Antenne optimering på NFC kortlæser

Project Report

Rasmus Mogensens Diplom Bachelor Projekt

Aalborg University
Electronics and IT

Here you can write something about which tools and software you have used for typesetting the document, running simulations and creating figures. If you do not know what to write, either leave this page blank or have a look at the colophon in some of your books.

Elektronik og IT
Aalborg Universitet
http://www.aau.dk

AALBORG UNIVERSITET
STUDENTERRAPPORT

Titel:
Antenne optimering

Tema:
NFC kommunkation

Projektperiode:
Forårssemesteret 2017

Projektgruppe:
Diplom Bachelor projekt

Deltager(e):
Rasmus Mogensen

Vejleder(e):
Hans Ebert

Oplagstal: 1

Sidetal: 40

Afleveringsdato:
14. april 2017

Abstract:

Conlan, et Aalborg baseret firma, der levere adgangskontrolsystemer til forskellige kunder rundt omkring i Danmark og Europa. Conlan har taget kontakt til Aalborg Universitet, med et problem omhandlende et af deres kortlæser produkter. Produktet har problemer med at læse adgangskort der anvender Mifare og DESFire protokollerne, selvom kortet er placeret helt op i mod antennen. Det formodes at være et problem med antennens matching til sende og modtage chippen som er en MFRC531 fra NXP. Det bliver undersøgt hvordan matchingen skal udregnes i forhold til applikationnoten, som leveres af NXP. Efter megen undersøgelse, finder man ud af at det ikke er muligt at se hvor godt antennen performer i forhold til bitfejlsandsynlighed. Der kommes frem til en løsning med en testplatform, som har til formål at finde bitfejlsandsynligheden. Det konkluders at det er muligt at lave en testplatfrom, som udregner bitfejlsandsynligheden.

Contents

Preface

Rappoten her kommer til at undersøge, hvilke muligheder der er for at optimere antennematching kredsløbet på en NFC kortlæser. Projektet er Rasmus Mogensens aflsuttende projekt på Aalborg Universitet. Aalborg University, April 14, 2017

Rasmus Mogensen
<rmoge12@student.aau.dk>

Chapter 1

Indledning

I dagens samfund er der et større og større krav om at kunne tracke og vide hvor mennersker befinder sig. Dette kan gøres på mange forskellige måder ved hjælp af GPS, RFID eller en af de mange andre muligheder som de forskellige gadgets som mange i dag har på sig hver dag.
Men som teknologien bliver bedre og bedre, opstår der andre udfordringer f.eks. i forhold til persondata og hvornår eller hvor lang tid man som person er tryg ved at bliver overvåget eller tracket. Fitnesscentre, gratis koncerter eller andre begivenheder med større menneskemængder kan have gavn af at holde øje med hvordan menneskemængden bevæger sig inde på koncertpladsen eller i fitnesslokalerne. Til koncerter kan det hjælpe med at øge sikkerheden så der ikke sker ulykker i stil med den der skete på Roskilde festival i år 2000 [7], dette sker ved at holde øje med hvor mange mennesker der er indenfor et bestemt område af koncertpladsen. Når koncerten så er overstået eller man går ud af fitnesscenteret er det et klart ønske om at man som person ikke vil overvåges eller trackes mere, da dette ikke har et formål i forhold til koncertsikkerhed eller fitnesscenterets analyse af folks fitness vaner. Dette ønske til ikke at blive overvåget hele tiden, er blevet diskuteret livligt i medierne efter Edward Snowden [1] lækkede oplysninger omkring masseovervågning udført af de amerikanske efterretningstjenester.
Både RFID og GPS teknologi kan bruges til at overvåge og tracke folkemængder med, og hver af de 2 teknologier har deres fordele og ulemper i forskellige situationer. Begge teknologier er trådløse og er derfor afhængige af antenner for at kunne overfører data.
En antenne kan i teroien bare være en ganske almindelig ledning, men den vil så have problemer med støj fra andre kilder. Så for at have en god antenne skal man kunne tune den ind til en bestemt frekvens for at undgå støj, men samtidig skal den være rigtigt god til at kunne forstærke den ønskede signalet på de ønskede frekvenser. Ved at lave forstærkning i antennen, kan man samtidig også nøjes med et mindre signal for at sende med samme styrke og derved opnå en energibespar-

else som i sidste ende giver bedre batterilevetid eller bare et mere grønt produkt. Strømforbruget er dog ikke altid vigtigt i forhold til et produkt. Adgangskortlæsere f.eks. er som regel altid koblet til en fast el-forsyning, så der er andre kriterier som er vigtigere. Ting som læseafstand og hvor hurtigt adgangskortet bliver læst er kriterier som giver en bedre brugeroplevelse, det skal være hurtigt og bekvemt at komme ind og ud af bygningen, istedet for at skulle vente 5 sekunder eller mere på at ens kort bliver læst. Det er også vigtigt at kortet bliver læst hvergang, så det ikke giver fejl uden grund.
Lige præcis dette er grundlaget for dette projekt, der bliver udført i samarbejde med Conlan. Conlan er et lille Aalborg firma, der lever af at lave adgangskontrol til bygninger. Disse produkter spænder over en lang række af forskellige metoder til at sikre det kun er autoriserede der har adgang til bygninger og pladser. Der anvendes kodelåse, adgangskortlæsere og fingeraftrykslæsere.
Conlan blev stiftet i 1991 af studerende fra Aalborg universitet og de er i dag 10 ansatte. Firmaet er lokaliseret på havnen i Aalborg, hvor al udvikling og test foregår, der er også begrænsede produktions muligheder. Grundet øget efterspørgsel er de dog blevet nødt til at få visse ting lavet eksternt af andre firmaer for at kunne følge med, så nu bliver visse print bestykket og loddet eksternt.
Conlan har i gennem en længere periode haft problemer med et produkt som indeholder en kortlæser, hvor læseafstanden til kortet er for kort. Dette giver så problemer med at kortene ikke bliver læst hver eneste gang og giver derfor en dårlig brugeroplevelse, ved at brugeren skal stå og fumle med kortet flere gange for at få lov til at komme ind.

1.1 Projektbeskrivelse

Fordi Conlan har haft disse problemer med antennen og ikke har capacitet og know-how nok til at løse problemet internt, har de bedt Aalborg Universite om hjælp. Det er så endt ud i dette projekt, som vil bliver beskrevet herunder.
Conlan vil gerne have lavet en værktøjskasse, til at kunne løse eventuelle fremtidige problemer med antenne og antennekredsløb. Derudover ønskes det at det nuværende problem med antennen eller antennekredsløb bliver løst. Samtidig ønskes der at antennen bliver dimensioneret så der opnåes en passende læseafstand mellem kort og læser.
Det produkt der er problemer med er CM3001, det er en kortlæser beregnet til adgangkontrol til døre. Produktet anvender en NXP chip (MFRC531) som styre kodning og dekodning af Mifare og DESFire protokollerene.

1.2 Mifare og DESFire

MIFARE dækker over en række teknologier som NXP har udviklet og ejer rettighederne til. Teknologien er baseret på ISO/IEC 14443 standarden som dækker over standarderne for 13.56 MHz kort og læsere.
MIFARE giver mulighed for at lagre data i kortene, så det er muligt at anvende dem til forskellige formål som ID-kort og billetter til koncerter eller offentligtransport mm. Der er forskellige mængder af data tilgængelig i kortene varierende fra 224 bytes i MIFARE mini til 3440 bytes i MIFARE 4K. Disse data er delt op i sektore hvor dataene så er beskyttet er 2 nøgler, som igen giver mulighed for at tildele de sektorene en specifik funktion så som om de kun kan skrives, læses f.eks.
DESFire er en del af MIRFARE linien og DES henviser til krypterings standarden Data Encryption Standard som bliver forkortet DES. DES blev udviklet af IBM i 1975 og har været i brugt i mange forskellige systemer siden, bla. har den været brugt det danske Dankort system i form af Triple DES implementeringen som skulle være mere sikker. Denne kryptering metode er dog ved at bliver erstattet af AES, som mindst er kryperet med 128 bit, da DES med en "simpel" 56 bit kryptering i dag er mulig at bryde på under 24 timer.

1.3 System beskrivelse

Systemet hvis man fokusere på kortlæseren og kortet, er ret simplet. Kortlæseren er, meget simplificeret, opbygget af en processor som har den overordnede kontrol af systemet. Det er den som modtager dataen fra kortet efter det er blevet dekodet, og ser på om det kort har adgang til hvad end kortlæseren sidder på og blokere adgang til. Kommunikationen med kortet bliver håndteret af en MFRC531 chip fra NXP, den er placeret før antennen og styrer alle signaler til og fra antennen.
Et blokdiagram af systemet ser ud som følgende:

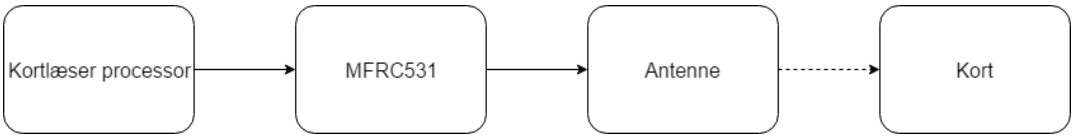


Figure 1.1: Blokdiagram over kortlæseren og hvordan antenne, MFRC531, antenne og kort sidder i forhold til hinanden.

Blokdiagrammet på figur 1.1 er meget simplificeret, det giver et overblik over systemet som bliver arbejdet med.

1.4 Conlans forventninger til projektet

Detter er hvad Conlan forventer at få ud af projektet. Conlans første prioritet med projektet, er at få et stykke værktøj som kan hjælpe dem med at finde en løsning på problemerne med læseafstanden på ders kortlæser.

- **Læseafstand:** Læseafstanden mellem læser og kort ønskes at være mellem 2 og 10 cm. Dette krav er vigtigt for hvis læseafstanden er for lille, vil kortet ikke kunne læses overhovedet. Er den derimod for stor så vil det gå hen og blive et sikkerhedsproblem at kortet kan bliver læst på uhensigtsmæssige tidspunkter, evt. ved en person går forbi med et kort i lommen.

- **Lille fejlrate for kortlæsning:** Det er vigtigt at kortet bliver læst hver gang det bliver holdt op til læseren. Hvis kortet ikke bliver læst kommer der ikke en melding tilbage til brugeren om kortet er godtaget eller afvist, derfor er det vigtigt at der er en succesrate på læsningen af kort på 95%.

- **"Toolbox" til Conlan:** Conlan vil gerne at der i forbindelse med dette projekt bliver udarbejdet en "toolbox", så de i fremtiden har mulighed for at fejlfinde og løse mulige problemer med antennen i fremtiden. Denne "toolbox" kan bestå af et stykke software, et excelark eller lign. som giver dem mulighed for at kunne taste forskellige parametre ind for antennen og så skal de have komponentværdier ud så antennen bliver optimeret.

Chapter 2

Måling af antenne

Antennen som Conlan har problemer med er en loop antenne der er lagt på printet som en del af printets layout, og da Conlan ikke har haft mulighed for at måle antennen og kender derfor ikke dens egenskaber. Så før det er muligt at forbedre antennen er det nødvendigt at måle den.

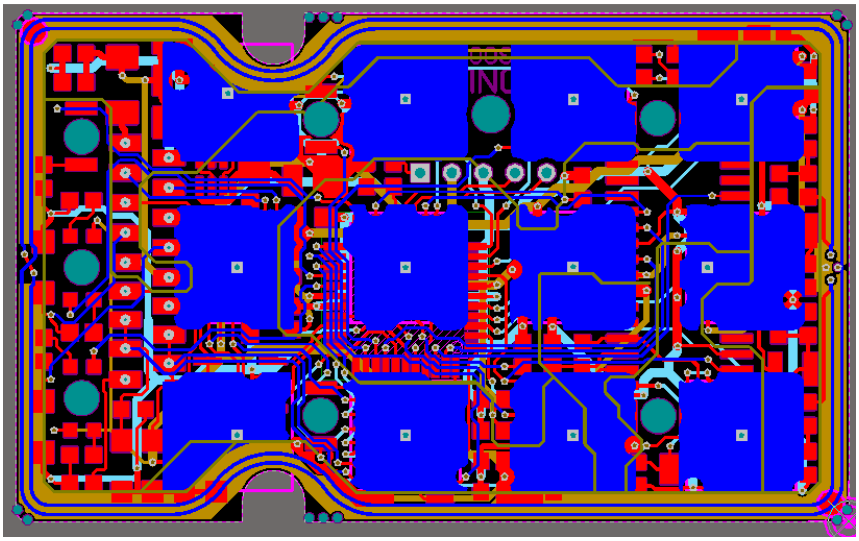
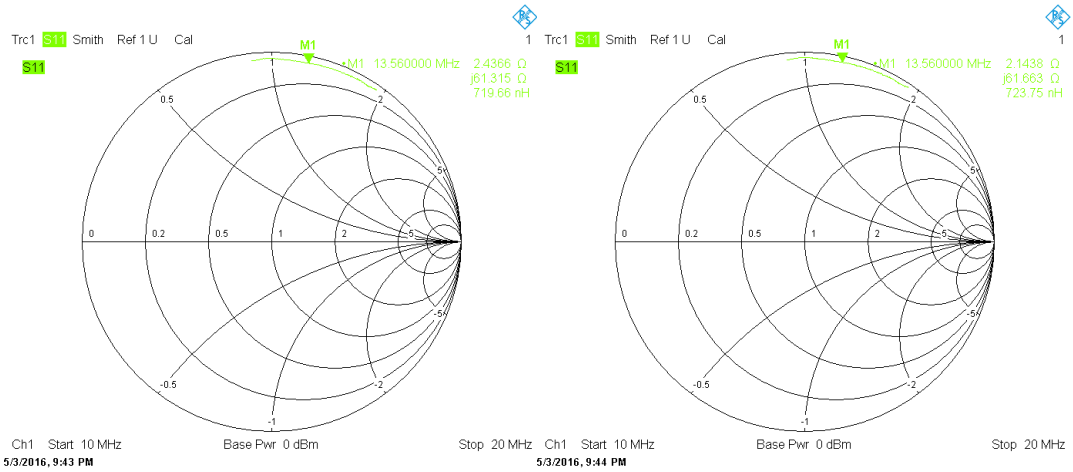


Figure 2.1: Print-layouttet for CM3001, antennen der skal forbedres er de 2 ringe der går rundt på kanten af printet.

Antennen som skal udmåles er antennen for Conlans produkt CM3100 og an-

tennen kan ses på figur 2.1, antennen er de 2 printbaner angivet med blå der løber rundt på kanten af printet.
Det er disse printbaner det er vigtigt at vide noget om for at vide hvordan antennen er og hvor god den er. For at finde ud af hvor god antennen er, er det nødvendigt at finde ud af antennens Q, induktans samt den ohmske modstand. Med disse værdier er det muligt at udregne det matchingkredsløb som skal sidde foran antennen, dette kredsløb gør at antennen kan omdanne den maksimale effekt om til radiobølger. Målingen af antennen foregår ved at isolere antennen fra resten af elektronikken, dette sker ved at lodde de 2 modstande af som forbinder antennen til resten af printet. Med antennen isoleret fra resten af elektronikken, er det nu muligt at måle antennen med en VNA (Vector Network Analyzer). Med en VNA er det muligt at se hvordan antennen den yder ved en forudbestemt frekvens, i dette tilfælde 13.56 MHz. Som målingerne viser på 2.2a og 2.2b, så giver VNA'ens



(a) Måling af antenne på print med komponenter

(b) Måling af antenne på print uden komponenter.

Figure 2.2: Måling af antenner på print med og uden komponenter, med antennen koblet af resten af printet.

målinger nu mulighed for at udregne antennens Q værdi. I application noten til MFRC531 chippen som bliver anvendt af Conlan til dette produkt, er det angivet hvordan Q værdien for en antenne bliver udregnet.

$$Q = \frac{2 * \pi * F * L}{R} \tag{2.1}$$

L i udregning 2.1 er antennens selvinduktans og R er den ohmske modstand i kobberbanen som udgår antennen. Begge disse værdier vises på målingerne fra VNA'en og det er disse vi vil udnytte i de følgende beregninger:

Ubestykket print:

$$Q = \frac{2 * \pi * 13.56\,\text{Hz} * 726\,\text{nH}}{2.1438\,\Omega} = 28,73 \tag{2.2}$$

Bestykket print:

$$Q = \frac{2 * \pi * 13.56\,\text{Hz} * 719\,\text{nH}}{2.4366\,\Omega} = 25,14 \tag{2.3}$$

Med disse værdier er det nu muligt at anvende den metode og de ligninger der er præsenteret i application-noten til den chip som Conlan har valgt at bruge, MFRC531 chippen fra NXP.

2.1 Udregninger af komponentværdier

Application-noten til chippen MFRC531 som Conlan anvender, fremviser en metode til at udregne matching kredsløbet til antennen. Det er nødvendigt at lave dette kredsløb så antennen kan udstråle mest mulig energi som radiobølger, hvis ikke kredsløbet er lavet korrekt kan der opstå uønsket fasedrej og er impedansen ikke korrekt vil energien blive afsat som varme i andre komponenter frem for elektrisk energi i antennen.

I application-noten [4] vises der at kredsløbet ønskes at ende med at have 500 Ω og 0° fasedrej, dette ses tydeligt på figur 2.3.

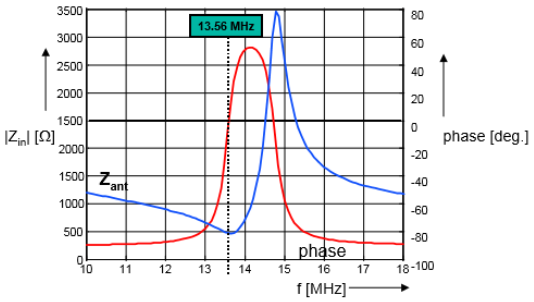
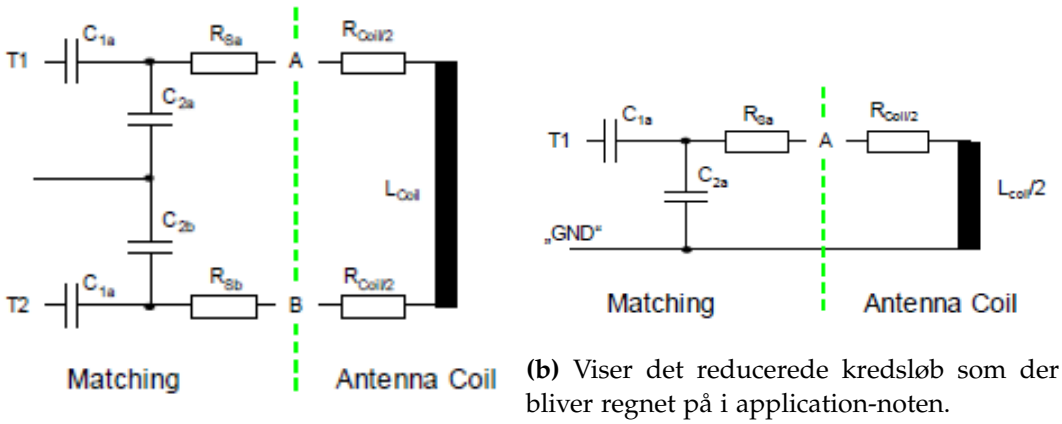


Figure 2.3: Figur fra application-note der viser ønsket fasedrej og impedans for matching kredsløbet.

I application-noten bliver der også sagt hvordan matchingkredsløbet skal opstilles i forhold til antennen, dette vises på figur 2.4a og viser samtidig at udregningerne bliver lavet for den ene side af antennen, da antennen har "virtual ground" i midten. Dette gør at kredsløbet kan reduceres til det som ses på figur 2.4b. I application-noten angiver de også hvordan de forskellige modstande og kondensatorer udregnes i forhold til dette kredsløb, modstandene bliver udregnet med



(a) Viser det fulde kredsløb som er det bliver vist på diagrammet.

(b) Viser det reducerede kredsløb som der bliver regnet på i application-noten.

Figure 2.4: Figurene her er taget fra application-noten til chippen MFRC531 som Conlan anvender.

udregning 2.4 og kondensatorne bliver udregnet med udregning **??** og **??**.

$$R_{Sa} = \frac{\Omega \cdot L}{2 \cdot Q} - \frac{R_{Coil}}{2} \tag{2.4}$$

R_{Sa} er den modstand der skal sættes foran antennen for at mindske antennens Q-værdi så den får en højere båndbredde, L er antennens induktans målt i Henry, Q er den ønskede Q-værdi for antennen (application-noten siger et Q på mindre end 22 er optimalt for højere bit-rates se side 26 i application-note.), R_{Coil} er den ohmske modstand i antennen.
Hvis værdierne fra målingen på figur 2.2a får man så at:

$$R_{Sa} = \frac{\omega \cdot 720\,\text{nH}}{2 \cdot 10} - \frac{2.43\,\Omega}{2} = 1.85\,\Omega \tag{2.5}$$

Med værdierne for R_{Sa} udregnet, kan den næste formel fra application-noten nu anvendes til at finde værdien for C2.

$$C_{2a} = \frac{1}{\omega \cdot \sqrt{(\frac{\omega \cdot L}{1-\frac{R}{Z_a}})^2 - \frac{R^2+\omega^2 \cdot L^2}{1-\frac{R}{Z_a}}} + \frac{\omega^2 \cdot L}{1-\frac{R}{Z_a}}} \tag{2.6}$$

Denne formel var lidt svær at gennemskue, da der er visse ting som ikke er gjort helt klar i application-noten. L er f.eks. kun det halve af induktansen for antennen, da det nu kun er det halve kredsløb der bliver regnet på som set på figur 2.4b. R er nu summen af $\frac{R_{Coil}}{2}$ og R_{Sa}, desuden skal Z_a også kun ses som halvdelen af den samlede impedans for matching kredsløbet. Så Z_a skal være halvdelen af de 500 Ω

som er blevet anbefalet på figur 2.3, altså 250 Ω. Indsættes tallene i udregning 2.6 kommer den til at se ud som dette:

$$C_{2a} = \frac{1}{\omega \cdot \sqrt{(\frac{\omega \cdot 360\text{nH}}{1-\frac{3\Omega}{250\Omega}})^2 - \frac{3\Omega^2+\omega^2 \cdot 360\text{nH}^2}{1-\frac{3\Omega}{250\Omega}}} + \frac{\omega^2 \cdot 360\text{nH}}{1-\frac{3\Omega}{250\Omega}}} = 368\text{pF} \tag{2.7}$$

Værdien for C_{2a} skal så anvendes til at finde ud af hvor stor C_{1a} skal være, dette gøres ved at anvende den følgende udregning:

$$C_{1a} = \frac{R^2 + (\omega \cdot L - \frac{1}{\omega \cdot C_{2a}})^2}{\frac{\omega \cdot L}{C_{2a}} \cdot (\frac{1}{\omega \cdot C_{2a}}) - \frac{R^2}{C_{2a}}} \tag{2.8}$$

Det samme for udregning 2.9 som for udregning 2.6, at L er det halve af den målte værdi og at R er R_{Sa} lagt sammen med $\frac{R_{Coil}}{2}$. Indsættes værdierne i udregningen får vi så:

$$C_{1a} = \frac{3\Omega^2 + (\omega \cdot 360\text{nH} - \frac{1}{\omega \cdot 368\text{pF}})^2}{\frac{\omega \cdot 360\text{nH}}{368\text{pF}} \cdot (\frac{1}{\omega \cdot 368\text{pF}}) - \frac{3\Omega^2}{368\text{pF}}} = 411\text{pF} \tag{2.9}$$

2.2 Måling af bitfejlsandsynlighed

Bitfejlsandsynligheden skal fastslåes ud fra de data som bliver sendt til og modtaget fra adgangskortet, dette kræver noget speciel software i Conlans produkt før disse bitstrømme kan bruges til at fastslå bitfejlsandsynligheden. Conlan har været behjælplige med at få lavet det test software som produktet skal bruge under testen, så de data der bliver sendt til kortet bliver læst tilbage igen med det samme. Dette gør det nemmere og hurtigere at fastslå bitfejlsandsynligheden, da det ikke altid er det samme som bliver sendt og modtaget fra kortet under normal drift.
Fordi det er rå bitsignaler som skal sammenlignes, er der mange forskellige platforme som opstilligen kan laves på, og jeg vil i det følgende gøre rede for valget af platform ved at lave en sammenligning mellem nogen af de logiske valg.

Krav til platformen:

- **Samplerate:** Det er yderst vigtig at sampleraten er høj nok til at data ikke går tabt, hvis data eller bits går tabt vil testen ikke give et reel billed af bitfejlsandsynligheden. Derfor er det vigtigt at sampleraten mindst er dobblet så stor som bitraten. Det er valgt at sampleraten skal være mindst det dobbelte af bitraten for at give lidt overhead til evt. at kunne skrive data ud til en fil eller lign.

- **Mulighed for automatisering:** Før bitfejlsandsynligheden kan findes skal der sendes en betragtelig mængde bits, set i forhold til hvor mange bits der

bliver sendt ved en enkelt læsning eller skrivning af adgangskort. Det gør at testen kan tage lang tid, så automatisering af testen vil gøre det muligt for en medarbejder at arbejde med andre ting mens testen kører. Så det eneste medarbejderen skal gøre er at sætte testen igang også skal testen køre af sig selv til der foreligger et resultat.

- **Overvågning af test:** Når testen så køre, kan det være en fordel hvis medarbejderen har mulighed for at undersøge hvor langt testen er, måske få et foreløblig resultat eller bare være sikker på at alt kører som det skal. Så en eller anden form for fjernovervågning kan gøre testopstillingen nemmere at arbejde med.

- **Samlet løsning:** Af praktiske årsager ville det være en fordel at al dataopsamling og behandling af data bliver foretaget af testopstillingen. Således undgår man at skulle have al den opsamlet data over på en anden computer for at få dem behandlet, så målet er at lave en samlet løsning.

- **Mulighed for udbygning af opstilling:** Dette punkt er mere en fremtidssikring af testopstillingen, da det i fremtiden kan være Conlan skal teste produktet og gerne vil have mere end bare bitfejlsandsynligheden ud. Derfor er det en fordel at testopstillingen har mulighed for at kunne foretage andre og måske mere avancerede tests.

Disse krav gør det muligt at vælge, hvilke platforme der vil være egnede til testopstillingen. Der vil i det følgende blive gjort rede for fordele og ulemper ved nogen af de bedst kendte platforme, samt blive taget stilling til hvilken platform der skal anvendes.

Platforme:

- **Arduino:** er en simpel platform i det der kun er mulighed for at køre c-programmer derpå, så man slipper for den ekstra kompleksitet som en platform med linux f.eks tilbyder. En Arduino tilbyder også i en vis grad mulighed for at udbygge testopstillingen, hvis Conlan har brug for at få nogen andre data ud samtidig med at testen kører. Dog kan det blive et problem hvis der ønskes datalogning med høj samplerate, da arduinoen ikke har ret meget RAM at gøre med. Så skal der logges meget data over længere tid bliver arduino'ens RAM fyldte og skal derfor skrives til SD-kort eller lignende, dette vil skabe en flaskehals da SK-kortet er meget langsommere at skrive til end RAM.

- **Raspberry Pi:** tilbyder det samme som arduinoen, dog med færre ulemper. Raspberry Pi tilbyder en hurtig processor med masser af regnkraft, mulighed for at udbygge testen og masser af RAM til hurtig datalogning. Derud over er

det muligt at køre Linux på Raspberry PI'en, selv med den ekstra kompleksitet som det medføre, giver det ekstra muligheder inden for databehandling og fjernovervågning af testen.

- **FPGA:** tilbyder den højeste samplehastighed af de 3 muligheder, da FPGA er et stykke hardware som bliver programmeret specifikt til opgaven. Dog kan databehandling på selv FPGA'en være besværlig, hvis man ikke vælger at lave en microprocessor til at håndtere det. Ulemper med FPGA'en er prisen, som er den højeste af de 3 muligheder. Der skal laves en microprocessor på FPGA'en eller der skal programmeres en microprocessor som kan stå for databehandling og håndtering af filer.

Ud fra disse 3 platforme, er det valgt at bruge en Raspberry Pi som platform til testopstillingen. Den er valgt ud fra at den formodentlig er rigeligt hurtig til at sample de ønskede signaler (dette bliver senere testet), samt der er rig mulighed for databehandling, datalogning og udbygning af testen med Linux og C-kode på Raspberry Pi'en.

2.2.1 Test af samplerate

Fordi signalet der bliver sendt ud på MFOUT benet er Manchester kodet, er det nødvendigt at dekode det før bitstrømmen kan sammenlignes. Dette sker ved hjælp af et c-program, som venter på at signalet skifter niveau. Planen er at programmet skal kunne afvikles hurtigt nok til at det er muligt niveauet kan blive læst, gemt i et array og så skal der stadig være tid til at vente indtil næste niveauskifte. Der bliver lavet en opstilling med det formål at undersøge hvordan hurtig bitstrømmen kan være, før Raspberry Pi'en ikke kan følge med mere og der bliver mistet bits. På Raspberry'en vil dette C-program blive kørt:

```
1 #include <bcm2835.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define Pin_in RPI_BPLUS_GPIO_J8_10
5 #define Pin_out RPI_BPLUS_GPIO_J8_13
6 uint8_t level_in;
7
8 int main()
9 {
10 if(!bcm2835_init()) //GPIO init
11   return 1;
12 bcm2835_gpio_fsel(Pin_in, BCM2835_GPIO_FSEL_INPT); //Set pin 10
     on J8 to input
13 bcm2835_gpio_fsel(Pin_out, BCM2835_GPIO_FSEL_OUTP); // set pin 13
     on J8 to output
14 bcm2835_delay(500);                                  //delay to
     ensure setup is finished
```

```
15 level_in = bcm2835_gpio_lev(Pin_in);                    //read
       digital level on input pin
16
17 while(1)
18 {
19   if(bcm2835_gpio_lev(Pin_in) != level_in){
20     level_in = bcm2835_gpio_lev(Pin_in);
21       if(level_in==1)
22       bcm2835_gpio_write(Pin_out, HIGH);
23       else
24       bcm2835_gpio_write(Pin_out, LOW);
25   }
26 }
27 bcm2835_close();
28 return(0);
29 }
```

Test_results/gpio_manchester.c

Programmet vil blive kørt på en Raspberry Pi model B+, som har en single core 700 MHz processor [6] og testopstillingen bliver vist på figur 2.5.

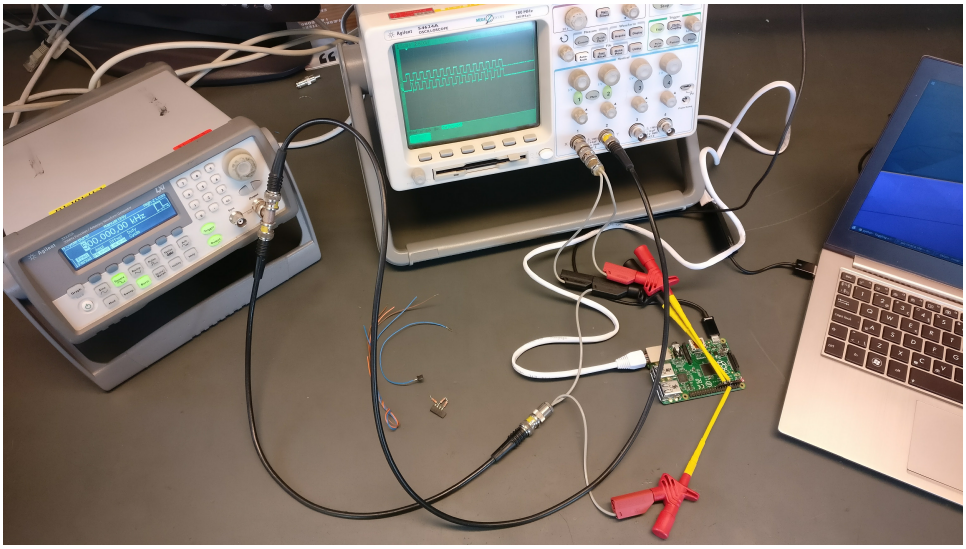
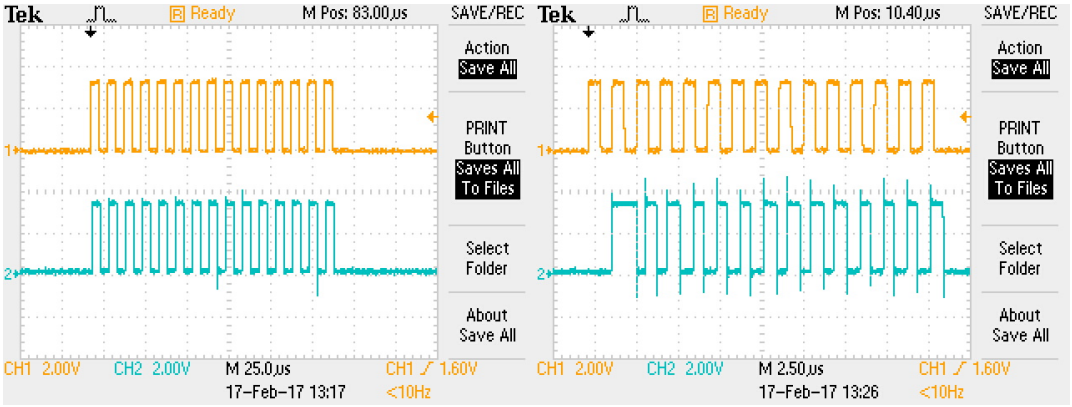


Figure 2.5: Testopstilling der måler Raspberry Pi GPIO hastighed.

Funktionsgeneratoren sender et signal på 15 firkant-pulser over til Raspberry'en som så sender signalet ud på en udgang osciloscopet måler. Frekvensen af firkantet-signalet bliver så øget indtil det ses på osciloscopet at der mangler en eller flere af de 15 pulser.



(a) 100 kHz signal på Raspberry Pi model B+ **(b)** 700 kHz signal på Raspberry Pi model B+

Figure 2.6: Osciloscope billeder der viser firkant signalet ved henholdsvis 100 kHz og 700 kHz målt på en Raspberry model B+. Det orange signal er det som funktionsgeneratoren har genereret, mens det blå signal er signalet der er kørt igennem Raspberry'en

Som det ses på figur 2.6b mangler den første bit, årsagen til dette kan være linux går ind og tager for meget CPU tid i forhold til hvor hurtig der skal samples ved 700 kHz. For at se om det kan lade sig gøre at sample hurtigere med en Raspberry PI, bliver den samme test udført på en Raspberry Pi model 3. Denne model har en 1 GHz quadcore processor [5]. Testen viste at på en Raspberry Pi model 3 er det muligt at sample et signal på 2 MHz som det ses på figur 2.7.

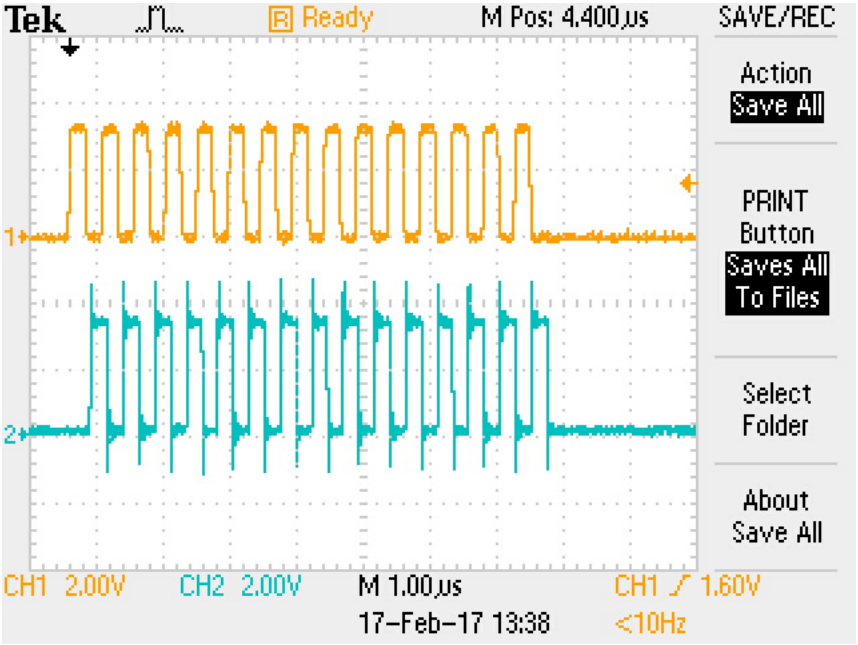


Figure 2.7: 2 MHz signal på en Raspberry Pi model 3.

Dette bekræfter hermed at Raspberry'en er hurtig nok til at kunne sample de signaler, der bliver sendt til og fra adgangskortene. Der blev dog oplevet nogen gange under testen af Raspberry'en at signalet, nogen gange ikke blev samplet ordentlig. Dette kan ses på figur 2.8.

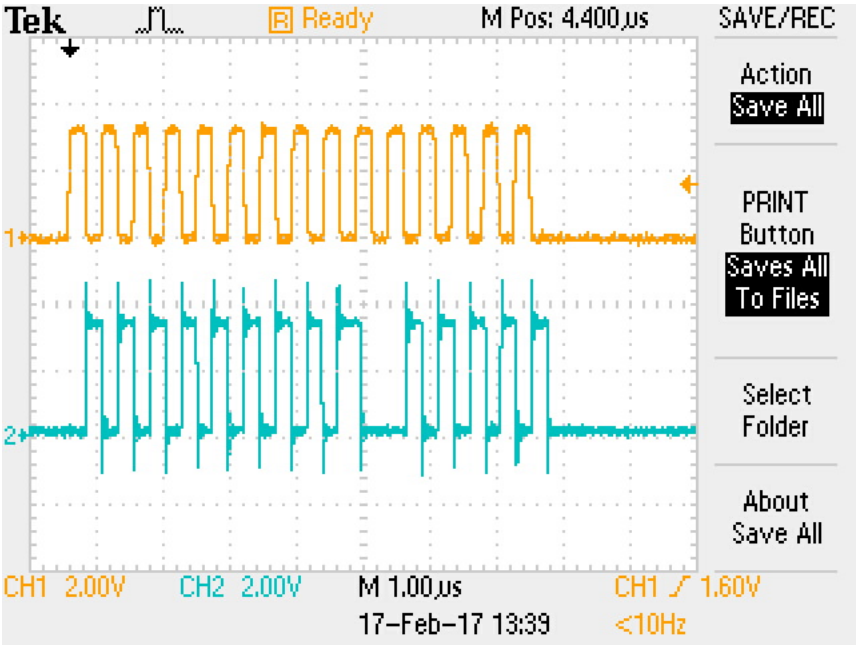


Figure 2.8: 2 MHz signal på en Raspberry Pi model 3, hvor der er et eller andet som forstyrre signalet.

Det formodes at det er linux systemet som gør krav på noget processor tid, lige som signalet skal samples. Derfor skal det undersøges om det er muligt at isolere en CPU-kerne og dedikere den til C-programmet, således linux ikke kan sende interrupts. Dette skal undersøges fordi der over en test på 2-3 timer, kan forekomme adskellige interrupts og dette vil så inducere fejl så vi ikke får det korrekte billed af bitfejlsandsynligheden. Der blev undersøgt adskellige forskellige muligheder for hvad det kunne være som afbrød C-programmet og ud fra de forskellige test, blev det konkluderet at det må være skeduleringen i Linux som giver problemet. Dette kan løses ved at køre en anden linux-distribution som f.eks. Realtime Linux eller andre lignende distributioner, som er lavet til realtids applikationer. En anden løsning er at anvende en seperart processor/microchip som tager sig af alt det tidskritiske og så sender data'en til at blive behandlet på raspberry'en. Løsningen som bliver valgt til dette projekt, er at anvende en seperart processor til at håndtere alt tidskritisk såsom sampling af data.

2.2.2 MiFare modulation og kodning

Inden opstillingen skulle fysisk laves og programmeres, blev der brugt lang tid på at undersøge hvor der skal måles for at få de bitstrømme som er nødvendige. Der er også blevet brugt tid på at forstå præcis hvordan signalerne er moduleret og kodet, da dette er vigtigt i forhold til at kunne forstå hvordan signalet skal håndteres i det program som bliver skrevet til det forskellige hardware som bliver anvendt. For at bedre kunne forstå hvordan signalet der bliver sendt er kodet og moduleret, blev der brugt tid på at undersøge og forstå hvordan MiFare virker. MiFare anvender en række koder til at styre om et kort skal vågne op eller gå i dvale, og signalet for at kortet skal vågne op er beskrevet som WUBA i ISO/IEC 14443-3 på side 14 [3]. Dette er forholdsvis simplet signal at starte med at dekode fordi signalet altid er det samme, nemlig en hexadecimal værdi på 52. Dette kan ses på figur 2.9. ISO standarden beskriver også at fordi kortlæseren er sat op til at overføre 106 kbit/sek, skal signalet være 100% ASK moduleret og køre modified miller kodning.

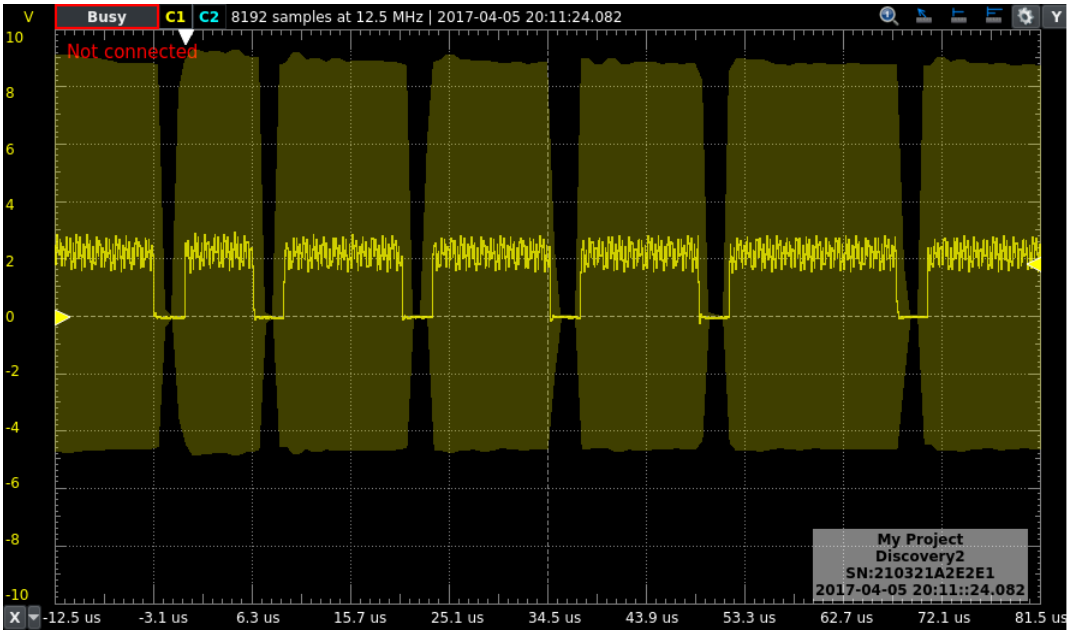


Figure 2.9: WUBA signalet målt på osciloscope på TX1 benet af MFRC531.

WUBA signalet som er afbilledet på 2.9 er moduleret med en 100 % ASK modulation, hvilket betyder at bærebølgen helt forsvinder når der skal afbilledes en bit. Signalet er også modified miller enkodet, det er en enkodning som deler en bit op i fire dele for at kunne afbillede 0 og 1.

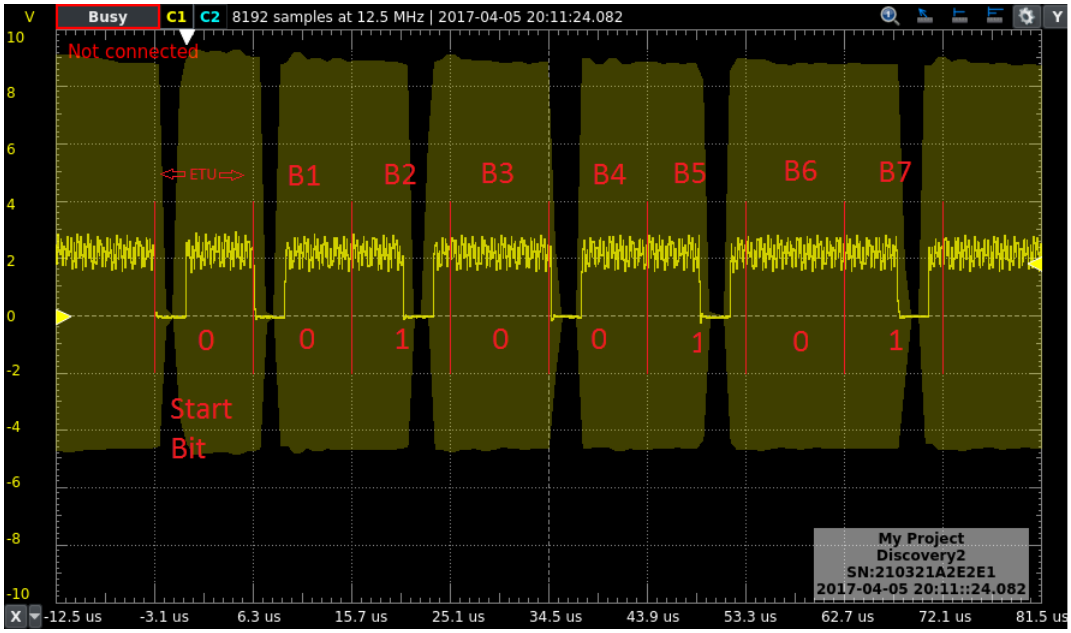


Figure 2.10: WUBA signalet forklaret og dekodet.

Figur 2.10 viser hvordan et modified miller kodet signal, skal dekodes. En bit bliver bestemt af hvordan signalet er mellem 2 røde streger på figur 2.10, denne tid kaldes for en ETU. En ETU skal i følge ISO/IEC 14443-2 side 14 [2] være delt op i fire lige store dele, et binært 1 bliver vist som en ETU hvor den 3. af de 4 dele som en ETU består af skal være lav. Et binært 0 kan være afbilledet på 2 måder, hvis det kommer efter et binært 1 sker der ingen ændring af signalet i løbet af den ETU som det binære 0 afbilledes i. Kommer der 2 binære 0'er efter hinanden, bliver det første binære 0 afbilledet som beskrevet tidligere, det efterfølgende 0 skal så afbilledes så den første 1/4 del af ETU'en er lav. Datavinduet bliver altid startet af et binært 0 og afsluttet af 2 binære 0'er og mellem 2 datavinduer skal der være mellem 10 eller 11 ETU. Ser vi bort fra start bit i bitstrømmen på figur 2.10, er det muligt at dekode signalet så bitstrømmen bliver 0100101. ISO standarden foreskriver at det er det mindst betydende bit som bliver sendt først, så når tallet omskrives til hexadecimal bliver det til 52. WUBA er en kommando som bliver brugt til at vågne et kort fra dvaletilstand og få kortet ud af HALT-state, som er en pause tilstand kortet kan sættes i hvor den kun lytter efter WUBA for at kommer igang igen. Når kortet har modtaget en WUBA kommando, skal kortet så svare med en ATQA kommando. Denne kommando bruges til at opfange eventuelle bitkollisioner, hvis flere kort svarer på WUBA kommandoen.

b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1
RFU				Proprietary coding				UID size bit frame		RFU	Bit frame anticollision				

Figure 2.11: Billed af hvordan ATQA bitvinduet er formateret, b1 er LSB og b16 er MSB. RFU bits skal være 0, og er b8 og b7 begge 1 betegnes de også som RFU og ATQA vinduet er dermed ugyldigt.

Figur 2.11 viser hvad ATQA sender fra kortet til kortlæseren, det er det mindst betydende bit som bliver sendt først. Bit 7 og 8 viser som kortlæseren hvor langt kortets UID er, og bestemmer hvordan kortlæseren reagere herefter. Kortets UID kan variere mellem single og triple strørrelse, og fordi der kun bliver sendt vinduer af en hvis størrelse hver gang kan det være nødvendigt at sende UID'et over flere vinduer derfor skal kortlæseren vide hvor stort det er. UID størrelsen er beskrevet i ISO/IEC 14443-3 tabel 10 [3]. Single UID størrelse er 4 bytes, double er 7 bytes og tiple er 10 byts. Figur 12 i samme ISO standard viser også hvordan UID'et bliver delt op i kaskade-niveauer så det kan sendes til kortlæseren. Data som er lagt ned i kortet datasektore er krypteret og fordi det er en kryptering algotritme som NXP har udvilket og meget gerne vil beskytte, er der ikke mulighed for at dekryptere data. Datakryptering foregår med 2 forskellige krypteringsnøgler, så data der bliver sendt fra kortlæseren til kortet vil ikke se ud på samme måde hvis det bliver sendt fra kortet til kortlæseren fordi kryperingsnøglerne ikke er de samme. På grund af det ikke er muligt at dekryptere det data, kommer testen på bitfejlsandsynligheden til at ske udelukkende med UID data, som ikke er krypteret.

2.2.3 Udvikling af testplatform

Fordi der er problemer med at håndtere realtids signaler på en Raspberry Pi, skal der udvælges en platform til at håndtere sampling af signalet der kommer fra kortlæseren. Kravene i forhold til samplerate er lidt mere lempelige nu hvor sampling og databehandling skal fortages af 2 forskellige platforme. De samme tests for samplerate køres for at sikre at sampleraten er hurtig nok på den valgte mikrochip / processor for at sikre at processoren der bliver valgt også kan håndtere signalet. En STM32F429I-DISCO er blevet valgt efter at have undersøgt om de forskellige krav for samplingrate og funktionalitet var overholdt. Den har rig mulighed for at udbygge testen så den er mere omfattende, masser af GPIO, timere, interrupts og ikke mindst hukommelse til at gemme data inden der er mulighed for at sende det videre til Rasberry'en. Det er samtidig et meget veldokumenteret board blandt hobbyister, så der er rig mulighed for at finde hjælp til at programmere det i tilfælde af der skulle opstå problemer med c-programmeringen. Selve testopstillingen er vist som et blokdiagram på figur 2.12, og viser hvordan de forskellige blokke kommer til at sidde i forhold til hinanden.

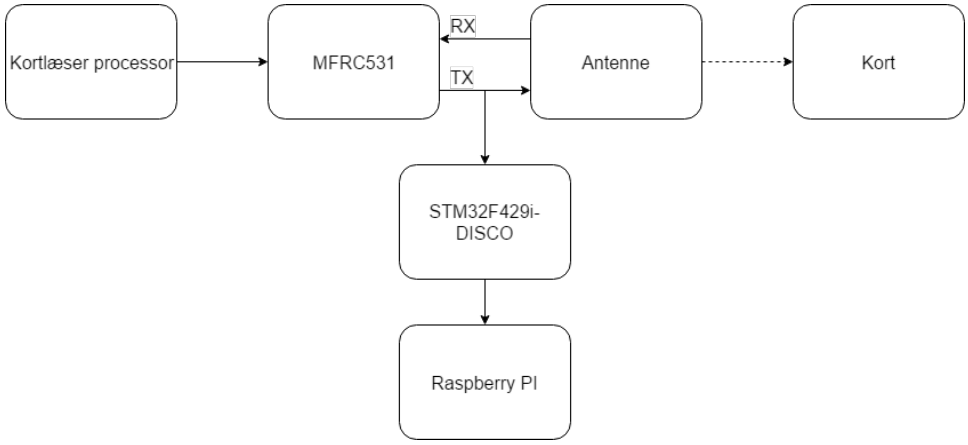


Figure 2.12: Blokdiagram over testopstillingen.

TX på figur 2.12 er styret af en MFRC531 chip. Denne chip håndtere alt MIFare og DesFire kommunikation med kortene, det er også denne chip som driver antennen. MiFare og DESFire er baseret på ISO/IEC 14443 standarden som beskriver i detajler hvordan kommunikationen mellem kort og kortlæser foregår, dette vil også blive gennemgået i næste afsnit af rapporten. Ideen bag at tage signalet på TX benet af MFRC531 chippen er, at det burde være muligt at se al kommunikation mellem kortet og kortlæseren. TX benet sidder på antennen og derfor skulle al kommunikation gerne kunne ses der også.
Det rå bit signal fra MFRC531 chippen er ikke mulig at sample på en STM32F429I, da signalets amplitude er for høj og vil med sikkerhed brænde porten af som det bliver sendt ind på. Det rå bit signal kan ses på figur 2.9, de gule skygger som ses på scopebilledet er bærebølgen på 13.56 MHz. Amplituduen er langt over de 5 V som hver indgang på STM32F29I kan klare, derfor er det nødvendigt at demodulere og tilpasse amplituden således det ikke brænder porten af på ARM processoren der sidder på STM32F429I boardet. Bærebølgen er det første problem der skal løses, for at fjerne den er det nødvendigt at anvende et lavpass RC-filter. Heldigvis er bærebølgen på 13.56 MHz meget højere end signalet, som bliver sendt med 106 kHz. Så derfor vil et første ordens RC-filter klare det fint i forhold til at filtrer bærebølgen fra, dog skal det ikke belaste udgangen af MFRC531 chippen. Derfor bliver det valgt at en meget lille kapacitet sammen med en forholdvis stor modstand vil kunne klare det som er ønsket. Kondensatoren bliver valgt til at være 1 pF og modstanden bliver valgt til at være 121 kΩm hvilket kommer til at give en knækfrekvens på:

$$f = \frac{1}{2 \cdot \pi \cdot r \cdot c} = \frac{1}{2 \cdot \pi \cdot 121\,\mathrm{k\Omega} \cdot 1\,\mathrm{pF}} = 1.32\,\mathrm{MHz} \tag{2.10}$$

Sættes dette RC-filter på TX benet af MFRC531 chippen kommer signalet til at se

ud på følgende måde:

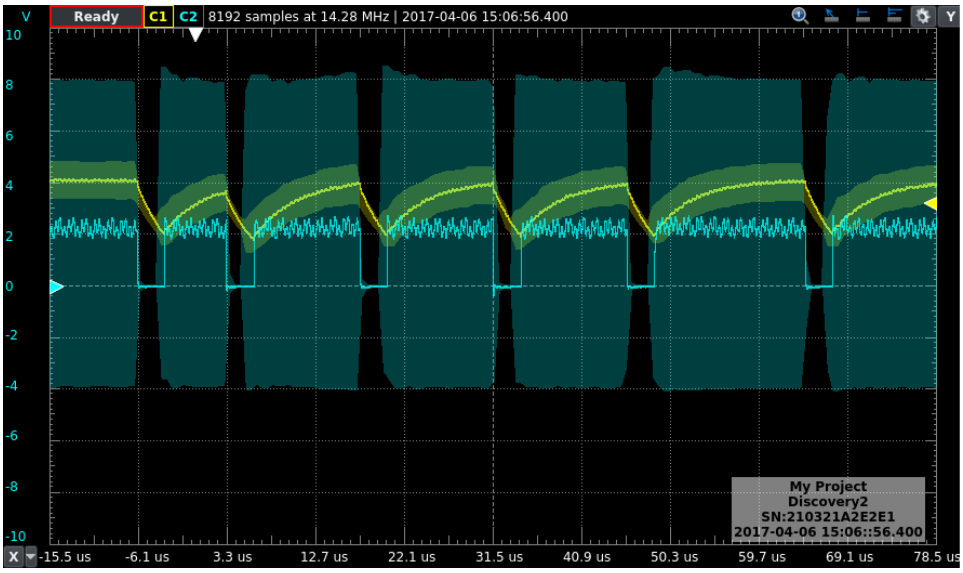


Figure 2.13: Det rå signal fra MFRC531 chippen og signalet efter RC-filteret.

Amplituden af signalet er stadig ikke helt optimalt i forhold til at det skal kunne samples ved hjælp af en inputport på en ARM processor, en lav værdi på en ARM processor er en spænding under 2.5 V, derfor kræves der en komperator for at sikre de rigtige amplitude niveauer.

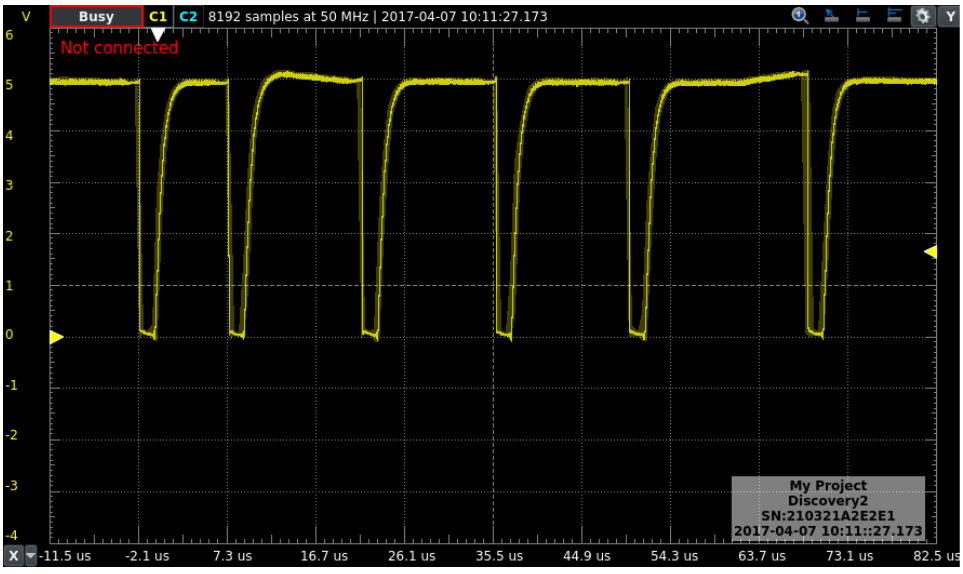


Figure 2.14: Bitsignalet efter komparator.

Signalet er nu kompatibelt med de 5 V tolerante input porte på ARM processoren, dette gør det nu muligt at gå videre med at undersøge hvordan signalet skal samples. Efter som signalet består af 2 forskellige tidsvinduer, en ETU og de 4 dele i en ETU, kan vi sætte timere op således at bits bliver samplet ens hver gang. 106 kbit/sek svare til 106000 ETU/sek, betyder det at timingen på ETU-timeren kommer til at køre med 53 kHz. Grunden til at timeren skal være halvdelen af ETU-frekvensen er at interruptet som timeren generer skal komme i starten af hver ETU og kun i starten af hver ETU, dette svare til at periodetiden for timeren kommer til at være 2 ETU og derfor den halve frekvens. En ETU er delt op i 4 dele, så derfor er det også nødvendigt at have en timer som er 4 gange hurtigere end ETU-timeren, altså 212 kHz. Derudover beskriver ISO/IEC 14443-3 at der skal være mindst 10-11 ETU mellem hvert datavindue, i den periode er det ikke nødvendigt at sample. Den ledige tid vil så blive brugt på at sende data til Raspberry Pi'en, og i stedet for at skulle tjekke om der er nye data hele tiden er det her smart at anvende et hardware interrupt. Dette hardware interrupt kommer til at markere starten af vores bitvindue og kan dermed bruges til at indstille timerne hver gang der kommer et nyt datavindue.

STM32 C kode

C koden for at få alt dette til at virke på en ARM processor som der sidder på en STM32F429I-DISCO, kræve at man skal initialisere alt det perifer hardware som skal bruges. Her menes GPIO-porte, UART, timere og interrupts som der også skal bruges i dette projekt.

```
1 /* Structures for peripheral initialsation */
2     GPIO_InitTypeDef  GPIO_InitStructure;
3     USART_InitTypeDef USART_InitStructure;
4     USART_ClockInitTypeDef USART_ClockInitStructure;
5     NVIC_InitTypeDef NVIC_InitStructure;
6     EXTI_InitTypeDef EXTI_InitStructure;
7     TIM_TimeBaseInitTypeDef timerInitStructure;
```

Her sættes alle strukturene op så de forskellige strukture op fra h-filerne. Disse giver mulighed for at bruge de forud defineret funktionskald til at sætte alting op som det skal være. GPIO er input og output porte, USART er til at sætte serielkommunikation og TIM er til at sætte timere op. NVIC står for Nested Vectored Interrupt Controller og står for interrupt håndteringen og fordi den er meget tæt bundt med CPU'en, giver det interruptene meget lav forsinkelse. EXTI er den externe interrupt kontroller og gør det muligt at anvende de mange input/output pins til interrupts. EXTI multiplexer externe interrupts til NVIC så antallet er interruptpins øges.

```
1  /* Enable SYSCFG clock */
2      RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
3      SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOG,
           EXTI_PinSource5);
4
5  /* GPIO init for CM1200 input Port G5 */
6      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE);
7      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
8      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
9      GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
10     GPIO_InitStructure.GPIO_Speed= GPIO_Speed_100MHz;
11     GPIO_Init(GPIOG, &GPIO_InitStructure);
12
13     /* Configure EXTI Line5 */
14     EXTI_InitStructure.EXTI_Line = EXTI_Line5;
           // PG5 is connected to EXTI9_5
15     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
           // Interrupt mode
16     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
17     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
           // Enable the interrupt
18     EXTI_Init(&EXTI_InitStructure);
           // Initialize EXTI
19
20     /* Enable and set priorities for the EXTI0 in NVIC */
21     NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
                          // Function name for interrupt
22     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =
           0;     // Set priority
23     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
                      // Set sub priority
24     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
                           // Enable the interrupt
25     NVIC_Init(&NVIC_InitStructure);
```

GPIO init viser hvordan GPIO porten sættes op til at være et input, men in intern pull up modstand. RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE) aktivere clocken til GPIO porten som tillader den at fungere. EXTI sætter input til en extern interrupt line med interrupt på en faldende flanke. NVIC sætter interruptet til interrupt rutine EXTI9_5_IRQn med højeste prioritet. Resten af det perifirer hardware omkring CPU'en bliver også initialiseret på samme måde.

```
1 int main(void) {
2     setSysTick();
3     initialize();
4     NVIC_DisableIRQ(TIM2_IRQn);
5     NVIC_DisableIRQ(TIM5_IRQn);
6     i=0;
7     while(1){
8         if(i==32){
9           NVIC_DisableIRQ(TIM2_IRQn);
10          NVIC_DisableIRQ(TIM5_IRQn);
11          for (x = 0; x <=32; x++) {
12
13          USART_SendData(USART1, Data[x]);
14          Delay(1);
15        }
16        NVIC_EnableIRQ(EXTI9_5_IRQn);
17        x=0;
18    }
19 }
20    return 0;
21 }
```

Dette er main funktionen, setSysTick() og initialize() er funktionerne lavet til at sætte processoren og alt perifirer hardware op. Timerene bliver disablet og i variablen bliver sat til 0. Dette sker for at holde styr på timerne, samt hvad i er inden der sker nogen interrupt. Variablen "i" bliver kun talt op efter hvert sample og dette sker i kun i interrupt rutinen. Interrupt rutinerne sker i en seperart c-fil, som skal linkes sammen med main filen når c-koden kompileres. variablen "i" viser hvor mange samples der er blevet taget, 32 samples svarer til 8 bit som er længden af WUBA kommandoen. Sampler kun WUBA kommandoen her, for at bruge det som en guideline om programmet virker efter hensigten. Efter de 32 samples er blevet sendt via UART, enables interruptet på inputet igen og programmet starter forfra.

```
1 void EXTI9_5_IRQHandler(void) {
2     // Make sure the interrupt flag is set for EXTI0
3     if(EXTI_GetITStatus(EXTI_Line5) != RESET){
4         NVIC_DisableIRQ(EXTI9_5_IRQn);
5         NVIC_EnableIRQ(TIM2_IRQn);
6         NVIC_EnableIRQ(TIM5_IRQn);
7         TIM_SetCounter(TIM2,0);
```

```
8           TIM_SetCounter(TIM5,99);
9           i=0;
10 }
11          // Clear the interrupt flag
12          EXTI_ClearITPendingBit(EXTI_Line5);
13 }
```

Dette er interrupt rutinen for det externe interrupt, som reagere på den nedad gående flanke på WUBA signalet fra kortlæseren. Her aktiveres timerne og timer TIM5 sættes til havldelen af maks værdien for timeren, således af samplingen bliver fremskudt for at samplingen ikke sker oven i en stigende eller faldende flanke. Interruptrutinen for TIM2, sætter TIM5 værdien til 99, så ledes den er kalibreret til hver bitvindue.

```
1 void TIM5_IRQHandler(void){
2 if (TIM_GetITStatus(TIM5, TIM_IT_Update) != RESET){
3       Data[i]=GPIO_ReadInputDataBit(GPIOG, GPIO_Pin_5);
4       i++;
5       TIM_ClearITPendingBit(TIM5, TIM_IT_Update);
6    }
7 }
```

Interruptruntinen for TIM5 sampler signalet og putter værdien af samplet ind i arrayet Data på plads i.
Dette er funktionen af c-koden på STM32F429I-DISCO boardet, det næste afsnit skal det testes om koden virker og om det er muligt at dekode signalet fra kortlæseren.

Chapter 3

Funktionalitets test af opstilling

Test af opstilling kommer til at foregå på følgende måde. Kortlæseren bliver koblet på STM32F429I-DISCO boardet, så signalet fra antenne kommer ind på GPIO pin G5. UARTen bliver koblet op til et Analog Discovery 2 US osciloscope, scopet har mulighed for at vise dataen som bliver sendt via UARTen.
Målet med denne test er at se om koden giver 4 værdier per bit, samt se om de bits der bliver sendt via UARTen giver den samme bitstrøm som på figur 2.14. Er den test succesfuld, er det næste mål at se om UIDet fra kortet kan læses. Outputet fra UARTen bliver vist som hex-værdier og hver værdi ser ud som dette: h01 for et 1 og h00 for et nul. Signalet der forventes for en WUBA kommando er:

{h00}{h01}{h01}{h01}-{h00}{h01}{h01}{h01}-{h01}{h01}{h00}{h01}-{h01}{h01}{h01}{h01}-
{h00}{h01}{h01}{h01}-{h01}{h01}{h00}{h01}-{h01}{h01}{h01}{h01}-{h01}{h01}{h01}{h01}.

Det er værd at bemærke her, at når signalet er korrekt dekodet kommer der aldrig 2 samples med {h00} efter hinanden. Sker dette er det en klar indikator på at koden ikke fungere efter hensigten, så dette kan bruges til hurtigt at skimte datastrømmen fra STM32 boardet. Første forsøg på at læse og dekode bitstrømmen ser ud således:

{h00}{h01}{h01}{h01}-{h00}{h01}{h01}{h01}-{h01}{h01}{h00}{h00}-{h01}{h01}{h01}{h01}-
{h00}{h01}{h01}{h01}-{h01}{h01}{h00}{h00}-{h01}{h01}{h01}{h01}-{h01}{h01}{h01}{h01}.

Som det ses på de røde værdier, så kommer der 2 0'er efter hinanden. Dette kan tyde på at timeren som styre sample tidspunkterne kræver lidt justering. Derfor forsøges der med at timerværdien for TIM5, bliver sat til 148 af en maks værdi på 199. Dette rykker tidspunktet for samplingen frem, så det sker tidligere. Signalet med den nye timerværdi ser ud som følgende:

{h00}{h01}{h01}{h01}-{h00}{h01}{h01}{h01}-{h01}{h01}{h00}{h01}-{h01}{h01}{h01}{h01}-

{h00}{h01}{h01}{h01}-{h01}{h01}{h00}{h01}-{h01}{h01}{h01}{h01}-{h01}{h01}{h01}{h01}.

Signalet matcher nu præcis det som forventes, dette gør at testen for at se om det er muligt at læse kortets UID nu udføres. Dog skal der laves et par ændringer i koden før dette kan lade sig gøre. Det første som ændres er i den if-statement som er i main funktionen, istedet for kun 32 samples skal der nu samples væsentligt flere bit for at fange hele UIDet fra kortet, som en start sættes if-statementen til først at blive aktiv efter 100 samples. Derudover skal det delay som kommer efter hver bit der bliver sendt på UARTen fjernes, da det i sidste ende vil koste alt for meget tid at vente 1 µs for hver bit som skal sendes. Med disse rettelse forsøges der nu at fange UID fra kortet.
Signalet som bliver sendt ud på UARTen har nu ændret karakter, så der hovedsagligt kommer værdier af {h00}. Så det er hurtigt at konkludere at signalet ikke bliver dekodet som det skal og en af de 2 ændringer som er blevet foretaget ødelægger dekodningen.
Der forsøges at sende de 100 samples, dog bliver delayet sat ind i if-statementen igen. Dette giver nu ikke flere {h00} efter hinanden, så der skal findes en løsning som giver mulighed for at sende mere data, uden at skulle inkludere et delay efter hver bit sendt. Det er heller ikke optimalt at der kun bliver sendt en bit ad gangen, dette tager meget CPU tid og giver ikke mulighed for at sende nok data til Raspberry Pi'en.
Efter at have undersøgt hvilke muligheder der er for at øge throughputtet på UARTen og undgå delayet i koden, ser det ud til at den bedste løsning vil være at koble UARTen op så den har mulighed for at læse data direkte fra hukommelsen på boardet via DMA. DMA står for Direct Memory Access og giver mulighed for at CPU kan lave andet i mens der bliver læst data fra hukommelse og sendt data via UART. Derudover ser det ud til at give mulighed for at sende mere end et enkelt bit ad gangen, så dette vil også øge throughput gevaldigt. Dog er det blevet opdaget for sent i projektet til at det kan implementeres i tide, så der er mulighed for at test det. Dermed er det heller ikke muligt at test hvordan testplatformen vil opfører sig under en fuld test, altså en test hvor der bliver samlet MegaBytes af data til at fastslå bitfejlsandsynligheden.

3.1 Acceptest

I sectionen 2.2 bliver der opstillet en række krav til testplatformen, som gerne skal opfyldes før det kan kaldes en success. Jeg vil her gøre rede for om kravene er opfyldt eller ej.

- **Samplerate:** Kravet til sampleraten er opfyldt. Det er blevet testet af det er muligt at sample hurtigt nok, til at signalet kan dekodes.
- **Mulighed for automatisering:** Dette krav er delvis opfyldt. Det forskellige hardware som er blevet valgt til projektet, har helt sikkert mulighed for at testen kan køre automatisk når den først er igangsat.
- **Overvågning af test:** Også delvis opfyldt. Som med mulighed for automatisering, så er der helt sikkert også mulighed for at programmere testplatformen således det er muligt at overvåge den.
- **Samlet løsning:** Kravet er delvis opfyldt. Det er delvis opfyldt på den måde at det ikke kræver man manuelt skal flytte data og så behandle dem på en arbejds PC, dog havde det været ideelt hvis alt kunne have kørt på Rasberry PIen. Da dette ikke var muligt, var det nødvendigt at inkludere STM32F429I-DISCO boardet.
- **Mulighed for udbygning af opstilling:** Selvom der kom et ekstra stykke hardware til under tilblivelsen af testplatformen, er der stadig mulighed for at udbygge testen yderligere hvis dette ønskes. Så kravet er opfyldt.

Ingen af kravene er fuldstændigt faldet igennem, så set i det store hele er produktet af dette projekt godkendt men heller ikke mere end det. Der er mange krav som kun er delvis opfyldt, samt det faktum at det ikke er muligt at læse alle de ønskede data fra kortet som testplatformen er udformet på nuværende tidspunkt, gør at projektet ikke kan kaldes en storslået success. Rigtigt meget af fodarbejdet der skal til før projektet kommer til at være en success er blevet gjort, så projektet er næsten i mål.

Chapter 4

Konklusion og perspektivering

I dette kapitel, vil jeg forsøge at konkludere og perspektivere over projektet. Perspektiveringen vil beskrive hvad fremtiden bringer for projektet, samt hvad der er af muligheder for at gøre projektet bedre. Konklusionen vil se tilbage på projektets forløb og kommer frem til en konklusion omkring hvordan projektet er blevet.

4.1 Konklusion

Set i det store hele, så er projektet kommet forholdvis langt med at kunne dekode data fra kortet. Dog mangler der enkelte dele som vil gøre at projektet var kommet noget længere. Var det lykkedes at få læst UID ud fra kortet, havde dette åbnet op for at der kunne køres længervarende test hvor det ville være muligt at fastslå bitfejlsandsynligheden. Med muligheden for at få resultatet af en længerevarende test, havde det også været muligt at se hvordan de forskellige komponentværdier der er udregnet i section 2.1 ville have påvirket bitfejlsandsynligheden. Det kan konkluderes at projektet er meget afhængig af at, en eller to dele af projektet virker. Det faktum at det er muligt at dekode signalet fra kortlæseren, men at UARTen sætter begrænsningen for om det er muligt at læse UID ud fra kortet indikere at meget står og falder med at få en eller to dele af projektet til at virke optimalt.

4.2 Perspektivering

Tager man et kig på hvad fremtiden bringer for dette projekt, så er der nogen ret åbenlyse ting som skal ordnes. DMA UART er nummer 1 på den liste, det ser ud til at kunne være løsningen på problemerne i forhold til at kunne få sendt al det nødvendige date i mellem hver data vindue. Det er også mere effektivt end den nuværende løsning hvor der kun sendes en bit ad gangen, og processoren bruger tid på hver bit. Derudover skal Conlan inkluderes i processen med at udvikle Raspberry Pi koden, således at interfacet til testplatformen bliver præcis som

de ønsker det. Dette er nødvendigt fordi testen formentligt kommer til at indgå i deres arbejdsgang, så det at Conlans ønsker til hvordan der bliver interfacet og arbejdet med testplatformen er meget vigtig.
Der er også rig mulighed for at udvikle testplatformen, så den kan assistere med test af alle Conlans platforme og de kommunikations protokoller som bliver anvendt.

Bibliography

[1] *Edward Snowden wiki*. https://da.wikipedia.org/wiki/Edward_Snowden.

[2] *ISO 14443-2*. Identification cards — Contactless integrated circuit(s) cards - Proximity cards — Part 3: Initialization and anticollision.

[3] *ISO 14443-3*. Identification cards — Contactless integrated circuit(s) cards - Proximity cards — Part 3: Initialization and anticollision.

[4] *MRC531 Application note*. http://www.nxp.com/documents/application_note/AN077925.pdf.

[5] *Raspberry Pi model 3 datasheet*. http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf.

[6] *Raspberry Pi model +datasheet*. http://docs-europe.electrocomponents.com/webdocs/1310/0900766b813109e6.pdf.

[7] *Ulykken under Pearl Jam koncert*. https://da.wikipedia.org/wiki/Roskilde_Festival_2000.

Appendix A

STM32F429I-DISCO c kode

A.1 Main.c

```
1 /*
2 External Interrupt (EXTI) Example
3     - Modified blinky with pushbutton code
4         to use EXTI instead of Polling
5 */
6 #include <stdio.h>
7 #include "stm32f4xx.h"
8 #include "stm32f4xx_exti.h"
9 #include "stm32f4xx_usart.h"
10 #include "stm32f4xx_gpio.h"
11 #include "misc.h"
12 #include "stm32f4xx_conf.h"
13 //#include "common.h"
14 int i;
15 int Data[100];
16
17 int x = 0;
18 volatile uint32_t msTicks;                      /* counts 1ms
    timeTicks       */
19 void SysTick_Handler(void) {
20     msTicks++;
21 }
22
23 //  Delays number of Systicks (happens every 1 ms)
24 static void Delay(__IO uint32_t dlyTicks){
25     uint32_t curTicks = msTicks;
26     while ((msTicks - curTicks) < dlyTicks);
27 }
28
```

```
29 void setSysTick(){
30     // ---------- SysTick timer (1ms) ---------- //
31     if (SysTick_Config(SystemCoreClock / 1000)) {
32         // Capture error
33         while (1){};
34     }
35 }
36
37 void initialize() {
38
39 /* Structures for peripheral initialsation*/
40     GPIO_InitTypeDef  GPIO_InitStructure;
41     USART_InitTypeDef USART_InitStructure;
42     USART_ClockInitTypeDef USART_ClockInitStructure;
43     NVIC_InitTypeDef NVIC_InitStructure;
44     EXTI_InitTypeDef EXTI_InitStructure;
45     TIM_TimeBaseInitTypeDef timerInitStructure;
46
47 /* Enable SYSCFG clock */
48     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
49     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOG, EXTI_PinSource5);
50
51 /* GPIO init for CM1200 Port G5 */
52     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE);
53     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
54     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
55     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
56     GPIO_InitStructure.GPIO_Speed= GPIO_Speed_100MHz;
57     GPIO_Init(GPIOG, &GPIO_InitStructure);
58
59     /* Configure EXTI Line5 */
60     EXTI_InitStructure.EXTI_Line = EXTI_Line5;                  //
           PG5 is connected to EXTI9_5
61     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;         //
           Interrupt mode
62     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
63     EXTI_InitStructure.EXTI_LineCmd = ENABLE;                   //
           Enable the interrupt
64     EXTI_Init(&EXTI_InitStructure);                             //
           Initialize EXTI
65
66     /* Enable and set priorities for the EXTI0 in NVIC */
67     NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
                          // Function name for interrupt
68     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
           // Set priority
```

```
69      NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
            // Set sub priority
70      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
                            // Enable the interrupt
71      NVIC_Init(&NVIC_InitStructure);
72
73 /* Timer for ETU-window interrupt 106kHz */
74      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
75      timerInitStructure.TIM_Prescaler= 1 - 1;
76      timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
77      timerInitStructure.TIM_Period = 790 - 1;
78      timerInitStructure.TIM_ClockDivision = 0;
79      //timerInitStructure.TIM_RepetitionCounter = 0;
80      TIM_TimeBaseInit(TIM2, &timerInitStructure);
81
82      NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
83      NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
84      NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
85      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
86      NVIC_Init(&NVIC_InitStructure);
87      TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
88      TIM_Cmd(TIM2, ENABLE); //timer should be enabled at bitstream
            start
89
90 /* Timer for Sampling-interrupt inside ETU-windows (sampling 4
       times inside ETU) */
91      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);
92      timerInitStructure.TIM_Prescaler= 1 - 1;
93      timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
94      timerInitStructure.TIM_Period = 198 - 1;
95      timerInitStructure.TIM_ClockDivision = 0;
96      //timerInitStructure.TIM_RepetitionCounter = 0;
97      TIM_TimeBaseInit(TIM5, &timerInitStructure);
98
99      NVIC_InitStructure.NVIC_IRQChannel = TIM5_IRQn;
100     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
101     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
102     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
103     NVIC_Init(&NVIC_InitStructure);
104     TIM_ITConfig(TIM5, TIM_IT_Update, ENABLE);
105     TIM_Cmd(TIM5, ENABLE); //timer should be enabled at bitstream
            start
106
107
108 /*UART INIT*/
109 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
110 RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```

```
111 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
112 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
113 GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
114 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
115 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
116 GPIO_Init(GPIOA, &GPIO_InitStructure);
117
118
119 GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
120 GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);
121
122 USART_InitStructure.USART_BaudRate = 115200;
123 USART_InitStructure.USART_WordLength = USART_WordLength_8b;
124 USART_InitStructure.USART_StopBits = USART_StopBits_1;
125 USART_InitStructure.USART_Parity = USART_Parity_No;
126 USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
127 USART_InitStructure.USART_HardwareFlowControl =
        USART_HardwareFlowControl_None;
128 USART_Init(USART1, &USART_InitStructure);
129
130 USART_Cmd(USART1, ENABLE);
131
132 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
133
134 NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
135 NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
136 NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
137 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
138 NVIC_Init(&NVIC_InitStructure);
139
140 /*GPIO output for test with led*/
141 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
142 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
143 GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
144 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
145 GPIO_Init(GPIOG, &GPIO_InitStructure);
146
147 }
148
149
150
151 int main(void) {
152     setSysTick();
153     initialize();
154     NVIC_DisableIRQ(TIM2_IRQn);
155     NVIC_DisableIRQ(TIM5_IRQn);
156     i=0;
```

```
157     while(1){
158         if(i==10){
159           NVIC_DisableIRQ(TIM2_IRQn);
160           NVIC_DisableIRQ(TIM5_IRQn);
161           GPIO_ToggleBits(GPIOG, GPIO_Pin_13);
162           for (x = 0; x <=32; x++) {
163
164           USART_SendData(USART1, Data[x]);
165           Delay(1);
166         }
167         NVIC_EnableIRQ(EXTI9_5_IRQn);
168         x=0;
169     }
170         //NVIC_EnableIRQ(EXTI9_5_IRQn);
171 }
172     return 0;
173 }
```

Test_software/BER_Test_software/BER_v1/main.c

A.2 STM32F4xx_it.c

Dette er filen hvor alle interruptrutiner er.

```
1  #include "stm32f4xx_it.h"
2  #include "stm32f4xx_conf.h"
3  //#include "common.h"
4  extern int Data[100];
5  extern int i;
6
7  /**
8    * @brief   This function handles NMI exception.
9    * @param   None
10   * @retval None
11   */
12 __attribute__((weak)) void NMI_Handler(void)
13 {
14 }
15
16 /**
17   * @brief  This function handles Hard Fault exception.
18   * @param   None
19   * @retval None
20   */
21 __attribute__((weak)) void HardFault_Handler(void)
22 {
23   /* Go to infinite loop when Hard Fault exception occurs */
24   while (1)
```

```
25   {
26   }
27 }
28
29 /**
30   * @brief  This function handles Memory Manage exception.
31   * @param  None
32   * @retval None
33   */
34 __attribute__((weak)) void MemManage_Handler(void)
35 {
36   /* Go to infinite loop when Memory Manage exception occurs */
37   while (1)
38   {
39   }
40 }
41
42 /**
43   * @brief  This function handles Bus Fault exception.
44   * @param  None
45   * @retval None
46   */
47 __attribute__((weak)) void BusFault_Handler(void)
48 {
49   /* Go to infinite loop when Bus Fault exception occurs */
50   while (1)
51   {
52   }
53 }
54
55 /**
56   * @brief  This function handles Usage Fault exception.
57   * @param  None
58   * @retval None
59   */
60 __attribute__((weak)) void UsageFault_Handler(void)
61 {
62   /* Go to infinite loop when Usage Fault exception occurs */
63   while (1)
64   {
65   }
66 }
67
68 /**
69   * @brief  This function handles SVCall exception.
70   * @param  None
71   * @retval None
```

```
72   */
73 __attribute__((weak)) void SVC_Handler(void)
74 {
75 }
76
77 /**
78   * @brief  This function handles Debug Monitor exception.
79   * @param  None
80   * @retval None
81   */
82 __attribute__((weak)) void DebugMon_Handler(void)
83 {
84 }
85
86 /**
87   * @brief  This function handles PendSVC exception.
88   * @param  None
89   * @retval None
90   */
91 __attribute__((weak)) void PendSV_Handler(void)
92 {
93 }
94
95 /**
96   * @brief  This function handles SysTick Handler.
97   * @param  None
98   * @retval None
99   */
100 __attribute__((weak)) void SysTick_Handler(void)
101 {
102
103 }
104
105 void EXTI9_5_IRQHandler(void) {
106     // Make sure the interrupt flag is set for EXTI0
107     if(EXTI_GetITStatus(EXTI_Line5) != RESET){
108         NVIC_DisableIRQ(EXTI9_5_IRQn);
109         NVIC_EnableIRQ(TIM2_IRQn);
110         NVIC_EnableIRQ(TIM5_IRQn);
111         TIM_SetCounter(TIM2,0);
112         TIM_SetCounter(TIM5,99);
113         i=0;
114 }
115         // Clear the interrupt flag
116         EXTI_ClearITPendingBit(EXTI_Line5);
117 }
118
```

```
119
120 void TIM2_IRQHandler(void){
121 if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
122 {
123   TIM_SetCounter(TIM5,99);
124 }
125   TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
126 }
127
128
129 void TIM5_IRQHandler(void){
130 if (TIM_GetITStatus(TIM5, TIM_IT_Update) != RESET){
131     Data[i]=GPIO_ReadInputDataBit(GPIOG, GPIO_Pin_5);
132     i++;
133     TIM_ClearITPendingBit(TIM5, TIM_IT_Update);
134   }
135 }
```

Test_software/BER_Test_software/BER_v1/stm32f4xx_it.c