal sendes.

Kort om virkemåten:
Slaver vil alltid høre på kanalen og sjekke om meldinger som mottas er adressert til dem. Dette gjøres ved å sammenligne ”Destination Address” med sin egen tildelt logiske adresse. Hvis destinasjonsadressen i meldingen og slaven sin adresse er like, dvs at meldingen er adressert til denne slaven, vil meldingen bli sendt videre til høyere lag hos slaven for prosessering, i motsatt fall blir meldingen droppet.

5.2.3 Nettverkslaget

5.2.3.1 Den valgte nettverkstopologien

Det ble valgt å utvikle vår protokoll for stjernenettverkstopologi. I et slikt nettverk er det en Master som skal være et knyttepunkt og styringsenhet for hele nettverket og mange slaver. Master bruker en aksessmetode for å distribuere retten til å kommunisere med forskjellige slaver. Denne sentraliserte aksessmetoden gjør at Master *poller* slaver en etter en for å se om de har noe å sende. *Polling* er en aksessmetode hvor en master tildeler slaver rett til å kommunisere, samt, i vårt system, begrenset tidsperiode (*tidsluke eller dedikerte tidsluke*) til å utføre denne kommunikasjonen på. Hver slave får tildelt en tidsluke til egen bruk ved registrering hos Master. Størrelsen på tidsluken og registreringsprosedyren vil bli beskrevet senere i rapporten.



Figur 5.4: *Den valgte nettverkstopologien*

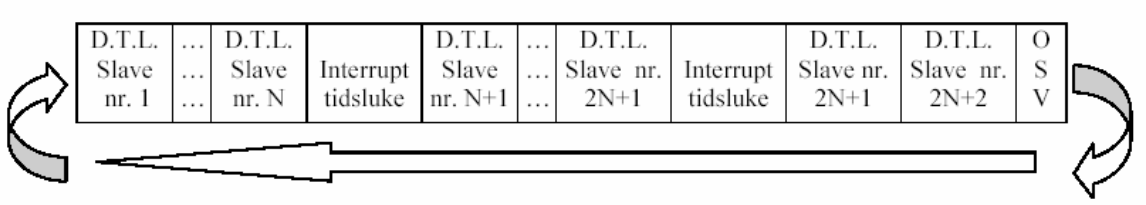
Siden denne typen av nettverkstopologi leverer kun ”best mulig service” vil det ikke alltid tilfredsstille krav til responstid for noen applikasjoner. Derfor for applikasjoner som krever en ”umiddelbar” respons vil det bli lagt inn en mulighet til å selv ta kontakt med Master uten at slaver behøver å vente på sin tur. Dette skal vi kalle for *interrupt*.

5.2.3.2 Multipleksing – dedikerte og interrupt-tidsluker

Slaver får sende når de får beskjed fra Master om det, dvs i løpet av sine tidsluker. En annen mulighet til å få sendt er interrupt. Slaver vil kun få sende interrupt i løpet av så kalte interrupt-tidsluker som slaver vil få beskjed om fra Master.

Interrupt-tidsluken skal være like stor som de dedikerte tidslukene. Det tas med CSMA for å takle interrupt-tidsluker. Dette er på grunn at det kan være flere slaver som vil prøve å sende interrupt samtidig. I så tilfelle vil slaver ødelegge for hverandre siden de vil snakke i munnen på hverandre.

Interrupt-tidsluker blir plassert mellom de dedikerte tidslukene med såpass liten mellomrom at slaver som har noe viktig å sende får en akseptabelt rask responstid.

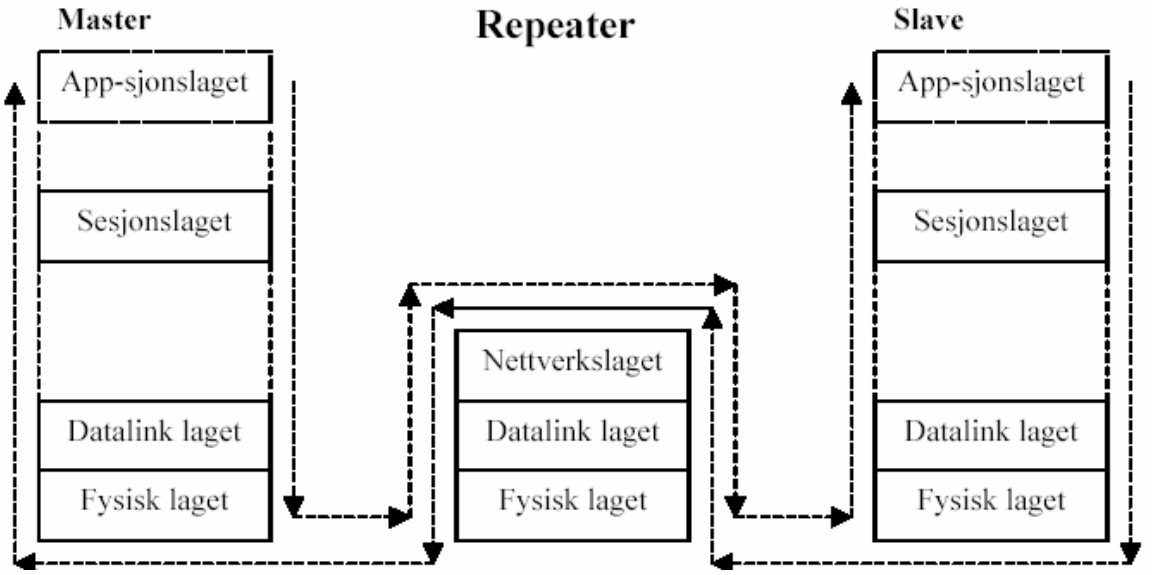


Figur 5.5: *Tidsmultipleksing (D.T.L. – dedikerte tidsluke)*

Antall tidsluker (N) innimellom *interrupt-tidslukene* vil bestemme hvor rask responstid for de viktigste meldingene skal være.

5.2.3.3 Multihopp

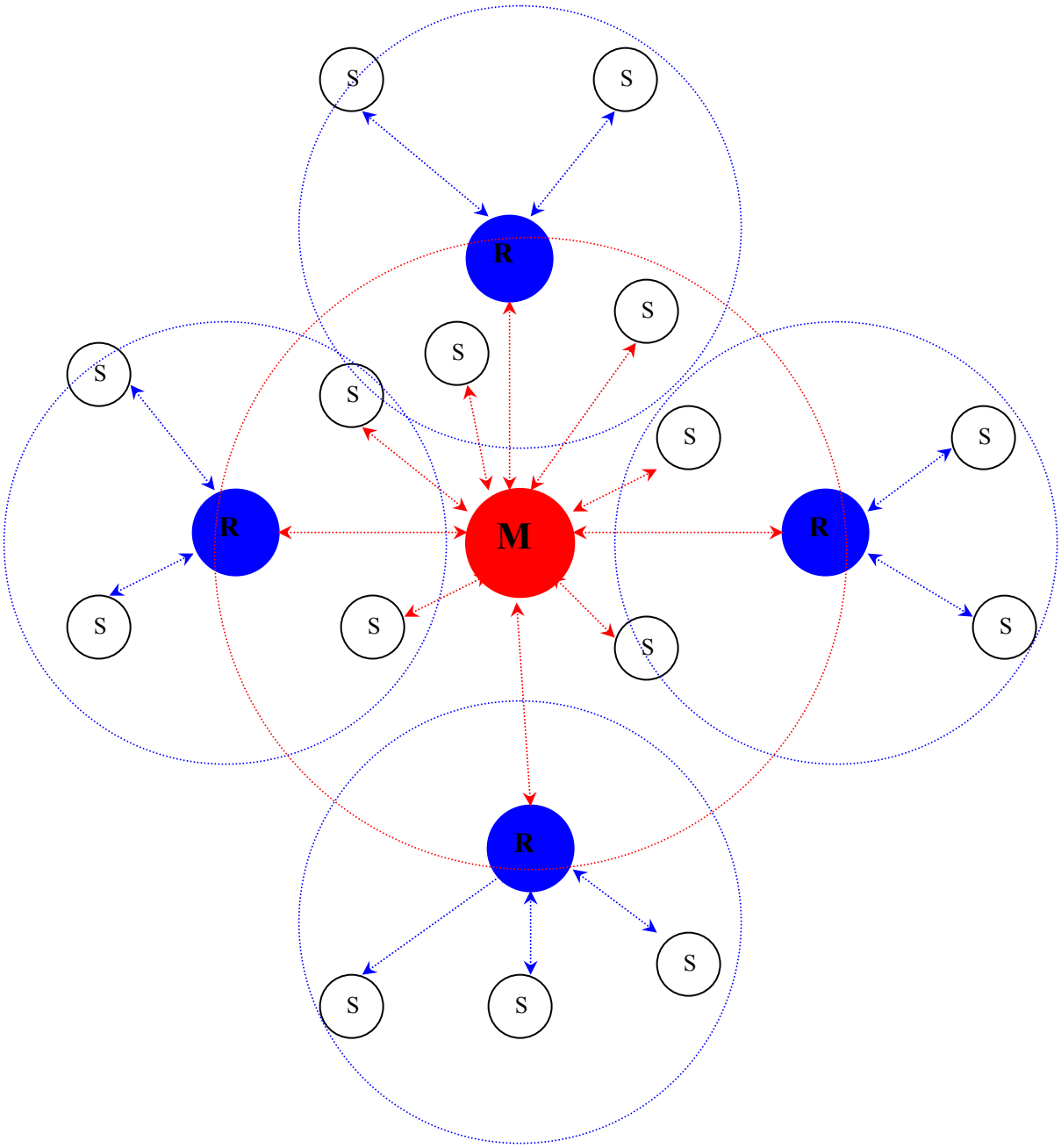
Nettverkslagets oppgave er å finne ut hvor man skal sende en pakke for å få den helt fra A til B. I vårt nettverk som er et eksempel på broadkast-nettverk er rutingen så simpel at det finnes ingen rutingstabeller verken hos Master eller slaver. Men utenom Master og slaver vil vi også ha repeatere som utvider nettverkets rekkevidde ved hjelp av multihopp-funksjonalitet.



Figur 5.6: *OSI-lag og kommunikasjonsflyt i MSP med Repeatere*

Repeatere vil trenge rutingstabeller siden det tenkes at de kun skal videresende meldinger som er adressert til slaver som ligger utenfor Masterens rekkevidde. Disse rutingstabeller vil bli kalt videre multihopptabeller. Multihopptabeller skal kun inneholde et felt med logiske adresser til slaver som er registrert hos repeater. Ved mottak av en melding skal repeater gå til sin multihopptabell og sjekke om slavens adresse står registrert i tabellen. Hvis den gjør det, så videresendes meldingen. Dette vil vi gå mer i detalj på i neste kapitel.

Slik kan nettverkets rekkevidde med repeatere se ut:



Figur 5.7: *Skjematisk skisse av nettverkets rekkevidde med repeatere*

5.5.4 Transportlaget

På transportlaget velges det nettverksforbindelse og pakker settes sammen i riktig rekkefølge. Transportlaget vil på forespørsel fra sesjonslaget gi beskjed om hva slags forbindelse som skal brukes. Metode for flytkontroll blir bestemt i dette laget.

Ingen av disse finneser trenges i vår protokoll, ganske enkelt på grunn av dens enkelhet. Derfor vil transportlaget være fraværende i vår protokoll.

5.5.5 Sesjonslaget

Sesjonslaget i vår protokoll vil ta seg av sikkerhet, med det mener vi at den skal passe på at en kommunikasjon fra en sender kommer fram til riktig mottakker. Dette kalles for autentisering. Autentiseringen er nærmere beskrevet i kap. 3.

5.5.6 Presentasjonslaget

Her formateres data slik at de underliggende lagene skal kunne sende forståelig data over til mottaker. Hvis vi tenker oss en nordmann og en russer som ønsker å kommunisere så kan applikasjonslagene sammenlignes med to tolker som kan henholdsvis norsk og russisk + engelsk. All kommunikasjon blir satt opp slik: Norsk til engelsk, engelsk til russisk, russisk til engelsk og engelsk til norsk. Skal sammenligningen være realistisk må vi forutsette at tolkene oversetter uten å kjenne ett ord av hva som blir sagt.

Dette laget tar seg av:

- Bestemmer hvilket format som skal brukes for å utveksle data
- Oversetter til et mellomliggende format
- Ansvarlig for konvertering av protokollene
- Kryptering
- Endre bokstavkode

Hvis det blir valgt å sette opp kryptering, blir dette gjort i presentasjonslaget. Da ved sending, vil presentasjonslaget kryptere meldingene og ved mottak – dekryptere.

5.5.7 Applikasjonslaget

Veldig enkelt og generelt sagt representerer applikasjonslaget det programmet som brukeren har foran seg. Mer inngående kan det beskrives med følgende punkter:

- Representerer tjenester som direkte støtter brukerapplikasjonene
- Håndterer generell nettverksadgang
- Flytkontroll
- Feiloppdaging

Vi skal ikke gå mer i detalj om applikasjonslaget siden dette laget ligger utenfor oppgavens omfang og dermed fraværende i vår protokolldefinisjon.

6 Hvordan fungerer MSP

Vi skal her beskrive hvordan MSP fungerer og skal forklare i detalj noen viktige meldingssekvenser og flytskjemaer.

6.1 Registrering av slaver hos Master

Alle slaver får tildelt et serienummer. Disse numrene skal tastes inn manuelt før registrering hos Master. Dette er som tidligere nevnt ment for blant annet autentisering av slavene.

CC1010 kort per i dag har ikke serienumre og vi vil gjøre et forsøk på å foreslå en hensiktsmessig størrelse på disse. To viktige faktorer som vil ha innvirke på lengden er CC1010 minnekapasitet og hvor unike serienumre blir hvis vi bruker en bestemt lengde.

MAC adresser slik vi kjenner dem i datakommunikasjon har lengde på 48 bit. Med denne lengden vil man kunne garantere unikhet for hvert enkelte kort i mange hundreår fremover (2^48 = 281 474 976 710 656 ulike serienumre). Grunnen som teller mot denne løsningen er at man generelt ikke trenger at et serienummer til et kretskort vil være så unik. En annen viktig grunn er at hvis man vil regne med at Master vil ha kjennskap til serienumre til alle slaver på nettverket, vil det ta en god del av tilgjengelig minnekapasitet. Vi må huske på at CC1010 bare har 2kByte RAM, og alt kan ikke bli brukt på serienumre heller. Med 255 mulige enheter i nettverket:

(48bit x 255) = 1530 bytes

Vårt forslag er å bruke 24 bit lange serienumre. I så tilfelle vil man få nokså unike serienumre (2^24 = 16 777 216 ulike serienumre) og det vil ikke ta så mye minnekapasitet som i tilfelle med 48 bit lange adresser:

(24bit x 255) = 765 bytes

Dette etterlatter 1235 bytes av RAM til andre formål.

Bruk av serienumre som adresseidentifikasjon for slaver ville unødvendig ta altfor mye båndbredde. Derfor vil Master registrere serienumre til slavene i en tabell, og tildele dem logiske adresser. Logiske adresser begynner fra 00000010, siden 00000000 og 00000001 er reservert til henholdsvis Broadkast og Master, og fortsetter fram til 11111111 (8 bit).

Tabellen etter registreringen vil se slik ut:

Tabell 6.2: Et eksempel for mapping mellom serienumre og logiske adresser

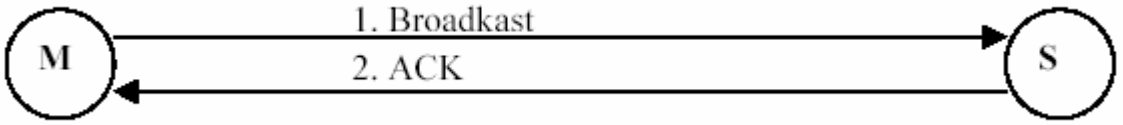
serienummer	logisk adresse
100100111010101001010101	00000010
100100111010101001000111	00000011
⋮	
serienummer N	N

Etter at manuelt registrering er utført, vil slaver i nettverket bli slått på og da vil Master informere slavene om deres tildelte logiske adresser (Kap. 6.2).

6.2 Mapping mellom serienumre og logiske adresser

Master sender ut broakast-meldinger med serienummer og logisk adresse som data. Disse meldingene krever ACK slik at Master kan registrere hvilke slaver som "virkelig" er med i nettverket. Det vil bli sendt like mange broadkast-meldinger som det var manuelt registrert slaver.

Dette kan illustreres med følgende figur:



Figur 6.1: *Meldingsutveksling ved registreringen*

1 Serienummer og logisk adresse sendes fra Master som Broadkast.
2 Slaven tar imot broadkast-meldingen, sjekker om innholdet i datafeltet er dens eget serienummer. Hvis det stemmer noterer slaven for seg denne logiske adressen og sender ACK.

Når Master er ferdig med registrering av første feltet i Tabell 6.1, går den videre til neste felt. Slik fortsetter Master til den har gått gjennom alle slaver i tabellen. Hvis det hender at noen slaver som var med i tabellen svarte ikke på broadkast-meldingen og retransmitterte broadkast-meldingen, regnes disse som fraværende og i ikke tas med i betraktning videre.

Problemer som kan forekomme og mulige løsninger:

- Broadkast meldingen nås ikke alle slaver, feil ved broadkast meldingen.

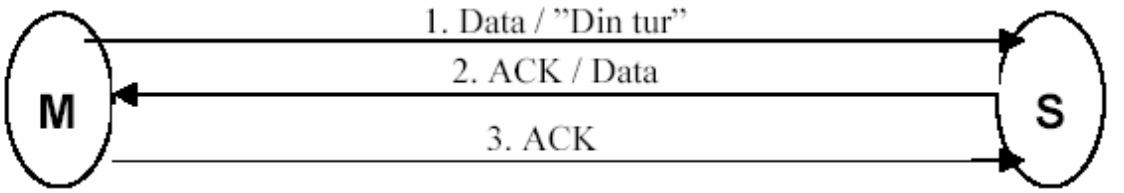
  Master har en tidsmåler som venter på ACK-meldingen. Hvis tida har gått ut før ACK er mottatt, så skal meldingen retransmitteres. Det må bestemmes hvor mange ganger Master skal retransmittere før den går til neste serienummer i tabellen, altså neste slave.

- ACK meldingen nås ikke fram til Master.

  På grunnlag av registreringen, får Master vite hvor mange slaver det er i nettverket og dermed kan konstruere "tidsmultipleksing system". Som nevnt tidligere går dette ut på å tildele hver slave sin egen tidsluke hvor den kan kommunisere med Master. Interrupt tidslukene kommer i tillegg.

6.3 Vanlig tidsluke

Master går gjennom tidslukene som den har fordelt til slavene. Master starter med den "laveste" logiske adressen og fortsetter videre til den har kommet til den "høyeste" logiske adressen som den registrerte. Når en tidsluke starter, starter også en tidsmåler for denne tidsluken. Når tidsluken er slutt eller at "samtalen" mellom Master og slave er ferdig før tidsluken er utgått, går Master videre til neste slave.
Meldingssekvensen for dette blir:



Figur 6.2: *Meldingsutveksling ved vanlig tidsluke*

1 Master sender data hvis det er noe å sende, ellers så sier Master ifra at slaven er inne i sin tidsluke.
2 Slaven sender ACK hvis melding 1. var data og sender Data hvis den var "Din tur".
3 Master sender ACK hvis melding 2 var data fra slaven.

Problemer som kan forekomme og mulige løsninger:

- Meldingene kommer ikke fram

  Når en melding sendes, startes det en *timeout-timer* for denne meldingen. Hvis *timeout-timer* går ut før senderside mottar respons fra mottakerside, blir meldingen retransmittert. I vanlige tidsluker skal meldinger retransmitteres kun en gang.

- Hva hvis slaven ikke har noe å sende, når den har mottatt ”Din tur” meldingen?

  Hvis slaven også ikke har noe å sende, så må Master bare vente til tidsmåleren til tidsluken går ut før den går til neste slave. Siden det betyr dårlig ressursutnyttelse, kunne alternativet vært å vente kun en del av denne tidsluken for å forsikre seg at slaven ikke har data å sende og så gå videre til neste slave.

- Hva hvis tidsmåleren til tidsluka har gått ut før samtalen er ferdig?

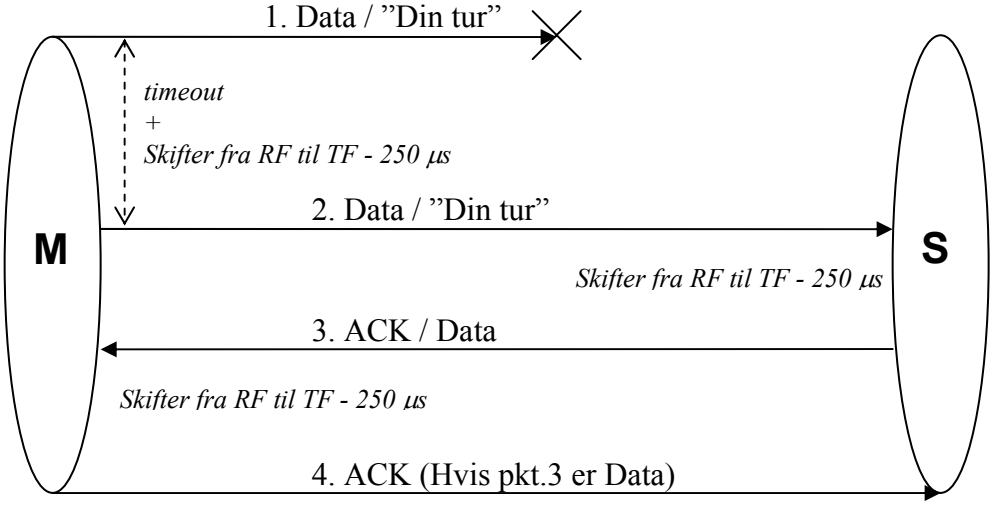
  Dette kan løses ved at også slaver har en tidsmåler for tidsluken. Denne tidsmåleren vil vise om slaven kan fortsette å sende eller ikke. Tidsmåleren hos slaver må være mindre enn den standard tid for tidsluken, som er lik tidsmåleren hos Master.

6.4 Tidsaspekter i vår protokolldesign

Det skal regnes ut tiden det tar for en enkel kommunikasjonssesjon mellom Master og slave. Dette underkapitelet tar ikke for seg av interrupt-tidsluker men kun vanlige tidsluker. Interrupt-tidsluker og hvordan de henger sammen med vanlige tidsluker blir beskrevet senere i underkapitel 6.5.

6.4.1 Aktuell situasjon

Kommunikasjonen mellom Master og slaver skjer på rundtur. Master tar kontakt med den aktuelle slaven for å sende data til den eller å be slaven om å sende data (Figur 6.3).



Figur 6.3: *En enkel kommunikasjonssekvens*

Det som kan skje er at Master blir nødt til å repetere den første meldingen på grunn av at meldingen ikke kom fram til slaven (som på figur 6.3) eller at ACK meldingen fra slaven ble mistet underveis. I begge tilfellene blir resultatet det samme: *timeout-timeren* som ble slått på hos Master da meldingen ble sendt utgår, og Master vil prøve å sende meldingen

en gang til. Det forutsettes at meldinger blir retransmittert kun en gang, og hvis det oppstår en transmisjonsfeil andre gang på rad, vil Master gå videre til neste slave.

6.4.2 Tidsberegninger

Tiden det tar å kjøre hele sekvensen består av

- tiden det tar å svitsje transmitter mellom TX og RX modi (= 250 μs)
- *timeout-tid* – tiden det ventes på kvittering for sendte meldingen før den retransmitteres
- tiden det tar for meldingen å dekke avstand mellom M og S (svært liten, kan egentlig ses bort fra) – *travel_time*
- tiden det tar å sende en melding fra en transeiver – *time_to_send_a_packet*

Da i følge figur 6.3 kan tiden for en enkel kommunikasjonssekvens regnes ut ved hjelp av følgende formel:

T = 250 μs * 3 + timeout + travel_time*3 + time_to_send_a_packet * 4

- **travel_time**
  maks avstand mellom M og S = 100 m
  travel_time = 100 m / *(3 * 10^8 m/s)* = 0,33 * 10^(-6) s = 0,33 μs
  travel_time * 3 ≈ 1 μs

  *(3 * 10^8 m/s)* er lysets hastighet. Travel_time ganges med tre fordi det kun er tre meldinger av de fire sendte som klarte å komme seg fram til mottakeren. Tiden som ble brukt på den som ble mistet underveis er blitt dekket av timeout-tid.

- **time_to_send_a_packet**
  pakke_lengde = Preamble (16 bit) + Sync (8 bit) + DAB (8 bit) + SAB (8 bit) + SQN (2 bit) + FLAG (4 bit) + NDB (3 bit) + CRC-headersjekk (8 bit) + DATA + CRC – datasjekk (16 bit)

  Følgende situasjon er oppstår ved tildeling av logiske adresser på grunnlag av serienumrene. Dette er maksimal størrelse på datafeltet.

  Data_felt = Serie_nmr + Logisk_adr = 48 bit + 8 bit = 56 bit

  pakke_lengde = 16 + 8 + 8 + 8 + 2 + 4 + 3 + 8 + 56 + 16 = 129 bit

  Nå regner vi ut tiden det tar å sende ut en bit

  τ = 1 bit / datarate

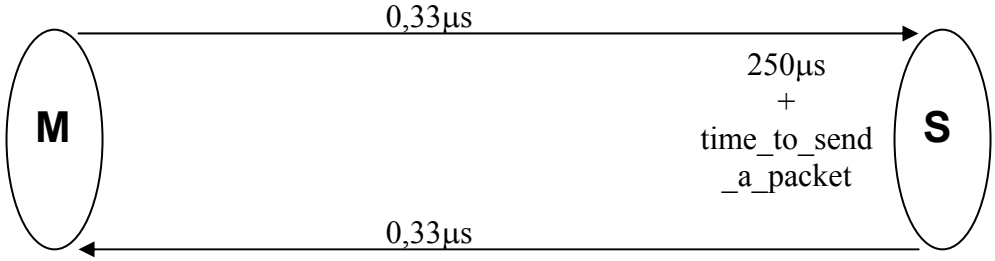
Det skal ikke gjøres utregning med maksimale og minimalle datarateverdier. Dette er grunnet at den maksimale verdien ikke vil være oppnåelig mesteparten av tiden og med den minimalle er altfor lav for å kunne brukes i såpass stort nettverk (viser til utregningen under).

$Datarate_{min}$ = 0,6 kbit/s
τ_1 = 1 bit / 0,6 kbit/s ≈ 1,7 ms
time_to_send_a_packet = τ_v * pakke_lengde = 1,7 ms * 129 bit = 219,3 ms
Det vil si at det vil ta ca. fjerdedel av sekund bare for å sende en pakke ut. Dette er selvfølgelig et uakseptabelt høy tall og det vil bli enda høyere hvis det blir tatt hensyn til andre tidsaspekter.

Da trenger vi å vi velge den laveste akseptabel dataraten i vårt nettverk.
Siden det regnes med at vanlig dataraten for CC1010 vil være 19,2 kbit/s, vil vi velge den laveste akseptabel verdien av dataraten til å være 10 kbit/s ($Datarate_{praktisk\_min}$). Det vil kun bli brukt denne dataraten videre i utregningen.

$Datarate_{praktisk\_min}$ = 10 kbit/s
$\tau\_{praktisk\_min}$ = 1 bit / 10 kbit/s ≈ 0,1 ms
time_to_send_a_packet = $\tau\_{praktisk\_min}$ * pakke_lengde = 0,1 ms * 129 bit = = 12,9 ms

- **timeout –** er tiden sender venter på kvittering fra mottaker før den vil prøve å sende på nytt.
  Størrelsen på timeout bør minst være lik den tiden i figur 6.4 og gjærne litt mer.



Figur 6.4 : *Utregning av timeout*

timeout = 0,33µs *2 + 250µs + time_to_send_a_packet

Som vi ser ut fra formellen over er **time_to_send_a_packet** helt avgjørende for å bestemme størrelsen på timeouten. **time_to_send_a_packet** er i sin tur avhengig av dataraten. Siden timeoutens oppgave er å skille det som menes til å være en akseptabel responstid fra det som menes til å være uakseptabel, bør vi bruke i timeouts beregning minimalt akseptabelt dataraten som er 10 kbit/s.

timeout = 0,33μs *2 + 250μs + 12,9 ms ≈ 13,151 ms ≈ 13,2 ms

Nå som vi har oppklart alle ledd i formellen kan vi endelig regne ut T.

T = 250 μs * 3 + timeout + travel_time*3 + time_to_send_a_packet * 4

$T_{praktisk\_min}$ = 250 μs * 3 + 13,2 ms + 1 μs + 12,9 ms * 4 ≈ 65,6 ms

En viktig kommentar:
Siden vi har bestemt oss for at dataraten lik 10 kbit/s vil være den minste tillate i nettverket, vil **$T_{praktisk\_min}$** være en viktig kriterium for å bestemme størrelsen på tidsluken. Størrelsen på tidsluken er den maksimale tiden som er tildelt for en kommunikasjon mellom Master og slave.

$T_{praktisk\_min}$ = Tidsluke = 65,6 ms

All kommunikasjon innenfor denne tidsintervall vil nå fram til mottaker som kan være enten en slave eller Master. Hvis Master-Slave paret klarer ikke å bli ferdige med sin kommunikasjon i løpet av denne tiden, må slaven ganske enkelt vente til neste gang Master kontakter den og prøve på nytt.

Nå har vi regnet ut hvor lang tid det vil ta for hver enkel kommunikasjon mellom en slave og Master. Men det som er mer interessant er hvor lang tid vil Master bruke for å *polle* alle slaver og komme tilbake til den første den hadde begynt med.

For maksimalt antall slaver (255) er omløpstiden (tiden fra en slave blir kontaktet til neste gang Master tar kontakt med den samme slaven)

$Omløpstid_{praktisk\_max}$ = 65,6 ms * 255 = 16728 ms ≈ 16,8 s

Resultatet over viser "the worth case", med lavest akseptabel datarate og det er dette resultatet som har praktisk betydning for om systemet vil fungere bra eller dårlig. $Omløpstiden_{praktisk\_max}$ = 16,8 s er ikke så lenge for mange applikasjoner som ikke trenger rask responstid og kan leve med en maksimal forsinkelse på 16,8 s. Men det vil alltid finnes applikasjoner hvor disse 16,8 s er altfor lenge å vente. Det kan være på grunn av at noen meldinger har verdi i veldig avgrenset tidsperiode eller på grunn av alvorlige konsekvenser en forsinkelse kan forårsake. Det er her interrupt kommer inn i bildet.

6.5 Interrupt

Applikasjoner som på grunn av sine egenskaper trenger en garanti på at data vil bli sendt uten forsinkelser, vil kunne bruke interrupt. Dermed slipper slaver, hvis applikasjon tilsier det, å vente på sine dedikerte tidsluker for å få sende.

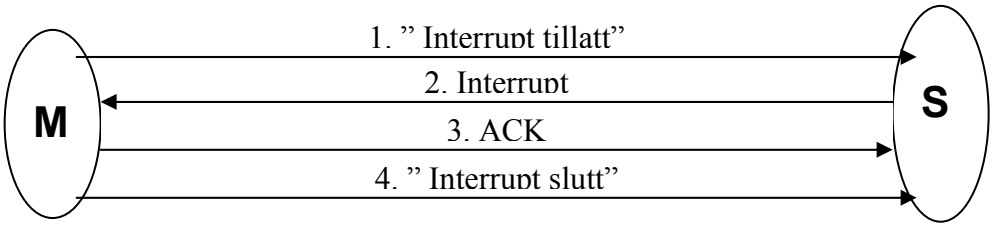
I vårt system vil slaver kun kunne sende interrupt i interrupt-tidsluker. I interrupt-tidslukene vil slaver som har noe viktig å sende, få sende uten å vente på sin tur. Det

brukes broadkast meldinger som kan høres av alle slaver, for å si ifra om interrupt-tidslukestart og -slutt.

På denne måten får systemet kontroll over interrupt-rutiner. Dette kan da neppe kalles for en" klassisk interrupt", som innebærer at interrupt kan bli sendt når som helst. Men vår løsning vil ikke ødelegge selve interrupt-prinsippet – interrupt vil uansett komme frem "umiddelbart" siden interrupt-tidsluker vil forekomme nokså ofte. Hvor ofte vil de egentlig inntreffe blir bestemt senere i avsnittet 6.5.2.

6.5.1 Interrupt-meldinger

Master sender broadkast meldinger i kombinasjon med markeringen av CMD bit, som vi skal se nærmere på senere i kapitlet, for å tillate interrupt. Master behandler kun én interrupt av gangen og tar ikke imot nye interrupt-meldinger mens den holder på med den gamle. Etter å ha sendt kvittering for interrupt-meldingen, avsluter Master interrupt-tidsluken med "Interrupt slutt" meldingen. Dvs at neste interrupt kan kun komme når Master tillater det neste gang, det vil si sender "Interrupt tillatt".
Sekvensen for dette er som følge:



Figur 6.5: *Meldingsutveksling ved Interrupt*

1 Master sier ifra at interrupt-tidsluken har begynt
2 Den slaven som har noe å sende – sender
3 Master sender ACK for mottatt interrupt
4 Master sier ifra at interrupt-tidsluken er slutt

Problemer som kan forekomme og mulige løsninger:

- Hva hvis "interrupt tillatt" og "interrupt slutt" ikke nås til alle slaver?
  Det at interrupt ikke vil komme frem til alle slaver har egentlig lite betydning, siden det kun er en slave som får sende interrupt per interrupt-tidsluke.

- Hva hvis to eller flere Interrupt-meldinger kommer samtidig?
  Dette løses som nevnt tidligere ved hjelp av nonpersistent CSMA.

- Hva hvis interrupt meldingen kommer samtidig som "interrupt slutt" meldingen?
  Her kan det løses med CSMA. Sannsynligheten for kollisjon i dette tilfelle er veldig liten hvis man bruker nonpersistent CSMA (Kap 3.3.1.4).

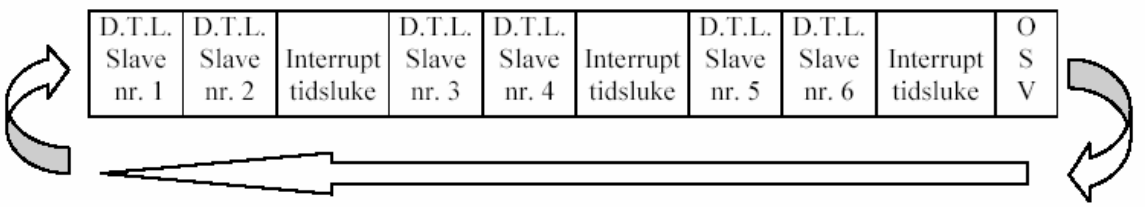
6.5.2 Tidsaspekter i interrupt-sekvensen

Dette avsnittet som navnet lyder vil ta seg av tidsaspekter og interrupt.
Det ble regnet ut at tidsluken vil være lik 65,6 ms. Ut ifra dette tallet vil vi bestemme oss for hvor ofte interrupt-tidsluker vil inntreffe. Tanken er at slaver må få rimelig liten ventetid når de vil sende meldinger utenom sine dedikerte tidsluker.

Hvis slaver får mulighet til å bruke interrupt en gang med mellomrom på to vanlige tidsluker, så ser det ut til å være et hensiktsmessig valg. Dette kan vi konkludere med større sikkerhet hvis vi regner ut denne tiden.

2* Tidsluke = 2 * 65,6 ms = 131,2 ms

Denne ventetiden ser ut til å være rimelig akseptabel. Slaven får i idealitet mulighet til å sende en gang i løpet av 131,2 ms. I praksis vil ting være mer komplekse, siden det kan være flere slaver som vil prøve å sende interrupt til enhver tid og ventetiden vil variere. Men hvis man kjører en streng interruptpolitikk og tildeler mulighet for interrupt kun til applikasjoner som absolutt trenger det, begrenser man med det samme antall slaver som vil prøve å sende interrupt til enhver tid.



Figur 6.6: *Meldingsutveksling ved interrupt*

6.6 Tidsberegninger – avslutting og konklusjon

Med verdiene vi har regnet ut kan vi bestemme hvor fort Master vil gå gjennom alle slaver og da inkludert med interrupt-tidsluker. Dette vil vi kalle for Periode.

Størrelsen på vanlige tidsluker og interrupt-tidsluker skal være like siden det vil i begge tilfeller sendes akkurat samme antall meldinger.

Utregning og resultatet er som følge:

Periode = Dedikerte tidsluker delen + Interrupt delen = 65,6 ms * 255 + 65,6 ms * 255 / 2 = 16728 ms + 8396,8 ms ≈ 16,8 s + + 8,4 s = 25,2 s

Det vil altså ta 25,2 s for en hel runde med interrupt tidsluker medregnet.

6.7 Multihopp

6.7.1 Hva er multihopp og hvorfor det brukes?

Med multihopp menes det at Master i MSP-nett har mulighet til å nå slaver selv om de kanskje befinner seg utenfor Masterens rekkevidde. I så tilfelle brukes det enheter som kan re-rute pakker videre til destinasjonsslaver og tilbake til Master. Dvs trafikken fra Master går ikke direkte til slike slaver, ganske enkelt på grunn av at signaler fra Master forplanter seg ikke så langt, men via enheter som skal bare videresende pakker som er adressert til slaver enheten "kjenner til". Den aller viktigste forutsetningen er at den ekstra funksjonaliteten ikke vil gjøre nettverket mer komplisert. Vi vil helst unngå omfatende videreutvikling av metoder som allerede er definert for nettverk uten multihopp. Fra slaven sin side må det være nesten usynlig om meldingen er kommet direkte fra Master eller via enheten som videresender meldinger.

6.7.2 Praktisk multihopp-løsning i MSP-nett

For å takle multihopp vil det bli brukt spesielle multihopp enheter som vi skal kalle repeatere. Repeatere skal være CC1010 kretskort som brukes for Master og slaver med repeater-program på. Repeatere skal være bindepunkter mellom Master og slaver som ligger utenfor CC1010 ordinær rekkevidde og med dette øke operasjonsrekkevidde med opptil 50% (Figur 5.7). Repeaterne's oppgave er å videresende pakker fra/til Master til/fra slaver. Av praktiske grunner og for enkelhetens skyld blir repeatere manuelt plassert slik at de vil dekke 360° rund Master. Dvs man kommer til å trenge 4 repeatere for å få til løsningen.

Det skal skilles mellom Gr.1 slaver og Gr.2 slaver. De første trenger ikke repeatere for å bli kontaktet, de andre kan kun bli kontaktet via repeatere. Utenom fysisk plassering finnes det ingen forskjell mellom disse to gruppene. Repeatere vil registrere Gr.2 slaver ved å registrere innslag i sine *multihopptabeller*.

Noen viktige poeng:

- Repeatere er CC1010 kretskort med repeater-program
- Repeatere har ingen adresser. Repeaterens funksjonalitet er å ta imot og videresende om nødvendig.
- Repeatere får ikke tildelt tidsluker. Repeatere skal bruke CSMA før de sender, for ikke å snakke i munnen på andre slaver.
- Alle meldingene som videresendes av repeatere skal ha "Multihopp flag" satt på. Det samme gjelder for ACK for disse meldingene.
- Repeatere skal videresende alle broadkast-meldinger den mottar. Broadkast brukes kun ved oppstart av nettverket, når Master ikke har mulighet til å vite om slaver hører til Gr. 1 eller Gr. 2, det er derfor alle meldinger blir videresendt.
- Repeatere skal kun videresende de meldingene som kom fra eller til registrerte slaver i multihopptabell (Unntak: Broadkast-meldinger). Dette er for ikke å

videresende meldinger som ikke trenger multihopp (fra Gr.1 slaver til Gr.1 slaver), siden repeatere også vil motta dem.

6.7.3 Virkemåten

Vi vil i kort form forklare virkemåten til multihopp.

Registreringssekvens

1. Master sender ut broadkast-meldinger ved oppstart.
2. Repeater tar imot broadkast-meldinger på samme måte som vannlige slaver gjør, setter ”Multihopp flag” på og videresender med broadkast adresse.
3. Slaven tar imot, sjekker om den har fått meldingen fra før (SQN-felt i pakkeformatet brukes her), deretter sjekker om ”Multihopp flag” er satt på, dvs at meldingen kom fra repeater.
    a. Flagget er på:
        i. Slaven skal også sette ”Multihopp flag” på når den vil sende ACK-meldingen.
        ii. Dette gjøres for å oppdatere multihopptabeller hos repeatere (pkt. 4).
    b. Flagget er av:
        i. Dvs meldingen kom direkte fra Master og ikke via repeater. I så tilfelle anvendes det regler for normal kommunikasjon mellom Master og slaver (Kap 6.3).
4. Repeater tar imot ACK-meldingen, sjekker om ”Multihopp flag” er satt på.
    a. Flagget er på:
        i. Slaven registreres (eller re-registreres) i rutingstabellen og ACK-meldingen videresendes til Master.
    b. Flagget er av:
        i. ACK-meldingen ignoreres.
5. Master tar imot ACK og registrerer slaven på vanlig måte som om det hadde vært Gr.1 slave (Kap. 6.2).

Etter at registreringen er blitt kjørt ferdig vil Repeatere operere på følgende måte.
Repeatere skal lese inn alle meldinger som når fram til dem. Hos alle mottatte meldinger skal det sjekkes destinasjonsadresse. Denne adressen skal sammenlignes med adresser som repeater har registrert i sin multihopptabell. Hvis det finnes tilsvarende innslag i tabellen, da vil repeateren sette på ”Multihopp flag” og videresende meldingen til destinasjonsslaven. Slaven mottar meldingen og ser i destinasjonsadressefelt at meldingen er adressert til den og leser ned inn videre. I responsmeldingen (Aknowledgement) vil slaven sette ”Multihopp flag” på eller av, avhengig av om ”Multihopp flag” ble satt på eller av i innkommet meldingen.

6.8 Kombinasjonen av flaggene

I dette avsnittet skal vi beskrive hva kombinasjoner av ACK-, CMD-, MTH- flagg og DAB feltet forteller oss om meldingen. Disse feltene viser hva slags service en melding vil få hos mottaker. Feltene kan ses på figur 6.7 som er repetisjon av figur 5.3 tidligere i rapporten. Forklaring til de aktuelle feltene er også tatt med fra Kap. 5.

Preamble	Sync.	DAB	SAB	SQN
16 bit	8 bit	8 bit	8 bit	2 bit

FLAG	NDB	CRC-headersjekk.	DATA	CRC - datasjekk
4 bit	3 bit	8 bit	Maks. 56 bit (7 byte)	16 bit

Figur 6.7: *Pakkerformat for MSP*

ACK – Acknowledge
Dette flagget viser om den mottatte meldingen trenger å bli kvittert for.
Hvis ”PÅ” (1), skal kvittering sendes,
hvis ”AV” (0), skal ikke kvittering sendes.
CMD – Command mode
Dette flagget har som formål å gjøre rede på hva slags melding er det som sendes: om det er Query- eller Reply-melding.
”0” er for Query.
”1” er for Reply.
Dette flagget og kombinert med DAB-feltet og ACK-flagget sier hva slags melding det er. Dette skal vi se nærmere på senere i rapporten.
MTH – Multihopp
Dette flagget skal vise om meldingen har kommet fram til mottaker direkte eller via en repeater. Hvis slaven mottar en melding med multihopp flagget ”PÅ” (1), så vet slaven at meldingen har kommet fra en repeater. Hvis flagget er ”AV” (0), vet slaven at meldingen kommer rett fra Master. Dette flagget er spesielt viktig for repeatere (Kap. 6) siden de må sette MTH-flagget på for alle meldinger de videresender.
DAB – Destination Address Byte
DAB er mottakerens logiske adressen. Den er 8 bit lang, med ”00000000” og ”00000001” reservert til henholdsvis broadkast- og masteradresse.

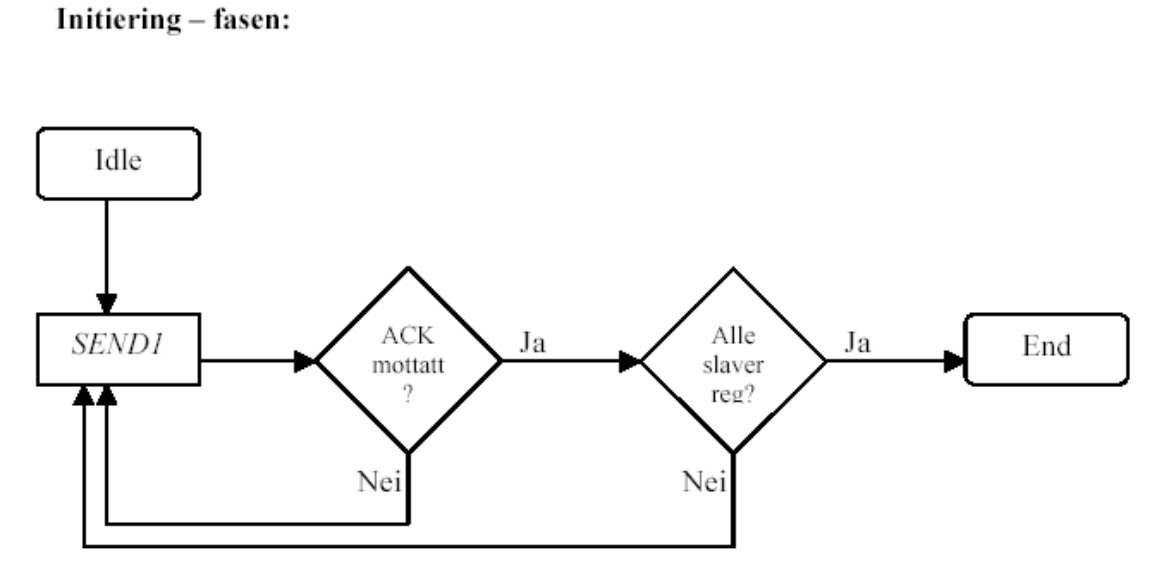
6.8.1 Kommunikasjon sett fra Master sin side

Master har tre typer meldinger som den skal ha kontroll over. Disse er initieringsmeldinger, vanlig tidsluke meldinger og interrupt-tidsluke meldinger. For å skille disse meldingene fra hverandre brukes det kombinasjonen av DAB feltet, CMD- og ACK-flagget.

For initiering-fasen har vi følgende kombinasjonen:

DAB settes lik 0 som er broadkast og CMD ikke er satt og ACK er satt

→ Broadkast && CMD=0 && ACK=1 → *SEND1*.



Figur 6.8: *Flytskjema for aktivitetsstrategi under initiering*

For tidsluke – fasen har vi to muligheter:

1. Vanlig tidsluke:

Innen vanlig tidsluke kan Master, som initierer kommunikasjon, enten sende data til slaven eller spørre slaven om å sende data.

- Når Master skal sende data til slaven:

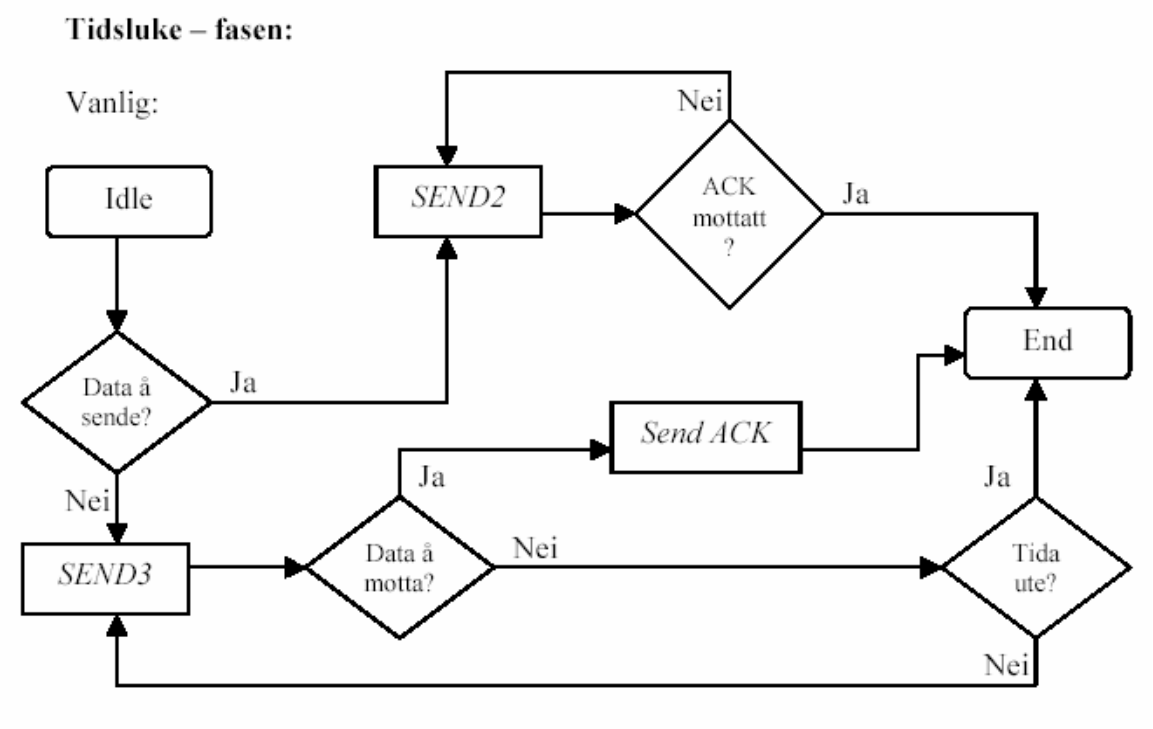
  DAB settes lik logiske adressen (Dest_Adr) til den slaven som Master skal kommunisere med og CMD ikke er satt og ACK er satt

  → Dest_Adr && CMD=0 && ACK=1 → *SEND2*.

- Når Master skal sende "Din tur" melding, dvs spørre slaven å sende data:

  DAB settes lik logiske adressen (Dest_Adr) til den slaven som Master skal kommunisere med og CMD er satt og ACK ikke er satt

  → Dest_Adr && CMD=1 && ACK=0 → *SEND3*.

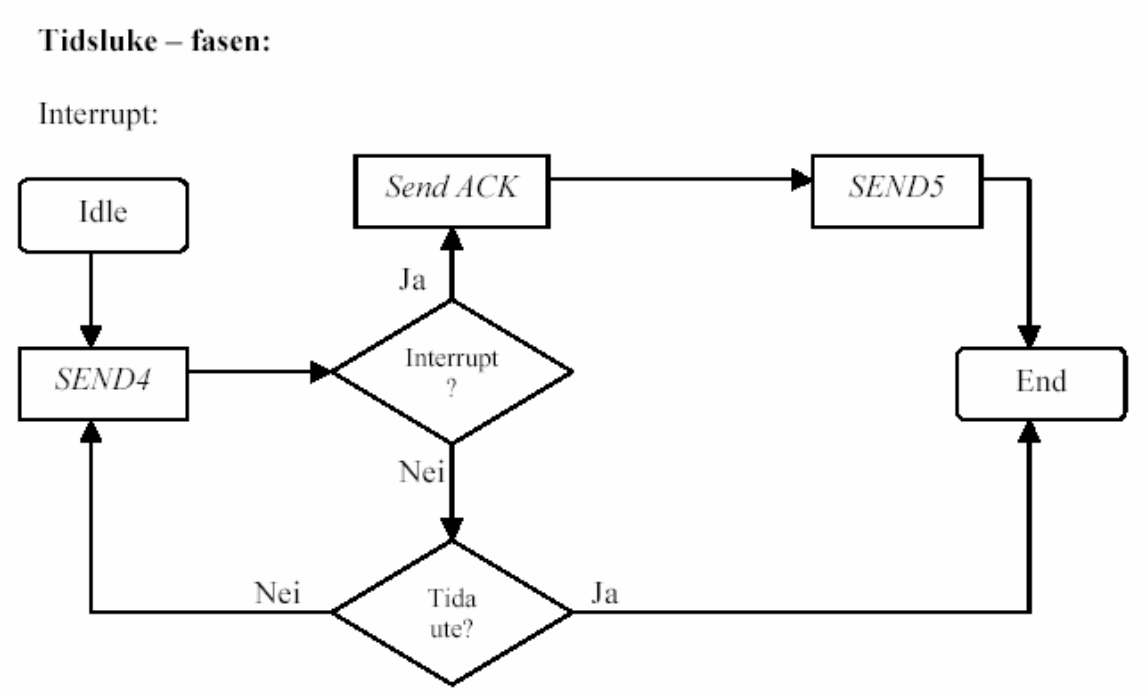


Figur 6.9: *Flytskjema for aktivitetsstrategi under vanlig tidsluke*

2. Interrupt-tidsluke:

- Når Master skal si ifra ”interrupt start”:
  DAB settes lik 0 som er broadkast og CMD er satt og ACK ikke er satt
  → Broadkast && CMD=1 && ACK=0 → *SEND4*.

- Når Master skal si ifra ”interrupt slutt”:
  DAB settes lik 0 som er broadkast og CMD ikke er satt og ACK ikke er satt
  → Broadkast && CMD=0 && ACK=0 → *SEND5*.



Figur 6.10: *Flytskjema for aktivitetsstrategi under interrupt-tidsluke*

6.8.2 Kommunikasjon sett fra slave sin side

Slaver har som oppgave å sjekke meldingene som de mottar fra Master og avgjør om hva Master tillater og/eller spør etter. I tillegg til de meldingene som sendes fra Master som er nevnt over, må de også sjekke multihopp-flagget. Slik at de vet om meldingen kom direkte fra Master eller via repeater. På sendersiden har slaver bare ACK- og Multihopp-flagg å ta seg av.

Fire følgende kombinasjoner av ACK- og Multihopp-flagg er mulig:

1. Send data med ACK-flagget satt på og multihopp-flagget satt på:
   → ACK=1 && multihopp=1 → *SEND6*

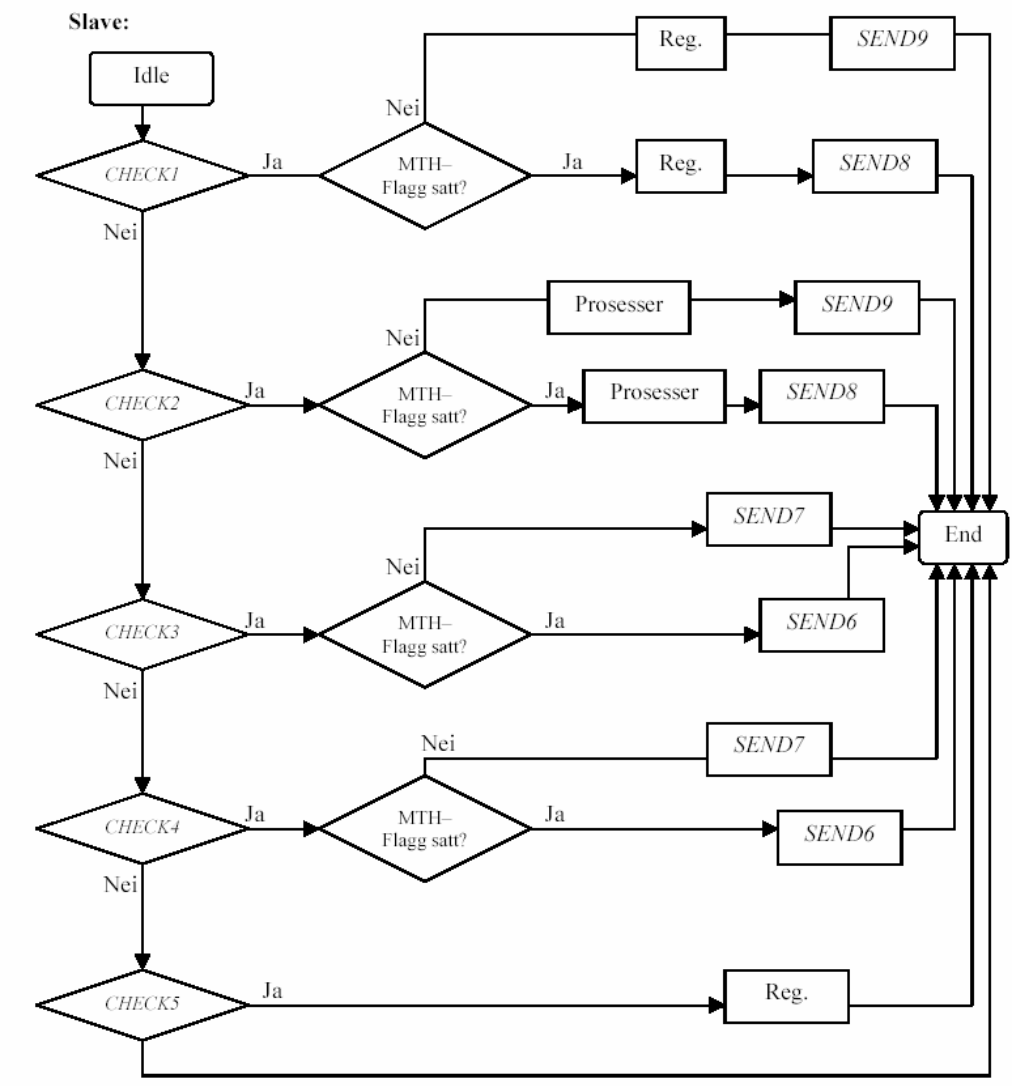
2. Send data med ACK-flagget satt på og multihopp-flagget ikke satt på:
   → ACK=1 && multihopp=0 → *SEND7*

3. Send data med ACK-flagget ikke satt på og multihopp-flagget satt på:
   → ACK=0 && multihopp=1 → *SEND8*

4. Send data med ACK-flagget satt på og multihopp-flagget ikke satt på:
   → ACK=0 && multihopp=0 → *SEND9*

De forskjellige sjekkene som slaver må foreta:

- Registrering melding? → *CHECK1*
- Data fra Master? → *CHECK2*
- "Din tur" melding fra Master? → *CHECK3*
- "Interrupt start" melding fra Master? → *CHECK4*
- "Interrupt slutt" melding fra Master? → *CHECK5*



Figur 6.11: *Flytskjema for slavens aktivitetsstrategi*

6.8.3 Kommunikasjon sett fra repeater sin side

Repeater har som sin viktigst oppgave å videresende meldinger til/fra Gr.2 slaver. Kun meldinger som er adressert til eller kommer fra slaver som er registrert hos repeaterens multihopptabell vil bli videresend. Veldig viktige i den sammenheng er DAB-feltet og multihopp-flagget. Det første innholder mottakerens adresse og det andre viser om meldingen er kommet direkte fra senderen eller via repeater.

Følgende kriterier avgjør om meldingen skal videresendes eller ikke:

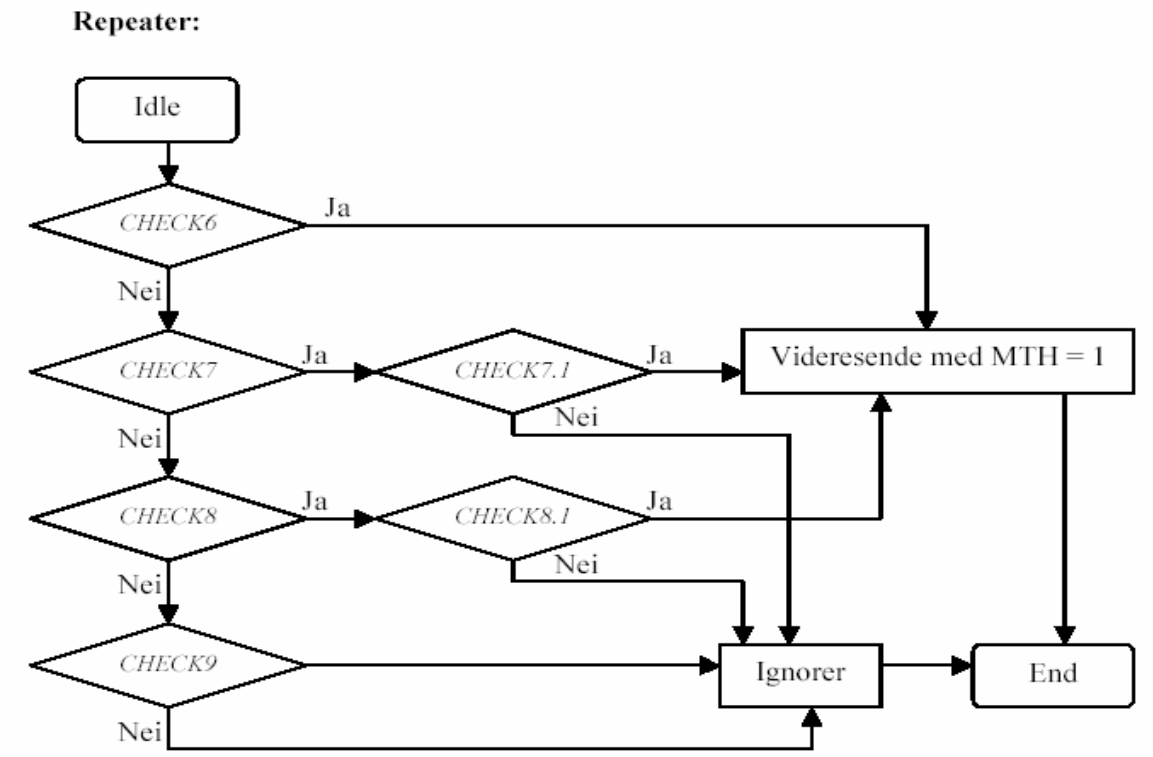
- DAB-feltet = broadkast adresse
  (*CHECK6)* → videresende
  Forklaring: Siden alle broadkast meldinger skal videresendes

- DAB-feltet = Dest_Adr, dvs meldingen er adressert til en av slavene.
  (*CHECK7)* → sjekker om "Dest_Adr" er i multihopptabellen (*CHECK7.1*), dvs om slaven er registrert hos repeater som Gr.2 slave. Hvis ja – videresender, hvis nei – ignorerer.

- DAB-feltet = Master og multihopp-flagget = 1
  (*CHECK8)* → sjekker om "Source_Adr" er i multihopptabellen (*CHECK8.1*),), dvs om slaven er registrert hos denne repeateren. Hvis ja – videresender, hvis nei – ignorerer.
  Forklaring: Det at multihopp-flagget er satt på, betyr ikke nødvendigvis at meldingen skal videresendes av akkurat denne repeateren. Det kan hende at slaven kan være registrert hos en annen repeater.

- DAB-feltet = Master og multihopp-flagget = 0
  (*CHECK9*) → ignorerer
  - Når denne meldingen mottas, får repeateren ut ifra multihopp-flagget vite at meldingen ikke er beregnet for videresending.

  Forklaring: Dette kan skje hvis det tilfeldigvis kommer en melding fra en Gr. 1 slave.



Figur 6.12: *Flytskjema for Repeaterens aktivitetsstrategi*

7 Drøfting

Å designe en protokoll har mange faktorer i seg. En kan grave seg langt ned i teknologi fronter og finne forskjellige metoder og tekniker som kan så brukes i protokolldesign. En annen fremgangsmåte er å finne en eller flere eksisterende protokoll(er) som kan tilfredstille noen aspekter i det man vil få til, for og så gå ut ifra disse og utvikle en ny protokoll til å kunne brukes til et aktuelt formål. Vi har benyttet oss av begge fremgangsmåter.

En viktig forutsetningen for å designe en protokoll er å bli først helt klar over hva den skal brukes til og hvordan den skal implementeres. En god forarbeid ble gjort, før vi kunne begynne med selve protokolldesign. Vi har tatt utgangspunkt i SSP og SNAP protokoller for å designe vår protokoll.

Vår protokoll er tenkt å bli brukt i hjemmeautomatiserings- og industristyringssystemer. Det ble foretatt valg mellom MS og PTP nettverkstyper til fordel av MS nettverk. Dette ble gjort på grunnlag av en karakteristisk trekk ved MS nettverk som tillater å samle beslutningstaking hos en enhet – Master. Dette gjør systemet enklere å utvikle, iverksette og håndtere. Generelt kan det også påstås at det er enklere å få til sikre transaksjoner på MS nettverk enn på PTP nettverk. En annen grunn var at MS nettverk er mer skalerbart, og det kommer godt med i hjemmeautomatiserings- og industristyringssystemer.

For multippel aksess metode ble TDMA valgt. Den ble valgt i første rekke på grunn av sin enkel fysisk implementering og lav priskostnad. I tillegg er TDMA et naturlig valg i systemer hvor noder ikke trenger å sende altfor ofte. Vår TDMA løsning går ut på å tildele slaver tidsluker når de kan foreta kommunikasjon med Master. Denne løsningen har den begrensningen at slaver er nødt til å vente på sin tur for å få sende. Det ble regnet ut at denne ventetiden kan maksimalt være 25,2 s. Denne tiden kan være grei nok for en del applikasjoner som ikke trenger lav responstid. Eksempel på en slik applikasjon kan være en varmeovn. Men det vil alltid finnes applikasjoner hvor forsinkelser kan få alvorlige konsekvenser. Derfor for applikasjoner som krever lav responstid vil det bli brukt interruptsrutiner som disse applikasjonene får bruke uten å vente på sin tidsluke. CSMA vil i den sammenheng bli brukt for å forhindre to eller flere slaver aktivisere sine interruptsrutiner samtidig. CSMA vil bli brukt i sin nonpersistent variant for å redusere yterligere sannsynligheten for kollisjoner. I følge utregningen som ble foretatt vil slaver ved å bruke denne metoden kunne få en ideell responstid på 131,2 ms. I virkeligheten vil denne ventetiden variere, men vil fortsatt være akseptabelt lav.

For feilkontrollmekanisme ble det valgt 8-bits CRC på headeren og 16-bits CRC på datafeltet. CRC er en veldig sikker feildetekterende metode som det i tillegg allerede finnes støtte for i SPP. Det at man kan gjenbruke koden som allerede er skrevet for SPP og at CRC-metoden oppdager 99% av alle feil var de største grunnene for hvorfor akkurat denne metoden ble valgt.

For påloggingsalgoritme og autentisering i hjemmeautomatiseringssystem ble det valgt å bruke fysisk autentisering og Autentisering med utvalgte MAC-adresser. Disse metodene beskytter godt mot den begrensede sikkerhetsfaren som er tilfelle i slike systemer. Ulempen er at man må manuelt registrere slaver hos Masteren. En annen

ulempe med dette valget er at det kan hende at ikke alle slaver er på etter manuell registreringen. Dette vil føre til at noen logiske adresser vil være unødig brukt opp. Et forslag for videreutvikling kan være å la slavene registrere seg automatisk når de blir slått på.

Sikkerhetsrisiko i fjernmåling- og styringssystemer i industriell sammenheng ligger på et helt annet nivå enn det den er i hjemmeautomatiseringssystem, siden utbytte for uvedkommende for å få kontroll over systemet eller data er mye større. Derfor blir det brukt mer sikker og avansert Delt nøkkel autentisering metoden i tillegg til fysisk autentisering.

Kryptering og nøkkelutveksling var det diskusjon om det trenges eller ikke. Som nevnt i Kap. 3.6 så er det et spørsmål om dataene som sendes er av sensitive typer. Systemet er tenkt til å kunne brukes både i hjemmeautomatisering og industriell sammenheng og disse bruksområder stiller forskjellige krav til sikkerhet. Derfor har vi gått for løsningen som tillater å slå på og av krypteringen ved behov. For krypteringsmetode ble det valgt DES-kryptering som er en symmetrisk krypteringsmetode. Grunnen var at det allerede finnes støtte til denne metoden i CC1010 hardware og dessuten er det en nokså sikker metode. Denne metoden bruker samme nøkkelen for både kryptering og dekryptering. Innenfor en lukket mengde med deltakerer, som i vårt tilfelle, kan nøkkeldistribusjon bli gjort manuelt.

Siden det er begrensninger for rekkevidde signalet i et trådløsnettverk kan nå fram, har vi tatt med mulighet for multihopp for å øke denne rekkevidden. Det ble innført ekstra enheter i nettverket som vi kaller reapetere. Repeatere skal være CC1010 kretskort, som skal være bindepunkter mellom Master og slaver som ligger utenfor CC1010 ordinær rekkevidde. Repeatere vil øke operasjonsrekkevidde med opptil 50%. Repeatere har i sin funksjon å videresende pakker. For å unngå unødig trafikk vil ikke reapetere videresende alle pakker som kommer inn men kun de pakkene som er adressert til slaver som er registrert i repeaterene's multihopptabeller. Det kan diskuteres om dette er det beste valget, men for oss så er det et bedre valg siden det ikke medfører ekstra kompleksitet i nettverket. MSP beregnet kun for å støtte et multihopp.

MSP protokollen som vi har kommet fram til har det som trenges for å kunne brukes i et MS nettverk med de kravene som var nevnt i Kap 4.1. Ulempen med protokollen vil være at den ikke er så standardisert slik at det kan brukes i alle MS nettverk. For eksempel så kan den ikke brukes i MS nettverk med mulighet for flere enn et multihopp, eller flere enn en Master.

Et alternativ som vi kunne ha brukt isteden for TDMA tidsluker for slaver, er å bruke CSMA prinsippet for alle slaver. Da hadde alle slaver fått mulighet til å ta kontakt med Master når de ville, bare at de først sjekket om mediet er ledig før de sendte. På denne måten hadde de ikke snakket i munnen på hverandre. Master kunne da bare tatt imot forespørsler og settet dem i en prioriteringstabell og prosessert fortløpende med høyest prioritering først. Master måtte da også sjekke mediet før sending. Dette alternativet ville stridde med et av kravene våre som tilsier at all kommunikasjon bestemmes og styres av Master. Det er på grunn av dette kravet at løsningen med tidsluker ble valgt framfor CSMA løsningen.

8 Konklusjon

I dette prosjektet er det designet en kortholdsradiokommunikasjonsprotokoll for bruk i hjemmeautomatiserings- og industrifjernmåling og styringssystemer. Protokollen ble kalt Master-Slave protokoll (MSP).

Valg av metoder som ble brukt i MSP ble tatt på grunnlag av teorier som vi har studert gjennom litteraturstudium. Enkelhet i implementering, drift og videreutvikling var et av de viktigste kriteriene ved valg av samtlige metoder. Disse valgene ble gjennomgått i drøftingen og ble tidligere foretatt etter vurdering av deres svake og sterke sider og om hvorvidt de passet inn i MSP-konseptet.

Det er aldri så simpelt å designe en enkel men fleksibel protokoll som tilfredstiller krav fra forskjellige applikasjoner. Simpelt var det heller ikke i tilfelle med MSP. Valg av MS nettverkstype sammen med TDMA aksessmetode og halv-duplex kommunikasjonsmodi har hjulpet oss med å holde systemet enkelt. To av mange flaskehalser var responstid og prinsipielle ulikheter i typiske applikasjoner MSP tenkes å bli brukt for. Responstiden ble det tatt hensyn til ved utforming av topologien, multippel aksess og pakkeformat. Det er også på grunn av responstiden at vi har innført interruptsrutiner, som da skal hjelpe til med å minske responstiden for applikasjoner som krever det.

MSP protokollen er designet i utgangspunktet for å kunne brukes i hjemmeautomatisering, men den kan også med noen små modifikasjoner, brukes i industrisammenheng. Innføring av forskjellige autentiseringsmetoder og valgfri kryptering ble gjort for å gi tilfredstillende service for disse ulike applikasjonsområder, som stiller prinsipielt forskjellige krav til sikkerhet.

Designing av multihopp-funksjonalitet har bydd på et annet problem, nemlig hvordan det kunne gjøres slik at systemet forble rimelig enkelt. Det ble brukt repeatere med multihopptabeller som har som sin funksjon å videresende meldinger til slaver som er registrert hos dem. Denne prosessen skal være nesten usynlig for både slaver og Master.

MSP vil i grunn gi en tilfredstillende service for mange applikasjoner. Imidlertid er MSP ingen protokoll som kan bli brukt overalt uten en nøye vurdering av påkrevde tjenester og om hvorvidt disse kunne blitt tilfredstilt av MSP. Spesielt viktig rolle i den sammenheng spiller responstid og hvor alvorlige konsekvenser eventuelle tidsforsinkelser kan få. Selv om interrupt i MSP i mange tilfeller vil sørge for en rimelig god responstid, vil den ikke garantere at responstiden alltid vil være akseptabelt lav. Dette er fordi det er mulig at flere slaver vil aktivisere sine interruptsrutiner samtidig og da er det den raskeste som får sende først. En mulig løsning på dette kunne vært et prioriteringssystem, slik at applikasjoner med høyere prioritet ville fått sende først. En annen ting som kunne komme godt med i MSP, er funksjonaliteten for sovemodus. Siden CC1010 kortene kan kjøres på batterier, er det derfor veldig gunstig at kortene ikke bruker strøm unødvendig når de ikke er med i kommunikasjon.

Det som gjenstår nå er å teste ut protokollen, altså programmeringsdelen og uttesting av programkoden. Det er på denne måten vi ville se om virkeligheten stemmer med det som vi har kommet fram til. Vi har så vidt begynt på programmeringen og hadde

ikke skrevet nok av rutiner til å kunne prøve ut systemet. Det er nå viktig at Chipcon kunne følge opp med programmeringen, for så å se om protokollen fungerer som den skal.

Anvendelsesområder til trådløse hjemmeautomatiseringssystemer og industrielle fjernmåling- og styringssystemer er tallrike. Med tanke på hvilke fordeler disse kan bringe oss, vil det bli et stort behov for utvikling av slike typer systemer i fremtiden. Det kan også godt tenkes at private folk og bedrifter vil bli villige til å kjøpe og bruke slike systemer både i arbeid og underholdning.

9 Forkortelser

ACK	Acknowledgement
ADC	Analog-Digital Converter
CDMA	Code Division Multiple Access
CMD	Command mode
CRC	Cyclic Redundancy code
CSMA	Carrier Sense Multiple Access
DAB	Destination address byte
DB	Data Bytes
DES	Data Encryption Standard
EB	Evaluation Board
EC	Encrypted
EM	Evaluation Module
FDD	Frequency Division Duplex
FDMA	Frequency Division Multiple Access
GUI	Grafic User Interface
HDB	The Header Definition Byte
HiA	Høgskolen i Agder
IDE	Integrated Development Environment
IKT	Informasjon Kommunikasjon Teknologi
I/O	Input/Output
ISO	International Organisation for Standartization

MAC	Media Access Control
MS nettverk	Master-Slave nettverk
MSP	Master Slave Protocol
MTH	Multihopp
NDB	Number of data byte
OSI	Open Systems Interconnection
PC	Personal Computer
PFB	Protocol specific Flag Bytes
PTP nettverk	Peer-to-Peer nettverk
RAM	Random Access Memory
RF	Radio Frequency
RTC	Real-time Clock
SAB	Source address byte
SNAP	Scaleable Node Address Protocol
SQN	Sequence number
SPP	Simple Packet Protocol
SRAM	Static Random Access Memory
SYNC	Synchronization byte
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
TDM	time-division multiplexing
TQFP	Thin Quad Flat Pack
UART	Universal Asynchronous Receiver/Transmitter

10 Referanser

Hovedveileder: Høgskolelektor, Arild Haglund

Andre ansatte ved HiA: Høgskolelektor, Stein Bergsmark

Fra Chipcon AS: Field Application Engineer, Karl Helmer Torvmark

10.1 Litteratur og tidskriftsartikler

Forfatter	Tittel
Andersen, Sven A.	*Protocol Design: Principles and Methods* Department of Telematics, Norwegian University of Science, 1998
Bjerstedt, Åke	*Å skrive rapport* Sebu forlag, Oslo, 1998
Davie, Bruce S. and Peterson, Larry L	*Computer Networks, A System Approach* (2nd edition) Morgan Kaufmann Publishers, San Francisco, 2000
Hall, Douglas V.	*Microprocessors and Interfacing* (2nd edition) McGraw-Hill, Singapore, 1992
Haglund, Arild, Monrad, Atle, Sørensen, Frode, og Køien, Geir	*Forelesningsnotater i IKT 2315 Signaleringsprotokoller og mobilitet* Høgskolen i Agder, Grimstad, 2002
Johnsen, Ragnar og Opland, Reidar	*Forelesningsnotater i DAT 2631 Telematikknett* Høgskolen i Agder, Grimstad, 2001
Keshav, Srinivasan	*An Engineering Approach to Computer Networking* Addison-Wesley, London, 2001
Sommerville, Jan	*Software Engineering* (5th edition) Addison-Wesley, Harlow, 1995
Stalheim, Arnie og Steen-Olsen, Geir	*Innføring i Nettverk-infrastruktur* Norge Books AS, Oslo, 1999
Stallings, William	*Business Data Communications* (4th edition) Prentice Hall, New Jersey, 2000
Sørensen, Frode	*Forelesningsnotater i IKT2200 Datakommunikasjon, videregående* Høgskolen i Agder, Grimstad, 2002
Torvmark, Karl H.	*Short-Range Wireless Design* **www.embedded.com** , 10/01/02

10.2 Internettsider (i den form disse var per 21.05.03)

1.	Datakommunikasjonsprotokoller: Online: **http://www.protocols.com/**
2.	Dataleksikon: Online: **http://www.windows.no/win_ordb.php**
3.	Datalinjen ved Oslo by Steinerskole, (Nettverksdelen): Online: **http://www.poppe.nu**
4.	Generelt om datakommunikasjon: Online: **http://olga.hials.no/datanet/**
5.	Hjemmeautomatiserings linker: Online: **http://www.ostenfeld.dtu.dk/~spacemanspiff/homecontrol/links.php**
6.	Kryptografi: Online: **http://www.uninett.no/arkiv/pca/html/innfoering.html**
7.	Oversikt over lokalnetteknologier: Online: **http://www.ia-stud.hiof.no/~ligr1/kap1_del1.html**
8.	S.N.A.P: Online: **http://www.cultdeadcow.com/**
9.	Short-Range Wireless Design: Online: **http://www.embedded.com/story/OEG20020926S0055**
10.	Whatis?com: Online: **http://whatis.techtarget.com/**

11 Vedlegg

11.1 CUL, Chipcon Ultility Library

```
/*****************************************************************************
*                                                                            *
*     **********                                                             *
*    ************                                                            *
*    ***      ***                                                           *
*    ***  +++  ***                                                          *
*    ***  + +  ***                                                          *
*    ***  +          CHIPCON CC1010                                         *
*    ***  + +  ***    CHIPCON UTILITY LIBRARY                                *
*    ***  +++  ***                                                          *
*    ***      ***                                                           *
*    ************                                                            *
*     *********                                                              *
*                                                                            *
*****************************************************************************
#ifndef CUL_H
#define CUL_H

#include <chipcon/reg1010.h>
#include <chipcon/hal.h>



/*****************************************************************************
*****************************************************************************
*                                                                            *
*   00000  00000  00000                                                      *
*   0    0  0 0                                                              *
*   0    00000  0          - CYCLIC REDUNDANCY CODE -                        *
*   0    0 0  0                                                              *
*   00000  0  0  00000                                                       *
*                                                                            *
*****************************************************************************
*************          COMMON CRC DEFINITIONS          *************
*****************************************************************************
*****************************************************************************/

 // Constants used by the CRC algorithms
#define CRC16_POLY 0x1021
#define CRC16_INIT 0xFFFF
#define CRC8_POLY 0x18
#define CRC8_INIT 0xFF

// A constant which can be used to check if a CRC check succeeded, e.g.:
// if (culSmallCRC8(lastDataByte, crcReg) == CRC_OK) ...
#define CRC_OK 0
```

```
/*****************************************************************************
 *****************************************************************************
 *************              FAST CRC-8               *************
 *****************************************************************************
 *****************************************************************************/


// CRC8 LUT (newCRC = crc8LUT[dataByte ^ oldCRC])
extern byte code crc8LUT[256];


//---------------------------------------------------------------------------
//  byte culFastCRC8(crcData, crcReg)
//          Macro FAST_CRC8(crcData, crcReg)
//
//          Description:
//                  A CRC-8 (DOW) implementation optimized for fast execution.
//                  The function should be called once for each byte in the data
//                  the CRC is to be performed on. Before the invocation on the first byte
//                  the FAST_CRC8_INIT() macro should be called. This final CRC-value
//                  should be added at the end of the data to facilitate a later CRC
//                  check. During checking the check should be performed on all the data
//                  AND the CRC-8 value appended to it. The data is intact if the value
//                  returned is 0.
//
//          Arguments:
//                  byte crcData
//                          The data to perform the CRC-8 operation on.
//                  byte crcReg
//                          The current value of the CRC register. For each additional byte
//                          the value returned for the last invocation should be supplied.
//
//          Return value (not for the macro):
//                  The updated value of the CRC8 register. This corresponds to the
//                  CRC-8 of the data supplied so far. During CRC checking, after working
//                  through all the data and the appended CRC-8 value, the value will be
//                  0 if the data is intact.
//---------------------------------------------------------------------------
byte culFastCRC8(byte crcData, byte crcReg);
#define FAST_CRC8(crcData, crcReg) (crcReg = crc8LUT[crcData ^ crcReg])
#define FAST_CRC8_INIT(crcReg) (crcReg = CRC8_INIT)
#define FAST_CRC8_BLOCK


//---------------------------------------------------------------------------
//  byte culFastCRC8Block(byte *crcData, word length, byte crcReg)
//
//          Description:
//                  An implementation of the above function (culFastCRC8) that operates
//     on blocks of data instead of single bytes.
```

```
//
//          Extra arguments:
//     word length
//          The number of bytes in this block (crcData[]).
//---------------------------------------------------------------------------
byte culFastCRC8Block(byte *crcData, word length, byte crcReg);



/*****************************************************************************
 *****************************************************************************
 ************             SMALL CRC-8              ************
 *****************************************************************************
 *****************************************************************************/


//---------------------------------------------------------------------------
//          byte culSmallCRC8(...)
//
//          Description:
//                    A CRC-8 (DOW) implementation optimized for small code size.
//                    The function should be called once for each byte in the data
//                    the CRC is to be performed on. For the invocation on the first byte
//                    the value CRC8_INIT should be given for _crcReg_. The value returned
//                    is the CRC-8 of the data supplied so far. This CRC-value should be
//                    added at the end of the data to facilitate a later CRC check. During
//                    checking the check should be performed on all the data AND the CRC-16
//                    value appended to it. The data is intact if the value returned is 0.
//
//          Arguments:
//                    byte crcData
//                              The data to perform the CRC-8 operation on.
//                    byte crcReg
//                              The current value of the CRC register. For the first byte the
//                              value CRC8_INIT should be supplied. For each additional byte the
//                              value returned for the last invocation should be supplied.
//
//          Return value:
//                    The updated value of the CRC8 register. This corresponds to the
//                    CRC-8 of the data supplied so far. During CRC checking, after working
//                    through all the data and the appended CRC-8 value, the value will be
//                    0 if the data is intact.
//---------------------------------------------------------------------------
byte culSmallCRC8(byte crcData, byte crcReg);


//---------------------------------------------------------------------------
//  byte culSmallCRC8Block(...)
//
//          Description:
//               A CRC-8 (DOW) implementation optimized for small code size on blocks
//     of data. For the invocation on the first and/or only block the value
//     CRC8_INIT should be given for _crcReg_. The value returned
```

```
//                  is the CRC-8 of the data supplied so far. This CRC-value should be
//                  added at the end of the data to facilitate a later CRC check. During
//                  checking the check should be performed on all the data AND the CRC-8
//                  value appended to it. The data is intact if the value returned is 0.
//
//          Arguments:
//                  byte* crcData
//                          A pointer to the block of data to perform the CRC-8 operation on.
//     word length
//          The number of bytes in this block.
//                  byte crcReg
//                          The current value of the CRC register. For the first block the
//                          value CRC8_INIT should be supplied. For each additional block the
//                          value returned for the last invocation should be supplied.
//
//          Return value:
//                  The updated value of the CRC8 register. This corresponds to the
//                  CRC-8 of the data supplied so far. During CRC checking, after working
//                  through all the data and the appended CRC-8 value, the value will be
//                  0 if the data is intact.
//---------------------------------------------------------------------------
byte culSmallCRC8Block(byte* crcData, word length, byte crcReg);



/*****************************************************************************
 *****************************************************************************
 *************             FAST CRC-16             *************
 *****************************************************************************
 *****************************************************************************/

extern word code crc16LUT[256];

//---------------------------------------------------------------------------
//          word culFastCRC16(crcData, crcReg)
//  Macro FAST_CRC16(crcData, crcReg)
//
//          Description:
//                  A CRC-16 (CCITT) implementation optimized for fast execution.
//                  The function should be called once for each byte in the data
//                  the CRC is to be performed on. Before the invocation on the first byte
//                  the FAST_CRC16_INIT() macro should be called. This final CRC-value
//                  should be added at the end of the data to facilitate a later CRC
//                  check. During checking the check should be performed on all the data
//                  AND the CRC-16 value appended to it. The data is intact if the value
//                  returned is 0.
//
//          Arguments:
//                  byte crcData
//                          The data to perform the CRC-16 operation on.
//                  word crcReg
//                          The current value of the CRC register. For each additional byte
//                          the value returned for the last invocation should be supplied.
```

```
//
//          Return value:
//                    The updated value of the CRC16 register. This corresponds to the
//                    CRC-16 of the data supplied so far. During CRC checking, after working
//                    through all the data and the appended CRC-16 value, the value will be
//                    0 if the data is intact.
//---------------------------------------------------------------------------
word culFastCRC16(byte crcData, word crcReg);
#define FAST_CRC16(crcData, crcReg) (crcReg = (crcReg << 8) ^
                                                                    crc16LUT[((byte)(crcReg
>> 8)) ^ crcData])
#define FAST_CRC16_INIT(crcReg) (crcReg = CRC16_INIT)


//---------------------------------------------------------------------------
//  word culFastCRC16Block(byte *crcData, word length, word crcReg)
//
//          Description:
//                    An implementation of the above function (culFastCRC16) that operates
//     on blocks of data instead of single bytes.
//
//          Extra arguments:
//     word length
//        The number of bytes in this block (crcData[]).
//---------------------------------------------------------------------------
word culFastCRC16Block(byte *crcData, word length, word crcReg);




/*****************************************************************************
 *****************************************************************************
 *************          SMALL CRC-16          *************
 *****************************************************************************
 *****************************************************************************/


//---------------------------------------------------------------------------
//          word culSmallCRC16(...)
//
//          Description:
//                    A CRC-16/CCITT implementation optimized for small code size.
//                    The function should be called once for each byte in the data
//                    the CRC is to be performed on. For the invocation on the first byte
//                    the value CRC16_INIT should be given for _crcReg_. The value returned
//                    is the CRC-16 of the data supplied so far. This CRC-value should be
//                    added at the end of the data to facilitate a later CRC check. During
//                    checking the check should be performed on all the data AND the CRC-16
//                    value appended to it. The data is intact if the value returned is 0.
//
//          Arguments:
//                    byte crcData
//                              The data to perform the CRC-16 operation on.
//                    word crcReg
```

```
//                          The current value of the CRC register. For the first byte the
//                          value CRC16_INIT should be supplied. For each additional byte the
//                          value returned for the last invocation should be supplied.
//
//          Return value:
//                  The updated value of the CRC16 register. This corresponds to the
//                  CRC-16 of the data supplied so far. During CRC checking, after working
//                  through all the data and the appended CRC-16 value, the value will be
//                  0 if the data is intact.
//---------------------------------------------------------------------------
word culSmallCRC16(byte crcData, word crcReg);


//---------------------------------------------------------------------------
//  word culSmallCRC16Block(...)
//
//          Description:
//             A CRC-16/CCITT implementation optimized for small code size on blocks
//     of data. For the invocation on the first and/or only block the value
//     CRC16_INIT should be given for _crcReg_. The value returned
//                  is the CRC-16 of the data supplied so far. This CRC-value should be
//                  added at the end of the data to facilitate a later CRC check. During
//                  checking the check should be performed on all the data AND the CRC-16
//                  value appended to it. The data is intact if the value returned is 0.
//     It is important that the CRC value is appended to the data in BIG
//     ENDIAN byte order. (Use the CRC16Append(...) function for this.)
//
//          Arguments:
//                  byte* crcData
//                          A pointer to the block of data to perform the CRC-16 operation on.
//     lword length
//        The number of bytes in this block.
//                  word crcReg
//                          The current value of the CRC register. For the first block the
//                          value CRC16_INIT should be supplied. For each additional block the
//                          value returned for the last invocation should be supplied.
//
//          Return value:
//                  The updated value of the CRC16 register. This corresponds to the
//                  CRC-16 of the data supplied so far. During CRC checking, after working
//                  through all the data and the appended CRC-16 value, the value will be
//                  0 if the data is intact.
//---------------------------------------------------------------------------
word culSmallCRC16Block(byte* crcData, word length, word crcReg);


//---------------------------------------------------------------------------
//  void culCRC16Append(...)
//
//          Description:
//             Appends a CRC value to a block of data in BIG ENDIAN byte order.
//
//          Arguments:
//                  byte* dataPtr
```

```
//                          A pointer to where to insert the CRC value.
//                  word crcValue
//                          The CRC value to insert.
//
//          Return value:
//      void
//---------------------------------------------------------------------------
void culCRC16Append(byte* dataPtr, word crcValue);


/*****************************************************************************/



/*****************************************************************************
 *****************************************************************************
 *                                                   *
 *    0   0  00000  00000                                  *
 *    00 00  0      0   0                                *
 *    0 0 0  00000  00000        - MASTER SLAVE  PROTOCOL -            *
 *    0   0      0  0                                    *
 *    0   0  00000  0                                      *
 *                                                   *
 *****************************************************************************
 *****************************************************************************
/*****************************************************************************
 *****************************************************************************
 *************              Setup                *************
 *****************************************************************************
 *****************************************************************************/

// Local settings
typedef struct {
        byte myAddress;                         // The address of this network node
        word rxTimeout;                         // Receive timeout (10s of msecs)
        word txAckTimeout;                      // Ack receive timeout (10s of msecs)
        byte txAttempts;                // Transmission attempts (applies only to ack'ed messages)
        byte txPreambleByteCount;       // Number of transmitted preamble bytes
} MSP_SETTINGS;

extern MSP_SETTINGS xdata mspSettings;

// The broadcast address
#define MSP_BROADCAST 0




//---------------------------------------------------------------------------
//  void mspSetupRF (...)
//
//  Description:
//      Set up MSP for transmission or reception.
//              Call this function to (re)calibrate the radio, or to switch between
```

```
//                  different RF settings.
//
//  Arguments:
//     RF_RXTXPAIR_SETTINGS code* pRF_SETTINGS
//                        RF settings (frequencies, modem settings, etc.)
//
//     RF_RXTXPAIR_CALDATA xdata* pRF_CALDATA
//                        Calibration results
//
//                  word clkFreq
//                        The XOSC clock frequency in kHz.
//
//                  bool calibrate
//                        Calibrate now. *pRF_CALDATA is written to when calibrate = TRUE,
//                        and read from otherwise. *pRF_CALDATA must always be valid.
//
//          Note: This function also enables timeouts, using timer 3.
//-----------------------------------------------------------------------
void mspSetupRF (
          RF_RXTXPAIR_SETTINGS code* pRF_SETTINGS,
          RF_RXTXPAIR_CALDATA xdata* pRF_CALDATA,
          word clkFreq,
          bool calibrate
);

// RF constants
#define MSP_PREAMBLE_SENSE_BITS   16          // Number of preamble bits to sense
//before RX (use min. 16)
#define MSP_ZEROPAD_COUNT                     2           // Number of trailing zero paddings



/*************************************************************************
 *************************************************************************
 *************          Communication              *************
 *************************************************************************
 *************************************************************************/

// Return values from mspSend() and mspReceive()
#define MSP_BUSY                              0
#define MSP_RX_STARTED                        1
#define MSP_TX_STARTED                        1

// MSP_TX_INFO/MSP_RX_INFO.flags
#define MSP_ACK_REQ                           0x01  // Acknowledge request (2-bit)
#define MSP_ACK                                     0x02  // Ack flag (used internally)
#define MSP_ENCRYPTED_DATA                    0x04  // The packet data is encrypted
#define MSP_SEQUENCE_BIT              0x08  // Toggled when a transmission finishes successfully (2-
bit)
#define MSP_MULTIHOP_BIT              0x10  // is 1 when a transmission comes from a rute (1-bit)
#define MSP_MODE_BIT                          0x20  // om det er "din tur"(1) eller data(0) (1-bit)

//-----------------------------------------------------------------------
```

```
// The transmit struct
typedef struct {
        byte destination;                       // The destination of the packet
        byte flags;                                     // Flags
        byte dataLen;                           // The length of the transmitted data
        byte xdata *pDataBuffer;        // The transmitted data
        byte status;                            // The transmission status (result)
} MSP_TX_INFO;

// MSP_TX_INFO.status:
#define MSP_TX_TRANSMITTING             0x01 // The packet is being transmitted
#define MSP_TX_WAITING_FOR_ACK  0x02 // Waiting for the ack packet
#define MSP_TX_ACK_INVALID      0x04 // An invalid ack was received
#define MSP_TX_ACK_TIMEOUT              0x08 // No response
#define MSP_TX_FINISHED                 0x10 // OK!
#define MSP_TX_RESET                    0x20 // Reset by mspReset()


//-------------------------------------------------------------------------
//  byte mspSend (MSP_TX_INFO xdata *pTXInfo)
//
//  Description:
//     If the transceiver is ready (in idle mode), the transmit section will
//                  be powered up and the RF interrupt enabled. The RF ISR will then
//                  transmit the packet (pTXInfo) and receive the ack (if requested). If
//                  requested (mspSettings.txAttempts = n), the packet will be re-
//                  transmitted (n-1) times, until the ack is received. When finished the
//                  transmit section will be powered down.
//
//                  This function will return immediately and the application can
//                  continue while the ISR transmits the packet. When finished,
//                  mspStatus() will return IDLE_MODE. During the transmission it will
//                  return TX_MODE or TXACK_MODE.
//
//                  After the transmission: Use pTXInfo->status to find out what happened:
//                          MSP_TX_ACK_INVALID = Something was received, but not the ack
//                          MSP_TX_ACK_TIMEOUT = No response
//                          MSP_TX_FINISHED
//
//                  mspSettings.txAckTimeout gives the ack timeout in msecs.
//
//  Arguments:
//     MSP_TX_INFO xdata *pTXInfo
//                          An MSP_TX_INFO struct must be prepared before the transmission,
//                          including the following values:
//                                  destination (MSP_BROADCAST or 2-255)
//                                  flags (MSP_ACK_REQ | MSP_ENCRYPTED_DATA)
//                                  dataLen (Length of *pDataBuffer, 0-255)
//                                  pDataBuffer (pointer to the transmission data buffer)
//
//  Return value:
//                  byte
//          MSP_TX_STARTED if OK
//                          MSP_BUSY if not ready
```

```
//-------------------------------------------------------------------------
byte mspSend (MSP_TX_INFO xdata *pTXInfo);



//-------------------------------------------------------------------------
// The receive struct:
typedef struct {
        byte source;                                    // The packet source address
        byte flags;                                             // Flags
        byte maxDataLen;                                // Maximum data length (size of *pDataBuffer)
        byte dataLen;                                   // The length of the received data
        byte xdata *pDataBuffer;   // The received data
        byte status;                                    // The reception status (result)
} MSP_RX_INFO;

// MSP_RX_INFO.status:
#define MSP_RX_WAITING                  0x01 // Waiting for packet
#define MSP_RX_RECEIVING    0x02 // Receiving packet
#define MSP_RX_ACKING                   0x04 // Transmitting ack
#define MSP_RX_TIMEOUT                  0x08 // Reception timed out
#define MSP_RX_TOO_LONG                 0x10 // The packet data was longer
//than the given maximum length (the buffer was not updated)
#define MSP_RX_FINISHED                 0x20 // OK!
#define MSP_RX_RESET            0x40 // Reset by mspReset()

//-------------------------------------------------------------------------
//  byte mspReceive (MSP_RX_INFO xdata* pRXInfo)
//
//  Description:
//      If the transceiver is ready (in idle mode), the receive section will
//                      be powered up and the RF interrupt enabled. The RF ISR will then
//                      receive the packet and transmit an ack if requested to. When finished,
//                      the receive section will be powered down.
//
//                      This function will return immediately and the application can
//                      continue while the ISR receives the packet. When finished, mspStatus()
//                      will return MSP_IDLE_MODE. During the transmission it will
//                      return RX_MODE or RXACK_MODE.
//
//                      After the reception: Use pRXInfo->status to find out what happened:
//                              MSP_RX_TIMEOUT =   Timeout (nothing received).
//                              MSP_RX_TOO_LONG =  dataLen > maxDataLen (the buffer is invalid).
//                              MSP_RX_FINISHED      =  source, dataLen and *pDataBuffer in *pRXInfo
//                                      are valid.
//
//  Arguments:
//      MSP_RX_INFO xdata* pRXInfo
//                              An MSP_RX_INFO struct must be prepared before the reception,
//                              including the following values:
//                                      maxDataLen (Length of the data buffer, 0-255)
//                                      pDataBuffer (pointer to the reception buffer)
//
```

```
//  Return value:
//                  byte
//                          MSP_RX_STARTED if OK
//                          MSP_BUSY if not ready
//-------------------------------------------------------------------------
byte mspReceive (MSP_RX_INFO xdata *pRXInfo);



//-------------------------------------------------------------------------
//  byte mspStatus (void)
//
//  Description:
//     Returns the status of the MSP finite state machine.
//
//  Return value:
//                  byte
//                          MSP_IDLE_MODE =  Ready to transmit or receive
//                          MSP_TX_MODE =    Transmitting a packet
//                          MSP_TXACK_MODE = Waiting for the ack
//                          MSP_RX_MODE =    Waiting for or receiving a packet
//                          MSP_RXACK_MODE = Transmitting the ack
//
//          NOTE: This function has also been implemented as a macro, MSP_STATUS(),
//          which is both smaller and faster.
//-------------------------------------------------------------------------
byte mspStatus(void);

// #define MSP_STATUS() ... is located in mspInternal.h

// Return values:
#define MSP_IDLE_MODE       0x00
#define MSP_TX_MODE         0x10
#define MSP_TXACK_MODE      0x20
#define MSP_RX_MODE         0x40
#define MSP_RXACK_MODE      0x80



//-------------------------------------------------------------------------
//  void mspReset (void)
//
//  Description:
//                  Stops a transmission or reception by
//                          - turning the transceiver off
//                          - entering IDLE mode (mspStatus())
//-------------------------------------------------------------------------
void mspReset (void);
```

```
/*****************************************************************************
 *****************************************************************************
 *************          Timer with Callback          *************
 *****************************************************************************
 *****************************************************************************/

// MSP supports 2 custom callbacks which can be set dynamically

//---------------------------------------------------------------------------
//  void mspSetTimerCB (void)
//
//      Description:
//              Add timer callback
//
//      Arguments:
//              byte cb
//                      Callback index - use MSP_CUSTOM_0_TIMER or
MSP_CUSTOM_1_TIMER
//                      The MSP finite state machine uses MSP_FSM_TIMER
//              void (*pF) ()
//                      Pointer to the function to call
//              word *pTicks
//                      The timeout in 10s of msecs
//
//      NOTE: *pTicks will be decremented by the Timer 3 interrupt
//---------------------------------------------------------------------------
void mspSetTimerCB (byte cb, void (*pF) (), word *pTicks);

// Available callbacks
#define MSP_FSM_TIMER           0 // used by the MSP
#define MSP_CUSTOM_0_TIMER      1 // free
#define MSP_CUSTOM_1_TIMER      2 // free



//---------------------------------------------------------------------------
//  word mspGetTime (void)
//
//      Description:
//              Returns the value of the 10-msec counter, which is started by
//              mspSetupRF(...)
//
//      Return value:
//              The current time in 10s of msecs
//---------------------------------------------------------------------------
word mspGetTime (void);



//---------------------------------------------------------------------------
//  MACRO: MSP_CHANGE_TICKS(var, value)
//
//      Description:
```

```
//                  Use this macro to change the tick counter variables used with the
//                  callback timer
//
//          Arguments
//                  (word) var
//                          The variable
//                  (word) value
//                          The new value
//---------------------------------------------------------------------------
#define MSP_CHANGE_TICKS(var, value)            \
        do {                                                                    \
                INT_ENABLE (INUM_TIMER3, INT_OFF);          \
                var = (word) value;                                     \
                INT_ENABLE (INUM_TIMER3, INT_ON);           \
        } while (0)

// sppInternal.h requires some of the type definitions listed above!
#include <chipcon/sppInternal.h>
#include <chipcon/mspInternal.h>
#endif // CUL_H
```

11.2 Internal Header File

```
/*****************************************************************************
*                                      *
*     **********                          *
*    ************                           *
*    ***      ***                        *
*    ***  +++  ***                         *
*    ***  + +  ***                        *
*    ***  +                CHIPCON CC1010          *
*    ***  + +  ***       CUL MASTER SLAVE PROTOCOL (MSP)    *
*    ***  +++  ***          INTERNAL HEADER FILE        *
*    ***     ***                          *
*     ***********                           *
*      *********                           *
*                                     *
*****************************************************************************
* Internal header file for the CUL master slave protocol.          *
*                                      *
*****************************************************************************/

#ifndef MSP_INTERNAL_H
#define MSP_INTERNAL_H

#include <chipcon/cul.h>


typedef void (code *pCBFM) ();
extern pCBFM xdata mspRFStateFunc;
```

```
// TX
void TX_START (void);
void TX_PREAMBLE (void);
void TX_SYNC (void);
void TX_DAB (void);
void TX_SAB (void);
void TX_DATA_LEN (void);
void TX_FLAG (void);
void TX_CRC8 (void);
void TX_DBX_START (void);
void TX_DBX (void);
void TX_CRC16_DATA_H (void);
void TX_CRC16_DATA_L (void);
void TX_ZEROPAD_START (void);
void TX_ZEROPAD (void);

// TXACK
void TXACK_START (void);
void TXACK_WAIT (void);
void TXACK_DAB (void);
void TXACK_SAB (void);
void TXACK_DATA_LEN (void);
void TXACK_FLAG (void);
void TXACK_CRC8 (void);

// RX
void RX_WAIT (void);
void RX_DAB (void);
void RX_SAB (void);
void RX_DATA_LEN (void);
void RX_FLAG (void);
void RX_CRC8_HEADER (void);
void RX_DBX_START (void);
void RX_DBX (void);
void RX_CRC16_DATA_H (void);
void RX_CRC16_DATA_L (void);

// RXACK
void RXACK_START (void);
void RXACK_PREAMBLE (void);
void RXACK_SYNC (void);
void RXACK_DAB (void);
void RXACK_SAB (void);
void RXACK_DATA_LEN (void);
void RXACK_FLAG (void);
void RXACK_CRC8 (void);
void RXACK_ZEROPAD (void);
//-------------------------------------------------------------------------



//-------------------------------------------------------------------------
// The internal data struct
```

```
typedef struct {
        MSP_RX_INFO xdata* pRXI;
        MSP_TX_INFO xdata* pTXI;
        RF_RXTXPAIR_SETTINGS code* pRF_SETTINGS;
        RF_RXTXPAIR_CALDATA xdata* pRF_CALDATA;
        byte counter;
        byte crc8;
        word crc16;
        byte mode;
        word toTicks;
        byte usedTxAttempts;
} MSP_INTERNAL_DATA;

// This struct is declared in mspISR.c
extern MSP_INTERNAL_DATA xdata mspIntData;
//-------------------------------------------------------------------------

// Faster version of void mspStatus (void)
#define MSP_STATUS() (mspIntData.mode)




//-------------------------------------------------------------------------
// Tranceiver macros:
#define MSP_POWER_UP_TX()

        do {

                halRFSetRxTxOff (RF_TX, mspIntData.pRF_SETTINGS,
mspIntData.pRF_CALDATA);
                RFCON|=0x01;

                RF_SEND_BYTE(RF_PREAMBLE_BYTE);

                RF_START_TX();

        } while (0)

// Fast version of MSP_POWER_UP_TX() (used to switch from RX to TX in the FSM)
#define MSP_FAST_POWER_UP_TX()
        do {
                PLL = mspIntData.pRF_SETTINGS->pll_tx;
                RFMAIN = 0xF0;
    TEST5 = 0x10 | mspIntData.pRF_CALDATA->vco_ao_tx;
    TEST6 = 0x3B;
    CURRENT = mspIntData.pRF_SETTINGS->current_tx;
                RFCON|=0x01;
                RF_SEND_BYTE(RF_PREAMBLE_BYTE);
                RF_START_TX();
        } while (0)
```

```
#define MSP_POWER_UP_RX()

        do {

                halRFSetRxTxOff (RF_RX, mspIntData.pRF_SETTINGS,
mspIntData.pRF_CALDATA);
                RF_START_RX();

        } while (0)

// Fast version of MSP_POWER_UP_RX() (used to switch from TX to RX in the FSM)
#define MSP_FAST_POWER_UP_RX()
        do {
                PLL = mspIntData.pRF_SETTINGS->pll_rx;
      RFMAIN = 0x30;
      TEST5 = 0x10 | mspIntData.pRF_CALDATA->vco_ao_rx;
      TEST6 = 0x20 | mspIntData.pRF_CALDATA->chp_co_rx;
      CURRENT = mspIntData.pRF_SETTINGS->current_rx;
                RF_START_RX();
   } while (0)

// equivalent to halRFSetRxTxOff(RF_OFF, 0, 0), but faster
#define MSP_POWER_DOWN() MSP_FAST_POWER_DOWN()

// Fast version of MSP_POWER_DOWN_TX()
#define MSP_FAST_POWER_DOWN() (RFMAIN = 0x38)
//-----------------------------------------------------------------------------




//-----------------------------------------------------------------------------
//  void mspStartTimer (void)
//
//  Description:
//              Run timer3 with a period of 10 msecs.
//
//      Arguments:
//              word clkFreq
//                      The XOSC clock frequency in kHz.
//-----------------------------------------------------------------------------
void mspStartTimer (word clkFreq);




//-----------------------------------------------------------------------------
//  void mspTimeout(void)
//
//  Description:
//              Timeout
//              Callback from sxpTimer: TIMER3_ISR()
//-----------------------------------------------------------------------------
void mspTimeout(void);
```

```
//---------------------------------------------------------------------------
//  MACRO: MSP_INIT_TIMEOUTS
//
//  Description:
//     Initialize timeouts by adding mspTimeout() to the sxpTimer callback.
//---------------------------------------------------------------------------
#define MSP_INIT_TIMEOUTS() (mspSetTimerCB(MSP_FSM_TIMER, mspTimeout,
&mspIntData.toTicks))


//---------------------------------------------------------------------------
//  MSP_ENABLE_TIMEOUT(int timeout), MSP_DISABLE_TIMEOUT()
//
//  Description:
//     Enables or disables the MSP timeout, which is used in the TXACK and RX
//                    modes. mspIntData.toTicks will count from _timeout down to 0.
//                    The timeout uses timer 3 (sxpTimer.c/h).
//
//          Arguments:
//                    int timeout
//                              The timeout (in 10s of msecs)
//---------------------------------------------------------------------------
#define MSP_ENABLE_TIMEOUT(x) (mspIntData.toTicks = x)
#define MSP_DISABLE_TIMEOUT() (mspIntData.toTicks = 0)


// The tick function:
void mspTimerTick (void);


#endif // MSP_INTERNAL_H
```

11.3 MSP Receive

```
/*****************************************************************************
 *                                        *
 *     **********                           *
 *    ************                            *
 *    ***      ***                          *
 *    ***  +++  ***                           *
 *    ***  + +  ***                          *
 *    ***  +                CHIPCON CC1010          *
 *    ***  + +  ***           CUL - msp              *
 *    ***  +++  ***                           *
 *    ***     ***                          *
 *    ************                            *
 *     *********                            *
 *                                        *
```

```
 ***************************************************************************/

#include <chipcon/mspInternal.h>

//---------------------------------------------------------------------------
//  byte mspReceive (MSP_RX_INFO xdata* pRXInfo)
//
//  Description:
//     If the transceiver is ready (in idle mode), the receive section will
//                    be powered up and the RF interrupt enabled. The RF ISR will then
//                    receive the packet and transmit an ack if requested to. When finished,
//                    the receive section will be powered down.
//
//                    This function will return immediately and the application can
//                    continue while the ISR receives the packet. When finished, mspStatus()
//                    will return IDLE_MODE. During the transmission it will
//                    return RX_MODE or RXACK_MODE.
//
//                    After the reception: Use pRXInfo->status to find out what happened:
//                              MSP_RX_TIMEOUT =   Timeout (nothing received).
//                              MSP_RX_TOO_LONG =  dataLen > maxDataLen (the buffer is invalid).
//                              MSP_RX_FINISHED      =  source, dataLen and *pDataBuffer in *pRXInfo
//                                        are valid.
//
//  Arguments:
//     MSP_RX_INFO xdata *pRXInfo
//                              An MSP_RX_INFO struct must be prepared before the reception,
//                              including the following values:
//                                        maxDataLen (Length of the data buffer, 0-255)
//                                        pDataBuffer (pointer to the reception buffer)
//
//  Return value:
//                    byte
//                              MSP_RX_STARTED if OK
//                              MSP_BUSY if not ready
//---------------------------------------------------------------------------
byte mspReceive (MSP_RX_INFO xdata *pRXInfo) {
          INT_GLOBAL_ENABLE (INT_OFF);

          // Begin reception only if we're in the idle state
          if (mspIntData.mode == MSP_IDLE_MODE) {

                    // Initialize the fsm
                    mspRFStateFunc = RX_WAIT;
                    mspIntData.mode = MSP_RX_MODE;
                    mspIntData.pRXI = pRXInfo;
                    mspIntData.pRXI->status = MSP_RX_WAITING;

                    // Activate receive timeout (disable timer 3 while we're changing the 16-bit number)
                    if (mspSettings.rxTimeout) {
                              MSP_ENABLE_TIMEOUT(mspSettings.rxTimeout);
                    } else {
                              MSP_DISABLE_TIMEOUT();
                    }
```

```
                INT_GLOBAL_ENABLE (INT_ON);

                // Enable transceiver interrupt and enter the FSM
                INT_SETFLAG (INUM_RF, INT_CLR);
                INT_PRIORITY (INUM_RF, INT_HIGH);
                INT_ENABLE (INUM_RF, INT_ON);

                // Power up for RX
                MSP_POWER_UP_RX();

                return MSP_RX_STARTED;
        } else {
                INT_GLOBAL_ENABLE (INT_ON);
                return MSP_BUSY;
        }
} // mspReceive
```

11.4 MSP Reset

```
/*****************************************************************************
*                                          *
*     **********                             *
*    ************                              *
*    ***      ***                            *
*    ***  +++  ***                             *
*    ***  + +  ***                            *
*    ***  +                CHIPCON CC1010          *
*    ***  + +  ***           CUL - msp               *
*    ***  +++  ***                             *
*    ***     ***                             *
*    ************                              *
*     *********                              *
*                                          *
*****************************************************************************
*                                          *
*****************************************************************************/

#include <chipcon/mspInternal.h>

//---------------------------------------------------------------------------
//  void mspReset (void)
//
//  Description:
//                  Stops a transmission or reception by
//                          - turning the transceiver off
//                          - entering IDLE mode (mspStatus())
//---------------------------------------------------------------------------
void mspReset (void) {

        // Turn off interrupts
        INT_ENABLE (INUM_RF, INT_OFF);
```

```
        INT_ENABLE (INUM_TIMER3, INT_OFF);
        MSP_DISABLE_TIMEOUT();
        INT_ENABLE (INUM_TIMER3, INT_ON);

        // Turn off RF
        MSP_POWER_DOWN();

        // Modify the packet status
        if (mspIntData.mode & (MSP_TX_MODE | MSP_TXACK_MODE)) {
                mspIntData.pTXI->status = MSP_TX_RESET;
        } else if (mspIntData.mode & (MSP_RX_MODE | MSP_RXACK_MODE)) {
                mspIntData.pRXI->status = MSP_RX_RESET;
        }

        // Return to idle mode
        mspIntData.mode = MSP_IDLE_MODE;

} // mspReset
```

11.5 MSP Send

```
/*****************************************************************************
 *                                      *
 *     **********                           *
 *    ************                           *
 *    ***     ***                          *
 *    ***  +++  ***                          *
 *    ***  ++  ***                          *
 *    ***  +                CHIPCON CC1010            *
 *    ***  ++  ***             CUL - msp                *
 *    ***  +++  ***                          *
 *    ***     ***                          *
 *     ***********                           *
 *     *********                            *
 *                                      *
 *****************************************************************************
 *****************************************************************************/

#include <chipcon/mspInternal.h>

//---------------------------------------------------------------------------
//  byte mspSend (MSP_TX_INFO xdata *pTXInfo)
//
//  Description:
//     If the transceiver is ready (in idle mode), the transmit section will
//                  be powered up and the RF interrupt enabled. The RF ISR will then
//                  transmit the packet (pTXInfo) and receive the ack (if requested). If
//                  requested (mspSettings.txAttempts = n), the packet will be re-
//                  transmitted (n-1) times, until the ack is received. When finished the
//                  transmit section will be powered down.
//
```

```
//                      This function will return immediately and the application can
//                      continue while the ISR transmits the packet. When finished,
//                      mspStatus() will return IDLE_MODE. During the transmission it will
//                      return TX_MODE or TXACK_MODE.
//
//                      After the transmission: Use pTXInfo->status to find out what happened:
//                              MSP_TX_ACK_INVALID = Something was received, but not the ack
//                              MSP_TX_ACK_TIMEOUT = No response
//                              MSP_TX_FINISHED
//
//                      mspSettings.txAckTimeout gives the ack timeout in msecs.
//
//  Arguments:
//     MSP_TX_INFO xdata *pTXInfo
//                              An MSP_TX_INFO struct must be prepared before the transmission,
//                              including the following values:
//                                      destination (MSP_BROADCAST or 1-255)
//                                      flags (MSP_ACK_REQ | MSP_ENCRYPTED_DATA)
//                                      dataLen (Length of *pDataBuffer, 0-255)
//                                      pDataBuffer (pointer to the transmission data buffer)
//
//  Return value:
//                      byte
//          MSP_TX_STARTED if OK
//                              MSP_BUSY if not ready
//-------------------------------------------------------------------------
byte mspSend (MSP_TX_INFO xdata *pTXInfo) {

        // Begin transmission only if we're in the idle state
        INT_GLOBAL_ENABLE (INT_OFF);
        if (mspIntData.mode == MSP_IDLE_MODE) {

                // Initialize the fsm
                mspRFStateFunc = TX_START;
                mspIntData.mode = MSP_TX_MODE;
                mspIntData.pTXI = pTXInfo;
                mspIntData.usedTxAttempts = 0;
                INT_GLOBAL_ENABLE (INT_ON);

                // Enable transceiver interrupt and enter the FSM
                INT_SETFLAG (INUM_RF, INT_CLR);
                INT_PRIORITY (INUM_RF, INT_HIGH);
                INT_ENABLE (INUM_RF, INT_ON);

                // Start TX
                MSP_POWER_UP_TX();

                return MSP_TX_STARTED;
        } else {
                INT_GLOBAL_ENABLE (INT_ON);
                return MSP_BUSY;
        }
} // mspSend
```

11.6 MSP RF Setup

```
/***************************************************************************
*                                      *
*     **********                         *
*    ************                          *
*    ***      ***                        *
*    ***  +++  ***                         *
*    ***  ++  ***                         *
*    ***  +              CHIPCON CC1010          *
*    ***  ++  ***          CUL - msp             *
*    ***  +++  ***                         *
*    ***     ***                         *
*     ***********                          *
*      *********                           *
*                                      *
***************************************************************************

#include <chipcon/mspInternal.h>

//-------------------------------------------------------------------------
//  void mspSetupRF (...)
//
//  Description:
//     Set up MSP for transmission or reception.
//                  Call this function to (re)calibrate the radio, or to switch between
//                  different RF settings.
//     Please note that this function calls mspStartTimer, which clears
//     all callback settings
//
//  Arguments:
//     RF_RXTXPAIR_SETTINGS code* pRF_SETTINGS
//                           RF settings (frequencies, modem settings, etc.)
//
//     RF_RXTXPAIR_CALDATA xdata* pRF_CALDATA
//                           Calibration results
//
//                  word clkFreq
//                           The XOSC clock frequency in kHz.
//
//                  bool calibrate
//                           Calibrate now. *pRF_CALDATA is written to when calibrate = TRUE,
//                           and read from otherwise. Use FALSE if *pRF_CALDATA is valid.
//
//          Note: This function also enables timeouts, using timer 3.
//-------------------------------------------------------------------------
void mspSetupRF (RF_RXTXPAIR_SETTINGS code *pRF_SETTINGS, RF_RXTXPAIR_CALDATA
xdata *pRF_CALDATA, word clkFreq, bool calibrate) {

          // Default synchronization settings
   RF_SET_PREAMBLE_COUNT(MSP_PREAMBLE_SENSE_BITS);
   RF_SET_SYNC_BYTE(RF_SUITABLE_SYNC_BYTE);
```

```
        // Update the internal pointers
        mspIntData.pRF_SETTINGS = pRF_SETTINGS;
        mspIntData.pRF_CALDATA = pRF_CALDATA;

        // Calibrate with the new/current settings
        if (calibrate) {
                halRFCalib (mspIntData.pRF_SETTINGS, mspIntData.pRF_CALDATA);
        }

        // Make sure that we're in IDLE mode
        mspIntData.mode = MSP_IDLE_MODE;

        // Enable the timeouts

        mspStartTimer(clkFreq);
   MSP_INIT_TIMEOUTS();


} // mspSetupRF
```

11.7 MSP Status

```
/*****************************************************************************
 *                                          *
 *     **********                             *
 *    ************                              *
 *    ***      ***                            *
 *    ***  +++  ***                             *
 *    ***  ++  ***                              *
 *    ***  +                 CHIPCON CC1010              *
 *    ***  ++  ***              CUL - msp                 *
 *    ***  +++  ***                               *
 *    ***     ***                               *
 *     ***********                               *
 *      *********                                *
 *                                          *
*****************************************************************************/

#include <chipcon/mspInternal.h>

//---------------------------------------------------------------------------
//  byte mspStatus (void)
//
//  Description:
//     Returns the status of the MSP.
//
//  Return value:
//                  byte
//                          MSP_IDLE_MODE =  Ready to transmit or receive
//                          MSP_TX_MODE =    Transmitting a packet
//                          MSP_TXACK_MODE = Waiting for the ack
//                          MSP_RX_MODE =    Waiting for or receiving a packet
```

```
//                              MSP_RXACK_MODE = Transmitting the ack
//-------------------------------------------------------------------------
byte mspStatus (void) {
        return (mspIntData.mode);
} // mspStatus
```

11.8 CSMA - Rutiner

```
/*****************************************************************************
 *                                          *
 *     **********                             *
 *    ************                              *
 *    ***      ***                            *
 *    ***  +++  ***                              *
 *    ***  + +  ***                             *
 *    ***  +                CHIPCON CC1010              *
 *    ***  + +  ***           HAL - RFReadRSSI              *
 *    ***  +++  ***                              *
 *    ***     ***                              *
 *    ***********                                *
 *     *********                                *
 *                                          *
 *****************************************************************************
#include <chipcon/hal.h>
#include <stdio.h>
//  halRFReadRSSI(...)
//
//  Description:
//     This function activates RSSI output on the AD2 pinn, activates the ADC
//     for channel 2 reads, the RSSI output voltage and converts it to an
//     approximation of the incoming signal's strength in dBm (range is appr.
//     -110 to -50 dBm.) The value obtained is approximate and most valid for
//     a signal @ 600 MHz. The accuracy should be no worse than +/- 10 dBm at
//     any frequency.
//     An RSSI filter circuit of one capacitor and one resistor is required to
//     be connected to the AD2 pin externally (see datasheet for details).
//     This function disrupts user ADC operation and does not restore the ADC
//     to its former state.
//     The function can be used as a carrier sense if the returned value is
//     compared to some relatively low threshold.
//
//  Arguments:
//
//  Return value:
//     char
//        The approximate signal strength of the incoming signal in dBm.
//-------------------------------------------------------------------------
char halRFReadRSSI() {
   byte val;
   FREND|=0x02;              // Activate RSSI output
   ADCON2=0x02;               // ADC divider i 1/48 (optimal for 11.0592 MHz
                     // CLK), Interrupt not enabled.
```

```
    ADCON=0x0A;                 // ADC power on, single-conversion mode,
                                // input from AD2 pin
    ADCON |= 0x04;              // Start conversion
    while (ADCON&0x04);         // Wait until sample is available

    val=ADC_GET_SAMPLE_8BIT();  // Get 8 MSB of sample

    ADCON=0x82;                 // Power down ADC

    val = -50-((val*55)>>8);
    return val;
}
```