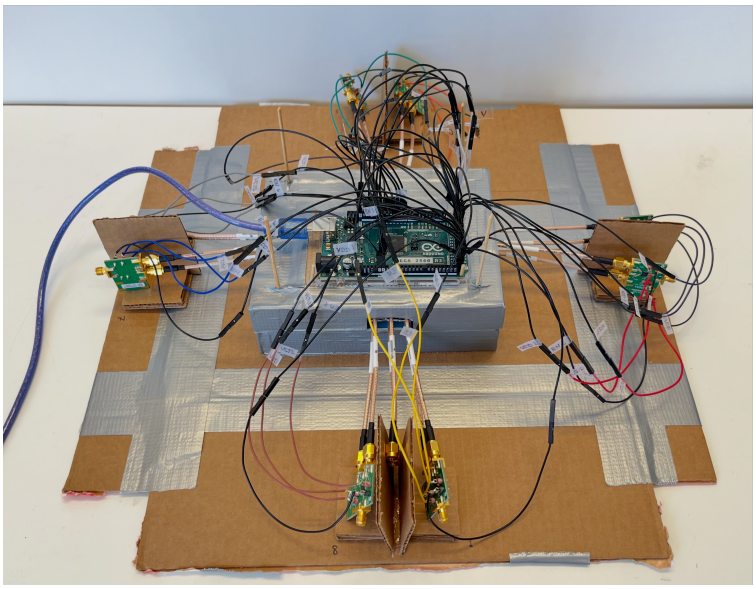
CHALMERS



Mikrovågsbaserad diagnostik av blödningar intrakraniellt och i buk/bröstkorg prehospitalt

Utveckling och utvärdering av ett portabelt mätsystem

Kandidatarbete EENX16-VT25-43

NATHALIE HÖGBERG
STINA ROBERTSSON
ALVA STRAND
SOFIA TEMPLIN
TOVA WIKLUND HELLSTRÖM

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025
www.chalmers.se

Kandidatarbete 2025

Mikrovågsbaserad diagnostik av blödningar intrakraniellt och i buk/bröstkorg prehospitalt

Utveckling och utvärdering av ett portabelt mätsystem

NATHALIE HÖGBERG
STINA ROBERTSSON
ALVA STRAND
SOFIA TEMPLIN
TOVA WIKLUND HELLSTRÖM

CHALMERS

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg 2025

Mikrovågsbaserad diagnostik av blödningar intrakraniellt och i buk/bröstkorg prehospitalt
Utveckling och utvärdering av ett portabelt mätsystem
NATHALIE HÖGBERG
STINA ROBERTSSON
ALVA STRAND
SOFIA TEMPLIN
TOVA WIKLUND HELLSTRÖM

Handledare: Andreas Fhager
Examinator: Xuezhi Zeng

Kandidatarbete 2025
Institutionen för Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Skriven i LaTeX
Göteborg 2025

Mikrovågsbaserad diagnostik av blödningar intrakraniellt och i buk/bröstkorg prehospitalt
Utveckling och utvärdering av ett portabelt mätsystem
NATHALIE HÖGBERG
STINA ROBERTSSON
ALVA STRAND
SOFIA TEMPLIN
TOVA WIKLUND HELLSTRÖM
Institutionen för Elektroteknik
Chalmers Tekniska Högskola

Sammandrag

Akuta blödningar i huvud, buk och bröstkorg är ett stort samhällsproblem, där tiden är avgörande för behandlingens effekt. Det finns därför ett starkt behov av att kunna diagnostisera sådana blödningar prehospitalt. Projektets syfte är att konstruera ett kompakt mätsystem som kan uppfylla den portabiliteten som krävs för prehospital användning, samt uppfylla jämförbar prestanda med en *Vector Network Analyzer* (VNA). Detta har gjorts genom konstruktion av en helt ny switch bestående av två stycken SP8T-switchar (*Single Pole, 8 Throw* som kopplar en ingång till åtta stycken utgångar), åtta stycken SPDT-switchar (*Single Pole, Double Throw* som kopplar en ingång till två utgångar) och en Arduino. Switchen har använts ihop med en *Software-defined radio* (SDR). Komponenterna strömförsörjdes med hjälp av Arduinon och systemet styrdes i ett LabVIEW-program. Dessutom har två stycken prototyper konstruerats, en utformad för detektion av intrakraniella blödningar och en för blödningar i buk/bröstkorg, vardera med åtta stycken tidigare framtagna monopolantenner omgivna av en dämpande gel. Switchen testades med en VNA och observerades dämpa signalen 5 dB och ha en isolation kring 80 dB. SDR:en konstaterades ha en isolation på runt 30 dB mellan sina kanaler, vilket gjorde att en 50 dB dämpare krävdes på referenssignalen för att undvika läckage. Switchen tillsammans med SDR:en kunde mäta signaler ned till 70 dB dämpning. Med mätsystemet kunde de två starkaste signalerna från prototypen för blödningar i buk/bröstkorg på en mänsklig buk urskiljas och liknade resultatet från motsvarande mätning med VNA:n. Detta visar att systemet fungerar och har stor potential. Vidareutvecklingen av systemet bör fokusera på att öka isoleringen mellan SPDT-switcharna, öka SDR:ens isolation mellan kanalerna och förbättra SDR:ens kapacitet att mäta svaga signaler.

Nyckelord: mikrovågor, blödningsdetektion, prehospital diagnostik, SP8T-switch, SPDT-switch, prototyp, monopolantenn

Microwave-based diagnostics of intracranial and abdominal/chest bleedings in prehospital settings
Development and evaluation of a portable measurement system
NATHALIE HÖGBERG
STINA ROBERTSSON
ALVA STRAND
SOFIA TEMPLIN
TOVA WIKLUND HELLSTRÖM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Acute bleedings in the head, abdomen, and chest are a major societal problem, where time is crucial for the effectiveness of treatment. Therefore, there is a strong need to be able to diagnose such bleedings in prehospital settings. The aim of the project is to construct a compact measurement system that can achieve the portability required for prehospital use, as well as comparable performance to a Vector Network Analyzer (VNA). This has been done by construction of a completely new switch consisting of two SP8T switches (Single Pole, 8 Throw, which connects one input to eight outputs), eight SPDT switches (Single Pole, Double Throw, which connects one input to two outputs) and an Arduino. The switch has been used together with a Software-defined radio (SDR). The components were powered by the Arduino and the system was controlled using a LabVIEW program. Furthermore, two prototypes have been constructed, one designed for the detection of intracranial bleedings and one for bleedings in the abdomen/chest, each with eight previously developed monopole antennas surrounded by a lossy gel. The switch was tested using a VNA, and was observed to attenuate the signal by 5 dB and have an isolation around 80 dB. The SDR was found to have an isolation of around 30 dB between its channels, which required a 50 dB attenuator on the reference signal to avoid leakage. The switch together with the SDR could measure signals down to 70 dB attenuation. The two strongest signals from the prototype for abdominal/chest bleedings, used on a human abdomen, could be distinguished with the measurement system. The signals also resembled the results from the corresponding measurement with the VNA. This showed that the system works and has great potential. Further development of the system should focus on increasing the isolation between the SPDT switches, increasing the isolation between the channels of the SDR and improving the SDR's ability to measure weak signals.

Keywords: microwaves, bleeding detection, prehospital diagnostics, SP8T switch, SPDT switch, prototype, monopole antenna

Förord

Vi vill ge ett stort tack till vår handledare Andreas Fhager för vägledning och stöttning under projektets gång. Vi vill även tacka vår examinator Xuezhi Zeng för värdefull kunskap och hjälp. Slutligen vill vi tacka Chalmers bibliotek och fackspråk för deras handledning med rapporten.

Nathalie Högberg, Stina Robertsson, Alva Strand, Sofia Templin, Tova Wiklund Hellström, Göteborg, maj 2025

Ordlista

Nedan följer en lista av ord och akronymer som används i rapporten, listade i alfabetisk ordning:

Agar	Sockermolekyler som tillsammans med vätska bildar en gel
Artefakt	Oönskad struktur i en bild som inte ingår i patientens anatomi
CT	Datortomografi
Dynamisk VI	Funktion som kan anropas i ett LabVIEW-program och ändras under körning
Flervägssignaler	Signaler från en antenn som inte går direkt genom det undersökta objektet utan tar alternativa vägar
MR	Magnetisk resonanstomografi
SDR	Software-defined radio
Signal-brusförhållande	Styrkan på en signal i förhållande till styrkan på bakgrundsbruset
SP8T-mottagarswitchen	Den SP8T-switch som användes för att ta emot signaler från SPDT-switcharna
SP8T-sändarswitchen	Den SP8T-switch som användes för att skicka ut signaler till SPDT-switcharna
SubVI	Funktion som integrerar ett LabVIEW-program i ett annat
TDMS	Technical Data Management Streaming
VNA	Vector Network Analyzer

Innehåll

1
Inledning

Akuta blödningar i huvud, buk och bröstkorg är ett stort samhällsproblem i form av många dödsfall och stora ekonomiska kostnader [1], [2]. Idag används datortomografi (CT) eller magnetisk resonanstomografi (MR) för att diagnostisera den här typen av blödningar på sjukhus. Tiden är avgörande för effekten av behandlingen och en stor möjlighet att förkorta tidsförloppet avsevärt utgörs av prehospital diagnostisering [1]. För närvarande finns dock ingen etablerad effektiv prehospital metod för att diagnosticera blödningar. Portabel och billig utrustning har potential att göra stor skillnad på platser utan tillgång till andra avancerade och dyra lösningar, exempelvis på glest befolkade platser eller för sjukvårdssystem med dåliga ekonomiska förutsättningar.

1.1 Bakgrund

En typ av akuta blödningar är intrakraniella blödningar, vilka kan delas in i traumatiska och spontana [3]. Traumatiska blödningar orsakas av direkt eller indirekt fysisk påverkan, till exempel slag eller tryckvågor mot huvudet [4]. Spontana blödningar orsakas av bristande blodkärl och ett exempel är hemorragisk stroke [1]. Det finns även en annan typ av stroke, ischemisk stroke, som orsakas av en blodpropp och är den vanligaste typen. Stroke står för ca 5 miljoner dödsfall årligen och är en av de största orsakerna till akut död i västvärlden. Dessutom har kostnaden för stroke uppskattats till 64,1 miljarder euro i Europa år 2010.

Tidig trombolysbehandling är idag etablerad som en gynnsam behandling för de patienter med ischemisk stroke. Trombolys är ett blodpropplösande läkemedel som ges intravenöst i syfte att lösa upp proppen [5]. För att påbörja behandlingen är det viktigt att typen av stroke är fastställd som ischemisk, eftersom det är förödande att ge blodförtunnande läkemedel till patienter med hemorragisk stroke. Behandlingen måste dessutom utföras inom 4,5 timme från symptomens uppkomst för att fördelarna ska överväga riskerna [1]. Således finns ett behov av att tidigt kunna skilja på ischemisk och hemorragisk stroke, vilket idag görs på sjukhus med CT och ibland MR. Framtagandet av alternativa prehospitala lösningar är däremot under utveckling [6]. Flertalet studier rörande portabla system för strokedetektion utnyttjar olika diagnostiseringstekniker, däribland olika tillämpningar av spektroskopi, ultraljud, elektroencefalografi (EEG) och mikrovågsteknik. Även om flertalet studier uppvisar lovande resultat kring identifiering och differentiering av stroke, är majoriteten av systemen fortfarande i utvecklingsfasen. Storskalig testning i prehospitala miljöer

och säkerställande av ett systems prestanda är enbart ett par aspekter att beakta innan klinisk användning blir aktuellt.

En annan typ av akuta blödningar kan uppstå i buken och är en vanlig dödsorsak för traumapatienter [2]. Vid en blödning i buken (hemoperitoneum) ökar risken för dödsfall med 1 % var tredje minut. Precis som vid stroke, är tidig upptäckt av hemoperitoneum viktigt för att undvika dödsfall. Ett prehospitalt mätsystem är därför användbart för att kunna upptäcka blödningar i tid. Etablerade metoder som idag kan användas vid diagnostisering av bukskador inkluderar ultraljud, CT och MR [7]. Ultraljud kan användas som en initial undersökning för att detektera hemoperitoneum. Undersökningen kan användas prehospitalt tack vare dess portabilitet och snabbhet, men eftersom metoden kräver en välutbildad operatör är sådan användning inte utbredd [2]. Sensitiviteten för ultraljudsdetektion är dessutom för låg för att helt utesluta hemoperitoneum. Användningen av MR vid traumaskador är idag begränsad, dels på grund av bristande tillgänglighet, dels på grund av lång bildtagningstid och monitoreringssvårigheter [7]. Den huvudsakliga undersökningen för att identifiera hemoperitoneum görs i stället med CT som utförs på sjukhus. På grund av dess höga kostnad och otymplighet är den dock inte lämplig för prehospital användning [2]. En alternativ metod skulle kunna vara att använda ett mikrovågsbaserat mätsystem, då ansamling av blod dämpar mikrovågssignalen, vilket möjliggör detektion av blödningar.

1.2 Syfte

Projektets syfte är att vidare undersöka användningen av mikrovågsteknik för att prehospitalt detektera blödningar i kroppen, med fokus på intrakraniella blödningar och blödningar i buk/bröstkorg. Mer specifikt är projektets huvudsyfte att konstruera ett fungerande kompakt mätsystem, där fokus ligger på att skapa en ny switch med låg dämpning och hög isolation. Vidare är målet att mätsystemet ska uppnå jämförbara resultat med en *Vector Network Analyzer* (VNA). Mätsystemet ska utvärderas med två egenutvecklade prototyper för blödningsdetektion. Prototyperna ska baseras på tidigare forskning och redan framtagna prototyper, med fokus på förbättrad stabilitet och flexibilitet. En av prototyperna ska vara utformad för detektion av intrakraniella blödningar och en ska vara utformad för blödningar i buk/bröstkorg.

1.3 Avgränsningar

Projektet innefattar konstruktion av två prototyper med monopolantenner. Den ena prototypen är anpassad för detektion av intrakraniella blödningar och den andra är anpassad för detektion av blödningar i buk/bröstkorg. Prototyper för detektion av blödningar i andra kroppsdelar undersöks inte. Avgränsningen är gjord eftersom blödningar intrakraniellt, i bröstkorgen och i buken anses vara mest livshotande och effektiv prehospital diagnostisering saknas idag.

Projektet omfattar inte analys eller förbättring av den redan existerande inlärningsalgoritmen för att detektera samt kategorisera stroke och blödningar. Det anses vara mer givande att först utforma ett portabelt mätsystem och två välfungerande prototyper innan vidare utvärdering av algoritmen utförs. Vidare rekonstrueras inga bilder från de erhållna resultaten, utan resultaten från mätningarna analyseras endast i MATLAB.

Slutligen genomförs ingen omfattande testning. Tester genomförs endast i syfte att validera och utvärdera mätsystemets olika komponenter samt de konstruerade prototyperna. En av de största bidragande faktorerna till den här avgränsningen är att tiden för projektet är begränsad. För att genomföra mer omfattande tester på människor behöver dessutom en klinisk studie utföras, vilket tar tid att planera, genomföra och analysera.

2
Teori

Kapitlet presenterar tidigare forskning inom mikrovågsbaserad diagnostik, följt av en teknisk bakgrund med relevant information om de komponenter som används i projektet.

2.1 Forskning inom mikrovågsbaserad diagnostik

Mikrovågsbaserad diagnostik är ett aktuellt forskningsområde som har visats kunna identifiera interna blödningar [1], [2], [8]. Tekniken bygger på en dielektrisk skillnad mellan olika kroppsvävnader och blod [1]. Mikrovågor har en liten strålningspåverkan på kroppen med lång våglängd och låg effekt, ~ 1 mW, vilket är betydligt lägre än mobiltelefoners högsta effekt [2]. Den långa våglängden gör också att mikrovågor, till skillnad från röntgenstrålning, inte är joniserande i kroppen [9]. En viktig aspekt inom diagnostik är avbildning [10]. Mikrovågsavbildning innefattar passiva och aktiva tekniker samt hybridtekniker som kombinerar olika passiva och aktiva metoder. Vid passiva tekniker mäts naturliga emissioner från målet för att få fram önskad information. Vid aktiva tekniker sänds riktade mikrovågor mot målet, och den uppmätta reflekterade signalen används sedan för att skapa bilder. De tre huvudtyperna av aktiva mikrovågsavbildningstekniker är mikrovågstomografi, radarbaserad avbildning och maskininlärningsbaserad avbildning [11]. Det går att konstruera kompakta användarvänliga mikrovågssystem till relativt låga kostnader, vilket möjliggör prehospitala användningsområden såsom ett system i varje ambulans [12].

2.1.1 Diagnostisering av intrakraniella blödningar

Flera tekniker för att skilja en blödning från en blodpropp i en prehospital miljö har undersökts [1]. I tidigare undersökningar av att utveckla ett mikrovågsbaserat system för strokedetektion har två prototyper konstruerats. I den första prototypen användes en cykelhjälm med tio antenner monterade inuti hjälmen. Antennerna var monterade så att en platt yta låg mot huvudet i syfte att skapa en bekväm passform. Inuti hjälmen användes även plastbehållare fyllda med vatten mellan antennerna för att möjliggöra en bra passform för alla sorters former och storlekar på huvuden. Ett problem med prototypen var att strukturerna som höll antennerna på plats hade en för svag mekanisk styrka i mer krävande miljöer. Den andra prototypen bestod av en specialbyggd hjälm med tolv antenner. Antennerna som användes i båda prototyperna var en tidigare modell av monopolantennerna som beskrivs längre fram, i avsnitt 2.1.4. Antennerna användes utan någon form av dämpande gel och hade en

triangulär form. Mätningarna för båda prototyperna utfördes genom att en antenn användes som sändare, medan de andra antennerna tog emot signaler. För den första prototypen utfördes mätningarna med en två-ports nätverksanalysator (Agilent E8362 B PNA) integrerad med en switch-matrismodul. Mätningarna för den andra prototypen utfördes i stället med en specialbyggd två-ports nätverksanalysator med switch-lösning. Utöver de tidigare nämnda undersökningarna har andra studier på mikrovågsbaserad strokedetektion genomförts. Gemensamt för dem är att ingen har använt någon form av dämpande gel.

2.1.2 Diagnostisering av blödningar i buken

I en tidigare studie konstruerades en portabel prototyp i form av ett bälte med åtta antenner som kunde upptäcka och monitorera hemoperitoneum [2]. För att möjliggöra flexibilitet var antennerna inte fixerade. Prototypen testades på sövda grisar med 500 ml eller 1000 ml injicerat blod i buken. Resultaten visade att signalen minskade jämfört med baslinjen när blodvolymen ökade. Klassifikationsanalysen hade högst precision (95 %) för den största blödningen. Detektionen försvårades dock av den stora skillnaden i baslinje mellan olika individer. En orsak till den stora skillnaden kan ha varit variationer i överhörning, som uppstår mellan närliggande antenner, på grund av varierande avstånd mellan antennerna. Antennerna vid midaxillarlinjen, en imaginär linje som delar kroppen i en främre och bakre del, var mest känsliga för att detektera blod eftersom det primärt ackumulerades där. Framöver bör antennerna placeras nära midaxillarlinjen samt tätt bakom på grund av gravitationen och att de flesta traumapatienter är positionerade i ryggläge.

2.1.3 Övriga tillämpningar inom medicinsk diagnostik

Mikrovågor kan tillämpas inom medicinsk diagnostik inom många olika områden. Till exempel kan mikrovågor även användas för diagnostisering och monitorering av muskelskador, där kunskapen kring läkningsprocessen inte är helt klarlagd [13]. Med mer kunskap finns förhoppningar om att minska risken för återkommande skador och förkorta rehabiliteringstiden. De metoder som idag används för att undersöka sådana skador utgörs främst av MR och ultraljud. Metoderna har dock brister både gällande ekonomi och tillgänglighet.

Mikrovågsteknik har också potential att diagnostisera bröstcancer [9]. Röntgenstrålning ger bäst kontraster i bilder av tunga vävnader som ben. Framförallt i bröst med mycket körtelvävnad försvåras diagnostiseringen på grund av bristande kontraster mellan olika typer av vävnad. Mikrovågsspektrat är bättre lämpat för att ge en tydlig kontrast mellan tumör och kroppsvävnad.

En ytterligare tillämpning är mikrovågsradiometri, vilket är en passiv metod som har visat potential att diagnostisera lunginflammation [14]. Studien avgränsades till lunginflammation hos COVID-19 patienter, där mikrovågsradiometri lyftes fram som ett potentiellt alternativ till CT-undersökning. Mikrovågsradiometri är ett alternativ som för med sig fördelar som ökad känslighet, noggrannhet och säkerhet,

eftersom det inte utsätter patienten för radioaktiv strålning. Samtidigt diskuteras begränsningar gällande djup detektering i lungorna och diagnostisering vid andra medicinska tillstånd.

2.1.4 Antenndesign

Utsända signaler som inte går direkt genom det undersökta objektet, exempelvis ytvågor eller reflektioner från bland annat stödjande strukturer, kallas flervägssignaler [13]. Signalerna kommer från den ursprungliga antennen och har därför samma frekvens som den sökta signalen, vilket gör dem svåra att filtrera bort. Antenner som är placerade nära varandra har större risk att påverkas av störningar från flervägssignaler [15]. En ny monopolantenn med en dämpande gel har tagits fram för att minska bildstörningar från flervägssignaler [13].

2.1.5 Utmaningar inom mikrovågsteknik

Vid klinisk användning av mikrovågsteknik finns störningar i form av interferens, bland annat från annan elektrisk utrustning och trådlösa kommunikationsenheter [16]. Andra icke-tolererbara störningar vid mikrovågsmätningar kan utgöras av strömförsörjande kablar eller utrustning utanför den kontrollerade miljön, i form av elektriska och magnetiska fält. Det medicintekniska systemet måste därför vara motståndskraftigt mot störningar och interferenser som kan uppkomma.

Den utsända signalen måste följa standarder och regler rörande interferens gentemot andra elektriska enheter, utsänd effekt och bandbredd [16]. Eftersom de valda frekvenserna har stor betydelse för systemets prestanda vid mikrovågsmätningar utgör ovannämnda standarder och regler en utmaning. De valda frekvenserna påverkar även penetrationsdjupet och upplösningen [15]. Vid ökad frekvens ökar upplösningen, medan penetrationsdjupet minskar. Avvägningen mellan upplösning och penetration utgör en utmaning, där mikrovågstekniker konkurrerar med andra avbildningstekniker som CT och MR [16]. Jämfört med CT och MR uppvisar mikrovågssystem fördelar gällande risker, portabilitet, tidsupplösning och kostnad, samtidigt som det finns nackdelar rörande exempelvis spatiell (rumslig) upplösning [15].

Andra utmaningar inkluderar borttagning av artefakter, exempelvis rörelseartefakter, och anpassning av mediet mellan objekt och sensor [16]. Vid stora skillnader i dielektriska egenskaper mellan vävnaden och mediet, uppkommer stora reflektioner i utrymmet mellan antennen och kroppen [15]. Genom att använda vatten eller ett matchande medium minskar reflektionen. En annan lösning innefattar att placera antennen ännu närmare kroppen.

Mikrovågsbaserad diagnostik bygger på jämförelse mellan mätdata och referensdata för att kunna identifiera avvikande dielektriska egenskaper hos mätobjektet [16]. En svårighet är att referensdata ofta är otillgänglig vid praktisk användning. Samtidigt är det svårt att utveckla allmänna datablad med referensdata eftersom det finns die-

lektriska skillnader mellan vävnader på individnivå. Skillnaderna påverkas av bland annat kön, ålder och sjukdomar men kan även bero på vävnaders ojämna tjocklek.

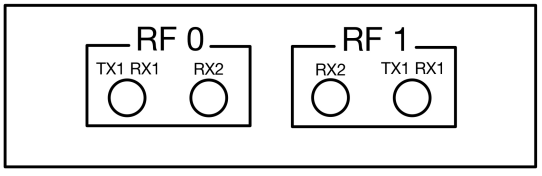
2.2 Teknisk bakgrund

Den tekniska bakgrunden beskriver de olika tekniska komponenterna som används i projektet. Det mikrovågsbaserade mätsystemet, som styrs i LabVIEW, är uppbyggt av en SDR, en Arduino, två sorters switchar och åtta monopolantenner. Avsnittet innehåller även information om VNA:n som används för validering och utvärdering av mätsystemet.

2.2.1 SDR

En *Software-defined radio* (SDR), på svenska mjukvarudefinierad radio, kan bland annat användas för att skapa och mäta mikrovågor [17]. Modellen USRP-2901 kan både skicka och ta emot vågor inom 70 MHz - 6 GHz, med steg på < 1 kHz och en noggrannhet på 2,5 ppm (*parts per million*). SDR:en har en brusfaktor på < 8 dB och strömförsörjs av 6 V vid 3 A via USB 2.0 eller USB 3.0. En SDR är ett flexibelt och relativt billigt instrument, men SDR:en har samtidigt flertalet brister [18]. Ett vanligt problem med en SDR är att isolationen mellan olika kanaler är låg, vilket resulterar i överhörning mellan sändar- och mottagarkanaler. Den här typen av överhörning bidrar till osäkerhet i mätdata, svårigheter med kalibrering och hindrar mätning av svaga signaler.

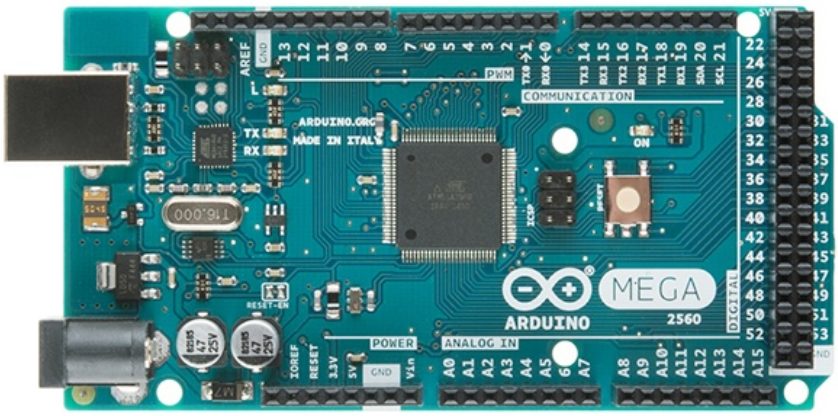
Ett annat problem är avsaknaden av fasstabilitet [18]. För varje ny mätning erhålls en ny fas, vilket innebär att mätdata behöver faskalibreras. I projektet utförs faskalibrering genom att jämföra mätdata med data erhållen på en referenskanal. SDR:en används för att skicka ut mikrovågor på en sändarkanal samtidigt som mikrovågor tas emot på två olika kanaler, mottagarkanal och referenskanal, i tur och ordning. Sändarkanalen har en inre förstärkare med kapacitet mellan -40 dB till +49,57 dB och mottagarkanalens förstärkare kan justeras mellan -15 dB till +61 dB [18]. I figur 2.1 visas portarna för de olika kanalerna i SDR:en. Varje huvudkanal (RF0 och RF1) har en kanal som kan användas som sändare eller mottagare (TX1 RX1) och en kanal som endast kan användas som mottagare (RX2). RF0 - TX1 används i projektet som sändarkanal, RF1 - RX1 och RF1 - RX2 används som referens- respektive mottagarkanal.



Figur 2.1: Bilden visar portarna för SDR:en med två huvudkanaler, RF0 och RF1. Varje huvudkanal har en kanal som kan användas som sändare eller mottagare (TX1 RX1) och en kanal som endast kan användas som mottagare (RX2).

2.2.2 Arduino

I projektet används en Arduino Mega 2560 Rev 3 för att styra och strömförsörja samtliga switchar som presenteras i nästkommande avsnitt. Arduinon, som visas i figur 2.2, har 54 digitala utgångar som opererar på 5 V (på eller av) och 5 utgångar för jord [19]. Arduinon behöver 12 V strömförsörjning och programmeras i Arduinos egna gränssnitt med C++-kod. Från gränssnittet laddas kod upp till Arduinon, och den uppladdade koden behålls sedan och körs på Arduinon så länge ingen ny kod laddas upp.



Figur 2.2: Bilden visar Arduino Mega 2560 Rev 3 [20]. CC BY 2.0.

2.2.3 SPDT-switch

Switchen av modell HMC8038 är en SPDT (*Single Pole, Double Throw*), vilket innebär att den kopplar en ingång, RFC, till två utgångar, RF1 och RF2 [21]. Switchen visas i figur 2.3.

Figur 2.3: Bilden visar SPDT-switchen.

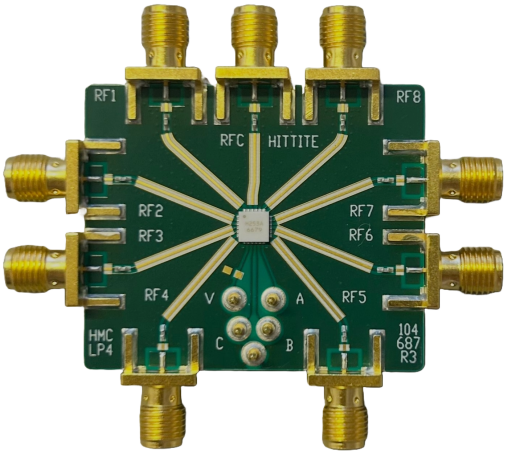
Modellen används i mätsystemet för att koppla varje antenn som sändare eller mottagare, alternativt stänga av antennen. Vilken utgång på SPDT-switchen (RF1 eller RF2) som är påkopplad styrs med strömtillförsel till VCTL och antennen kan helt isoleras genom att tillföra ström till EN, se tabell 2.1. Switchen behöver en matningsspänning på $V_{DD} = 5$ V, kopplat till uttaget märkt ”VDD”, och har en antireflekterande 50 Ω design. Switchen har även ett uttag för jord märkt ”GND”. Isolationen mellan kanalerna är typiskt 60 dB och inkopplingsförlusten 0,8 dB.

Tabell 2.1: Tabell över styrsignaler och den resulterande utsignalen [21].

Styrsignal		**Utsignal**	
$V_{\mathbf{CTL}}$	**EN**	**RFC till RF1**	**RFC till RF2**
Låg	Låg	Av	På
Hög	Låg	På	Av
Låg	Hög	Av	Av
Hög	Hög	Av	Av

2.2.4 SP8T-switch

Switchen av modell HMC253ALC4 är en SP8T (*Single Pole, 8 Throw*), vilket innebär att den kopplar en ingång, RFC, till åtta stycken utgångar, RF1-8. Switchen visas i figur 2.4.



Figur 2.4: Bilden visar SP8T-switchen.

Modellen används i projektet för att koppla ihop samtliga SPDT-switchar till SDR:ens sändar- och mottagarkanal. Vilken av utgångarna på SP8T-switchen (RF1-8) som är påkopplad styrs med olika kombinationer av styrsignaler till tre stycken styrpinnar på switchen märkta A, B, C enligt tabell 2.2 [22]. Switchen behöver en matningsspänning på $V_{DD} = 5$ V, kopplat till uttaget märkt ”V”, och har en antireflekterande design. Switchen har även ett uttag för jord. Isolationen mellan kanalerna är typiskt

43 dB vid frekvenser mellan DC - 2,0 GHz och inkopplingsförlusten är typiskt 1,1 dB för samma frekvensintervall.

Tabell 2.2: Tabellen beskriver vilka kombinationer av insignaler till styrpinnarna i SP8T-switchen som resulterar i vilken utgående port [22].

Styrsignaler			**Utsignal**
A	**B**	**C**	**Påkopplad RF**
Låg	Låg	Låg	RF1
Hög	Låg	Låg	RF2
Låg	Hög	Låg	RF3
Hög	Hög	Låg	RF4
Låg	Låg	Hög	RF5
Hög	Låg	Hög	RF6
Låg	Hög	Hög	RF7
Hög	Hög	Hög	RF8

2.2.5 VNA

I projektet används en VNA av modellen R&S®ZNBT8 för att mäta isolationen mellan SPDT- och SP8T-switcharna i switchen och jämföra mätdata från VNA:n med mätdata från mätsystemet. Genom att separera samt mäta infallande och reflekterade signaler erhålls mätdata i VNA:n i form av spridningsparametrar, vanligen kallade S-parametrar [23], [24]. S-parametrar beskriver hur signaler sprids eller reflekteras när de passerar genom ett nätverk [24].

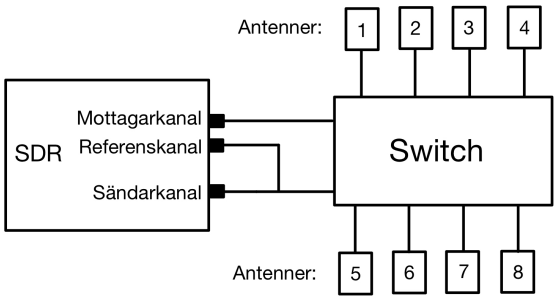
Modellen R&S®ZNBT8 från Rhode & Schwarz opererar i frekvensintervallet 9 kHz till 8,5 GHz [25]. VNA:n har 24 portar som alla kan användas för datainsamling samtidigt. Modellen har även en hög temperaturstabilitet på 0,01 dB/K, vilket innebär att mätvariationen påverkas väldigt lite av temperaturvariationer under mätning. En hög temperaturstabilitet innebär också att VNA:n behöver kalibreras mer sällan. I projektet gjordes en automatisk kalibrering av portarna som används på VNA:n med en kalibreringsenhet av modell R&S®ZN-Z152.

3
Metod

Kapitlets första avsnitt inkluderar en överblick av mätsystemet ihop med konstruktion och programmering av den nya switchen. Därefter beskrivs hur de två prototyperna konstruerades. Sista avsnittet i kapitlet inkluderar tester av mätsystemets olika delar och de två prototyperna.

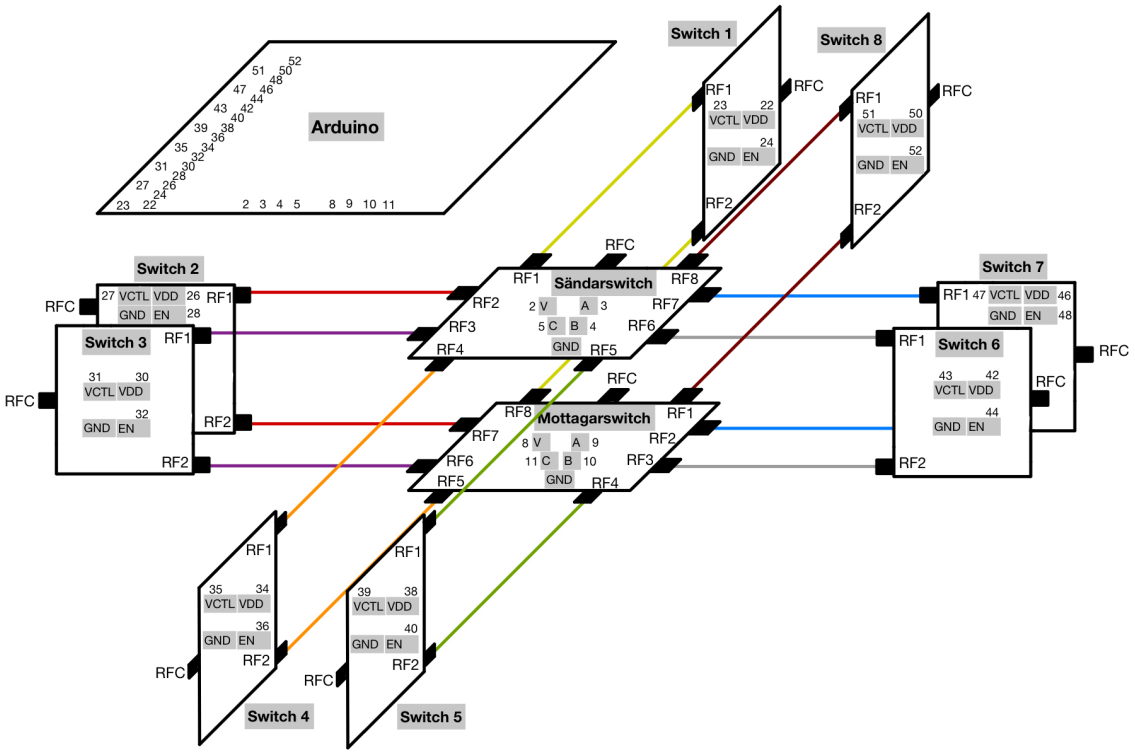
3.1 Konstruktion av mätsystem

Mätsystemet konstruerades med en SDR, en ny switch samt åtta antenner. En översiktlig skiss över mätsystemet visas i figur 3.1.



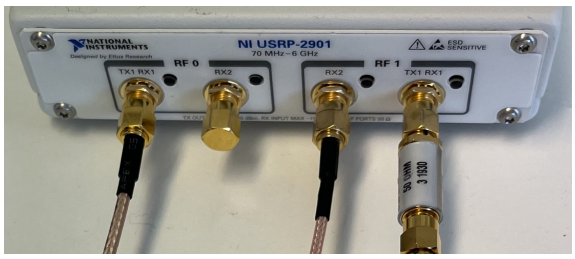
Figur 3.1: Skiss över mätsystemets olika delar; SDR:en, den nya switchen och de åtta antennerna.

En ritning av den nya switchen visas i figur 3.2. Switchen består av åtta SPDT-switchar av modell HMC8038 där varje RFC-port på switcharna kopplades via mikrovågskablar till varsin monopolantenn med dämpande gel, beskrivna i avsnitt 2.1.4. Alla RF1-portar, respektive RF2-portar, på SPDT-switcharna kopplades med mikrovågskablar till varsina SP8T-switchar av modell HMC253ALC4. De två SP8T-switcharna placerades i mitten med undersidorna mot varandra. Samtliga SPDT- och SP8T-switchar styrdes med 5 V med en Arduino Mega 2560 Rev 3. Arduinon programmerades med C++ för att kunna förse samtliga SPDT- och SP8T-switchar med V_{DD}-spänning samt ge spänning till rätt styrpinnar på switcharna utifrån extern input.



Figur 3.2: Den nya switchens konstruktion, där kopplingarna för samtliga ingående switchar och Arduinon visas. Utgångarna på Arduinon visualiseras med siffror som stämmer överens med styrpinnarna på de olika SPDT- och SP8T-switcharna för att illustrera hur de är sammankopplade med kablar.

De två SP8T-switcharna kopplades från respektive RFC-port till en SDR av modell USRP-2901, där portarna för SDR:en visas i figur 3.3. Vid SDR-utgången (RF0 - TX1, senare benämnd som ”sändarkanal”) användes en signaldelare där den ena signalen kopplades vidare till den SP8T-switchen som styrde RF1-portarna på SPDT-switcharna (senare benämnd som ”SP8T-sändarswitchen”). Den andra signalen kopplades in i en av ingångarna på SDR:en (RF1 - RX1) via en dämpare. SP8T-switchen som styrde RF2-portarna på SPDT-switcharna (senare benämnd som ”SP8T-mottagarswitchen”) kopplades in i den andra ingången på SDR:en (RF1 - RX2, senare benämnd som ”mottagarkanal”). Signalen som gick direkt från SDR:ens utgång, via signaldelaren, till SDR:ens ingång användes för kalibrering (senare benämnd som ”referenskanal”). En översikt av kopplingarna mellan SDR:en och switchen visas i figur 3.1.



Figur 3.3: Figuren visar portarna för SDR:en av modell USRP-2901.

På alla SPDT-switchar löddes fyra kopplingskablar fast på styrpinnarna VCTL, VDD, EN och GND. Fem kopplingskablar löddes även fast på SP8T-switcharna på styrpinnarna V, A, B, C och G. Samtliga kopplingskablar var i andra änden inkopplade i Arduinon enligt figur 3.2. Alla GND-styrpinnar på SPDT-switcharna och G-styrpinnar på SP8T-switcharna var även inkopplade till jord på Arduinon.

3.2 Styrning av mätsystem

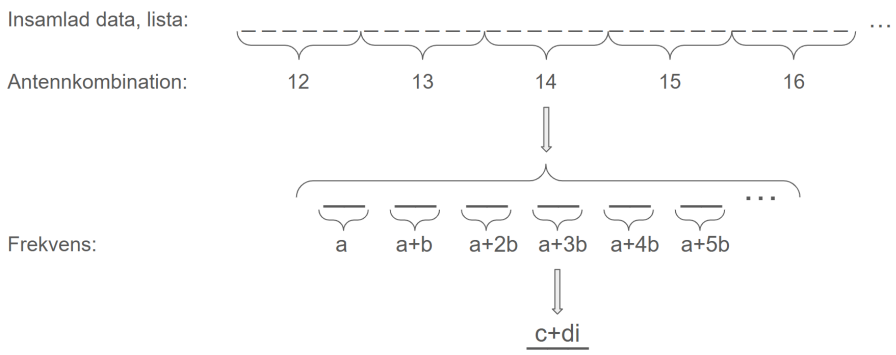
Både Arduinon och SDR:en styrdes med hjälp av LabVIEW. Arduinon styrdes med en SubVI vid namn `Arduino.SubVI` som presenteras i appendix C.1. Programmet konstruerades så att ett kommando skickades till Arduinon med information om vilken antenn som skulle kopplas som sändare respektive mottagare. Den första siffran definierade sändarantennen, medan den andra siffran stod för mottagarantennen. Till exempel innebär ”12” att antenn 1 sänder ut och antenn 2 tar emot. Totalt finns det 28 antennkombinationer eftersom samma SPDT-switch inte kan vara sändare och mottagare samtidigt, samt eftersom speglingskombinationer, exempelvis ”12” och ”21”, räknas som samma kombination. SubVI:n var sammankopplad med den C++-kod som var uppladdad till Arduinon. På så sätt ku Arduinon kunde läsa av vilken antennkombination som skickades från LabVIEW och använda informationen för att ge spänning till rätt styrpinnar på SPDT- och SP8T-switcharna.

Ett första test utfördes för att verifiera att Arduino-koden fungerade korrekt. En antennkombination bestämdes i LabVIEW och en multimeter användes för att mäta spänningen i de olika utgångarna på Arduinon som användes i systemet.

3.2.1 LabVIEW-styrning utan faskalibrering

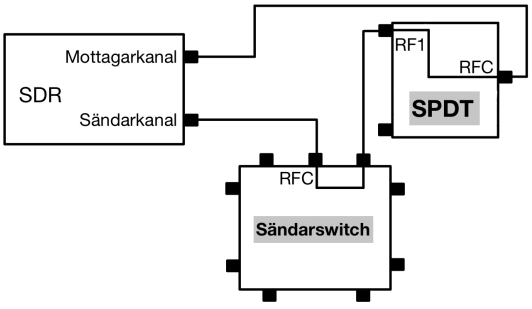
Vidare integrerades SubVI:n för Arduinon i ett redan existerande LabVIEW-program för SDR:en, `SDR_RI2582.vi`, skrivet av Laura Guerrero Orozco och Xuezhi Zeng. LabVIEW-koden för `SDR_RI2582.vi` presenteras i appendix C.3. `SDR_RI2582.vi` loopade igenom alla valda antennkombinationer och gjorde en frekvenssvepsmätning per kombination. Programmet utförde endast enkalansmätningar. Det vill säga att mätningar endast utfördes på mottagarkanalen på SDR:en utan referenskanalen inkopplad. `SDR_RI2582.vi` styrde SDR:en genom att först producera den signal som skickades till sändarantennen, och sedan ta emot signalen från mottagarantennen. Data från den mottagna signalen sparades från LabVIEW i en TDMS-fil (*Technical data management system*).

TDMS-filens data är hierarkiskt strukturerad i nivåerna fil, grupp och kanal. Vid enkanalsmätningen sparas all uppmätt data i en lång lista utifrån ordningen den uppmäts. Ordningen är till följd av detta primärt sorterad efter antennkombination och sekundärt efter frekvens, från lägst till högst utifrån ett frekvenssvep med ökande frekvens. Vid varje frekvens insamlas flera mätpunkter av signalen, varpå ett medelvärde av mätpunkterna sparas. Ett uppmätt datavärde sparas i form av ett komplext tal. I figur 3.4 illustreras TDMS-filernas struktur vid enkanalsmätning.

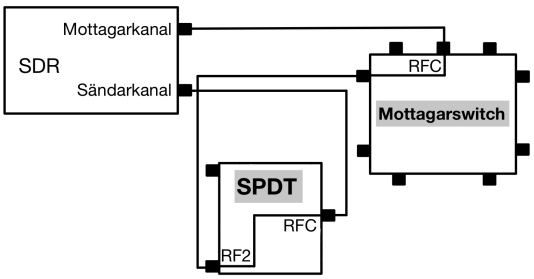


Figur 3.4: Figuren visar hur data i TDMS-filen vid en enkanalsmätning i första hand är strukturerad efter antennkombination och i andra hand efter frekvens. Vid frekvenssvepet i figuren beskriver a startfrekvens och b steglängd. Varje individuellt mätpunkt är ett komplext tal och representerat i figuren som c+di.

För att verifiera att en signal sändes ut från SDR:en, gick genom switchen och togs emot i SDR:en igen enligt de instruktioner som gavs i LabVIEW, gjordes tester med LabVIEW-koden `SDR_RI2582.vi`. Till en början kopplades SP8T-sändarswitchen direkt till sändarkanalen på SDR:en medan mottagarkanalen på SDR:en kopplades till en SPDT-switch i taget, se figur 3.5a. Mätningar utfördes för att bekräfta att signalen tog sig fram till samtliga SPDT-switchar. Mätningarna upprepades även med SP8T-mottagarswitchen för att verifiera att signalen tog sig tillbaka från samtliga SPDT-switchar. Här var SP8T-mottagarswitchen inkopplad i mottagarkanalen på SDR:en och SPDT-switcharna var i stället inkopplade i sändarkanalen på SDR:en, se figur 3.5b. Slutligen bekräftades den sammansatta funktionen av switchen med mätningar där båda SP8T-switcharna användes. SP8T-sändarswitchen var inkopplad i sändarkanalen på SDR:en, medan SP8T-mottagarswitchen var inkopplad i mottagarkanalen på SDR:en, se figur 3.6. Två olika SPDT-switchar var direkt sammankopplade i olika kombinationer för att bekräfta att signalen tog sig igenom hela switchen då mätningar genomfördes. I samtliga tester med `SDR_RI2582.vi` uppstod en tydlig topp av signalen vid antennkombinationen med direktkopplingen, vilket bekräftade att signalen tog sig igenom systemet på korrekt sätt.

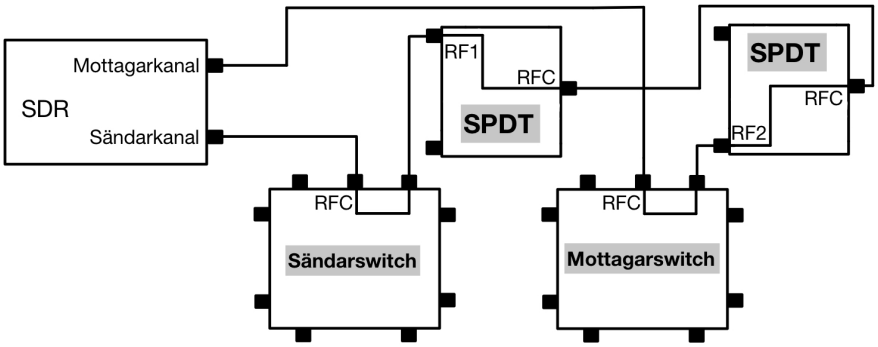


(a) Kopplingsschema för test med SP8T-sändarswitchen.



(b) Kopplingsschema för test med SP8T-mottagarswitchen.

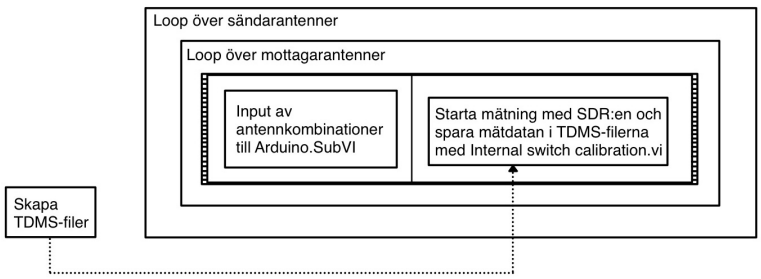
Figur 3.5: Kopplingsschema för test med båda SP8T-switcharna.



Figur 3.6: Figuren visar kopplingschemat för direktkoppling mellan två SPDT-switchar.

3.2.2 LabVIEW-styrning med faskalibrering

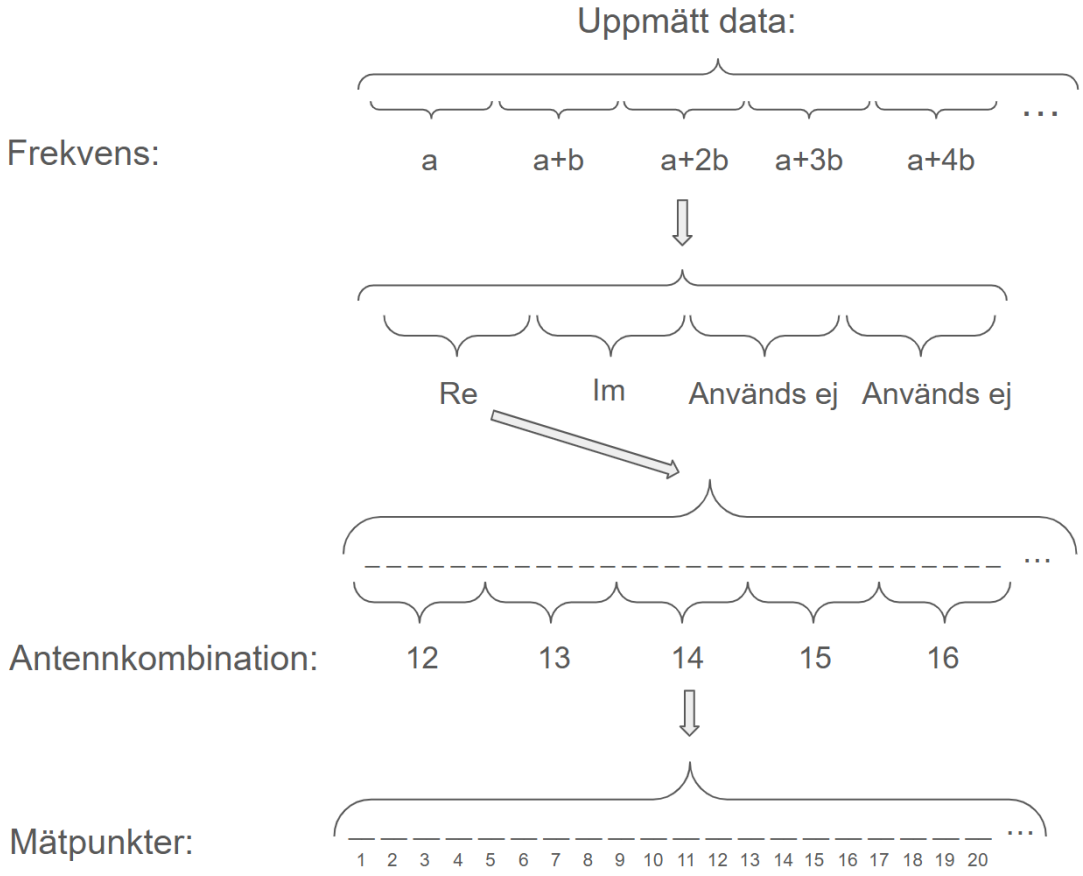
SDR:en kräver faskalibrering för att resultaten ska kunna processeras vidare. Därför behövde SubVI:n för Arduinon användas tillsammans med en LabVIEW-kod skriven av Laura Guerrero Orozco och Xuezhi Zeng, vid namn `Internal switch calibration.vi`. Faskalibreringen gjordes i ett nytt LabVIEW-program som döptes till `Arduino_TwoChannel.vi`, se appendix C.4 för kod. I `Arduino_TwoChannel.vi` integrerades SubVI:n för Arduinon i början av en tidssekvensstruktur. I efterföljande steg i tidssekvensstrukturen integrerades `Internal switch calibration.vi` som en dynamisk VI. En skiss över kodstrukturen i `Arduino_TwoChannel.vi` visas i figur 3.7. Med SubVI:n `GenIQ.vi` i `Internal switch calibration.vi` skapades signalen som skickades till sändarantennen, se LabVIEW-koden för `GenIQ.vi` i appendix C.2. Signalen togs sedan emot först från referenskanalen och därefter mottagarkanalen. Programmet loopade därefter över ett bestämt frekvensintervall. I `Internal switch calibration.vi` bestämdes antal mätpunkter per frekvens till 16384 och förstärkningen vid sändarkanalen och mottagarkanalen till 15 dB respektive -15 dB. En omskrivning av `Internal switch calibration.vi` gjordes för att de mottagna signalernas data skulle sparas i två TDMS-filer i stället för två nya TDMS-filer för varje frekvens. Den uppdaterade versionen öppnade i stället de två filer som skapades i början av `Arduino_TwoChannel.vi`. Därmed erhölls en TDMS-fil för referenskanalen och en TDMS-fil för mottagarkanalen. Den uppdaterade versionen av `Internal switch calibration.vi` presenteras i appendix C.5.



Figur 3.7: Skiss över kodstrukturen i LabVIEW-programmet `Arduino_TwoChannel.vi`.

Med två loopar i `Arduino_TwoChannel.vi` erhölls mätdata från samtliga antenn-

kombinationer. Mätdata från referens- och mottagarkanalen var organiserad på liknande sätt som vid enkanalsmätningen, utifrån frekvens och antennkombination. Till skillnad från enkanalsdata sparades realdelen och imaginärdelen separat i två olika listor, samt ytterligare två listor som inte användes i projektet. Totalt sparades fyra listor per frekvens i frekvenssvepet. Varje lista innehöll nu även alla uppmätta mätpunkter och inte enbart ett medelvärde. Alla mätpunkter tillhörande samma antennkombination låg efter varandra, se figur 3.8. Detta är den stora skillnaden gentemot originalprogrammet i LabVIEW, där enbart en sektion fanns i varje fil. I den uppdaterade versionen var de här sektionerna i sin tur sorterade efter ordningen som antennkombinationerna mättes. Mätdata från de två kanalerna var sorterade utifrån samma princip, i varsin TDMS-fil, och mätpunkter vid samma position i respektive datafil utgjorde alltså ett par.



Figur 3.8: Figuren visar strukturen hos en TDMS-fil från en kanal vid en tvåkanalsmätning. Data strukturerades först efter frekvens, där a är startfrekvens och b är steglängd. Därefter strukturerades data efter real- och imaginärdel. Slutligen är uppmätt data sorterad efter antennkombination, där data inom en antennkombination är sorterad efter ordningen som mätpunkterna uppmättes.

3.3 Hantering av data från mätsystem

Mätdata från LabVIEW erhölls som TDMS-filer. En MATLAB-funktion kallad `SDR_1channel.m` skapades för att hantera resultatet från en enkanalsmätning, se appendix D.1 för MATLAB-kod. Funktionen är exempelvis kompatibel med LabVIEW-

programmet SDR_RI2582.vi. TDMS-filerna från SDR_RI2582.vi konverteras först med hjälp av funktionen convertTDMS.m (appendix D.2.4), skriven av Laura Guerrereo Orozco och Xuezhi Zeng. Funktionen organiserar och sparar data i en datastruktur som är kompatibel med MATLAB. Absolutbeloppet av mätvärdena sammanställs därefter i grafer. SDR_1channel.m användes för att visualisera resultaten från testerna av LabVIEW-styrningen utan faskalibrering, beskrivna i avsnitt 3.2.1.

Efter omarbetning av LabVIEW-koden sparades totalt fyra filer per antennkombination, en för uppmätt data och en för uppmätt referensdata samt en hjälpfil till vardera fil. Därmed krävdes omskrivning av MATLAB-funktionerna, GetSparSDR.m (appendix D.2.2), opsdr.m (appendix D.2.3) och openblood.m, skrivna av Laura Guerrero Orozco och Xuezhi Zeng. Eftersom funktionen convertTDMS.m enbart konverterar TDMS-filerna, behövde den inte skrivas om.

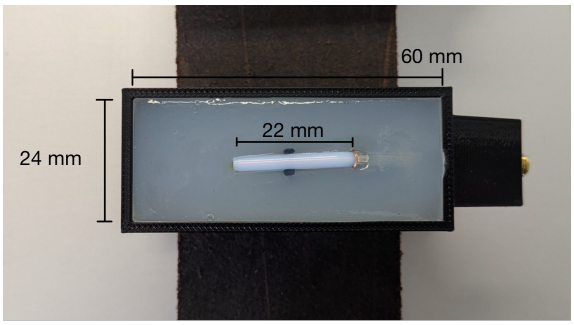
Filen openblood.m, som vidare anropar resterande funktioner i flera led, skrevs om till en funktion kallad SDR_2channel.m för att göra koden mer anpassningsbar vid olika typer av mätningar. SDR_2channel.m presenteras i appendix D.2. Flertalet förutbestämda variabler i koden ändrades till anpassningsbara variabler eller beräknades utifrån filerna. Variablerna ska specificeras vid anrop av funktionen SDR_2channel.m som tar in flera parametrar i form av de två filerna med uppmätt data, startfrekvens, steglängd, slutfrekvens, antal mätpunkter och antal antenner.

SDR_2channel.m anropar GetSparSDR.m som beräknar mottagen effekt i dB med hjälp av opsdr.m. Filen opsdr.m öppnar och konverterar ett TDMS-filpar med hjälp av convertTDMS.m och beräknar medelvärdet från referenskanalen och medelvärdet av uppmätt data. Funktionerna anropades med samma struktur som openblood.m tidigare gjort med skillnaden att opsdr.m nu enbart anropas en gång i stället för 28 gånger. Det krävdes endast ett anrop till opsdr.m eftersom mätpunkterna från varje antennkombination i de nya TDMS-filerna lade sig i kronologisk ordning i samma fil. För att hantera ändringen, omskrevs opsdr.m med hjälp av indexering och en cellmatris-struktur. GetSparSDR.m ändrades för att använda värdena sparade i cellmatrisen i stället för att anropa opsdr.m. Beräknad data skickades slutligen tillbaka till SDR_2channel.m som sedan ritade upp resultatet. Filen opsdr.m ändrades även så att de sista värdena i uppmätt data inte användes i beräkningen av medelvärdet för att undvika eventuella avvikande värden.

3.4 Prototyp för detektion av blödningar i buk/bröstkorg

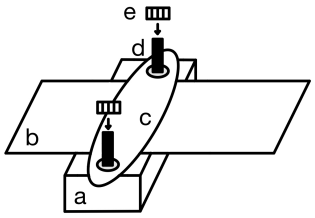
Prototypen för buk/bröstkorg konstruerades med hjälp av en 4 mm tjock läderrem med en bredd på 50 mm, ett 50 mm brett spännband i nylon och två klickspännen med längdjustering. Två 30 cm långa band skars från läderremmen och åtta hål borrades längs med läderbandens kortsidor. Varje kortsida på läderbanden syddes ihop med ett ungefär 40 cm långt spännband. Spännbanden träddes in i klickspännena som möjliggjorde smidig på- och avtagning av prototypen. Det fanns även

möjlighet att spänna åt spännbanden för att göra prototypen mindre och vice versa. Längdjusteringen var nödvändig då prototypen behövde passa för både bröstkorgen och buken samt för olika kroppsstorlekar. Åtta monopolantenner placerades i 3D-printade rektangulära behållare som fylldes med en dämpande gel för att dämpa flervägssignaler. Gelen, tillverkad av kranvatten, 1,5 wt% agar och 10 wt% NaCl, täckte hela antennen utom den sida som hade kontakt med huden, se figur 3.9.

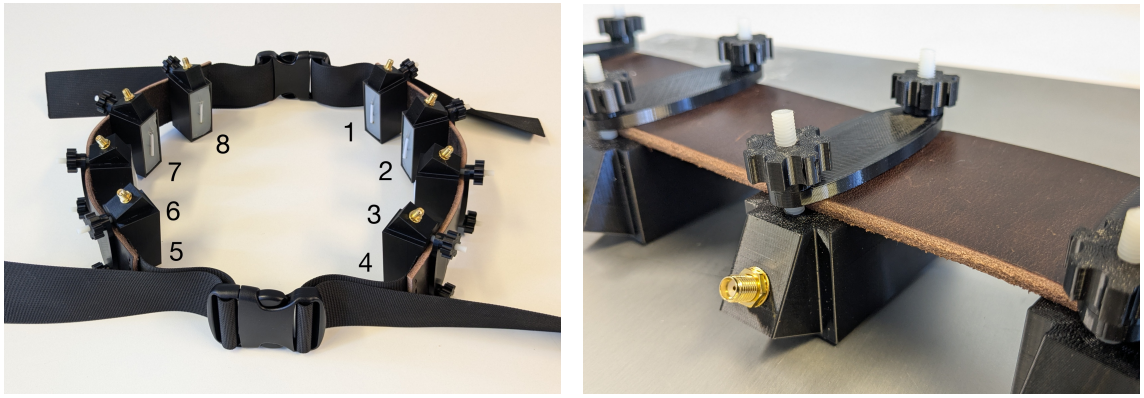


Figur 3.9: Figuren visar en antennbehållare för buk-/bröstkorgsprototypen med gel bakom antennen.

Fyra antennbehållare monterades på vardera läderband, med möjlighet att flytta antennerna längs med banden. Monteringen av antennerna illustreras i figur 3.10. Antennerna monterades genom att två plastskruvar limmades fast på baksidan av varje antennbehållare. Läderbandet placerades sedan mellan de två skruvarna och därefter placerades en täckbricka av plast med två hål på skruvarna. Till sist skruvades två muttrar i plast fast för att spänna ihop antennbehållare, läderband och täckbricka. En bild på prototypen visas i figur 3.11, där bilden till vänster visar prototypen med numrerade antenner och bilden till höger visar hur antennbehållarna var monterade.



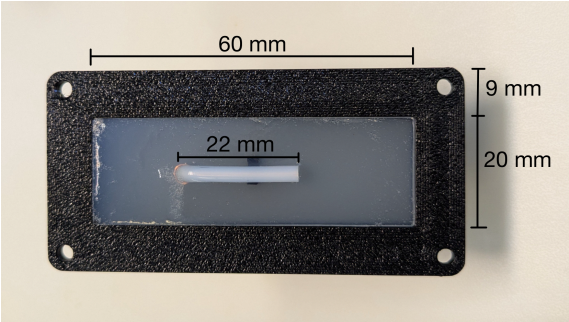
Figur 3.10: Figuren visar en skiss över hur prototypen för bröstkorgen och buken konstruerades där man kan se a) en antennbehållare, b) ett läderband, c) en täckbricka, d) två skruvar och e) två muttrar.



Figur 3.11: Prototypen för detektion av blödningar i buk/bröstkorg.

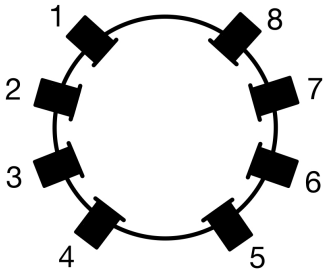
3.5 Prototyp för detektion av intrakraniella blödningar

För att bygga prototypen för intrakraniella blödningar användes en flexibel hjälm i ett skumliknande material. Hål skars i hjälmen för att ge plats åt åtta monopolantenner. På samma sätt som i avsnitt 3.4, fylldes 3D-printade antennbehållare med gel, se figur 3.12.



Figur 3.12: Figuren visar en antennbehållare för huvudprototypen med gel bakom antennen. Hålen användes för att sätta fast antennerna i hjälmen med buntband.

Fyra stycken antennbehållare placerades i en horisontell halvcirkel på vänster, respektive höger, sida av huvudet. Monteringen av antennerna illustreras i figur 3.13. Antennbehållarna sattes fast med hjälp av buntband som drogs genom hålen på antennbehållarnas kanter och genom hål som gjordes i hjälmen. Buntbanden gjorde det möjligt att smidigt kunna plocka bort alla antennbehållare för att enkelt kunna byta ut gelen. Hjälmens snörning där bak möjliggjorde justering av hjälmens storlek, vilket gjorde den anpassningsbar efter olika huvudens storlekar och former. En bild på prototypen visas i figur 3.14. I figuren visas snörningen till höger i bilden, vilken möjliggjorde åtdragning av hjälmen så att antennerna fick kontakt med huvudet. Det fanns även ett spännband runt hakan på hjälmen, vilket ytterligare möjliggjorde en stabil passform.



Figur 3.13: Figuren visar en skiss, sedd ovanifrån, över hur prototypen för huvudet konstruerades där man kan se hur fyra stycken antennbehållare placerades i en halvcirkel på båda sidor. Siffrorna beskriver antennernas positionering.

Figur 3.14: Prototypen för detektion av intrakraniella blödningar.

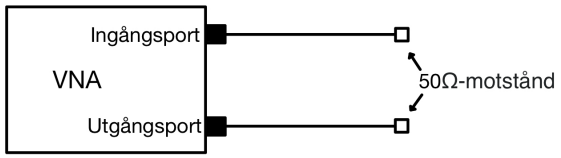
3.6 Tester

Andra delen av projektet innefattade testning av switchen, SDR:en och de två prototyperna. På switchen gjordes tester med en VNA i syfte att undersöka dämpningen och isolationen i switchen. Tester utfördes även på SDR:en, här i stället med LabVIEW, för att undersöka överhörningen mellan kanalerna och SDR:ens mätintervall. Tester med dämpare utfördes på switchen ihop med SDR:en för att undersöka deras kombinerade funktion. Tester på prototyperna utfördes med VNA:n och mätsystemet.

3.6.1 Switchens dämpning och isolation

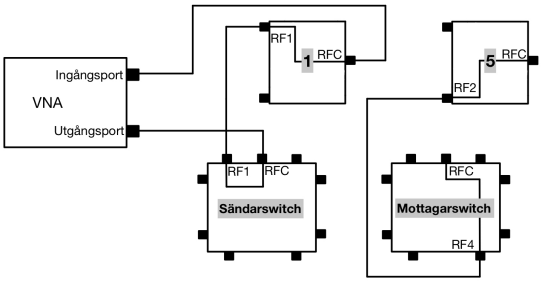
Initialt kalibrerades kablar för två portar i VNA:n enligt beskrivning i avsnitt 2.2.5. För att undersöka svagaste signalen som VNA:n kunde mäta gjordes en bakgrundsmätning med kablarna i de två portarna. Änden av respektive sladd kopplades till

varsitt 50 Ω-motstånd för att minska reflektioner, se kopplingen i figur 3.15 och resultat i avsnitt 4.2.1.

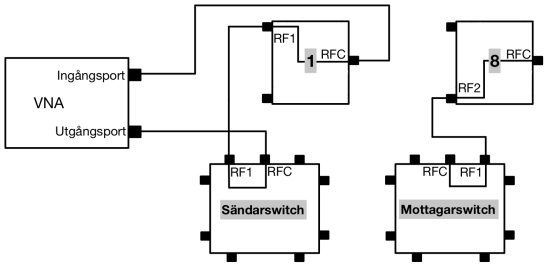


Figur 3.15: Kopplingsschemat för bakgrundsmätningen med VNA:n.

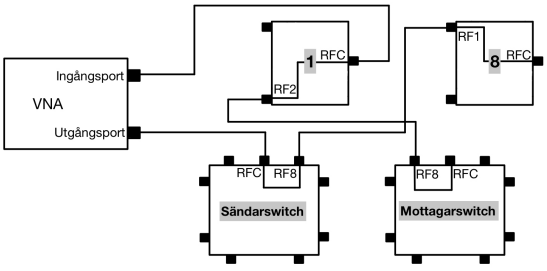
För att testa switchens dämpning kopplades SP8T-sändarswitchen till kabeln i utgångsporten på VNA:n och RFC-porten på switch 1 till kabeln i ingångsporten på VNA:n, vilket illustreras i figur 3.16. Switch 1 valdes som sändare med hjälp av koden i `Arduino.SubVI`, varpå en mätning startades med VNA:n mellan 0,4 GHz - 5 GHz. Kopplingsschemat för mätningen visas i figur 3.16a. Mätningen upprepades på resterande switchar, 2 till 8, i syfte att säkerställa att signalen dämpades lika lite oberoende av SPDT-switch. Kopplingsschemat för mätningen som utfördes på switch 4 visas i figur 3.17a. Som mottagare för samtliga mätningar användes en switch placerad långt ifrån den switch som mättes med VNA:n.



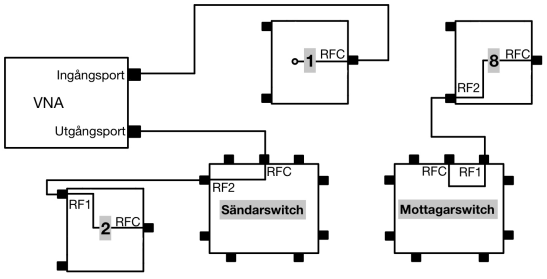
(a) Kopplingsschemat då SPDT-switch 1 användes som sändare och SPDT-switch 5 användes som mottagare.



(b) Kopplingsschemat då SPDT-switch 1 användes som sändare och SPDT-switch 8 användes som mottagare.



(c) Kopplingsschemat då SPDT-switch 8 användes som sändare och SPDT-switch 1 användes som mottagare.

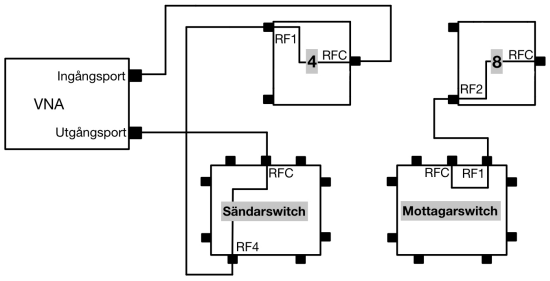


(d) Kopplingsschemat då SPDT-switch 2 användes som sändare och SPDT-switch 8 användes som mottagare.

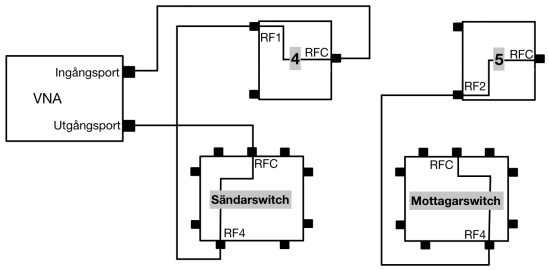
Figur 3.16: Fyra olika kopplingsscheman för mätningar med VNA då SPDT-switch 1 var inkopplad i ingångsporten på VNA. De kopplingar som visas ovan är de som är öppna för signal. Switchen är fortfarande kopplad enligt ritningen i figur 3.2.

Hela proceduren upprepades med SP8T-mottagarswitchen inkopplad i VNA för att bekräfta att den uppnådde samma dämpning som SP8T-sändarswitchen. Efterföljande mätningar utfördes endast med SP8T-sändarswitchen inkopplad i VNA:n.

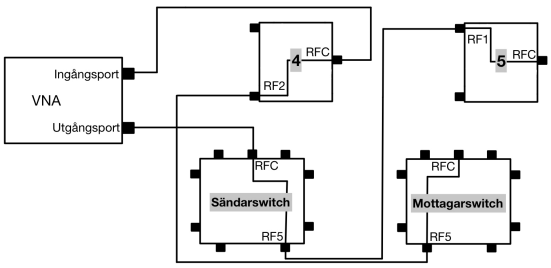
För att undersöka om signalen påverkades av vilken SPDT-switch som användes som mottagare utfördes en mätning på switch 1 där den närmaste switchen, switch 8, i stället användes som mottagare. Kopplingsschemat för mätningen visas i figur 3.16b. Mätningen utfördes även på switch 4, då switch 5 i stället användes som mottagare. Kopplingsschemat för mätningen visas i figur 3.17b.



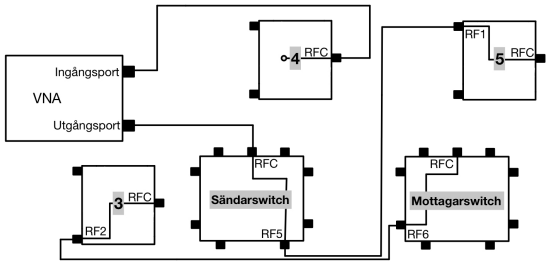
(a) Kopplingsschemat då SPDT-switch 4 användes som sändare och SPDT-switch 8 användes som mottagare.



(b) Kopplingsschemat då SPDT-switch 4 användes som sändare och SPDT-switch 5 användes som mottagare.



(c) Kopplingsschemat då SPDT-switch 5 användes som sändare och SPDT-switch 4 användes som mottagare.



(d) Kopplingsschemat då SPDT-switch 5 användes som sändare och SPDT-switch 3 användes som mottagare.

Figur 3.17: Fyra olika kopplingsscheman för mätningar med VNA då SPDT-switch 4 var inkopplad i ingångsporten på VNA. De kopplingar som visas ovan är de som är öppna för signal. Switchen är fortfarande kopplad enligt ritningen i figur 3.2.

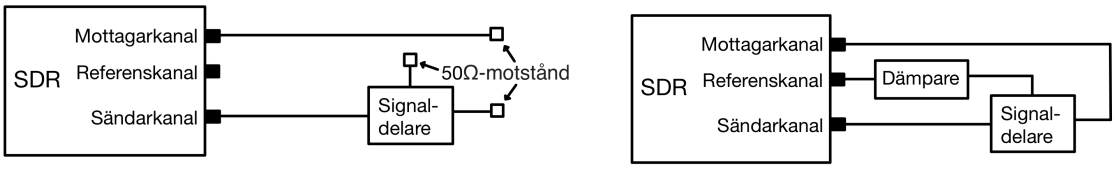
För att mäta överhörningen mellan SPDT-switcharna gjordes en mätning på switch 1 då switch 8 användes som sändare, medan switch 1 i stället användes som mottagare. Kopplingsschemat för mätningen visas i figur 3.16c. En likadan mätning gjordes på switch 4 med switch 5 som sändare. Kopplingsschemat för mätningen visas i figur 3.17c. Till sist mättes överhörningen till switch 1 då switchen var helt avstängd. Detta gjordes genom en mätning på switch 1 där switch 2 användes som sändare och switch 8 som mottagare. Kopplingsschemat för mätningen visas i figur 3.16d. En liknande mätning gjordes även på switch 4, men med switch 5 som sändare och switch 3 som mottagare. Kopplingsschemat för mätningen visas i figur 3.17d.

3.6.2 SDR:ens isolation och mätintervall

När switchens dämpning och isolation hade säkerställts utfördes tester på SDR:en i frekvensintervallet 500 MHz - 1500 MHz. Syftet med testerna på SDR:en var att mäta isolationen i kanalerna på SDR:en samt bestämma mätintervallet för SDR:en.

Kablarna som gick från SDR:en ut till de två SP8T-switcharna kopplades initialt till varsina motstånd på 50 Ω enligt kopplingsschemat i figur 3.18a. Referenskanalen kopplades ur SDR:en och kopplades i stället till ett 50 Ω-motstånd. Detta gjordes för att kontrollera nivån på bakgrundsbruset och på så sätt erhålla en nedre gräns på hur svag signal som gick att mäta. Signaldelaren som dämpade signalen med 10 dB satt kvar på sändarkanalen. På så sätt kunde brusmätningen jämföras med övriga mätningar där referenskanalen var inkopplad.

Därefter kopplades referenskanalen tillbaka in i SDR:en och ytterligare brusmätningar gjordes med 30 dB, 40 dB, 50 dB respektive 70 dB dämpare på referenskanalen. Mätningarna gjordes för att mäta isolationen i referenskanalen. Till sist kopplades kablarna med 50 Ω-motstånd ihop enligt kopplingsschemat i figur 3.18b. På så sätt erhölls en övre gräns på hur stark signal som gick att få. Testerna utfördes med 30 dB, 40 dB, 50 dB och 70 dB dämpare på referenskanalen. Resultatet från mätningarna presenteras i avsnitt 4.2.2.



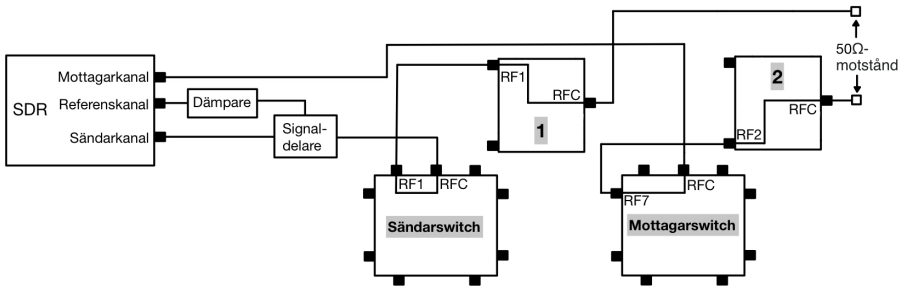
(a) Testet på SDR:en då sändarkanalen och mottagarkanalen var frånkopplade med 50 Ω-motstånd i kabeländarna. Referenskanalen var urkopplad.

(b) Testet på SDR:en då sändarkanalen och mottagarkanalen var ihopkopplade och referenskanalen inkopplad via en dämpare.

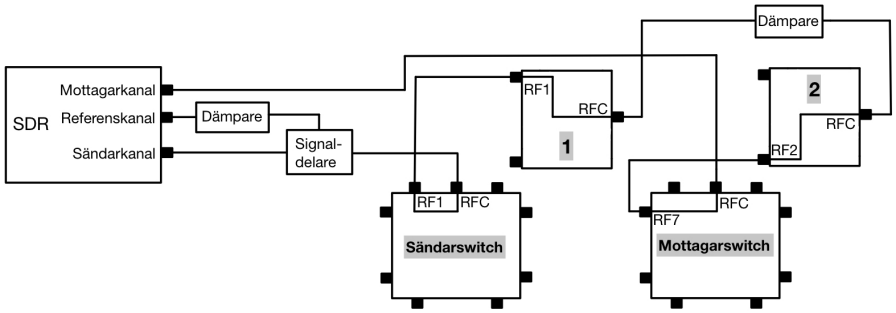
Figur 3.18: Kopplingsscheman för testerna på SDR:ens isolation och mätintervall.

3.6.3 Switchen ihop med SDR:en

Efter testerna på SDR:en gjordes mätningar på mätsystemet utan antenner. Syftet med testerna var att kontrollera hur låga signaler switchen tillsammans med SDR:en klarade av att mäta. För att mäta brusnivån för systemet kopplades kablarna från SDR:en tillbaka in i de två SP8T-switcharna. En dämpare på 50 dB kopplades in på referenskanalen. Därefter kopplades switch 1 och switch 2 via mikrovågskablar till varsina motstånd på 50 Ω och en mätning kördes. Kopplingsschemat för brusmätningen visas i figur 3.19a. Sedan kopplades switch 1 och switch 2 ihop med en mikrovågskabel enligt kopplingsschemat i figur 3.19b och en mätning kördes. Mätningen kördes först utan dämpare och sedan med en dämpare på 20 dB, 40 dB, 60 dB respektive 70 dB inkopplad mellan switch 1 och 2. Resultatet för samtliga mätningar med dämpare presenteras i avsnitt 4.2.3.



(a) Kopplingsschema för brusmätningen.



(b) Kopplingsschema för testerna med dämpare.

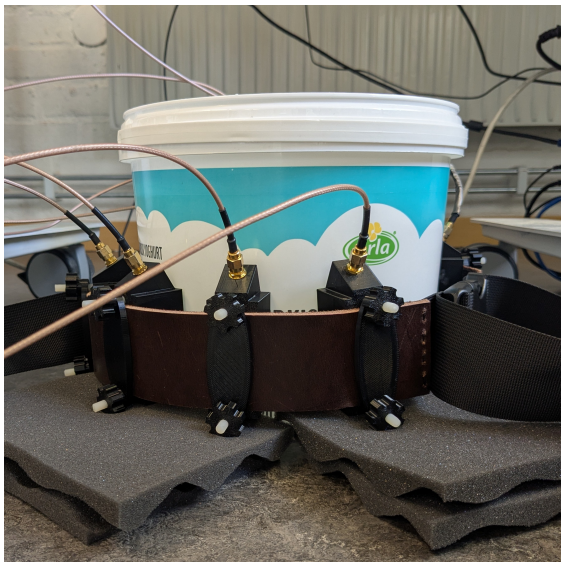
Figur 3.19: Kopplingsscheman över testerna utförda på switchen och SDR:en med olika dämpare.

3.6.4 Prototyper

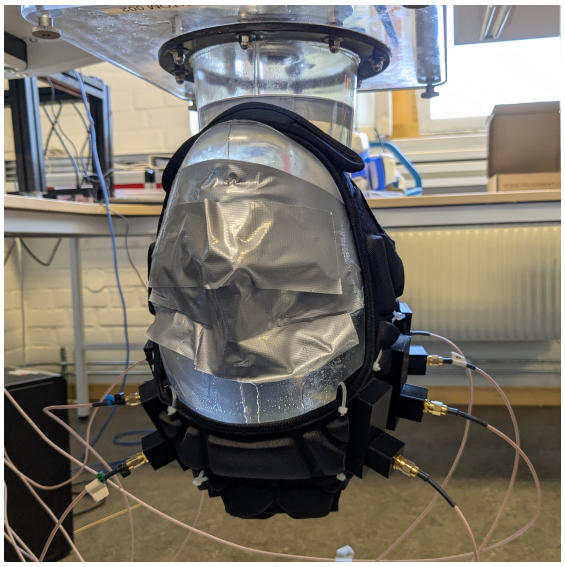
Fortsatt testning av hela mätsystemet utfördes på fantommodeller i syfte att efterlikna en mer verklig situation. En översikt av mätsystemet visas i figur 3.1. Fantommodellerna konstruerades genom att blanda vatten och glycerol. För att efterlikna de olika kroppsvävnadernas dielektriska egenskaper justerades andelen glycerol. Fantommodellen för buken/bröstkorgen konstruerades med 90 wt% glycerol och 10 wt% vatten för att uppnå en konduktivitet på 0,8 S/m och en relativ permittivitet på 20 [26]. Blandningen hälldes i en femlitershink och bältet fästes runt hinken, se figur 3.20a. För att bältet inte skulle glida ned längs hinkens något sluttande yttervägg lades skumdynor in under. Därefter kalibrerades åtta kablar i VNA:n enligt beskrivning i avsnitt 2.2.5 och kopplades in i antennerna i prototypen. För att testa signalernas styrka genom fantomen mättes S-parametrarna för samtliga antennkombinationer mellan 0,4 GHz - 1,5 GHz, se resultatet när antenn 2 sänder i avsnitt 4.12 och samtliga kombinationer i appendix A. För att testa hur väl fantomen efterliknade den mänskliga buken upprepades mätningen sedan med bältesprototypen på en frivillig gruppmedlem, se resultatet från när antenn 2 sänder i avsnitt 4.13 och samtliga kombinationer i appendix A. För att jämföra mätsystemet med VNA:n kopplades antennerna i prototypen om till switchen och en mätning på den mänskliga buken av alla antennkombinationer då SPDT-switch 2 sände gjordes med SDR:en från LabVIEW. Resultatet visas i avsnitt 4.15.

Fantommodellen för huvudet konstruerades med 70 wt% glycerol och 30 wt% vatten, vilket vid 1 GHz borde ge en konduktivitet på ungefär 1 S/m och en relativ

permittivitet på 40 [26]. Blandningen hälldes i en huvudform av plast som tätades med silvertejp och hängdes upp och ned för att motverka läckage, se figur 3.20b. Därefter kopplades de kalibrerade kanalerna i VNA:n in i antennerna i huvudprototypen. För att testa signalernas styrka genom fantomen mättes S-parametrarna för samtliga antennkombinationer mellan 0,4 GHz - 1,5 GHz, se resultatet från när antenn 2 sänder i avsnitt 4.2.5 och samtliga kombinationer i appendix A.



(a) Figuren visar prototypen för buken/ bröstkorgen på dess tillhörande fantom.



(b) Figuren visar huvudprototypen på huvudfantomen.

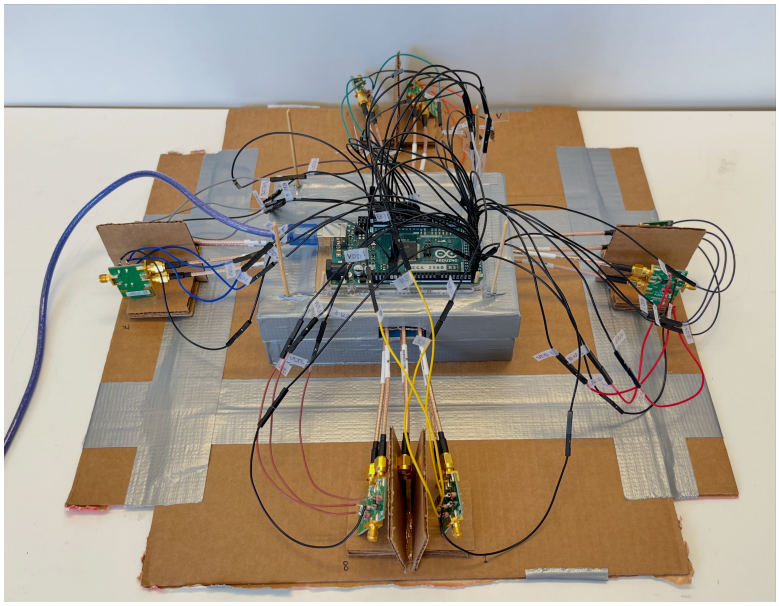
Figur 3.20: Figuren visar prototyperna på deras respektive fantommodeller.

4
Resultat

Kapitlet innehåller mätsystemets konstruktion, dess styrning och de tester som har utförts för att säkerställa dess funktion och prestanda. I kapitlet analyseras även resultaten från de utförda mätningarna.

4.1 Switchens konstruktion

Switchen, som konstruerades enligt avsnitt 3.1 i metoden, kan ses i figur 4.1. I mitten av systemet befinner sig SP8T-mottagarswitchen längst ner, monterad upp och ned. Ovanpå är SP8T-sändarswitchen placerad rättvänd och högst upp sitter Arduinon. De två SP8T-switcharna är inbyggda i en kartonglåda och en kartongbit är även placerad mellan switcharna. I kartonglådan är hål utskurna för att kunna koppla mikrovågskablar mellan SP8T-switcharna och SPDT-switcharna. SPDT-switcharna är upphöjda med kartongbitar för att mikrovågskablarna ska vara raka. Det sitter även kartongbitar mellan de SPDT-switcharna som är närmast varandra för att undvika att de kommer i kontakt. Switchen väger 764 g och har måtten 45 cm x 45 cm x 15 cm.



Figur 4.1: Figuren visar den konstruerade switchen.

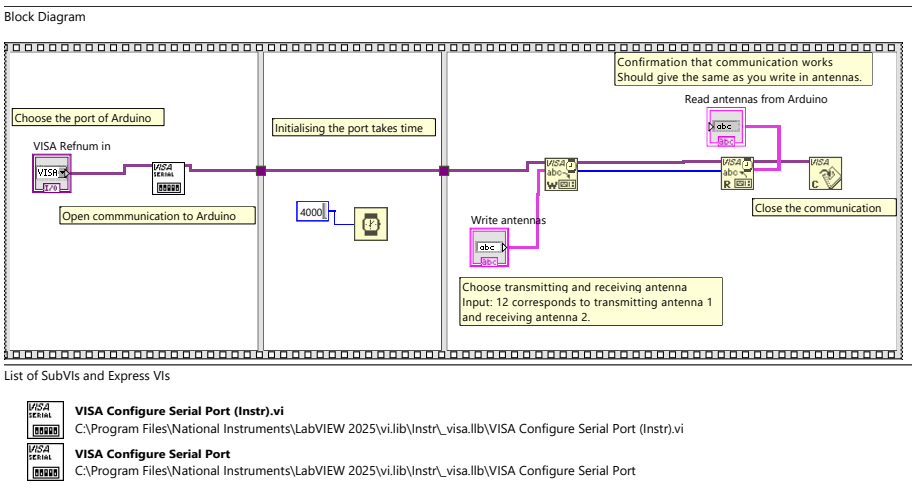
Arduinon programmerades för att ge spänning till rätt utgångar som var kopplade till styrpinnarna på SPDT- och SP8T-switcharna baserat på input från LabVIEW.

Arduinokodens uppbyggnad illustreras i form av en pseudokod nedan. Notera att modulo 2 på rad 19 är en kompensering för att SP8T-mottagarswitchen är monterad upp och ned relativt SP8T-sändarswitchen. Detta gör att alla nollor blir ettor och vice versa eftersom RF_case på rad 5 är baserad på SP8T-sändarswitchen. Den fullständiga Arduinokoden med kommentarer kan ses i appendix B.

```
1  setup: {
2      sätt alla utgångar på Arduino som output
3      sätt på spänning på alla utgångar för V_DD
4      starta kommunikation med LabVIEW
5      skapa matris RF_case över kombinationerna för styrpinnarna till SP8T-switchar
6      skapa matris switching_antenna över kombinationerna på SPDT-switcharnas ↩
           styrpinnar så att SPDT-switcharna är av
7      }
8
9  loop : {
10     om information AB tas emot från LabVIEW:
11         Tx = A (antennen som ska sända signalen)
12         Rx = B (antennen som ska ta emot signalen)
13         ändra switching_antenna så att antenn Tx sätts som sändarantenn
14         ändra switching_antenna så att antenn Rx sätts som mottagarantenn
15         skicka tillbaka Tx och Rx till LabVIEW
16         sätt på/av styrpinnar till SP8T-sändarswitch efter RF_case[Tx]
17         sätt på/av styrpinnar till SP8T-mottagarswitch efter RF_case[Rx]%2
18         sätt på/av styrpinnnar till SPDT-switcharna efter switching_antenna
19         återställ switching_antenna till ursprungsfallet i setup
20     fördröjning 1 sek
21 }
```

4.1.1 Styrning av mätsystem

Switchens uppgift är att koppla på och av rätt antenner under mätningen. För att göra detta skapades ett LabVIEW-program kallat `Arduino.SubVI`. Koden för LabVIEW-programmet presenteras i figur 4.2.

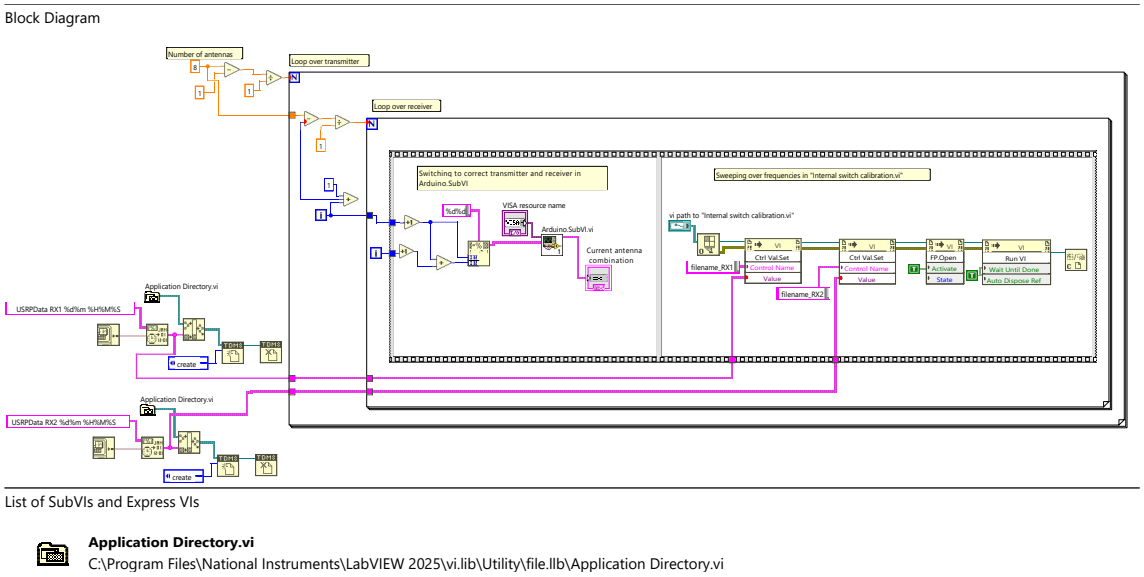


Figur 4.2: Figuren visar blockdiagrammet för LabVIEW-koden `Arduino.SubVI` som skickar instruktioner till Arduinon. För en tydligare bild se appendix C.1.

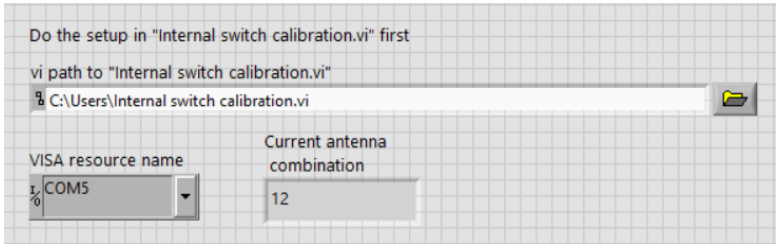
LabVIEW-programmet upprättar kommunikation med Arduinon och skickar därefter input med information om vilka antenner som ska användas. Koden är skriven i

en sekvensstruktur med en tidsfördröjning i mitten eftersom det tar tid att upprätta kommunikationen mellan LabVIEW och Arduinon.

`Arduino.SubVI` integrerades som en SubVI vid introducering av referenskanal i `Arduino_TwoChannel.vi`. Programmet `Arduino_TwoChannel.vi` styrde hela mätningen inklusive antennbyte och insamling av data i TDMS-filer. Blockdiagrammet för programmet visas i figur 4.3 och frontpanelen visas i figur 4.4. Programmet är uppbyggt så att två filer, en för referenssignalen och en för mottagarsignalen, skapas först och döps med en tidsstämpel. Därefter ligger en loop-struktur som itererar över samtliga antennkombinationer. I varje iteration används `Arduino.SubVI` för att koppla på rätt antennkombination i switchen och de två filerna öppnas i `Internal switch calibration` där mätningen sker och mätdata från samtliga frekvenser sparas.



Figur 4.3: Figuren visar blockdiagrammet för LabVIEW-koden `Arduino_TwoChannel.vi` som användes för att köra mätningar med mätsystemet.



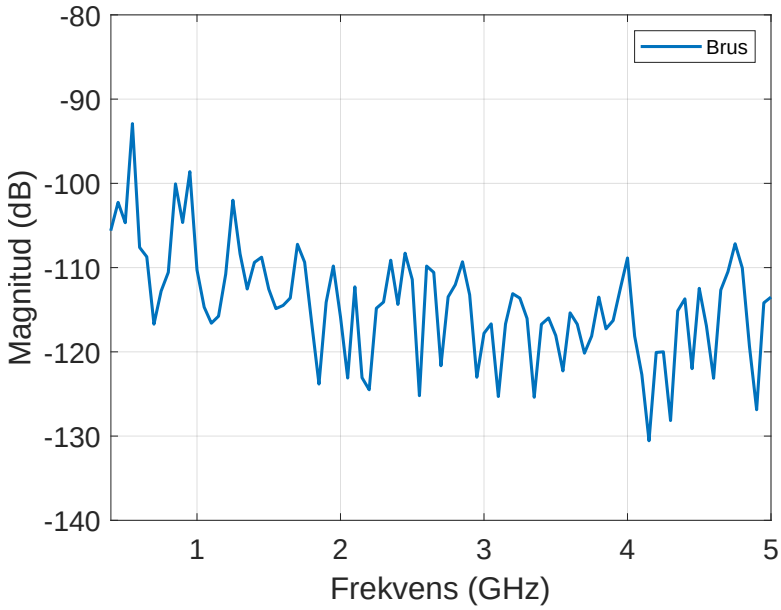
Figur 4.4: Figuren visar frontpanelen för LabVIEW-koden `Arduino_TwoChannel.vi` som användes för att köra mätningar med mätsystemet.

4.2 Tester

För att utvärdera mätsystemet har flera olika typer av tester utförts. I följande avsnitt presenteras resultaten från de tester som utförts med VNA:n på switchen, de tester som utförts på SDR:en, de tester som utförts på switchen ihop med SDR:en samt de tester som utförts på prototyperna.

4.2.1 Switchens dämpning och isolation

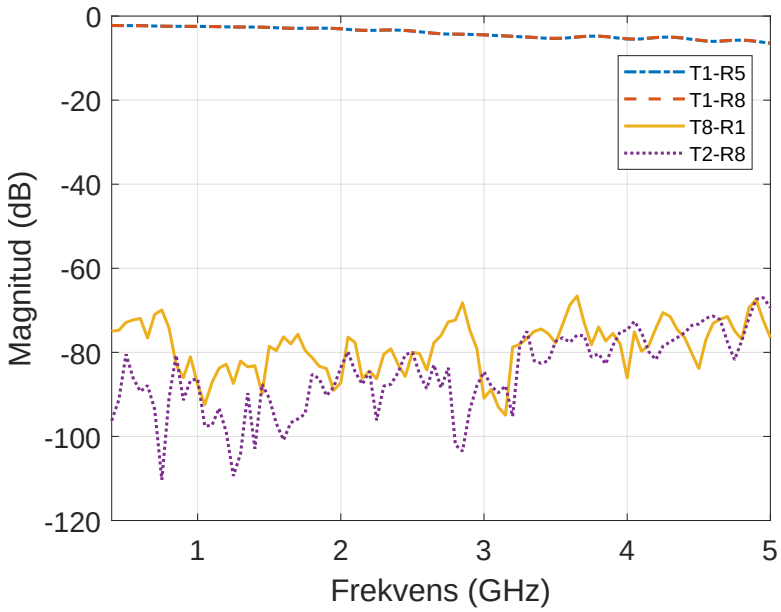
För att utvärdera dämpningen och isolationen i switchen utfördes tester med en VNA. Först gjordes en brusmätning med VNA:n enligt kopplingsschemat i figur 3.15 för att ta reda på hur svaga signaler VNA:n kan mäta. Resultat från mätningen visas i figur 4.5. Från figuren utläses att bruset startar på -105 dB och sjunker sedan med ökad frekvens till -120 dB. Resultatet innebär att VNA:n klarar av att mäta signaler ned till en dämpning på 105 dB - 120 dB.



Figur 4.5: I diagrammet visas magnituden på S-parameter för VNA:ns bakgrundsbruset som funktion av frekvensen.

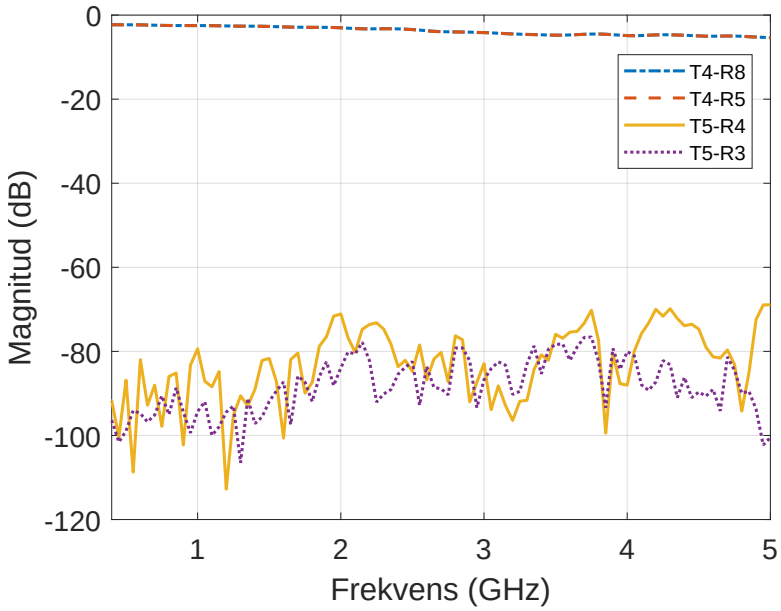
Switchens dämpning och isolation undersöktes sedan med hjälp av VNA:n. Till att börja med var SP8T-sändarswitchen och SPDT-switch 1 inkopplade i VNA:n enligt figur 3.16. I figuren visas kopplingsscheman för de fyra olika mätningarna som gjordes då olika SPDT-switchar användes som sändare respektive mottagare. I figur 4.6 plottas resultatet som erhölls från samtliga fyra mätningar. I diagrammet i figur 4.6 betecknas de olika kombinationerna som TA-RB, det vill säga att SPDT-switch A var sändare och SPDT-switch B var mottagare. I T1-R5 var switch 1 sändare och switch 5 mottagare. I frekvensintervallet 0,5 GHz - 1,5 GHz ligger kurvan T1-R5 kring -2,5 dB, vilket innebär att signalen som går genom SP8T-sändarswitchen ut till en SPDT-switch dämpas med 2,5 dB. Switch 1 var även sändare i mätningen T1-R8 då switch 8 agerade mottagare. Även kurvan T1-R8 ligger kring -2,5 dB.

Resultatet innebär att dämpningen av signalen inte påverkas av hur stort avstånd det är mellan de SPDT-switchar som agerar sändare respektive mottagare eftersom switch 8 ligger mycket närmare switch 1 jämfört med switch 5. Kurvorna för T8-R1 och T2-R8 ligger lägre och varierar kring -80 dB respektive -90 dB. I och med att varken switch 2 eller switch 8 var inkopplade i VNA:n kommer den låga signalen ifrån överhörningen mellan switch 8 och switch 1 respektive switch 2 och switch 1. Det går att se att kurvan för T2-R8 ligger lite längre ner, något som är rimligt eftersom switch 1 ligger längre ifrån switch 2 än switch 8.



Figur 4.6: Diagrammet visar fyra olika mätningar som gjordes då SP8T-sändarswitchen och SPDT-switch 1 var kopplade till VNA:n. En kurva TA-RB visar resultatet då SPDT-switch A var i sändarläge och SPDT-switch B var i mottagarläge.

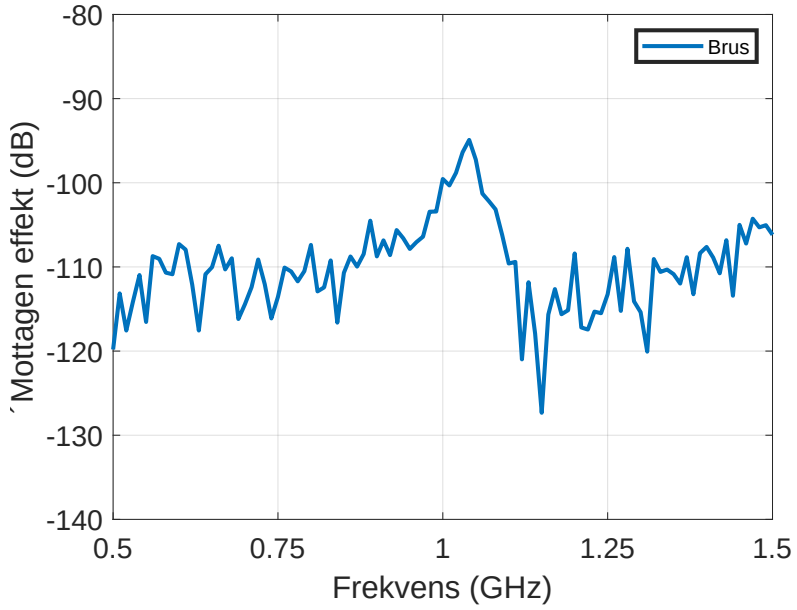
Liknande mätningar som utfördes på SPDT-switch 1 med VNA:n gjordes även på SPDT-switch 4 för att säkerställa att det erhållna resultatet gällde för samtliga SPDT-switchar. Under mätningarna på SPDT-switch 4 var SP8T-sändarswitchen och switch 4 inkopplade i VNA:n enligt figur 3.17. I figuren visas kopplingsscheman för de fyra olika mätningarna som gjordes då olika SPDT-switchar användes som sändare respektive mottagare. Resultaten från de fyra mätningarna visas i figur 4.7. Från figuren ses att kurvorna T4-R8 och T4-R5, då switch 4 var sändare och switch 8 respektive 5 var mottagare, ligger runt -2,5 dB. Resultatet, ihop med motsvarande mätningar på switch 1 (T1-R5 och T1-R8 presenterade ovan), innebär att dämpningen i signalen som går genom SP8T-sändarswitchen ut till en SPDT-switch alltid dämpas 2,5 dB oavsett vilka SPDT-switchar som agerar sändare respektive mottagare. Kurvorna T5-R4 och T5-R3 är lägre och ligger båda kring -80 dB. Eftersom switch 5 inte var inkopplad i VNA:n kommer signalerna från överhörningen mellan switch 5 och switch 4. Att T5-R4 inte är högre än T5-R3 tyder på att överhörningen primärt sker mellan SPDT-switcharna och inte SP8T-switcharna. Detta eftersom det inte spelar någon roll om kanalen från SP8T-mottagarswitchen till switch 4 är öppen eller stängd. Kurvorna visar att isolationen för den konstruerade switchen ligger på ungefär 80 dB, något som även stärks av resultaten i figur 4.6.



Figur 4.7: Diagrammet visar fyra olika mätningar som gjordes då SP8T-sändarswitchen och SPDT-switch 4 var inkopplade i VNA:n. En kurva TA-RB visar resultatet då SPDT-switch A var i sändarläge och SPDT-switch B var i mottagarläge.

4.2.2 SDR:ens isolation och mätintervall

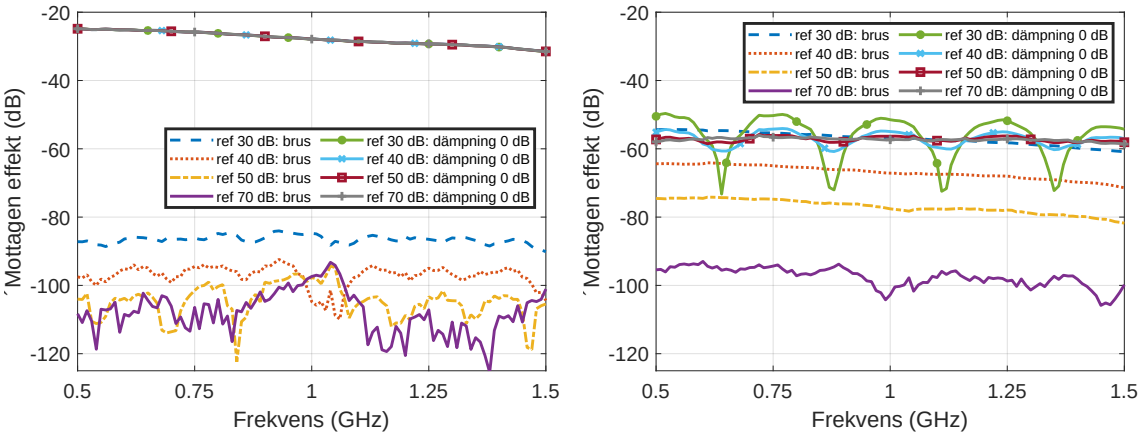
För att kontrollera isolationen och mätintervallet på SDR:en gjordes först en brusmätning när referenskanalen inte användes, se kopplingsschema i figur 3.18a. Brusmätningen gjordes utan referenskanalen för att säkerställa att signalen från referenskanalen inte påverkade brusnivån. Resultatet kan ses i figur 4.8 där brusnivån ligger runt -110 dB.



Figur 4.8: Figuren visar brusnivån för SDR:en när ingen referenskanal användes.

Ytterligare brusmätningar gjordes med referenskanalen inkopplad via dämpare på

30 dB, 40 dB, 50 dB respektive 70 dB för att undersöka isolationen i referenskanalen. Därefter gjordes mätningar med sändar- och mottagarkanalen ihopkopplade och med olika dämpare, 30 dB, 40 dB, 50 dB respektive 70 dB, på referenskanalen (se kopplingsschema i figur 3.18b). Resultaten från dessa mätningar tillsammans med brusmätningarna plottas i figur 4.9. Signalen erhållen på mottagarkanalen och referenskanalen är uppdelade i varsina plottar, 4.9a respektive 4.9b.



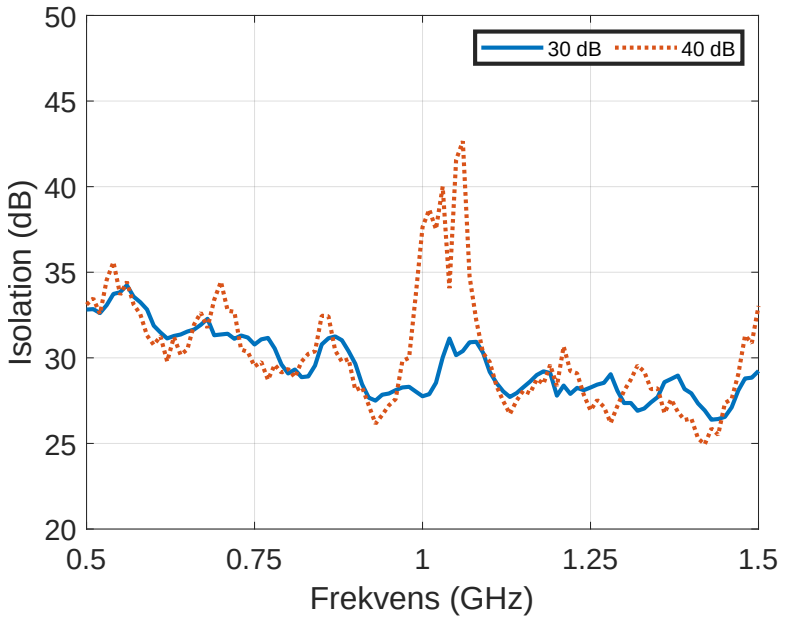
(a) Mottagarsignal **(b)** Referenssignal

Figur 4.9: Figuren visar mottagarsignalen i a) och referenssignalen i b) för SDR:en då signalen var ihopkopplad med 0 dB dämpning samt då signalen var frånkopplad. Mätningarna har genomförts med 30 dB, 40 dB, 50 dB samt 70 dB dämpning på referenskanalen.

I figur 4.9a visas den mottagna effekten på SDR:ens mottagarkanal. De fyra övre kurvorna visar signalen då sändar- och mottagarkanalen är direkt ihopkopplade och referenskanalen har 30 dB, 40 dB, 50 dB respektive 70 dB dämpning. De fyra nedre kurvorna visar signalen då sändar- och mottagarkanalen är frånkopplade och referenskanalen har 30 dB, 40 dB, 50 dB respektive 70 dB dämpning. De kurvor erhållna från de ihopkopplade mätningarna har samma utseende samt ligger på samma mottagna effekt, -25 dB till -32 dB. Anledningen till det samstämmiga utseendet är att signalen på mottagarkanalen är mycket stark för de ihopkopplade mätningarna, vilket gör att den inte påverkas av läckage från referenskanalen. De fyra kurvorna erhållna från brusmätningarna skiljer sig mer. Brusmätningen då referenskanalen hade 30 dB dämpning ligger mellan -85 dB och -90 dB och brusmätningen då referenskanalen hade 40 dB dämpning ligger mellan -95 dB och -100 dB. De sista två brusmätningarna ligger omkring -105 dB. Om kanalerna i SDR:en hade varit helt isolerade från varandra hade alla fyra kurvor legat på samma nivå och nivån hade stämt överens med brusmätningen i figur 4.8. Eftersom 30 dB brusmätningen och 40 dB brusmätningen ligger högre upp tyder det på att det förekommer signalläckage från referenskanalen till mottagarkanalen i SDR:en. 50 dB brusmätningen och 70 dB brusmätningen stämmer däremot förhållandevis bra överens med brusmätningen i figur 4.8, vilket tyder på att det endast förekommer ytterst lite alternativt inget signalläckage från referenssignalen vid de här mätningarna.

I figur 4.9b visas den mottagna effekten på SDR:ens referenskanal. De fyra kurvorna märkta med symboler visar signalen då sändar- och mottagarkanalen är ihopkopplade och referenskanalen har 30 dB, 40 dB, 50 dB respektive 70 dB dämpning. Det kan observeras att kurvorna för 40 dB, 50 dB respektive 70 dB dämpning är mycket lika vilket visar att det inte är referensens signal som mäts utan läckaget från mottagarsignalen. Det ska dock noteras att detta är ett teoretiskt test; i en verklig situation bör mottagarsignalen aldrig vara större än referenssignalen och därmed inte orsaka läckage. Kurvorna ligger också på samma nivå som brusmätningen med 30 dB dämpning på referenskanalen, vilket tyder på att mottagarkanalen på SDR:en har en isolation på 30 dB vid en signal på -25 dB. Isolationen gör att läckaget kan interferera med referensignalen när sändar- och mottagarkanalen är ihopkopplade och dämpningen på referenskanalen är 30 dB. Interferensen kan ses på formen av grafen ”ref 30 dB: dämpning 0 dB” i figur 4.9b. Tendensen av interferens syns även när 40 dB är kopplat på referenskanalen. De fyra kurvorna utan symboler visar signalen då sändar- och mottagarkanalen är frånkopplade och dämpningen på referenskanalen är 30 dB, 40 dB, 50 dB respektive 70 dB. Det kan observeras att skillnaden i mottagen effekt mellan kurvorna motsvarar skillnaderna på dämpningen i referenskanalen, vilket är väntat.

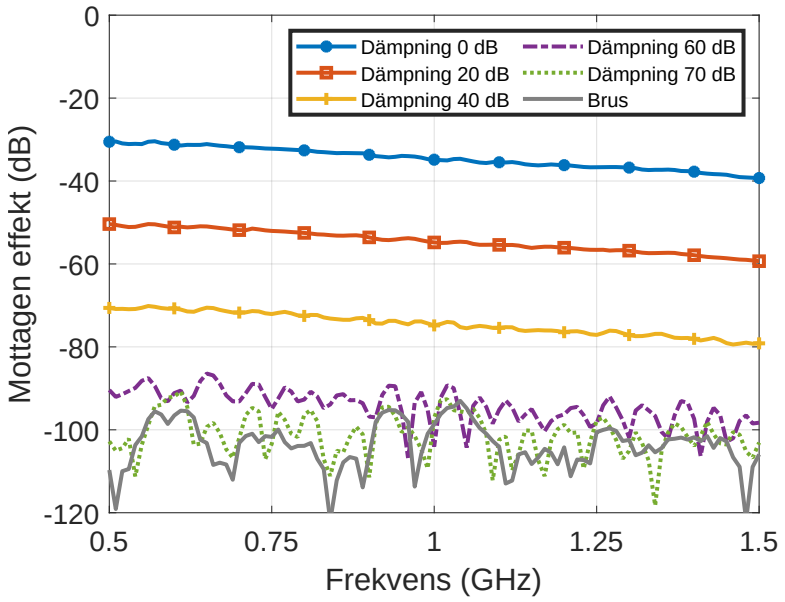
I figur 4.10 visas en sammanställd bild av resultatet i figur 4.9 där kurvorna visar isolationen i referenskanalen när referenskanalen har 30 dB respektive 40 dB dämpning. Isolationen har beräknats genom att ta skillnaden mellan signalen erhållen på referenskanalen för brusmätningen (från figur 4.9b) och signalen erhållen på mottagarkanalen för brusmätningen (från figur 4.9a). Kurvorna visar att isolationen i referenskanalen ligger omkring 30 dB och är oberoende av hur mycket dämpning som sitter på referenskanalen. Isolationen 30 dB innebär att läckaget från referenskanalen till mottagarkanalen är signalen som går in i referenskanalen minus 30 dB. Fortsättningsvis har alla mätningar med SDR:en genomförts med 50 dB dämpning på referenssignalen eftersom inget läckage då förekommer från referenskanalen till mottagarkanalen. 50 dB valdes i stället för 70 dB eftersom lägre dämpning på referenskanalen ger en högre referenssignal och därmed fås ett högre signal-brusförhållande. Ett högt signal-brusförhållande är att föredra då det leder till en mer korrekt och konsekvent faskalibrering. Då 50 dB dämpning används på referenskanalen utläses det ur figur 4.9a att mätintervallet för signalen ligger mellan -25 dB och -105 dB. Då referenskanalen har 50 dB dämpning går det alltså att mäta signaler ned till en dämpning på 80 dB med SDR:en.



Figur 4.10: Diagrammet visar en sammanställning av isolationen i referenskanalen på SDR:en framtaget från resultaten i figur 4.9.

4.2.3 Switchen ihop med SDR:en

För att undersöka hur låga signaler switchen ihop med SDR:en klarade av att mäta utfördes mätningar med olika dämpare samt en brusmätning enligt kopplingsscheman i figur 3.19. Figur 4.11 visar de erhållna resultaten för mätningarna. Kurvorna visar signalen erhållen på mottagarkanalen som funktion av frekvensen. Alla mätningar är utförda med SPDT-switch 1 kopplad som sändare och SPDT-switch 2 kopplad som mottagare. Uppifrån och ned visar diagrammet den erhållna signalen då SPDT-switcharna är ihopkopplade med 0 dB dämpning, 20 dB dämpning, 40 dB dämpning, 60 dB dämpning samt 70 dB dämpning. Den sista kurvan visar en brusmätning då SPDT-switcharna var kopplade till varsina 50 Ω-motstånd. Kurvan då dämpningen var 0 dB startar på -30 dB och går till -39 dB. Som en jämförelse låg signalen mellan -25 dB och -32 dB för mätningen då SDR:ens sändar- och mottagarkanal var ihopkopplade (figur 4.9a). Signalen tappar alltså ungefär 5 dB när den går genom switchen, något som stämmer väl överens med den uppmätta dämpningen på switchen som visade att signalen tappar 2,5 dB enkel väg (figur 4.6 och 4.7). Vidare ses i figur 4.11 att effekten minskar med 20 dB för varje 20 dB dämpare som kopplas in mellan SPDT-switcharna vilket är ett väntat resultat. Vid 60 dB går kurvan att urskilja från bruset, även om den börjar bli mer oförutsägbar än de med mindre dämpning. Vid 70 dB ligger kurvan precis ovanför bruset vid lägre frekvenser men vid högre frekvenser går det inte längre att urskilja signalen från bruset. Sammantaget visar resultatet att switchen ihop med SDR:en klarar av att mäta signaler ned till 70 dB dämpning. För svagare signaler begränsar både SDR:ens kapacitet att mäta svaga signaler och switchens isolation mätningen.

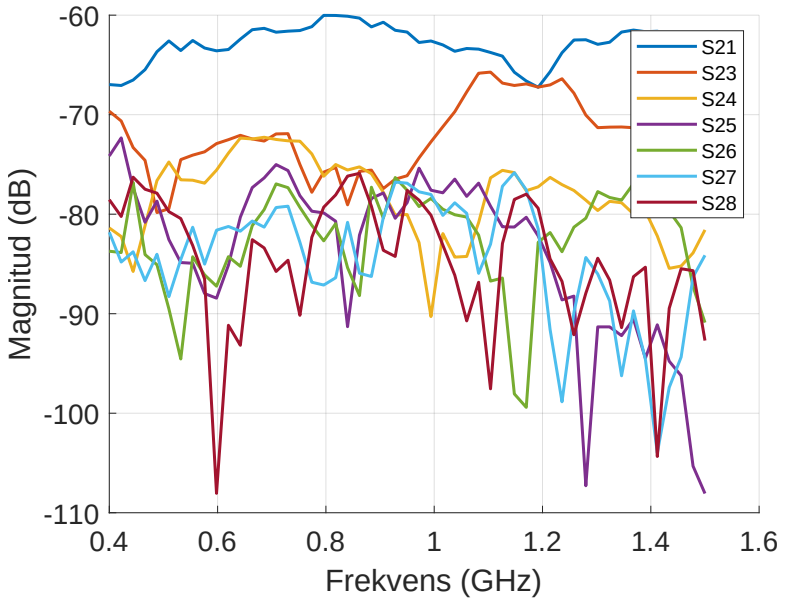


Figur 4.11: Diagrammet visar sex olika mätningar utförda med switchen och SDR:en. Samtliga mätningar är utförda med SPDT-switch 1 som sändare och SPDT-switch 2 som mottagare. Uppifrån och ned visar diagrammet den erhållna signalen på mottagarkanalen, då SDPT-switcharna är ihopkopplade med olika stor dämpning. Den sista kurvan visar en brusmätning då SPDT-switcharna var frånkopplade.

4.2.4 Prototyp för buk/bröstkorg

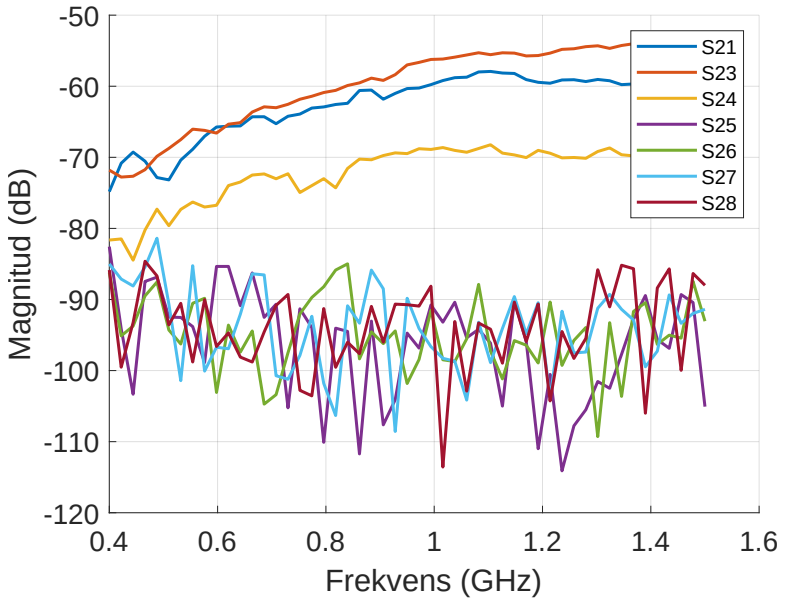
Med bältet gjordes tre olika typer av tester; två med VNA:n samt ett test med mätsystemet. Det ena testet med VNA:n gjordes på fantommodellen för buk/bröstkorg, medan det andra testet gjordes på en frivillig gruppmedlem. Även testet med mätsystemet gjordes på den frivilliga gruppmedlemmen.

Buk-/bröstkorgsprototypen fästes på bukfantomen enligt figur 3.20a och signalernas styrka genom fantomen testades med VNA:n genom att koppla in de åtta antennerna på bältet direkt till VNA:n. Figur 4.12 visar magnituden på S-parametrarna för de erhållna signalerna som funktion av frekvensen då frekvensen går mellan 0,4 GHz - 1,5 GHz. Varje kurva i figuren har en beteckning SAB som beskriver vilka antenner signalen går emellan. SAB innebär att signalen skickades ut i antenn A och togs emot i antenn B. Diagrammet visar samtliga antennkombinationer då signalen skickades ut genom antenn 2, se figur A.1 i appendix för resultaten på övriga antennkombinationer. I diagrammet ses att kurvan S21 har högst magnitud, följt av kurvan S23. Notera att S21 är ekvivalent med S12. Det är rimligt att magnituderna för signalerna mellan antenn 2 och de antenner som sitter närmast antenn 2 på prototypen, S21 och S23, har en högre magnitud än de övriga antennkombinationerna. I frekvensintervallet 0,4 GHz - 1 GHz ligger kurvan S23 på ungefär samma magnitud som kurvorna för de antennkombinationer där avståndet mellan antennerna är större. Möjliga anledningar till att signalen för S23 är lägre än förväntat diskuteras i avsnitt 5.5.



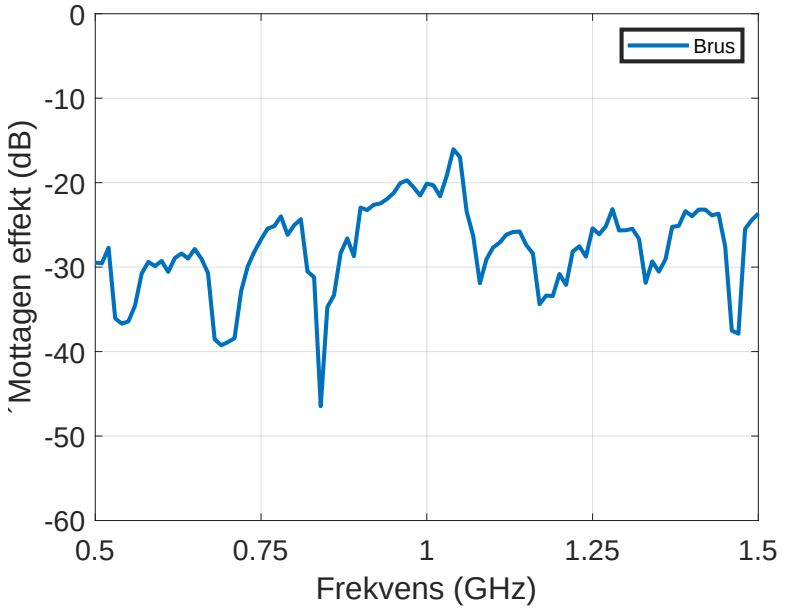
Figur 4.12: Diagrammet visar magnituden på S-parametrarna för de erhållna signalerna från en mätning utförd med buk-/bröstkorgsprototypen med VNA:n då antenn 2 var sändare. För magnituden på S-parametrarna för samtliga signaler se figur A.1 i appendix. Prototypen var under mätningen placerad på fantommodellen för buk/bröstkorg.

Det testades även hur väl fantomen efterliknar den mänskliga buken genom att upprepa ovanstående mätning men med prototypen på en frivillig gruppmedlems buk. I figur 4.13 visas de erhållna resultaten från antennkombinationerna S21-S28, se mätningens resultat från samtliga antennkombinationer i figur A.2 i appendix. I diagrammet i figuren avläses att magnituden på signalen mellan antenner som sitter nära varandra på prototypen, S21 och S23, har högst magnitud och har liknande form. Även magnituden på signalen mellan antenn 2 och 4, S24, går tydligt att urskilja från övriga antennkombinationer. Så var inte fallet i mätningen då buk-/bröstkorgsprototypen var placerad på fantomen (se figur 4.12). Om kurvorna S21 och S23 jämförs mellan försöket med buk-/bröstkorgsprototypen på fantommodellen och försöket på en mänsklig buk, inses att magnituderna är högre då prototypen satt på en mänsklig buk. Anledningar till avvikelser mellan mätningen på fantomen och mätningen på den mänskliga buken diskuteras vidare i avsnitt 5.5. Signalen för de antenner som satt på motsatt sida av buken gentemot antenn 2 (S25, S26, S27 och S28) har lägre magnitud och ligger på ungefär samma nivå.



Figur 4.13: Figuren visar magnituden på S-parametrarna för de erhållna signalerna från en mätning utförd med VNA:n då buk-/bröstkorgsprototypen satt på en mänsklig buk och antenn 2 var sändare. För magnituden på S-parametrarna för samtliga signaler se figur A.2 i appendix.

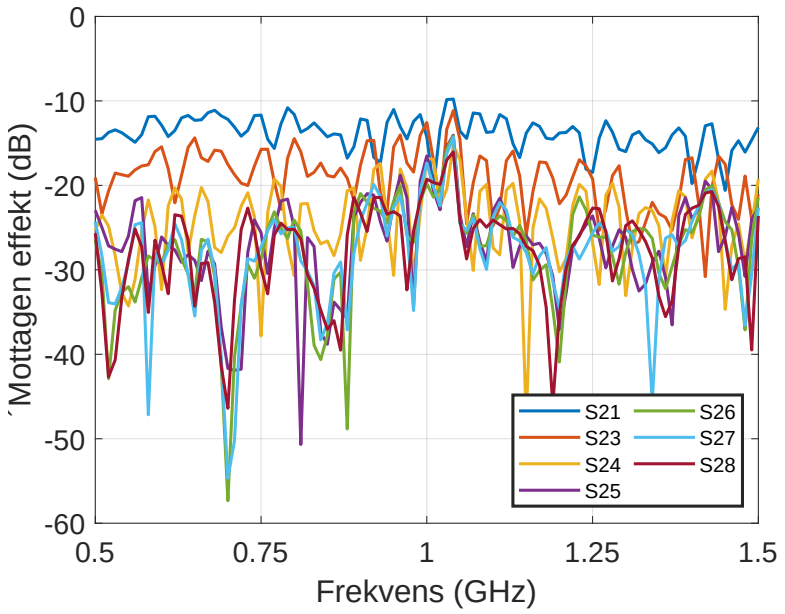
För att undersöka det faskalibrerade bruset från switchen ihop med SDR:en gjordes en mätning med koppling enligt figur 3.19a. Resultatet från mätningen kan ses i figur 4.14 där kurvan visar att brusnivån för den faskalibrerade signalen ligger runt -30 dB.



Figur 4.14: Figuren visar en brusmätning från switchen ihop med SDR:en då signalen är faskalibrerad.

För att jämföra mätsystemet och VNA:n kopplades bältesprototypen till switchen när prototypen satt kvar på den frivilliga gruppmedlemmen. En översikt av kopplingen visas i figur 3.1. I figur 4.15 visas det erhållna resultatet då antenn 2 användes

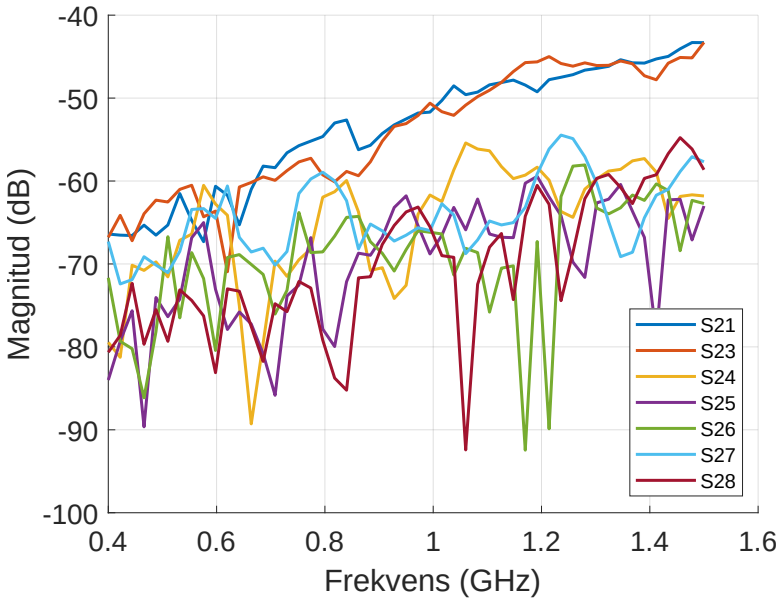
som sändare. Från diagrammet ses att signalerna mellan de antenner som sitter närmast varandra i prototypen, S21 och S23, utmärker sig med en högre mottagen effekt. Resterande kurvor (S24, S25, S26, S27 och S28) går inte att urskilja från bruset då de ligger på samma nivå som bruset för switchen och SDR:en som visas i figur 4.14.



Figur 4.15: Figuren visar mätresultaten erhållna med mätsystemet då buk-/bröstkorgsprototypen satt på en mänsklig buk och antenn 2 var sändare.

4.2.5 Prototyp för huvud

Prototypen för huvudet fästes på huvudfantomen enligt figur 3.20b och signalens styrka genom fantomen mättes med VNA:n genom att koppla in de åtta antennerna direkt till VNA:n. Resultatet illustreras i figur 4.16 som visar magnituden på S-parametrarna för de erhållna signalerna för olika antennkombinationer som funktion av frekvensen då frekvensen går mellan 0,4 GHz - 1,5 GHz. Beteckningen SAB indikerar att kurvan kommer från mätningen då signalen skickades ut i antenn A och togs emot i antenn B. Här illustreras samtliga antennkombinationer då signalen skickades ut på antenn 2. Se figur A.3 i appendix för resultatet från samtliga antennkombinationer. I diagrammet ses att kurvorna S21 och S23 har en högre magnitud än resterande kurvor, och de ligger dessutom väldigt nära varandra. Att det är starkare signal för antennkombinationerna 21 och 23 än för övriga antennkombinationer är förväntat eftersom antenn 1 och 3 är de antenner som är placerade närmast antenn 2 på prototypen. Antennerna 1 och 3 är symmetriskt placerade på varsin sida om antenn 2, vilket stämmer väl överens med att kurvorna S21 och S23 har liknande form.



Figur 4.16: Diagrammet visar magnituden på S-parametrarna för de erhållna signalerna från en mätning utförd med huvudprototypen med VNA:n då antenn 2 var sändare. För magnituden på S-parametrarna för samtliga signaler se figur A.3 i appendix. Under mätningen var prototypen placerad på fantommodellen för huvudet.

5
Diskussion

I kapitlet diskuteras innebörden av de erhållna resultaten. Switchen, SDR:en, hela mätsystemet samt prototyperna utvärderas. Det tas även upp möjliga felkällor och vidareutvecklingar av projektet.

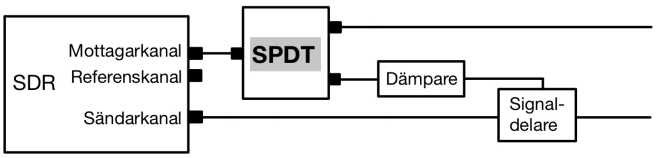
5.1 Switchen

Switchens fysiska konstruktion har utvecklingspotential. SP8T-switcharna och Arduinon sitter fast väl i konstruktionen och riskerar inte att lossna vid användning eller förflyttning av mätsystemet. Hela mätsystemet är dock byggt på en kartongbotten som är relativt svag, vilket fungerar bra i testmiljö, men behöver förbättras innan prehospital användning. SPDT-switcharna är placerade i par med en kartongvägg emellan, men sitter för övrigt endast fast i kablarna som är kopplade till SP8T-switcharna och kablarna till Arduinon. Stabiliteten för SPDT-switcharna behöver därför också förbättras innan prehospital användning. Den konstruerade switchen är portabel. Switchen väger 764 g och har storleken 45 cm x 45 cm x 15 cm. Framförallt är Arduinon och de två SP8T-switcharna ihopbyggda på en liten yta. SPDT-switcharna är däremot placerade lite längre ifrån resterande delar eftersom längden på mikrovågskablarna begränsar hur nära SPDT-switcharna kan sitta SP8T-switcharna. Med kortare kablar har systemet möjlighet att bli mer kompakt eftersom SPDT-switcharna då hade hamnat närmare systemets centrum. Med SPDT-switcharna närmare centrum har systemet även möjlighet att bli mer robust eftersom SPDT-switcharna inte hade varit lika utsatta vid exempelvis förflyttning.

Vid VNA-mätningar av switchen, se avsnitt 4.2.1, fastslogs att switchen dämpar signaler med cirka 5 dB, vilket är ett bra resultat. Isolationen för switchen låg däremot runt 80 dB, vilket uppkommer från överhörningen mellan SPDT-switcharna. För att möjliggöra bättre mätningar med prototyperna behöver således isolationen på switchen förbättras. För att göra detta behöver överhörningen mellan SPDT-switcharna i mätsystemet minskas genom att addera extern isolation runt varje switch. Ett förslag är att bygga in switchen i metall eller något annat ledande material och jorda materialet. Det ledande materialet stänger in mikrovågorna så att de inte kan ta sig ut till resterande SPDT-switchar. Däremot skulle lösningen eventuellt kunna skapa problem med den enskilda SPDT-switchens interna isolation då det är fler mikrovågor i rörelse runt switchen. Ett annat förslag på en lösning är att bygga in switchen i ett absorberande material. På så sätt absorberas mikrovågorna så att de inte plockas upp av de andra SPDT-switcharna.

5.2 SDR:en

I resultatet för SDR:en fastslogs att en relativt stor dämpning på referenskanalen krävs för att undvika överhörning från referenskanalen till mottagarkanalen. För att förbättra systemets prestanda behöver överhörningen minskas i SDR:en för lägre dämpningar på referenskanalen. Med lägre dämpning blir referenssignalen högre vilket gör att signal-brusförhållandet blir högre och därmed fås en mer pålitlig faskalibrering. Det finns olika sätt att minska överhörningen i SDR:en. En lösning kan vara att sätta en SPDT-switch vid mottagaringången på SDR:en och koppla in mottagarkanalen och referenskanalen i den. Se figur 5.1 för ett kopplingsschema över den tilltänkta förändringen. Genom användning av en SPDT-switch tas signalen emot på endast en kanal i SDR:en, vilket förhindrar överhörning i SDR:en. SPDT-switchen har en isolation på typiskt 60 dB, vilket är avsevärt högre än isolationen mellan kanalerna i RF1 i SDR:en som uppmättes till 30 dB. Den här lösningen är att föredra eftersom det skulle kunna tillåta en sänkning av dämpningen från 50 dB till 30 dB på referenssignalen.

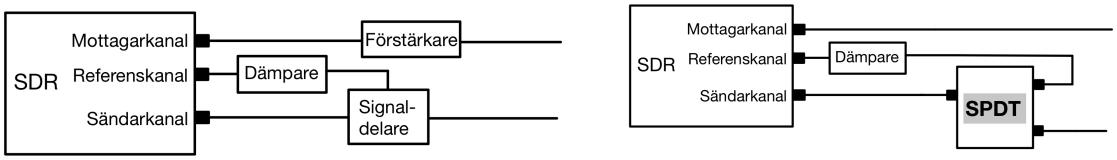


Figur 5.1: Kopplingsschema över en potentiell förbättring av SDR:ens överhörning. Mottagarsignalen och referenssignalen går via en SPDT-switch till en ingång på SDR:en.

I resultatet för SDR:en konstaterades även att SDR:en kan mäta signaler ned till en dämpning på 80 dB. Då signalen dämpas mer än så går det inte att urskilja signalen från SDR:ens brus. En lösning för att kunna mäta signaler med högre dämpning än 80 dB är att koppla in en förstärkare på mottagarkanalen, se kopplingsschema i figur 5.2a. På så sätt kan de mottagna signalerna förstärkas så att de hamnar i SDR:ens mätintervall. En förstärkare på mottagarkanalen kan också göra att den mottagna signalen blir större än läckaget från referenskanalen, vilket hade kunnat lösa problemet med överhörning mellan kanalerna i SDR:en.

En annan möjlig förbättring för att öka SDR:ens kapacitet att mäta svaga signaler och minska överhörningen i SDR:en är att koppla in en SPDT-switch i stället för signaldelaren på sändarkanalen, vilket visas i kopplingsschemat i figur 5.2b. Signaldelaren som användes dämpar båda signalerna 10 dB medan en SPDT-switch inte dämpar signalen märkbart. På så sätt förstärks signalen 10 dB om signaldelaren byts till en SPDT-switch. Bytet skulle troligtvis också minska överhörningen i SDR:en eftersom SPDT-switchen har en bättre isolation. Den höga isolationen i SPDT-switchen gör att SDR:en endast tar emot den signal som ska mätas. En möjlig komplikation med att byta signaldelaren mot en SPDT-switch är att SPDT-switchen behöver styras med Arduinon. Eftersom SPDT-switchen behöver kopplas om då mätningen ska ta emot signaler på mottagarkanalen i stället för referenskanalen kommer ett kommando behöva skickas från LabVIEW till Arduinon. Först

efter omkopplingen kan mätningen börja ta emot mätdata på mottagarkanalen. Omkopplingen innebär att LabVIEW behöver kommunicera med Arduinon dubbelt så många gånger som i nuläget, det vill säga två gånger för varje antennkombination. Något som tar tid. Eftersom en mätning över alla antenner i nuläget kan ta upp till 30 minuter, beroende på hur många frekvenser som mäts, behöver mättiden förbättras om SPDT-switchen implementeras.



(a) Kopplingsschema där en förstärkare är insatt innan mottagarkanalen.

(b) Kopplingsschema där signaldelaren är utbytt mot en SPDT-switch.

Figur 5.2: Kopplingsscheman över potentiella förbättringar av SDR:ens kapacitet att mäta svaga signaler och SDR:ens överhörning.

5.3 Switchen ihop med SDR:en

I resultatet för switchen ihop med SDR:en som utvärderades med hjälp av olika dämpare fastslogs att mätsystemet klarar av att mäta signaler ned till 70 dB dämpning. Signaler som dämpas mer än 70 dB går inte att urskilja från bruset. Brusnivån uppkommer som ett resultat av switchens isolation och SDR:ens kapacitet att mäta svaga signaler. Om mätsystemet ska klara av att mäta svagare signaler behöver således dessa två begränsningar förbättras. Förslag på förbättringar presenterades i avsnitten 5.1 och 5.2.

5.4 Prototyper

Prototyperna konstruerades med justeringsmöjligheter för att kunna anpassas efter individen vid varje mätning. Dock finns begränsningar gällande anpassningsgraden. Storleken på hjälmen är anpassningsbar med ett snöre, medan antennerna ej går att justera. Vid hjälmstorlekens ytterlägen finns det därför risk att antennerna inte är utspridda optimalt runt huvudet, vilket skulle kunna påverka mätresultaten negativt. Alla komponenter och antenner är väl fästa på båda prototyperna som förväntas vara robusta vid det tänkta användningsområdet.

Bältet har större justeringsmöjligheter än hjälmen, i form av både bältesstorlek och antennjusteringar. Storleken är enkel att justera med spännen, medan antennerna sitter fast med ett skruvsystem. Det är viktigt att antennerna placeras symmetriskt, vilket kan kräva flera antennförflyttningar vid justering. För att möjliggöra en mindre bältesstorlek är läderbanden endast 30 cm långa, vilket begränsar ytan där antennerna kan placeras. Som en följd placeras antennerna närmare än önskat då maximal bältesstorlek används och intressanta mätområden kan därmed missas.

Utifrån projektets begränsade testning anses justeringen av prototypernas passform fungera väl, men vid fortsatt utveckling behövs vidare testning på fler individer för att säkerställa passformen. Dessutom håller gelen som omger antennerna enbart i några dagar. Förbättring gällande hållbarhet för gelen är därför aktuellt vid fortsatt utveckling av prototyperna.

5.5 Prototyptester

I resultatet från VNA-mätningarna med buk-/bröstkorgsprototypen på bukfantomen konstaterades att signalen med högst magnitud erhölls mellan de antenner som var placerade bredvid varandra i prototypen. Även i VNA-mätningarna med buk-/bröstkorgsprototypen på den mänskliga buken erhölls starkast signaler mellan de intilliggande antennerna i prototypen. På den mänskliga buken kunde även signalen mellan antenn 2 och 4 urskiljas trots att de inte satt direkt bredvid varandra på prototypen. En starkare signal mellan antenn 2 och 4 på den mänskliga buken tyder på att buk-/bröstkorgsfantomen inte helt efterliknar en mänsklig buk. En förklaring till detta kan vara att yttersta delen av bukfantomen bestod av en plasthink, vilken omgav blandningen med vatten och glycerol. Eftersom plasten var hård och inte formade sig efter prototyperna blev det en glipa mellan plasten och den dämpande gelen. Det var därmed svårt att få antennerna helt omslutna av gelen, vilket kan ha orsakat att mikrovågorna propagerade i oönskade riktningar. Även formen på hinken kan ha orsakat problem under mätningarna. Diametern på hinkens botten var mindre än diametern på toppen av hinken, vilket gjorde att buk-/bröstkorgsprototypen enkelt gled nedåt. Detta gjorde att prototypen inte var tätt åtsittande runt bukfantomen, vilket gav sämre kontakt mellan bukfantomen och antennerna med dämpande gel. En ytterligare förklaring kan vara symmetri, där en mänsklig buk är mer ovalformad än bukfantomen som var placerad i en cirkulär hink.

Sammantaget från testerna med VNA på buk-/bröstkorgsprototypen bedöms prototypen fungera i enlighet med förväntningarna, men att bukfantomen var bristfällig. Ett förslag för att förbättra bukfantomen är att använda agar för att skapa en fantom gjord av ett gelématerial. En fantom i gelé är mer formbar, som den mänskliga buken, vilket ger bättre kontakt mellan fantomen och antennerna. Det bör också göras en djupare undersökning av vilken andel glycerol som är optimal att använda.

Vid jämförelse mellan mätningarna med buk-/bröstkorgsprototypen på en mänsklig buk med VNA kontra mätsystemet konstateras att resultaten är liknande men att VNA:n kan urskilja fler signaler, se figur 4.13 och figur 4.15. Resultatet är förväntat baserat på SDR:ens mätintervall och switchens isolation. Det går att se i båda figurerna att antennerna närmast varandra ger starkast signal. Detta visar på att mätsystemet fungerar och har stor potential. Vidareutveckling av systemet bör fokusera på att öka isolationen i switchen och förbättra SDR:ens kapacitet att mäta svaga signaler, förslagsvis genom de metoder som presenterades ovan i avsnitt 5.1 och 5.2.

Vid VNA-mätningar på huvudfantomen med hjälmprototypen konstaterades att

signalen med högst magnitud erhölls på mottagarantenn 1 och 3 då antenn 2 var sändare, se figur 4.16. Resultatet är rimligt eftersom antenn 1 och 3 ligger närmast antenn 2 utifrån deras placering i hjälmprototypen. De ligger dessutom på ungefär samma avstånd ifrån varandra utifrån hjälmens symmetriska utformning, vilket förklarar deras kurvors lika utseende. Vid jämförelse mellan mätningarna på de två olika fantomerna med VNA:n konstateras att resultatet på huvudfantomen uppvisar starkare signaler. Resultatet kan bero på kortare avstånd mellan antenner, men eftersom huvudfantomen uppvisar bättre resultat vad gäller likheter mellan S21 och S23 tros antennerna på huvudprototypen ha bättre kontakt med fantomen. Dessutom var formen på huvudfantomen mer lik ett mänskligt huvud än vad bukfantomen var lik en mänsklig buk. Således antas huvudprototypen fungera väl även på ett mänskligt huvud.

Nästa steg i form av tester skulle bestå av att genomföra motsvarande mätningar med fantomer som även innehåller någon typ av simulerad blödning. Genomförande av sådana mätningar skulle bekräfta hur blödningar potentiellt skulle påverka mätdata, en viktig aspekt vid fortsatt utveckling av mätsystemet.

5.6 Felkällor

Det finns ett antal statistiska felkällor som identifierades under testningen, vilka påverkar reproducerbarheten av mätningarna. Det upptäcktes skillnader i signalen vid likadana mätningar fast med olika kablar. En avgörande faktor utgjordes av kablarnas längd, vilket är förväntat, men även kablar med samma längd fungerade olika bra vid flertalet likadana mätningar. Detta förväntas inte påverka resultaten gällande funktion av systemet, men är en aspekt att ta med sig till framtida utveckling av projektet. Ett behov av separat kontrollering av kablars funktion kan därför inte uteslutas.

VNA:n temperaturregleras för att minska variationer mellan olika mätningar vilket saknas hos SDR:en. Temperaturskillander i rummet tros därför också kunna påverka stabiliteten av SDR:ens kalibrering genom att amplituden påverkas. Inverkan på resultatet minskas genom att det görs en ny kalibrering vid varje mätning.

Vid upprepade tillfällen har LabVIEW gett felmeddelandet ”Error - 1074118643 occurred at niUSRP Write Tx Data (CDB).vi. Possible reason: Packet had timestamp that was late (or too early).”. Konsekvensen är att mätningen vid den frekvensen inte kan sparas och programmet måste därför startas om. Någon tydlig anledning till varför problemet uppkommer har inte hittats men ett visst samband till parametern ”Time to Receive & Transmit First Sample”, som bestämmer en tidsgräns för att skicka och ta emot första mätpunkten, har observerats. En del av mätningarna kunde köras med parametern på 0,5 sekunder, medan andra fick ha 0,7 sekunder för att undvika felmeddelandet. All annan aktivitet på den aktuella datorn undveks också för att minska risken för felet. Vid enstaka tillfällen har LabVIEW också stängt ned hela programmet mitt i en mätning utan någon uppenbar anledning. Det anses därför relevant att vidare undersöka programmets pålitlighet inför en framtida

medicinsk tillämpning.

5.7 Vidareutveckling

Styrningen av SDR:en och kommunikationen med Arduinon kräver flera olika fördröjningar (delays) i LabVIEW-koden för att fungera korrekt. Tiden för en mätning förkortades avsevärt med hjälp av bättre anpassade fördröjningar. Samtidigt finns ytterligare möjligheter att effektivisera systemet med hjälp av noggrannare optimeringar av fördröjningarna. En liten förändring gör stor skillnad då sekvenser av koden upprepas flera gånger, exempelvis delar som upprepas för alla 28 antennkombinationer.

Mätningar utförs med LabVIEW som styr både SDR:en och Arduinon. Arduinon behöver i sin tur förprogrammeras i C++. Då mätdata sedan hanteras i MATLAB finns förbättringspotential gällande integreringen mellan alla programmen som krävs för att systemet ska fungera. Styrning av LabVIEW genom MATLAB eller liknande skulle underlätta vid vidare omfattande tester som krävs för fortsatt validering av systemet.

Även filhanteringen i systemet skulle kunna utvecklas. Omskrivningar i projektet resulterade i en minskning från över 100 filer per mätning, vilket alla datorer inte klarar av att spara, till fyra stycken. Detta bör däremot gå att halvera genom att lägga referens- och mätdata efter varandra i samma fil. Andra förbättringar som har gjorts gällande datahanteringen i MATLAB utgjordes av mer anpassningsbara funktioner som öppnar upp för en bredare användning gällande olika mätparametrar och inställningar. I nuläget behöver dock parametrar, som information gällande frekvenssvepet och antal antenner, sparas separat för att data ska kunna analyseras korrekt i MATLAB. Det finns därför en förbättringsmöjlighet genom utläsning av parametrar från datafilerna. Informationen skulle kunna sparas på ett smidigare sätt, förslagsvis genom mindre förändringar i LabVIEW-koden. De utvecklade funktionerna i MATLAB som hanterar mätresultaten skulle på så sätt kunna omarbetas så att färre parametrar krävs. Funktionerna skulle på så sätt bli mer lätthanterliga vilket skulle underlätta vid mer omfattande testning där mycket resultathantering krävs.

Utöver tidigare nämnda förbättringar av befintligt mätsystem finns flera möjligheter att utveckla systemet och inkludera fler funktioner. Systemet skulle på sikt kunna integreras med en redan existerande algoritm som kan rekonstruera bilder utifrån uppmätt data med systemet. Avbildning är ett viktigt hjälpmedel vid diagnostik. En sådan utveckling skulle öppna upp för en bredare användning av systemet inom diagnostik och ta projektet vidare, ett steg närmare det slutgiltiga målet, klinisk användning.

5.8 Slutsats

Utifrån resultaten från VNA-mätningar uppvisar den konstruerade switchen potential rörande användning vid mikrovågsbaserad diagnostisering prehospitalt. Switchen är kompakt och uppvisar en låg dämpning (5 dB) av signalen då den färdas genom switchen. En viss förbättringspotential finns vad gäller minskning av överhörningen mellan SPDT-switcharna eftersom switchens isolation uppmättes till 80 dB. En möjlig lösning för att minska dämpningen är att isolera SPDT-switcharna från varandra genom att bygga in dem.

Även överhörningen mellan kanalerna i SDR:en har förbättringspotential utifrån de erhållna resultaten. För att undvika överhörning mellan SDR:ens kanaler krävdes en dämpare på 50 dB på referenskanalen. Genom att förbättra isolationen mellan kanalerna skulle en dämpare på 30 dB kunna användas, vilket hade förbättrat resultatet vid en framtida bildframställning. Sedan behöver även SDR:ens kapacitet att mäta svaga signaler förbättras. Flera möjliga lösningar finns och inkluderar bland annat användning av en extra SPDT-switch.

Delvis jämförbara resultat uppmättes mellan VNA:n och mätsystemet. De övergripande trenderna i VNA-mätningarna kan i flertalet fall även ses i mätningarna med mätsystemet, men begränsas delvis av switchens och SDR:ens prestanda. Likheterna i resultaten tyder på att mätsystemet har en stor potential att i framtiden kunna användas för blödningsdetektion. Om mätsystemet ska kunna mäta signaler under -70 dB behöver både isolationen mellan SPDT-switcharna förbättras och SDR:ens förmåga att mäta svaga signaler ökas. Båda förbättringarna anses vara fullt möjliga att genomföra.

De två konstruerade prototyperna, en för blödningar i buk/bröstkorg och en för intrakraniella blödningar, uppnådde sina förväntade funktioner vad gäller passform och robusthet. Utifrån testerna anses även att bra mätsignaler går att erhålla med prototyperna.

Litteratur

[1] M. Persson, A. Fhager, H. D. Trefna m. fl., "Microwave-based stroke diagnosis making global prehospital thrombolytic treatment possible," *IEEE Transactions on Biomedical Engineering*, årg. 61, nr 11, 2014, ISSN: 15582531. DOI: 10.1109/TBME.2014.2330554.

[2] S. Candefjord, L. Nguyen, R. Buendia m. fl., "A wearable microwave instrument can detect and monitor traumatic abdominal injuries in a porcine model," *Scientific Reports*, årg. 11, nr 1, 2021, ISSN: 20452322. DOI: 10.1038/s41598-021-02008-5.

[3] Janusinfo. "Antikoagulantiabehandling efter intrakraniell blödning." (nov. 2024), URL: https://janusinfo.se/behandling/expertgruppsutlatanden/hjartochkarlsjukdomar/hjartochkarlsjukdomar/antikoagulantiabehandlingefterintrakraniellblodning.5.78ae827d1605526e94b8e780.html (hämtad 2025-02-13).

[4] 1177 Vårdguiden. "Traumatisk hjärnskada (THS)." (juni 2023), URL: https://vardpersonal.1177.se/kunskapsstod/vardforlopp/traumatisk-hjarnskada-ths/ (hämtad 2025-02-13).

[5] Hjärnfonden. "Så här behandlas stroke." (juli 2019), URL: https://www.hjarnfonden.se/2019/07/sa-har-behandlas-stroke/ (hämtad 2025-02-27).

[6] S. Chennareddy, R. Kalagara, C. Smith m. fl., "Portable stroke detection devices: a systematic scoping review of prehospital applications," *BMC Emergency Medicine*, årg. 22, nr 1, s. 111, juni 2022, ISSN: 1471-227X. DOI: 10.1186/s12873-022-00663-z.

[7] D. Weishaupt, A. M. Grozaj, J. K. Willmann, J. E. Roos, P. R. Hilfiker och B. Marincek, "Traumatic injuries: imaging of abdominal and pelvic injuries," *European Radiology*, årg. 12, nr 6, s. 1295–1311, juni 2002, ISSN: 0938-7994. DOI: 10.1007/s00330-002-1462-7.

[8] R. Hood, M. Persson, A. Fhager m. fl., "Evaluating the sensitivity and specificity of a microwave based tool to support and enhance stroke triage," *International Journal of Stroke*, årg. 17, nr 3_suppl, s. 64–65, okt. 2022, Abstract O136 / #1314, ISSN: 1747-4930. DOI: 10.1177/17474930221125973.

[9] IngaBritt och Arne Lundbergs Forskningsstiftelse, "MIKROVÅGOR GÖR DET ENKLARE ATT UPPTÄCKA TUMÖRER OCH BLÖDNING," tekn. rapport, 2020.

[10] N. Khalid, M. Zubair, M. Q. Mehmood och Y. Massoud, "Emerging paradigms in microwave imaging technology for biomedical applications: unleashing the power of artificial intelligence," *npj Imaging*, årg. 2, nr 1, juni 2024. DOI: 10.1038/s44303-024-00012-8.

[11] J. Liu, L. Chen, H. Xiong och Y. Han, "Review of microwave imaging algorithms for stroke detection," *Medical & Biological Engineering & Computing*, årg. 61, nr 10, s. 2497–2510, okt. 2023, ISSN: 0140-0118. DOI: 10.1007/s11517-023-02848-5.

[12] A. Fhager, S. Candefjord, M. Elam och M. Persson, "Microwave Diagnostics Ahead: Saving Time and the Lives of Trauma and Stroke Patients," *IEEE Microwave Magazine*, årg. 19, nr 3, s. 78–90, maj 2018, ISSN: 1527-3342. DOI: 10.1109/MMM.2018.2801646.

[13] L. G. Orozco, L. Peterson och A. Fhager, "Microwave Antenna System for Muscle Rupture Imaging with a Lossy Gel to Reduce Multipath Interference," *Sensors*, årg. 22, nr 11, 2022, ISSN: 14248220. DOI: 10.3390/s22114121.

[14] B. Emilov, A. Sorokin, M. Seiitov m. fl., "Diagnostic of Patients with COVID-19 Pneumonia Using Passive Medical Microwave Radiometry (MWR)," *Diagnostics*, årg. 13, nr 15, s. 2585, aug. 2023, ISSN: 2075-4418. DOI: 10.3390/diagnostics13152585.

[15] R. Chandra, H. Zhou, I. Balasingham och R. M. Narayanan, "On the Opportunities and Challenges in Microwave Medical Sensing and Imaging," *IEEE Transactions on Biomedical Engineering*, årg. 62, nr 7, s. 1667–1682, juli 2015, ISSN: 0018-9294. DOI: 10.1109/TBME.2015.2432137.

[16] D. Singh, M. Särestöniemi och T. Myllylä, "Microwave Technique for Brain Monitoring: State of the Art, Promising Applications, Challenges, and Future Directions," *IEEE Access*, årg. 13, s. 35 183–35 202, 2025, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2025.3544032.

[17] National Instruments, *USRP-2901 Specifications*, maj 2023. URL: https://www.ni.com/docs/en-US/bundle/usrp-2901-specs/page/specs.html?srsltid=AfmBOop45cv5DNOUZMI4Aou7_ew7RRnqULmeoIuaM-YxZQmB-6pVir_j.

[18] X. Zeng och L. G. Orozco, "Measurement quality of a software defined radio system for medical diagnostics," *The Journal of Engineering*, årg. 2022, nr 12, s. 1162–1172, dec. 2022, ISSN: 2051-3305. DOI: 10.1049/tje2.12196.

[19] Arduino, *Arduino® Mega 2560 Rev3*, april 2025. URL: https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf.

[20] SparkFun Electronics, *Arduino Mega 2560 R3*, sept. 2015. URL: https://www.flickr.com/photos/41898857@N04/20658673643 (hämtad 2025-05-09).

[21] Analog Devices, *HMC8038: High Isolation, Silicon SPDT, Nonreflective Switch, 0.1 GHz to 6.0 GHz*, version Rev. A, 2015. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/HMC8038.pdf (hämtad 2025-02-07).

[22] Analog Devices, *HMC253ALC4: GaAs MMIC SP8T Non-Reflective Switch, DC - 3.5*, version v00.0413, 2015. URL:

https://www.analog.com/en/products/hmc253alc4.html#documentation (hämtad 2025-02-07).

[23] M. Sayed och J. Martens, "Vector network analyzers," i *Modern RF and Microwave Measurement Techniques*, V. Teppati, A. Ferrero och M. Sayed, utg. Cambridge University Press, 2013, s. 98–129.

[24] R. Pollard och M. Sayed, "Transmission lines and scattering parameters," i *Modern RF and Microwave Measurement Techniques*, V. Teppati, A. Ferrero och M. Sayed, utg. Cambridge University Press, 2013, s. 3–20.

[25] Rohde & Schwarz, *R&S®ZNBT VECTOR NETWORK ANALYZER: Specifications*, version 14.00, aug. 2024. URL: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/pdm/cl_brochures_and_datasheets/specifications/3606_9727_22/ZNBT_specs_en_3606-9727-22_v1400.pdf.

[26] P. M. Meaney, C. J. Fox, S. D. Geimer och K. D. Paulsen, "Electrical Characterization of Glycerin: Water Mixtures: Implications for Use as a Coupling Medium in Microwave Tomography," *IEEE Transactions on Microwave Theory and Techniques*, årg. 65, nr 5, s. 1471–1478, maj 2017, ISSN: 0018-9480. DOI: 10.1109/TMTT.2016.2638423.

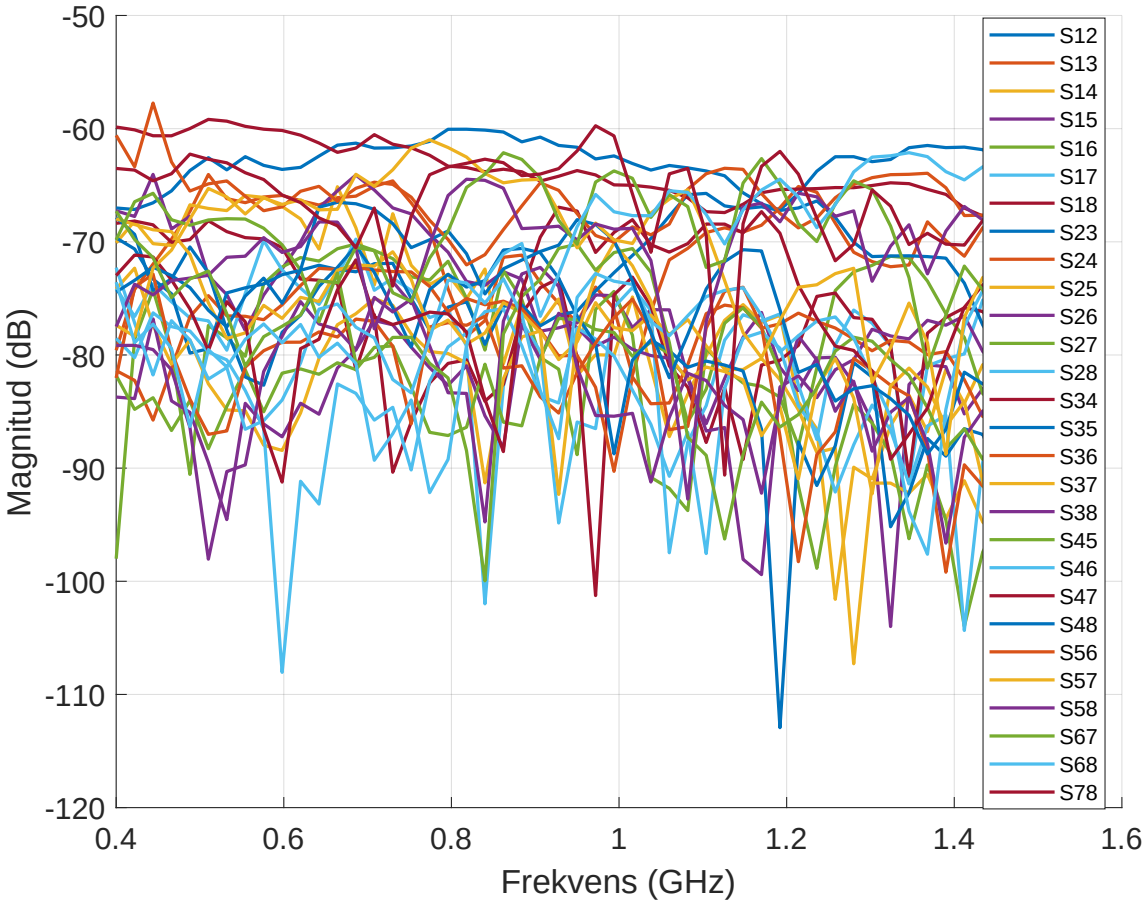
A

Appendix: Mätdata

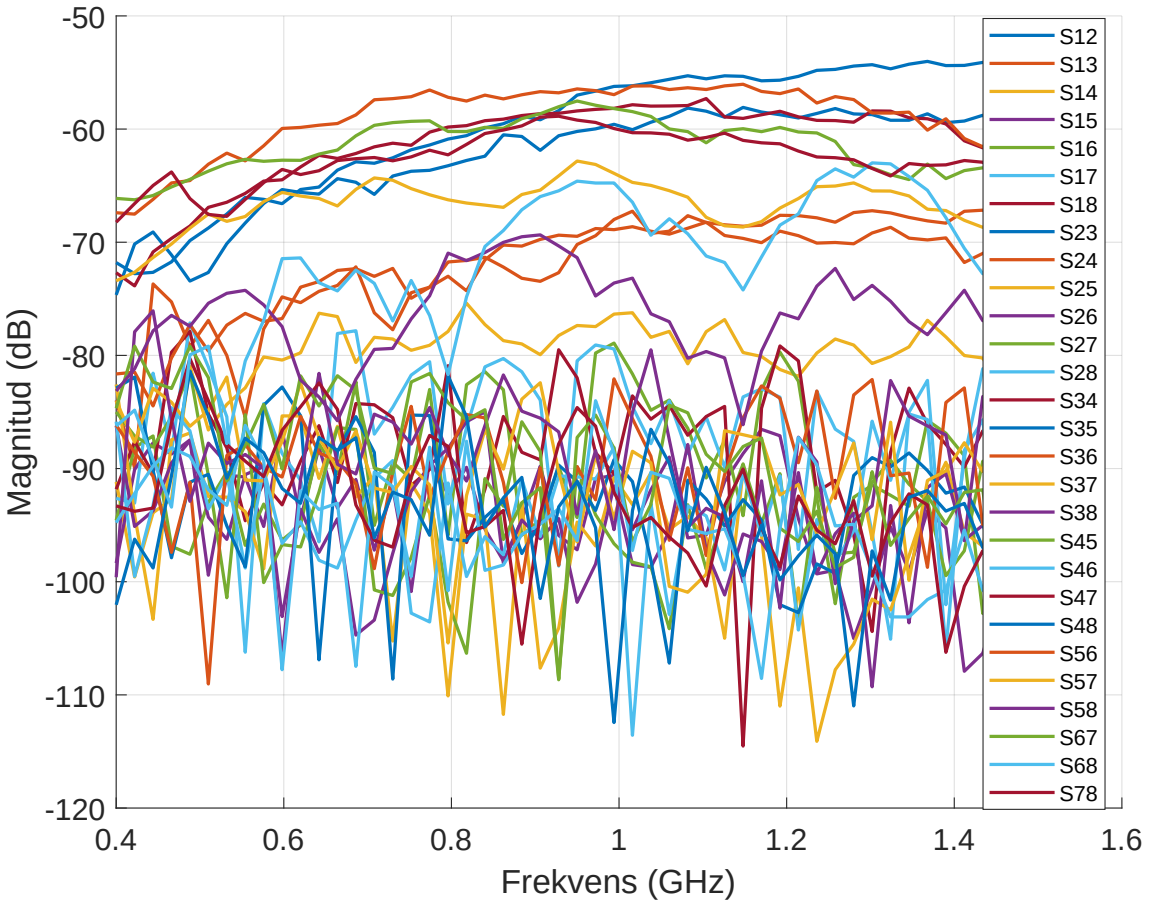
Nedan presenteras all uppmätt data. Ett utdrag presenteras i resultatet, alla mätningar inkluderande antenn 2, se 4.2.4 och 4.2.5.

A.1 Buk-/bröstkorgsprototyp

Mätdata från mätning med VNA på en bukfantom ses i figur A.1. Motsvarande mätning på en mänsklig buk visas i figur A.2.



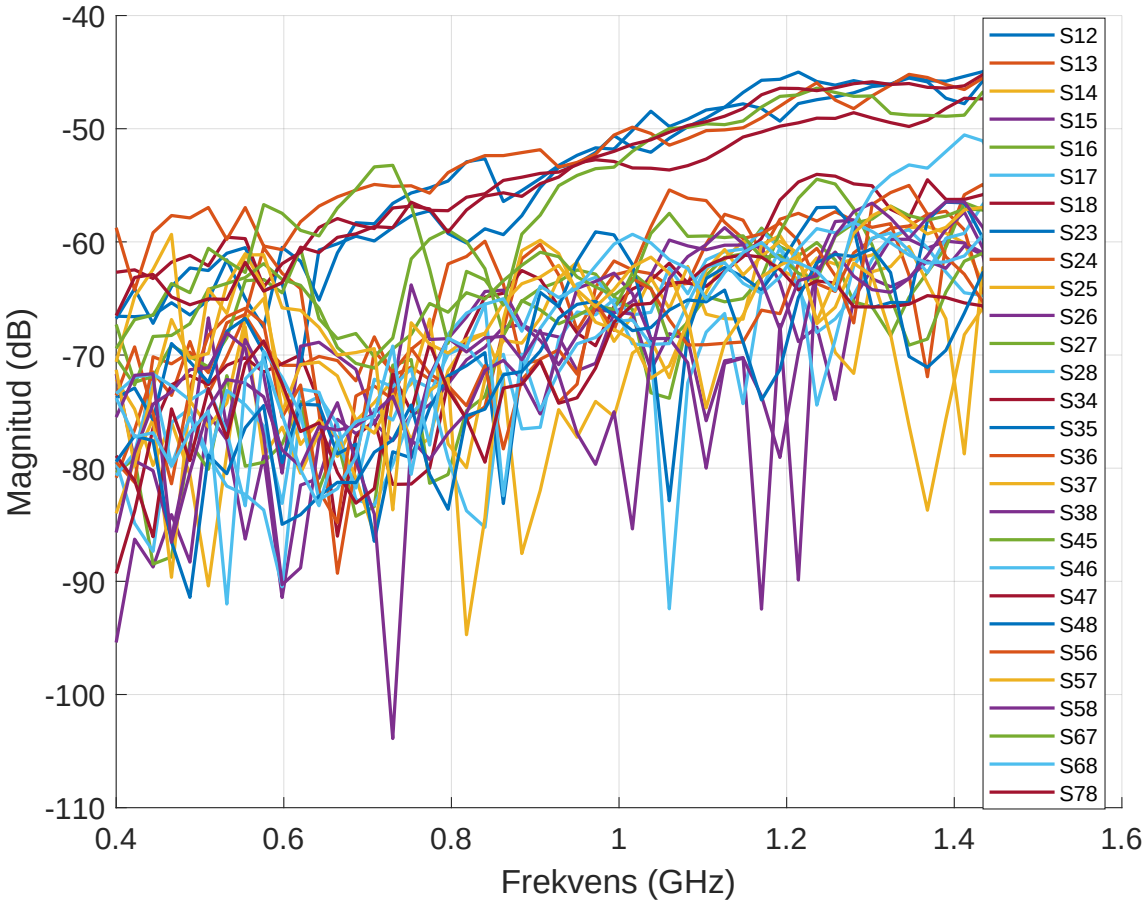
Figur A.1: Diagrammet visar magnituden för de erhållna signalerna då en mätning utförts med buk-/bröstkorgsprototypen med VNA:n. Prototypen var under mätningen placerad på fantommodellen för buken/bröstkorgen.



Figur A.2: Figuren visar mätresultaten erhållna från VNA:n då buk-/bröstkorgsprototypen satt på en mänsklig buk.

A.2 Huvudprototyp

En mätning utförd på en huvudfantom med hjälmprototypen och VNA:n visas i figur A.3.



Figur A.3: Diagrammet visar de erhållna signalernas magnitud för en mätning utförd med huvudprototypen med VNA:n. Under mätningen var prototypen placerad på fantommodellen för huvudet.

B

Appendix: Arduino-kod

Nedan presenteras Arduino-koden som har skrivits för att styra switcharna.

```
// Choose pins
const int output_pins[] = {2,3,4,5,6,7, 8,9,10,11,12, 13,22,23,24,25, 26,27,28,29,30, 31,32,33,34,35, 36,37,38,39,40, 41,42,43,44,45, 46,47,48,49,50,51,52,53};
const int Vdd_pins[] = {2,8,22,26,30,34,38,42,46,50};

void setup() {
  for (int i = 0; i < 44; i++) {
    pinMode(output_pins[i], OUTPUT);  // Put all the pins as output
  }
  for (int i = 0; i < 11; i++) {
    digitalWrite(Vdd_pins[i],1);  // Vdd power to all switches
  }
  Serial.begin(9600); // Open communication to LabVIEW
}

const int RF_case[8][3] = {
        {0, 0, 0},
        {1, 0, 0},
        {0, 1, 0},
        {1, 1, 0},
        {0, 0, 1},
        {1, 0, 1},
        {0, 1, 1},
        {1, 1, 1}
    };  // 8 cases for ABC pins to controll RF on the transmitter and receiver switch
int Switching_antenna[8][2] = {
        {0,1}, {0,1}, {0,1}, {0,1}, {0,1}, {0,1}, {0,1}, {0,1} // basecase, off, for the 8 antennas, {Vctl,EN}
    };

void loop() {
  if (Serial.available() > 0){
    String var = Serial.readString(); // reading variables from LabVIEW
    int Tx = var[0] - 48; // transmitting antenna number,-48 due to ASCII code
    int Rx = var[1] - 48; // receiving antenna number, -48 due to ASCII kod

    Switching_antenna[Tx-1][0] = 1; // Turning on the transmitting antenna
    Switching_antenna[Tx-1][1] = 0; // Turning on the transmitting antenna
    //Switching_antenna[Rx-1][0] = 0; // Turning on the receiving antenna
    Switching_antenna[Rx-1][1] = 0; // Turning on the receiving antenna

    Serial.print(Tx); //sending variables back to LabVIEW for confirmation
    Serial.print(Rx);
    for (int i = 1; i < 4; i++) {
      digitalWrite(output_pins[i], RF_case[Tx-1][i-1]); // supplies voltages to transmitter switch according to the case from RF_case to switch on the correct RF-port
    }

    for (int i = 7; i < 11; i++) {
```

```
47        digitalWrite(output_pins[i], (RF_case[Rx-1][i-7] + 1) % 2); // supplies ↩
              voltages to receiver switch according to the case from RF_case to ↩
              switch on the correct RF-port
48      }
49
50      for (int i = 13; i < 44; i+=4) { // supplies voltages to the 8 antenna ↩
            switches according to the case from Transmitting_antenna
51        digitalWrite(output_pins[i], Switching_antenna[(i-13)/4][0]);   // Vctl
52        digitalWrite(output_pins[i+1], Switching_antenna[(i-13)/4][1]);   // EN
53      }
54      Switching_antenna[Tx-1][0] = 0; // Returning list to basecase
55      Switching_antenna[Tx-1][1] = 1;
56      //Switching_antenna[Rx-1][0] = 0;
57      Switching_antenna[Rx-1][1] = 1;
58      }
59    delay(1000);
60  }
```

C

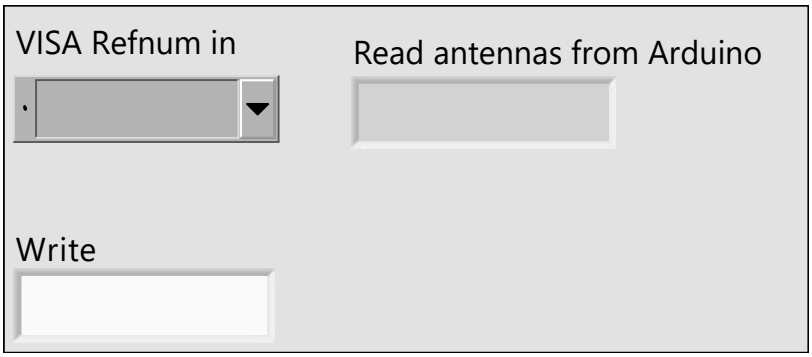
Appendix: LabVIEW-program

Nedan följer bilder på alla LabVIEW-strukturer som har använts vid mätningar. Alla programmen utom `Arduino.SubVI` och `Arduino_TwoChannel.vi` är ursprungligen skrivna av Laura Guerrero Orozco och Xuezhi Zeng men har omarbetats vid behov. Tillsammans utgör programmen all LabVIEW-kod som har krävts för att utföra enkanals- och tvåkanalsmätningar. Notera att en del bilder även förekommer i resultatet, dock i ett mindre format. För en mer detaljerad beskrivning av programmen, se avsnitt 3.2.

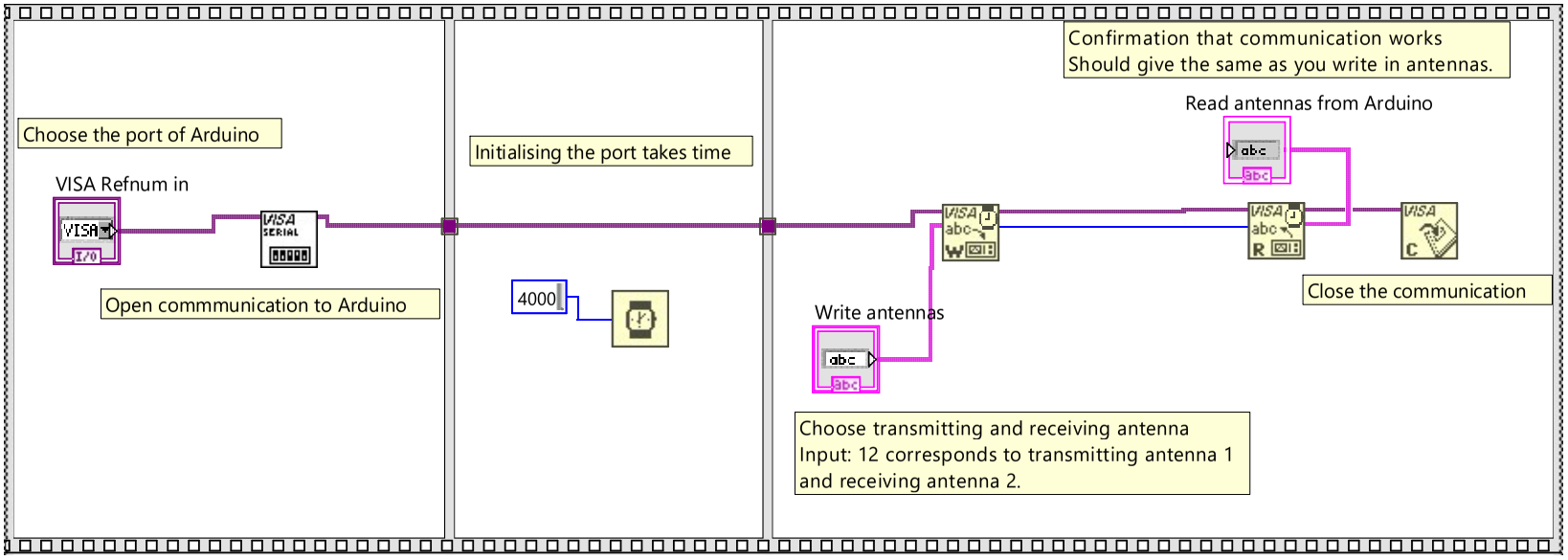
C.1 `Arduino.SubVI`

`Arduino.SubVI` skickar och tar emot information från Arduinon som i sin tur styr switcharna. Programmet presenteras även i ett mindre format i resultatet, se 4.1.1. Nedan följer frontpanel, blockdiagram och en lista över SubVIs samt Express VIs.

Front Panel



Block Diagram



List of SubVIs and Express VIs

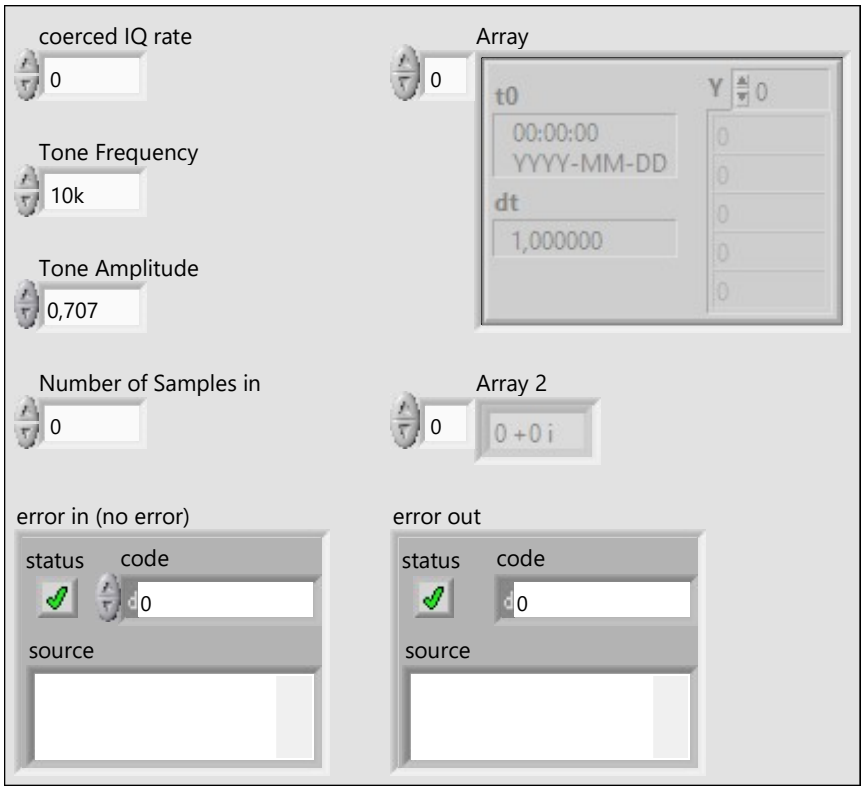
VISA Configure Serial Port (Instr).vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Instr_visa.llb\VISA Configure Serial Port (Instr).vi

VISA Configure Serial Port
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Instr_visa.llb\VISA Configure Serial Port

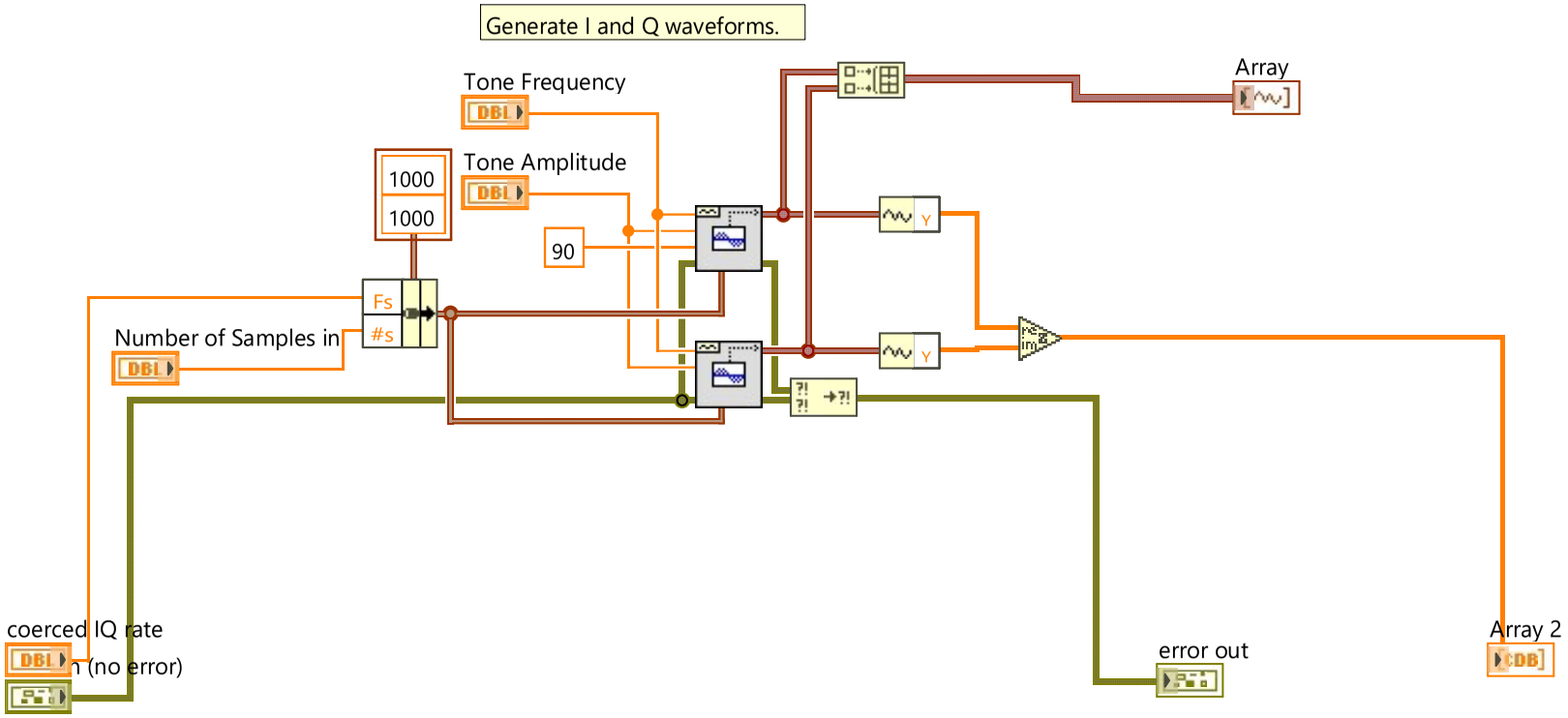
C.2 `GenIQ.vi`

`GenIQ.vi` är en SubVi som anropas vid både enkanal- och tvåkanalsmätning. Den genererar en signal utifrån intagna parametrar och returnerar sedan den genererade signalen. Nedan följer frontpanel, blockdiagram och en lista över SubVIs samt Express VIs.

Front Panel



Block Diagram

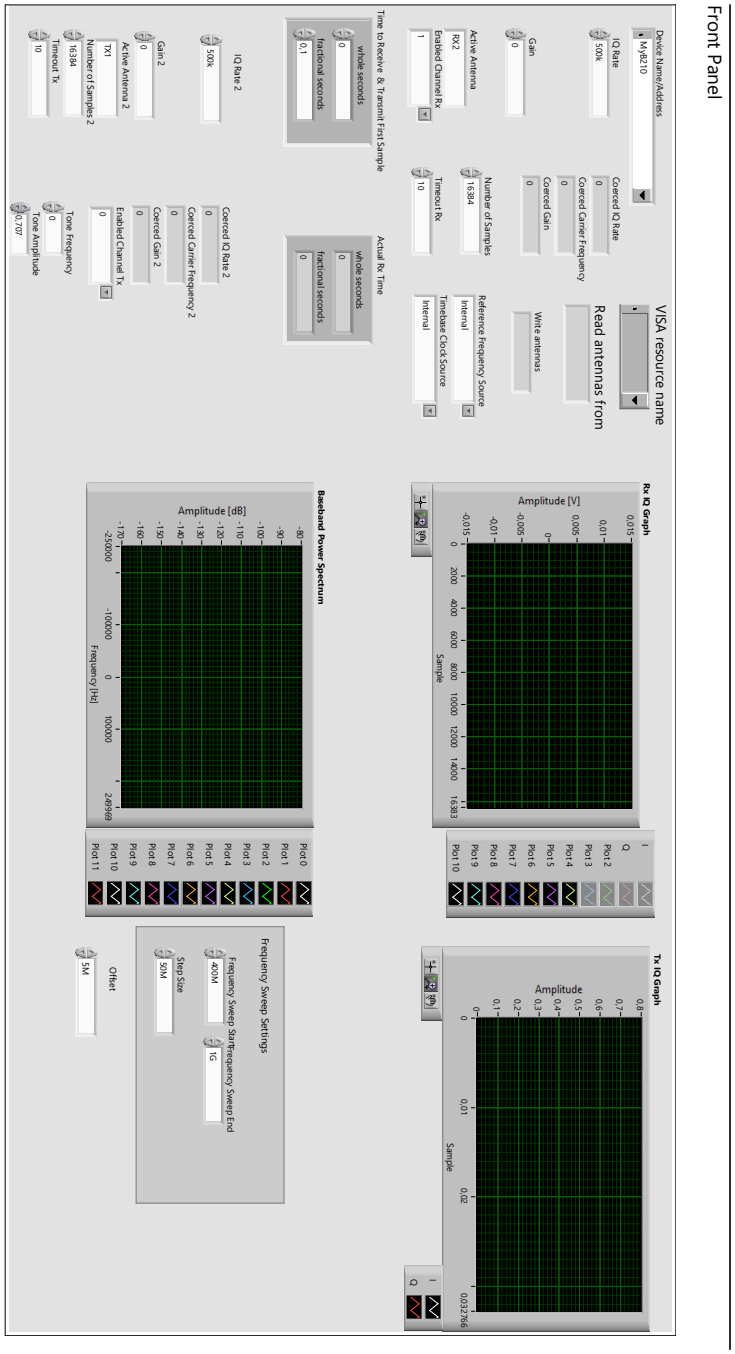


List of SubVIs and Express VIs

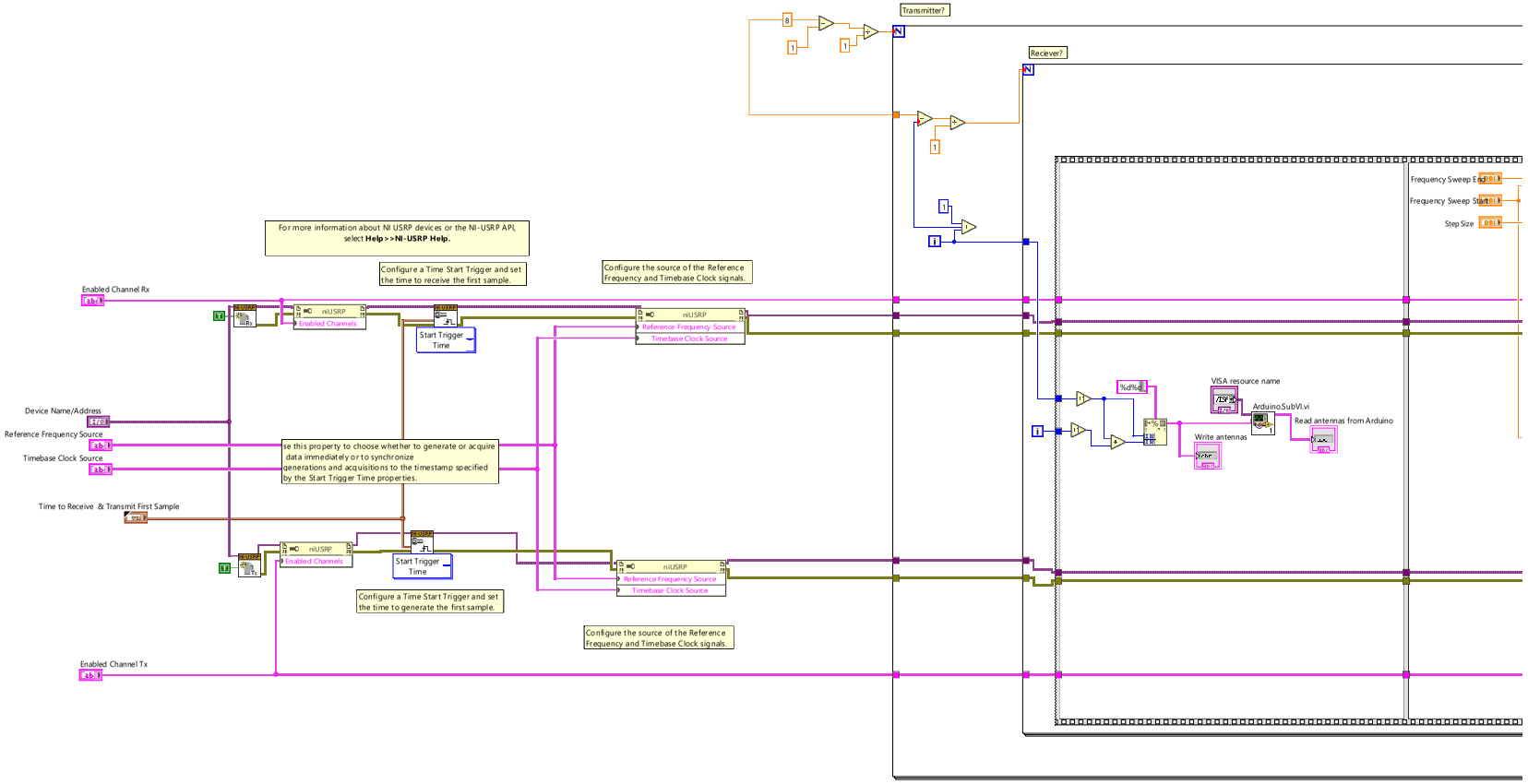
NI_MABase.lvlib:Sine Waveform.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\measure\masignal.llb\Sine Waveform.vi

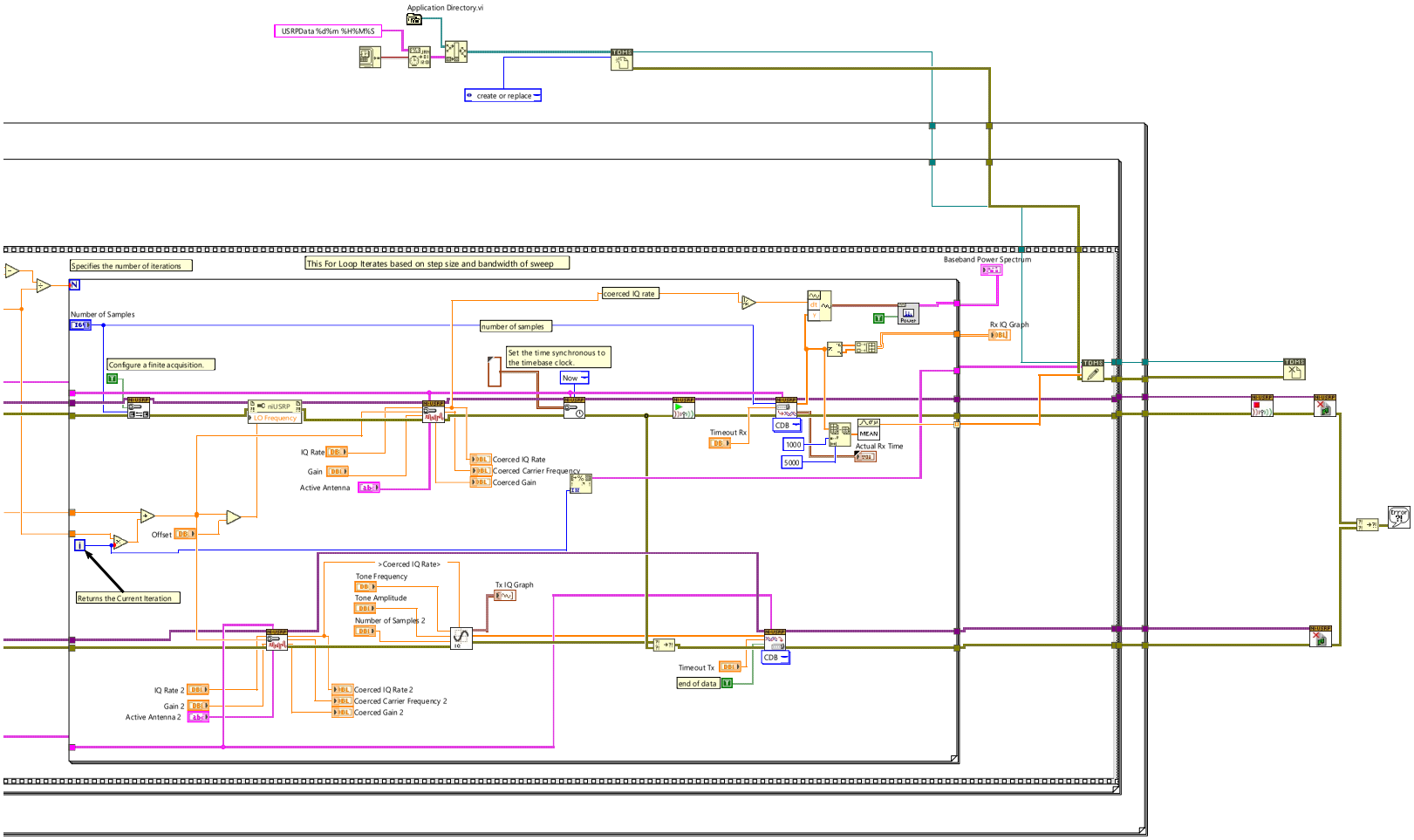
C.3 `SDR_RI2582.vi`

`SDR_RI2582.vi` utför en enkanalsmätning. Programmet styr SDR:en och `Arduino.SubVI` samt sparar insamlad data i en TDMS-fil. Nedan följer frontpanel, blockdiagram och en lista över SubVIs samt Express VIs till `SDR_RI2582.vi`.



Block Diagram




Application Directory.vi
USRPData %d%m %H%M%S
create or replace
Specifies the number of iterations.
This For Loop Iterates based on step size and bandwidth of sweep
Baseband Power Spectrum
Number of Samples
Configure a finite acquisition.
number of samples
coerced IQ rate
Set the time synchronous to the timebase clock.
Now
Rx IQ Graph
niUSRP
LO Frequency
IQ Rate
Gain
Active Antenna
Coerced IQ Rate
Coerced Carrier Frequency
Coerced Gain
Timeout Rx
CDB
1000
5000
MEAN
Actual Rx Time
Offset
Returns the Current Iteration
>Coerced IQ Rate>
Tone Frequency
Tone Amplitude
Number of Samples 2
Tx IQ Graph
IQ Rate 2
Gain 2
Active Antenna 2
Coerced IQ Rate 2
Coerced Carrier Frequency 2
Coerced Gain 2
Timeout Tx
end of data

List of SubVIs and Express VIs

NI_AALBase.lvlib:Mean (CDB).vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Analysis\baseanly.llb\Mean (CDB).vi

NI_MAPro.lvlib:FFT Power Spectrum for 1 Chan (CDB).vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\measure\maspectr.llb\FFT Power Spectrum for 1 Chan (CDB).vi

NI_AALBase.lvlib:Mean.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Analysis\baseanly.llb\Mean.vi

NI_MAPro.lvlib:FFT Power Spectrum.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\measure\maspectr.llb\FFT Power Spectrum.vi

niUSRP Fetch Rx Data (CDB).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Fetch Rx Data (CDB).vi

niUSRP Timestamp.ctl
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Timestamp.ctl

niUSRP Fetch Rx Data (poly).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Fetch Rx Data (poly).vi

niUSRP Initiate.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Initiate.vi

niUSRP Set Time.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Set Time.vi

niUSRP Configure Signal.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Signal.vi

niUSRP Configure Number of Samples.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Number of Samples.vi

Arduino.SubVI.vi
C:\Users\alvas\OneDrive\Dokument\kandidatarbete\Labview\Arduino.SubVI.vi

Application Directory.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Utility\file.llb\Application Directory.vi

niUSRP Open Tx Session.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Open Tx Session.vi

niUSRP Configure Time Start Trigger.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Time Start Trigger.vi

niUSRP Configure Trigger.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Trigger.vi

niUSRP Open Rx Session.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Open Rx Session.vi

Simple Error Handler.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Utility\error.llb\Simple Error Handler.vi

niUSRP Abort.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Abort.vi

niUSRP Close Session.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Close Session.vi

niUSRP Write Tx Data (CDB).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Write Tx Data (CDB).vi

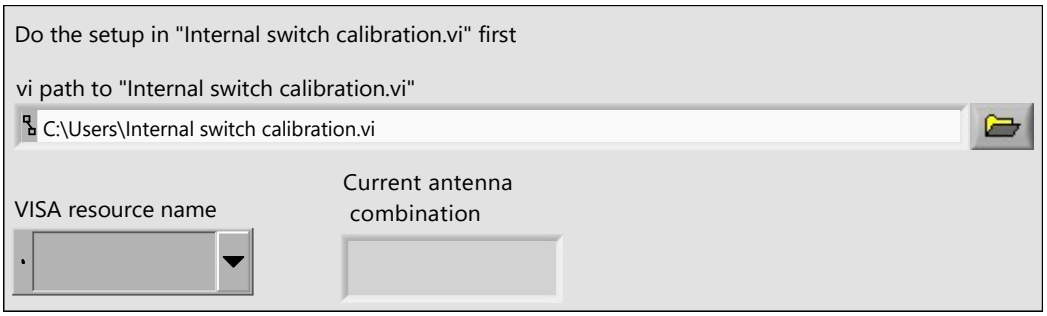
niUSRP Write Tx Data (poly).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Write Tx Data (poly).vi

GenIQ.vi
C:\Users\alvas\OneDrive\Dokument\kandidatarbete\Labview\GenIQ.vi

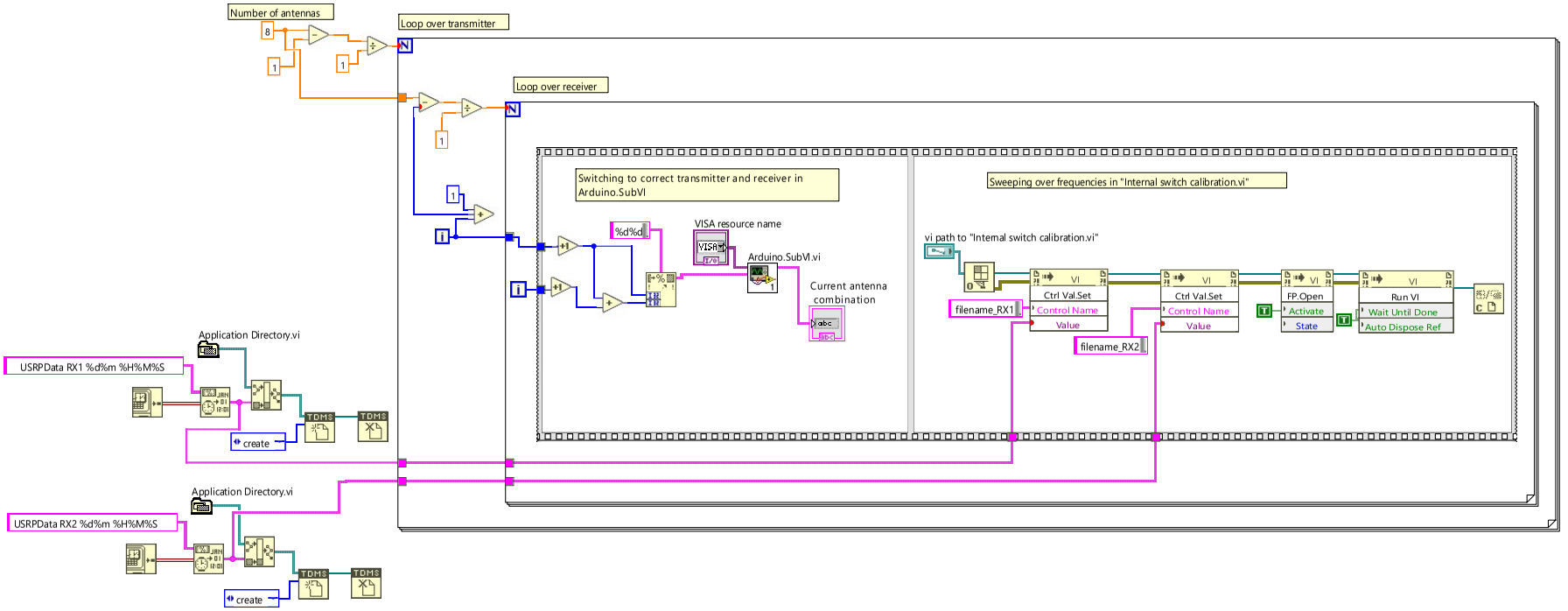
C.4 `Arduino_TwoChannel.vi`

`Arduino_TwoChannel.vi` hanterar styrningen av `Arduino.SubVI` och SDR:en genom att anropa `Internal switch calibration.vi` vid en tvåkanalsmätning. Nedan följer frontpanel, blockdiagram och en lista över SubVIs samt Express VIs. Blockdiagrammet presenteras även i ett mindre format i resultatet, se 4.1.1.

Front Panel



Block Diagram



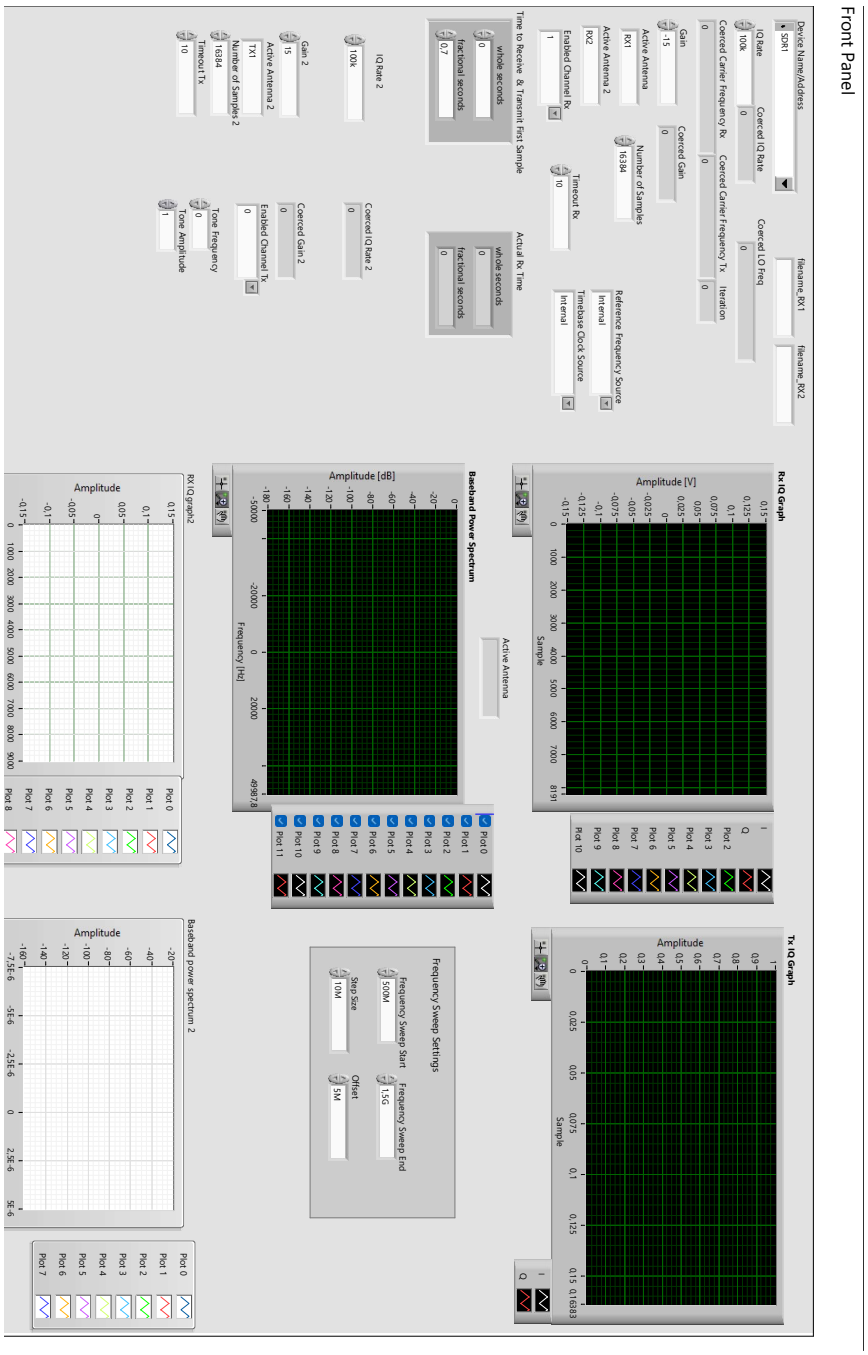
List of SubVIs and Express VIs

Application Directory.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Utility\file.llb\Application Directory.vi

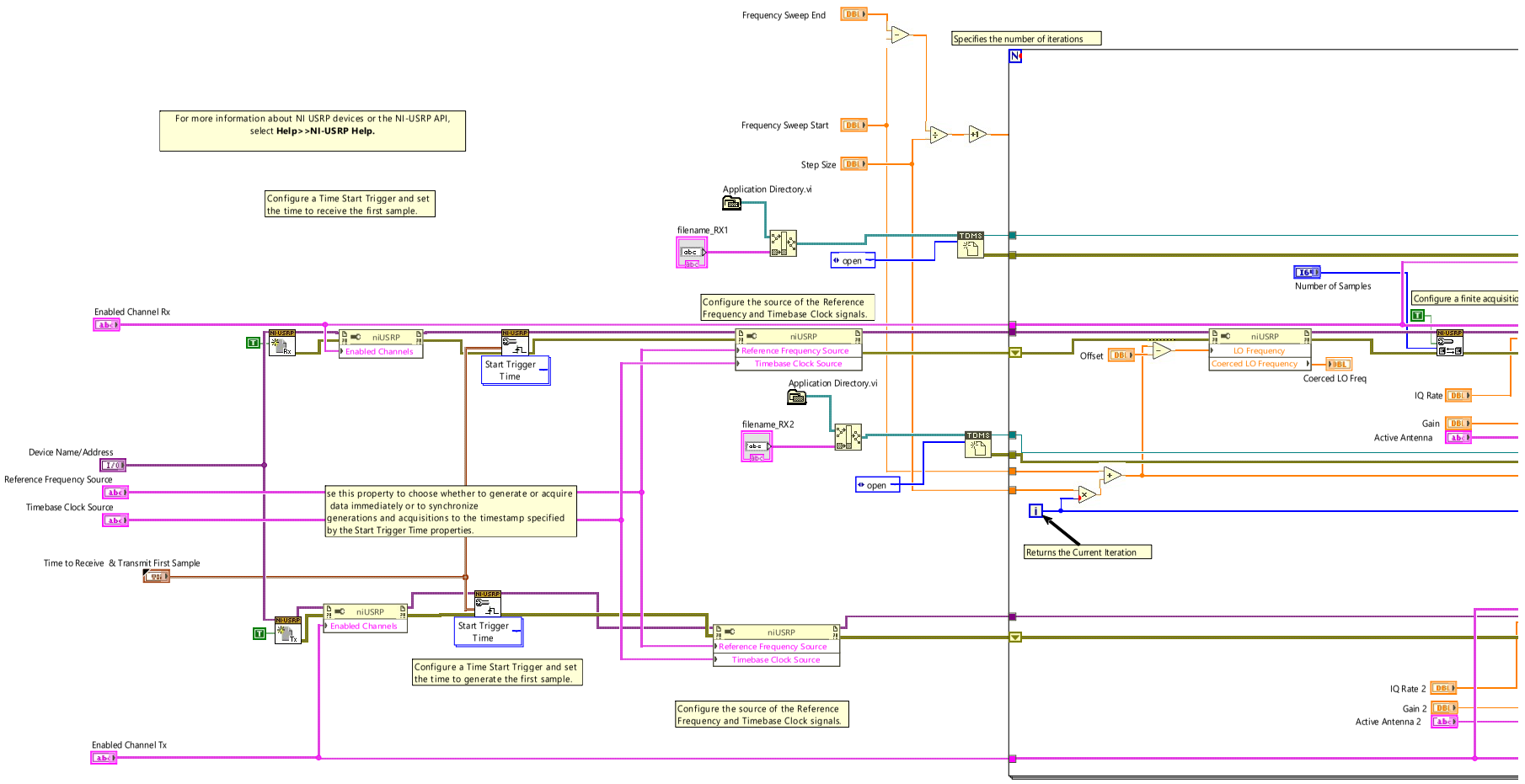
Arduino.SubVI.vi
C:\Users\alvas\OneDrive\Dokument\kandidatarbete\Labview\Arduino.SubVI.vi

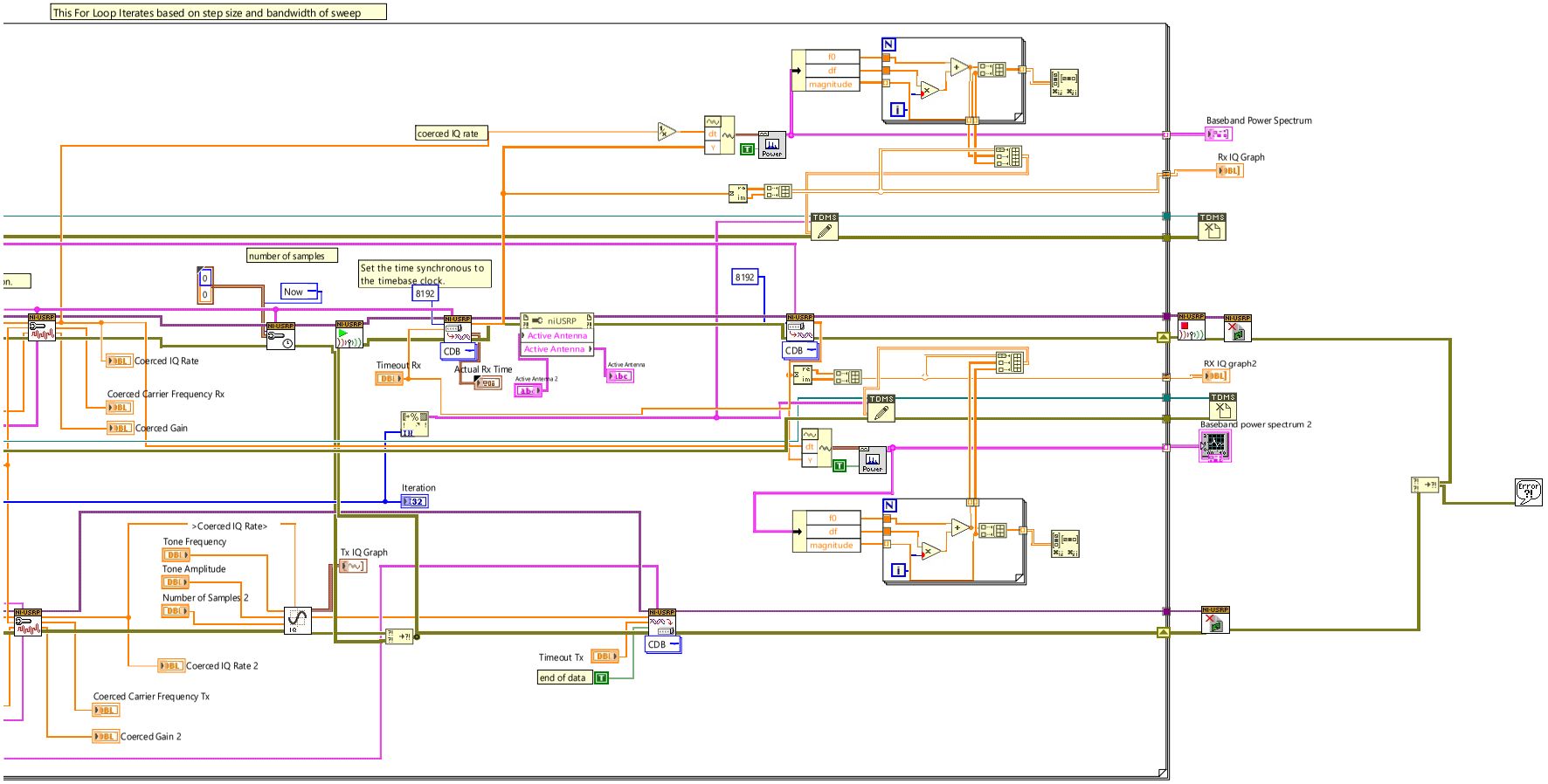
C.5 `Internal switch calibration.vi`

`Internal switch calibration.vi` styr ett frekvenssvep med en SDR och sparar uppmätt data i två TDMS-filer, en per använd kanal. Nedan följer frontpanel, blockdiagram och en lista över SubVIs samt Express VIs.



Block Diagram




This For Loop Iterates based on step size and bandwidth of sweep
coerced IQ rate
number of samples
Set the time synchronous to the timebase clock.
8192
Now
Coerced IQ Rate
Coerced Carrier Frequency Rx
Coerced Gain
Timeout Rx
Actual Rx Time
niUSRP
Active Antenna
CDB
Iteration
>Coerced IQ Rate>
Tone Frequency
Tone Amplitude
Number of Samples 2
Tx IQ Graph
Coerced IQ Rate 2
Coerced Carrier Frequency Tx
Coerced Gain 2
Timeout Tx
end of data
f0
df
magnitude
Power
TDMS
Baseband Power Spectrum
Rx IQ Graph
RX IQ graph2
Baseband power spectrum 2

List of SubVIs and Express VIs

NI_MAPro.lvlib:FFT Power Spectrum for 1 Chan (CDB).vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\measure\maspectr.llb\FFT Power Spectrum for 1 Chan (CDB).vi

NI_MAPro.lvlib:FFT Power Spectrum.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\measure\maspectr.llb\FFT Power Spectrum.vi

niUSRP Fetch Rx Data (CDB).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Fetch Rx Data (CDB).vi

niUSRP Timestamp.ctl
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Timestamp.ctl

niUSRP Fetch Rx Data (poly).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Fetch Rx Data (poly).vi

niUSRP Initiate.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Initiate.vi

niUSRP Set Time.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Set Time.vi

niUSRP Configure Signal.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Signal.vi

niUSRP Configure Number of Samples.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Number of Samples.vi

Application Directory.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Utility\file.llb\Application Directory.vi

niUSRP Open Tx Session.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Open Tx Session.vi

niUSRP Configure Time Start Trigger.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Time Start Trigger.vi

niUSRP Configure Trigger.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Configure Trigger.vi

niUSRP Open Rx Session.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Open Rx Session.vi

Simple Error Handler.vi
C:\Program Files\National Instruments\LabVIEW 2025\vi.lib\Utility\error.llb\Simple Error Handler.vi

niUSRP Abort.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Abort.vi

niUSRP Close Session.vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Close Session.vi

niUSRP Write Tx Data (CDB).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Write Tx Data (CDB).vi

niUSRP Write Tx Data (poly).vi
C:\Program Files\NI\LVAddons\niusrp\1\instr.lib\niUSRP\niUSRP Write Tx Data (poly).vi

GenIQ.vi
C:\Users\alvas\OneDrive\Dokument\kandidatarbete\Labview\GenIQ.vi

D

Appendix: MATLAB-funktioner

Nedan är alla omskrivna MATLAB-funktioner samlade, ursprungligen skrivna av Laura Guerrero Orozco och Xuezhi Zeng. De är uppdelade efter respektive användningsområde, enkanals- eller tvåkanalsmätning. Notera att funktionen `convertTDMS`, ej har ändrats men presenteras i avsnitt D.2.4 ändå. För en mer detaljerad beskrivning av funktionerna, se avsnitt 3.3.

D.1 Enkanalsmätning

Funktionen `SDR_1channel` återger resultatet från en enkanalsmätning i form av två grafer. Båda graferna skildrar den uppmätta amplituden men utifrån ett frekvens- eller antennkombinationsperspektiv. Funktionen kräver information i form av filnamn/filväg och antalet antennkombinationer som användes vid mätning samt tillgång till `convertTDMS`-funktionen. Funktionen är kompatibel med mätdata från LabVIEW-koden `SDR_RI2582.vi`, se appendix C.3.

```
function SDR_1channel(file, combinations)
data_struct = convertTDMS(0, file); % konvertera data till en struktur
data = data_struct.Data.MeasuredData(3).Data; % hämta mätdata från strukturen

num_freq = length(data)/combinations; % beräkna antalet frekvenser i ↩
    frekvenssvepet
position = 1:num_freq:length(data); % lista med alla datapositioner för första ↩
    frekvensen
comb = 1:combinations; % lista över alla kombinationer
freq = 1:num_freq; % lista med antalet frekvenser

figure;
%sgtitle('Amplitud i förhållande till antennkombination och frekvens');

subplot(2,1,1)
for i = 1:num_freq % Gå igenom alla frekvenser
    plot(comb,abs(data(position+i-1))) % plot alla uppmätta värden för frekvens i
    hold on;
end
xlabel('Antennkombination')
ylabel('Amplitud')
xlim([1,combinations])
%title('Amplitud i förhållande till antennkombination')

subplot(2,1,2)
for i=1:combinations % Gå igenom alla antennkombinationer
    plot(freq,abs(data(freq+(i-1)*num_freq))) % plot alla värden för en ↩
        antennkombination
    hold on;
end
xlim([1,num_freq])
xlabel('Frekvensmätning')
```

```
ylabel('Amplitud')
%title('Amplitud i förhållande till frekvens')
end
```

D.2 Tvåkanalsmätning

Vid en tvåkanalsmätning används flera funktioner i MATLAB för att visualisera resultatet. Funktionerna är kompatibla med varandra och data uppmätt med LabVIEW `Arduino_TwoChannel.vi`, se appendix C.4.

D.2.1 `SDR_2channel`

`SDR_2channel` är en omskrivning av funktionen `openblood`, som därmed inte är bifogad. `SDR_2channel` anropar `GetSparSDR`-funktionen som returnerar en matris, där raderna representerar olika frekvenser och för varje antennkombination finns en kolumn. Returnerad data ritas sedan upp i samma graf med en linje per antennkombination.

```
function [SB] = SDR_2channel(fileRX1,fileRX2,startfreq,stepsize,endfreq,↩
    numsamples,numantennas)

combinations = numantennas*(numantennas-1)/2; % calculate antenna combinations
[SB] = GetSparSDR(fileRX1,fileRX2,startfreq,stepsize,endfreq,numsamples,↩
    numantennas,combinations); % save the returned matrix, every row represents a↩
     frequency and every column an antenna combination
freq = ((startfreq:stepsize:endfreq)*1e6).'; % frequencies in the frequency sweep


h = figure;
for a = 1:combinations % loop through every combination
    plot(freq/1e9,20*log10(abs(SB(:,a))),'linewidth',2) % plot for antenna ↩
        combination a
    hold on
end

antennacombinations = strings(1, combinations); % create for legend text
c = 1; % counter
for i = 1:numantennas
    for j = 1:numantennas
        if j>i
          antennacombinations(c) = "S" + string(i) + string(j); % create ↩
              textstring if recieve antenna j is bigger than transmit i
          c=c+1;
        end
    end
end

% all plot settings below
lg = legend(antennacombinations,'linewidth',2,'NumColumns',4);

xlabel('Frequency(GHz)')
ylabel('Received power(dB)')
%ylim([-80 -15])
xticks([0.5 0.75 1 1.25 1.5])
set(gca,'FontSize', 14)
set(lg,'FontSize',14)

oldunits = h.Units;
set(h, 'PaperUnits', 'centimeters', 'Units', 'centimeters');
figpos = get(h, 'Position');
```

```
set(h, 'PaperSize', figpos(3:4), 'Units', oldunits);
% set(h,'papersize',[5.1 3.2]);
print(h,'RXSDRtwoamp','-dpdf','-bestfit')
end
```

D.2.2 GetSparSDR

Funktionen anropar `opsdr` som returnerar data strukturerad i en cellmatris. Data tillhörande mottagarkanalen divideras med motsvarande data från referenskanalen. Resultatet samlas i en matris som returneras när funktionen anropas.

```
function [SB] = GetSparSDR(fileRX1,fileRX2,startfreq,stepsize,endfreq,numsamples,↩
    numantennas,combinations)

S_cellarray = opsdr(fileRX1,fileRX2,startfreq,stepsize,endfreq,numsamples,↩
    combinations); % all data from the TDMS files

for i = 1:combinations
 Rx1b = S_cellarray{i}{1};
 Rx2b = S_cellarray{i}{2};
 Rx1bV(:,i) = Rx1b.';
 Rx2bV(:,i) = Rx2b.';
end

iterations = ((endfreq-startfreq)/stepsize)+1; % iterations in the frequency ↩
    sweep
SB = Rx2bV(1:iterations,:)./Rx1bV(1:iterations,:); % divide received data by ↩
    reference data
end
```

D.2.3 opsdr

`opsdr` anropar `convertTDMS`, en gång för referensfilen och en gång för mottagarfilen för att konvertera mätpunkterna till ett format kompatibelt med MATLAB. Utifrån den konverterade datastrukturen beräknas ett medelvärde per antennkombination och frekvens. Samtliga medelvärden sammanställs i en cellmatris med en cell per kombination, som i sin tur innehåller två celler (en för referensdata och en för mottagardata). Vardera cell består av en lista med alla medelvärden från frekvenssvepet, där värdena är sorterade i ordning utifrån ökande frekvens. Vid anrop av funktionen returneras hela den skapade cellmatrisen.

```
function S_cellarray = opsdr(fileRX1,fileRX2,startfreq,stepsize,endfreq,↩
    numsamples,combinations)

iterations = ((endfreq-startfreq)/stepsize)+1; % iterations in frequency sweep
endrow = ((iterations)*5)+1; % endrow in file calculated from number of ↩
    iterations in sweep, every iteration uses 5 rows

TDMSRX1 = convertTDMS(0,fileRX1); % reference channel
TDMSRX2 = convertTDMS(0,fileRX2); % receiver channel

S_cellarray = {}; % for saving the data

for j = 1:combinations
    c = 1;
    for i = 3:5:endrow  % loop throguh all iterations/frequencies
```

```
        % get all data from reference channel, frequency i and combination j, add↩
             the real and imagenary parts together
        sref = TDMSRX1.Data.MeasuredData(i).Data(((j-1)*numsamples/2)+1:j*↩
            numsamples/2) + 1i * TDMSRX1.Data.MeasuredData(i+1).Data(((j-1)*↩
            numsamples/2)+1:j*numsamples/2);

        % get all data from reciever channel, frequency i and combination j, add ↩
            the real and imagenary parts together
        s = TDMSRX2.Data.MeasuredData(i).Data(((j-1)*numsamples/2)+1:j*numsamples↩
            /2) + 1i * TDMSRX2.Data.MeasuredData(i+1).Data(((j-1)*numsamples/2)↩
            +1:j*numsamples/2);

        Sref(c) = mean(sref(1000:6000)); % save mean in list Sref
        S(c) = mean(s(4000:8000)); % save mean in list S
        c=c+1;
    end
    S_cellarray{j}{1} = Sref;
    S_cellarray{j}{2} = S;

end
end
```

D.2.4 `convertTDMS`

Funktionen `convertTDMS` konverterar innehållet i en TDMS-filer till en datastruktur kompatibel med MATLAB.

```
function [ConvertedData,ConvertVer,ChanNames,GroupNames,ci]=convertTDMS(varargin)
%Function to load LabView TDMS data file(s) into variables in the MATLAB ↩
    workspace.
%An *.MAT file can also be created.  If called with one input, the user selects
%a data file.
%
%   TDMS format is based on information provided by National Instruments at:
%   http://zone.ni.com/devzone/cda/tut/p/id/5696
%
% [ConvertedData,ConvertVer,ChanNames]=convertTDMS(SaveConvertedFile,filename)
%
%       Inputs:
%               SaveConvertedFile (required) - Logical flag (true/false) that
%                 determines whether a MAT file is created.  The MAT file's name
%                 is the same as 'filename' except that the 'TDMS' file extension↩
    is
%                 replaced with 'MAT'.  The MAT file is saved in the same folder
%                 and will overwrite an existing file without warning.  The
%                 MAT file contains all the output variables.
%
%               filename (optional) - Filename (fully defined) to be converted.
%                 If not supplied, the user is provided a 'File Open' dialog box
%                 to navigate to a file.  Can be a cell array of files for bulk
%                 conversion.
%
%       Outputs:
%               ConvertedData (required) - Structure of all of the data objects.
%               ConvertVer (optional) - Version number of this function.
%               ChanNames (optional) - Cell array of channel names
%               GroupNames (optional) - Cell array of group names
%               ci (optional) - Structure of the channel index (an index to
%                   where all of the information for a channel resides in a
%                   file.
%
%
%'ConvertedData' is a structure with 'FileName', 'FileFolder', 'SegTDMSVerNum',
%'NumOfSegments' and 'Data' fields'. The 'Data' field is a structure.
%
%'ConvertedData.SegTDMSVerNum' is a vector of the TDMS version number for each
```

```
%segment.
%
%'ConvertedData.Data' is a structure with 'Root' and 'MeasuredData' fields.
%
%'ConvertedData.Data.Root' is a structure with 'Name' and 'Property' fields.
%The 'Property' field is also a structure; it contains all the specified ↩
    properties
%(1 entry for each 'Property) for the 'Root' group. For each 'Property' there are
%'Name' and 'Value' fields. To display a list of all the property names, input
%'{ConvertedData.Data.Root.Property.Name}'' in the Command Window.
%
%'ConvertedData.Data.MeasuredData' is a structure containing all the channel/↩
    group
%information. For each index (for example, 'ConvertedData.Data.MeasuredData(1)'),
%there are 'Name', 'Data' and 'Property' fields.  The list of channel names can
%be displayed by typing 'ChanNames' in the Command Window.  Similarly, the list
%of group names can be displayed by typing 'GroupNames' in the Command Window.
%The 'Property' field is also a structure; it contains all the specified ↩
    properties
%for that index (1 entry in the structure for each 'Property'). Any LabView ↩
    waveform
%attributes ('wf_start_time', 'wf_start_offset', 'wf_increment' and 'wf_samples')↩
     that
%may exist are also included in the properties. For each 'Property' there are '↩
    Name'
%and 'Value' fields.  To display a list of all the property names, input
%'{ConvertedData.Data.MeasuredData(#).Property.Name}'' in the Command Window
%where '#' is the index of interest.
%
%If you recieve an error that DAQmxRaw data cannot be converted, this is due
%to the details on parsing this data type not being published by NI.  the
%work around is to use a VI that reads in the data file and then writes it
%to a new file.  See: https://decibel.ni.com/content/docs/DOC-32817
%
%  See Also: simpleconvertTDMS

%--------------------------------------------------------------------------
%Brad Humphreys - v1.0 2008-04-23
%ZIN Technologies
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Brad Humphreys - v1.1 2008-07-03
%ZIN Technologies
%-Added abilty for timestamp to be a raw data type, not just meta data.
%-Addressed an issue with having a default nsmaples entry for new objects.
%-Added Error trap if file name not found.
%-Corrected significant problem where it was assumed that once an object
%    existsed, it would in in every subsequent segement.  This is not true.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Grant Lohsen - v1.2 2009-11-15
%Georgia Tech Research Institute
%-Converts TDMS v2 files
%Folks, it's not pretty but I don't have time to make it pretty. Enjoy.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Jeff Sitterle - v1.3 2010-01-10
%Georgia Tech Research Institute
%Modified to return all information stored in the TDMS file to inlcude
%name, start time, start time offset, samples per read, total samples, unit
%description, and unit string.  Also provides event time and event
%description in text form
%Vast speed improvement as save was the previous longest task
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Grant Lohsen - v1.4 2009-04-15
```

```
%Georgia Tech Research Institute
%Reads file header info and stores in the Root Structure.
%---------------------------------------------------------------------------

%---------------------------------------------------------------------------
%Robert Seltzer - v1.5 2010-07-14
%BorgWarner Morse TEC
%-Tested in MATLAB 2007b and 2010a.
%-APPEARS to now be compatible with TDMS version 1.1 (a.k.a 4712) files;
%   although, this has not been extensively tested.  For some unknown
%   reason, the version 1.2 (4713) files process noticeably faster. I think
%   that it may be related to the 'TDSm' tag.
%-"Time Stamp" data type was not tested.
%-"Waveform" fields was not tested.
%-Fixed an error in the 'LV2MatlabDataType' function where LabView data type
%   'tdsTypeSingleFloat' was defined as MATLAB data type 'float64' .  Changed
%   to 'float32'.
%-Added error trapping.
%-Added feature to count the number of segments for pre-allocation as
%   opposed to estimating the number of segments.
%-Added option to save the data in a MAT file.
%-Fixed "invalid field name" error caused by excessive string lengths.
%---------------------------------------------------------------------------

%---------------------------------------------------------------------------
%Robert Seltzer - v1.6 2010-09-01
%BorgWarner Morse TEC
%-Tested in MATLAB 2010a.
%-Fixed the "Coversion to cell from char is not possible" error found
%  by Francisco Botero in version 1.5.
%-Added capability to process both fragmented or defragmented data.
%-Fixed the "field" error found by Lawrence.
%---------------------------------------------------------------------------

%---------------------------------------------------------------------------
%Christian Buxel - V1.7 2010-09-17
%RWTH Aachen
%-Tested in Matlab2007b.
%-Added support for german umlauts (ï¿½,ï¿½,ï¿½,ï¿½,ï¿½,ï¿½,ï¿½) in 'propsName'
%---------------------------------------------------------------------------

%---------------------------------------------------------------------------
%Andrï¿½ Rï¿½egg - V1.7 2010-09-29
%Supercomputing Systems AG
%-Tested in MATLAB 2006a & 2010b
%-Make sure that data can be loaded correctly independently of character
% encoding set in matlab.
%-Fixed error if object consists of several segments with identical segment
% information (if rawdataindex==0, not all segments were loaded)
%---------------------------------------------------------------------------

%---------------------------------------------------------------------------
%Robert Seltzer - v1.7 2010-09-30
%BorgWarner Morse TEC
%-Tested in MATLAB 2010b.
%-Added 'error trapping' to the 'fixcharformatlab' function for group and
% channel names that contain characters that are not 'A' through 'Z',
% 'a' through 'z', 0 through 9 or underscore. The characters are replaced
% with an underscore and a message is written to the Command Window
% explaining to the user what has happened and how to fix it. Only tested
% with a very limited number of "special" characters.
%---------------------------------------------------------------------------

%---------------------------------------------------------------------------
%Robert Seltzer - v1.8 2010-10-12
%BorgWarner Morse TEC
%-As a result of an error found by Peter Sulcs when loading data with very
% long channel names, I have re-written the sections of the function that
% creates the channel and property names that are used within the body of
% the function to make them robust against long strings and strings
```

```
% containing non-UTF8 characters.  The original channel and property
% names (no truncation or character replacement) are now retained and
% included in the output structure.  In order to implement this improvement,
% I added a 'Property' field as a structure to the 'ConvertedData' output
% structure.
%-Added a more detailed 'help' description ('doc convertTDMS') of the
% returned structure.
%-List of channel names added as an output parameter of the function.
%-Corrected an error in the time stamp converion. It was off by exactly
% 1 hour.
%-Tested in MATLAB 2010b with a limited number of diverse TDMS files.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Robert Seltzer - v1.8 2010-10-19
%BorgWarner Morse TEC
%-Fixed an error found by Terenzio Girotto with the 'save' routine.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Robert Seltzer - v1.8 2010-10-25
%BorgWarner Morse TEC
%-Fixed an error with channels that contain no data.  Previously, if a
% channel contained no data, then it was not passed to the output structure
% even if it did contain properties.
%-Added 'GroupNames' as an optional output variable.
%-Fixed an error with capturing the properties of the Root object
%--------------------------------------------------------------------------


%--------------------------------------------------------------------------
%Philip Top - v1.9 2010-11-09
%John Breneman
%-restructured code as function calls
%-seperated metadata reads from data reads
%-preallocated space for SegInfo with two pass file read
%-preallocated index information and defined segdataoffset for each segment
%-preallocate space for data for speedup in case of fragmented files
%-used matlab input parser instead of nargin switch
%-vectorized timestamp reads for substantial speedup
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Robert Seltzer - v1.9 2010-11-10
%BorgWarner Morse TEC
%-Fixed an error error in the 'offset' calculation for strings
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Philip Top - v1.95 2011-5-10
%Fix Bug with out of order file segments
%Fix some issues with string array reads for newer version files,
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Brad Humphreys - v1.96 2013-10-9
%Fixed problem with error catch messages themselves (interleaved, version,
%big endian) using the TDMSFileName variable which was not available
%(passed into) the getSegInfo function.  Added ability to work with
%interleaved and big endian files.
%So function now covers:
%       -v1.0-v2.0
%       -Interleaved and Decimated Data Formats
%       -Big and Little Endian storage
% Does not work with DAQmxRawData
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Brad Humphreys - v1.97 2013-11-13
%Added information to help documentation on how to deal with DAQmxRaw Data.
```

```
%--------------------------------------------------------------------------


%--------------------------------------------------------------------------
%Brad Humphreys - v1.98 2014-5-27
%Per G. Lohsen's suggestion, added check to verify that first caharters are
% TDMs.  If not, errors out and lets user know that the selected file is
% not a TDMS file.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Sebastian Schwarzendahl (alias Haench) - v1.99 2014-10-23
% Added support for complex data types
% CSG - tdsTypeComplexSingleFloat=0x08000c
% CDB - tdsTypeComplexDoubleFloat=0x10000d)
% This feature was added in LV2013 (I believ) and produced an error in
% the previous version of this code.
%--------------------------------------------------------------------------
%Initialize outputs
ConvertVer='1.98';    %Version number of this conversion function
ConvertedData=[];

p=inputParser();

p.addRequired('SaveConvertedFile',@(x) islogical(x)||(ismember(x,[0,1])));
p.addOptional('filename','',@(x) iscell(x)||exist(x,'file'));
p.parse(varargin{:});

filename=p.Results.filename;
SaveConvertedFile=p.Results.SaveConvertedFile;

if isempty(filename)

    %Prompt the user for the file
    [filename,pathname]=uigetfile({'*.tdms','All Files (*.tdms)'},'Choose a TDMS ↩
        File');
    if filename==0
        return
    end
    filename=fullfile(pathname,filename);
end


if iscell(filename)
    %For a list of files
    infilename=filename;
else
    infilename=cellstr(filename);
end

for fnum=1:numel(infilename)

    if ~exist(infilename{fnum},'file')
        e=errordlg(sprintf('File ''%s'' not found.',infilename{fnum}),'File Not ↩
            Found');
        uiwait(e)
        return
    end

    FileNameLong=infilename{fnum};
    [pathstr,name,ext]=fileparts(FileNameLong);
    FileNameShort=sprintf('%s%s',name,ext);
    FileNameNoExt=name;
    FileFolder=pathstr;

    if fnum==1
        fprintf('\n\n')
    end
    fprintf('Converting ''%s''...',FileNameShort)
```

```
    fid=fopen(FileNameLong);

    if fid==-1
        e=errordlg(sprintf('Could not open ''%s''.',FileNameLong),'File Cannot Be↩
             Opened');
        uiwait(e)
        fprintf('\n\n')
        return
    end

    % Build a database with segment info
    [SegInfo,NumOfSeg]=getSegInfo(fid);

    % Build a database with channel info
    [channelinfo SegInfo]=getChannelInfo(fid,SegInfo,NumOfSeg);

    % Add channel count to SegInfo
    %SegInfo=addChanCount(SegInfo,channelinfo);

    % Get the raw data
    ob=getData(fid,channelinfo,SegInfo);  %Returns the objects which have data.  ↩
        See postProcess function (appends to all of the objects)
    fclose(fid);

    %Assign the outputs
    ConvertedData(fnum).FileName=FileNameShort;
    ConvertedData(fnum).FileFolder=FileFolder;

    ConvertedData(fnum).SegTDMSVerNum=SegInfo.vernum;
    ConvertedData(fnum).NumOfSegments=NumOfSeg;
    [ConvertedData(fnum).Data,CurrGroupNames]=postProcess(ob,channelinfo);

    GroupNames(fnum)={CurrGroupNames};

    TempChanNames={ConvertedData(fnum).Data.MeasuredData.Name};
    TempChanNames(strcmpi(TempChanNames,'Root'))=[];
    ChanNames(fnum)={sort(setdiff(TempChanNames',CurrGroupNames))};
    if SaveConvertedFile
        MATFileNameShort=sprintf('%s.mat',FileNameNoExt);
        MATFileNameLong=fullfile(FileFolder,MATFileNameShort);
        try
            save(MATFileNameLong,'ConvertedData','ConvertVer','ChanNames')
            fprintf('\n\nConversion complete (saved in ''%s'').\n\n',↩
                MATFileNameShort)
        catch exception
            fprintf('\n\nConversion complete (could not save ''%s'').\n\t%s: %s\n↩
                \n',MATFileNameShort,exception.identifier,...
                exception.message)
        end
    else
        fprintf('\n\nConversion complete.\n\n')
    end
end
ci=channelinfo;
end


%%
function [SegInfo,NumOfSeg]=getSegInfo(fid)
% Go through the whole file and build a database of the segment lead-in
% information:
%1) Count the total number of segments in the file.  (NumOfSeg)
%2) Get all of the "Lead In" Headers from all of the segements and store
%   that in the SegInfo variable:
%         SegInfo.SegStartPosn: Absolute Starting Position of Segment in file
%         SegInfo.MetaStartPosn: Absolute Starting Position of Meta Data in file
%         SegInfo.DataStartPosn: Absolute Starting Position of Raw Data in file
%         SegInfo.DataLength: Number of bytes of Data in segment
%         SegInfo.vernum: LV version number (4712 is v1.0, 4713 is v2.0)
%         SegInfo.NumChan: number of channels in this segement.  This is
```

```
%           only instatated in this function to 0.  The addChanCount function
%           updates this to the actual value later.
%
%         SegInfo.SegHasMetaData:  There is Meta Data in the Segement
%         SegInfo.SegHasRawData: There is Raw Data in the Segment
%         SegInfo.SegHasDaqMxRaw: There is DAQmxRaw Data in the Segment
%         SegInfo.SegInterleaved: 0: Contigous Data, 1: Interleaved Data
%         SegInfo.SegBigEndian:  0: Little, 1:Big (numeric, leadin, raw, and meta)
%         SegInfo.SegHasNewObjList: New object list in Segment
%
%3) While doing the above, also include error trapping for incompatibity

%Find the end of the file
fseek(fid,0,'eof');
eoff=ftell(fid);
frewind(fid);

segCnt=0;
CurrPosn=0;
LeadInByteCount=28; %From the National Instruments web page (http://zone.ni.com/devzone/cda/tut/p/id/5696) under
%the 'Lead In' description on page 2: Counted the bytes shown in the table.
while (ftell(fid) ~= eoff)

    Ttag=fread(fid,1,'uint8','l');
    Dtag=fread(fid,1,'uint8','l');
    Stag=fread(fid,1,'uint8','l');
    mtag=fread(fid,1,'uint8','l');

    if Ttag==84 && Dtag==68 && Stag==83 && mtag==109
        %Apparently, this sequence of numbers identifies the start of a new segment.

        segCnt=segCnt+1;

        %ToC Field
        ToC=fread(fid,1,'uint32','l');
        kTocBigEndian=bitget(ToC,7);
        if kTocBigEndian
            kTocEndian='b';
        else
            kTocEndian='l';
        end

        %TDMS format version number
        vernum=fread(fid,1,'uint32',kTocEndian);

        %From the National Instruments web page (http://zone.ni.com/devzone/cda/tut/p/id/5696) under the 'Lead In'
        %description on page 2:
        %The next eight bytes (64-bit unsigned integer) describe the length of the remaining segment (overall length of the
        %segment minus length of the lead in). If further segments are appended to the file, this number can be used to
        %locate the starting point of the following segment. If an application encountered a severe problem while writing
        %to a TDMS file (crash, power outage), all bytes of this integer can be 0xFF. This can only happen to the last
        %segment in a file.
        nlen=fread(fid,1,'uint64',kTocEndian);
        if (nlen>2^63)
            break;
        else

            segLength=nlen;
        end
        TotalLength=segLength+LeadInByteCount;
        CurrPosn=CurrPosn+TotalLength;
```

```
        status=fseek(fid,CurrPosn,'bof');       %Move to the beginning position ↩
            of the next segment
        if (status<0)
            warning('file glitch');
            break;
        end
    else  %TDSm should be the first charaters in a tdms file.  If not there, ↩
        error out to stop hunting.
        fclose(fid);
        error('Unable to find TDSm tag. This may not be a tdms file, or you ↩
            forgot to add the .tdms extension to the filename and are reading the↩
             wrong file');

    end

end

frewind(fid);

CurrPosn=0;
SegInfo.SegStartPosn=zeros(segCnt,1);
SegInfo.MetaStartPosn=zeros(segCnt,1);
SegInfo.DataStartPosn=zeros(segCnt,1);
SegInfo.vernum=zeros(segCnt,1);
SegInfo.DataLength=zeros(segCnt,1);
SegInfo.NumChan=zeros(segCnt,1);

SegInfo.SegHasMetaData=false(segCnt,1);
SegInfo.SegHasRawData=false(segCnt,1);
SegInfo.SegHasDaqMxRaw=false(segCnt,1);
SegInfo.SegInterleaved=false(segCnt,1);
SegInfo.SegBigEndian=false(segCnt,1);
SegInfo.SegHasNewObjList=false(segCnt,1);
segCnt=0;


while (ftell(fid) ~= eoff)

    Ttag=fread(fid,1,'uint8','l');
    Dtag=fread(fid,1,'uint8','l');
    Stag=fread(fid,1,'uint8','l');
    mtag=fread(fid,1,'uint8','l');

    if Ttag==84 && Dtag==68 && Stag==83 && mtag==109
        %Apparently, this sequence of numbers identifies the start of a new ↩
            segment.
        %Leaving the above comment in, because it reflects that state of
        %the NI documenation on TDMS files when the contributers first started
        %developing this code.

        segCnt=segCnt+1;

        if segCnt==1
            StartPosn=0;
        else
            StartPosn=CurrPosn;
        end

        %ToC Field
        ToC=fread(fid,1,'uint32','l');
        kTocBigEndian=bitget(ToC,7);
        kTocMetaData=bitget(ToC,2);
        kTocNewObjectList=bitget(ToC,3);
        kTocRawData=bitget(ToC,4);
        kTocDaqMxRawData=bitget(ToC,8);
        kTocInterleavedData=bitget(ToC,6);

        if kTocBigEndian
            kTocEndian='b';
        else
```

```
            kTocEndian='l';
        end

        %         if kTocInterleavedData
        %             error([sprintf(['\n Seqment %.0f of the above file has ↩
            interleaved data which is not supported with this '...
        %                 'function. '],segCnt),'Interleaved Data Not Supported↩
            ']);
        %             fclose(fid);
        %         end

%         if kTocBigEndian
%             error(sprintf(['\n Seqment %.0f of the above file uses the big-↩
    endian data format which is not supported '...
%                 'with this function. '],segCnt),'Big-Endian Data Format Not ↩
    Supported');
%             fclose(fid);
%         end

        if kTocDaqMxRawData
            error(sprintf(['\n Seqment %.0f of the above file contains data in ↩
                the DAQmxRaw NI datatype format which is not supported '...
                'with this function. See help documentation in convertTDMS.m for ↩
                    how to fix this. '],segCnt),'DAQmxRawData Format Not ↩
                    Supported');
            fclose(fid);
        end

        %TDMS format version number
        vernum=fread(fid,1,'uint32',kTocEndian);
        if ~ismember(vernum,[4712,4713])
            error(sprintf(['\n Seqment %.0f of the above file used LabView TDMS ↩
                file format version %.0f which is not '...
                'supported with this function (%s.m).'],segCnt,vernum),...
                'TDMS File Format Not Supported');
            fclose(fid);
        end

        %From the National Instruments web page (http://zone.ni.com/devzone/cda/↩
            tut/p/id/5696) under the 'Lead In'
        %description on page 2:
        %The next eight bytes (64-bit unsigned integer) describe the length of ↩
            the remaining segment (overall length of the
        %segment minus length of the lead in). If further segments are appended ↩
            to the file, this number can be used to
        %locate the starting point of the following segment. If an application ↩
            encountered a severe problem while writing
        %to a TDMS file (crash, power outage), all bytes of this integer can be 0↩
            xFF. This can only happen to the last
        %segment in a file.
        segLength=fread(fid,1,'uint64',kTocEndian);
        metaLength=fread(fid,1,'uint64',kTocEndian);
        if (segLength>2^63)
            fseek(fid,0,'eof');
            flen=ftell(fid);
            segLength=flen-LeadInByteCount-TotalLength;
            TotalLength=segLength+LeadInByteCount;
        else
            TotalLength=segLength+LeadInByteCount;
            CurrPosn=CurrPosn+TotalLength;
            fseek(fid,CurrPosn,'bof');      %Move to the beginning position of ↩
                the next segment
        end


        SegInfo.SegStartPosn(segCnt)=StartPosn;
        SegInfo.MetaStartPosn(segCnt)=StartPosn+LeadInByteCount;
        SegInfo.DataStartPosn(segCnt)=SegInfo.MetaStartPosn(segCnt)+metaLength;
        SegInfo.DataLength(segCnt)=segLength-metaLength;
        SegInfo.vernum(segCnt)=vernum;
```

```
        SegInfo.SegHasMetaData(segCnt)=kTocMetaData;
        SegInfo.SegHasRawData(segCnt)=kTocRawData;
        SegInfo.SegHasDaqMxRaw(segCnt)=kTocDaqMxRawData;
        SegInfo.SegInterleaved(segCnt)=kTocInterleavedData;
        SegInfo.SegBigEndian(segCnt)=kTocBigEndian;
        SegInfo.SegHasNewObjList(segCnt)=kTocNewObjectList;


    end

end
NumOfSeg=segCnt;
end


%%
function [index SegInfo]=getChannelInfo(fid,SegInfo,NumOfSeg)
%Loop through the segements and get all of the object information.

% name:  Short name such as Object3
% long_name: The path directory form of the object name
% rawdatacount: number of segments that have raw data for this object
% datastartindex:  Absolute position in file of the begining of all raw
%       data for the segment.  Add the rawdata offset to get to this object's
%       start point.
% arrayDim:  array dimension of raw data (for now should always be one as
%       NI does not support multi-dimension
% nValues:  number of values of raw data in segment
% byteSize: number of bytes for string/char data
% index:  segment numbers that contain raw data for this object
% rawdataoffset: offset from datastartindex for the begining of raw data
%       for this object
% multiplier: As part of LV's optimization, it will appenend raw data writes to ↩
    the
%       TDMS file when meta data does not change, but forget to update the
%       nValues.  multiplier is the number of times the base pattern
%       repeats.
% skip: When multiple raw data writes are appended, this is the number of
%   bytes between the end of a channel in one block to the begining of that ↩
    channel in
%   the next block.
% dataType:  The data type for this object. See LV2MatlabDataType function.
% datasize:  Number of bytes for each raw data entry.
% Property Structures:  Structure containg properties.


%Initialize variables for the file conversion
index=struct();
objOrderList={};


for segCnt=1:NumOfSeg

    %Go to the segment starting position of the segment
    fseek(fid,SegInfo.SegStartPosn(segCnt)+28,'bof');
    % +28 bytes: TDSm (4) + Toc (4) + segVer# (4) + segLength (8) + metaLength ↩
        (8)

    %segVersionNum=SegInfo.vernum(segCnt);
    kTocMetaData=SegInfo.SegHasMetaData(segCnt);
    kTocNewObjectList=SegInfo.SegHasNewObjList(segCnt);
    kTocRawData=SegInfo.SegHasRawData(segCnt);
    kTocInterleavedData=SegInfo.SegInterleaved(segCnt);
    kTocBigEndian=SegInfo.SegBigEndian(segCnt);

    if kTocBigEndian
        kTocEndian='b';
    else
```

```
        kTocEndian='l';
    end

    %% Process Meta Data

    %If the object list from the last segment should be used....
    if (kTocNewObjectList==0)
        fnm=fieldnames(index);   %Get a list of the objects/channels to loop ↩
            through
        for kk=1:length(fnm)
            ccnt=index.(fnm{kk}).rawdatacount;
            if (ccnt>0)   %If there is raw data info in the previous segement, ↩
                copy it into the new segement
                if (index.(fnm{kk}).index(ccnt)==segCnt-1)
                    ccnt=ccnt+1;
                    index.(fnm{kk}).rawdatacount=ccnt;
                    index.(fnm{kk}).datastartindex(ccnt)=SegInfo.DataStartPosn(↩
                        segCnt);
                    index.(fnm{kk}).arrayDim(ccnt)=index.(fnm{kk}).arrayDim(ccnt↩
                        -1);
                    index.(fnm{kk}).nValues(ccnt)=index.(fnm{kk}).nValues(ccnt-1)↩
                        ;
                    index.(fnm{kk}).byteSize(ccnt)=index.(fnm{kk}).byteSize(ccnt↩
                        -1);
                    index.(fnm{kk}).index(ccnt)=segCnt;
                    index.(fnm{kk}).rawdataoffset(ccnt)=index.(fnm{kk}).↩
                        rawdataoffset(ccnt-1);
                    SegInfo.NumChan(segCnt)=SegInfo.NumChan(segCnt)+1;
                end
            end
        end
    end


    %If there is Meta data in the segement
    if kTocMetaData
        numObjInSeg=fread(fid,1,'uint32',kTocEndian);
        if (kTocNewObjectList)
            objOrderList=cell(numObjInSeg,1);
        end
        for q=1:numObjInSeg

            obLength=fread(fid,1,'uint32',kTocEndian);                          %↩
                Get the length of the objects name
            ObjName=convertToText(fread(fid,obLength,'uint8','l'))';    %Get the ↩
                objects name

            if strcmp(ObjName,'/')
                long_obname='Root';
            else
                long_obname=ObjName;

                %Delete any apostrophes.  If the first character is a slash (↩
                    forward or backward), delete it too.
                long_obname(strfind(long_obname,''''))=[];
                if strcmpi(long_obname(1),'/') || strcmpi(long_obname(1),'\')
                    long_obname(1)=[];
                end
            end
            newob=0;
            %Create object's name.  Use a generic field name to avoid issues with↩
                 strings that are too long and/or
            %characters that cannot be used in MATLAB variable names.  The actual↩
                 channel name is retained for the final
            %output structure.
            if exist('ObjNameList','var')
                %Check to see if the object already exists
                NameIndex=find(strcmpi({ObjNameList.LongName},long_obname)==1,1,'↩
                    first');
                if isempty(NameIndex)
```

```
                newob=1;
                %It does not exist, so create the generic name field name
                ObjNameList(end+1).FieldName=sprintf('Object%.0f',numel(↩
                    ObjNameList)+1);
                ObjNameList(end).LongName=long_obname;
                NameIndex=numel(ObjNameList);
            end
        else
            %No objects exist, so create the first one using a generic name ↩
                field name.
            ObjNameList.FieldName='Object1';
            ObjNameList.LongName=long_obname;
            NameIndex=1;
            newob=1;
        end
        %Assign the generic field name
        obname=ObjNameList(NameIndex).FieldName;

        %Create the 'index' structure
        if (~isfield(index,obname))
            index.(obname).name=obname;
            index.(obname).long_name=long_obname;
            index.(obname).rawdatacount=0;
            index.(obname).datastartindex=zeros(NumOfSeg,1);
            index.(obname).arrayDim=zeros(NumOfSeg,1);
            index.(obname).nValues=zeros(NumOfSeg,1);
            index.(obname).byteSize=zeros(NumOfSeg,1);
            index.(obname).index=zeros(NumOfSeg,1);
            index.(obname).rawdataoffset=zeros(NumOfSeg,1);
            index.(obname).multiplier=ones(NumOfSeg,1);
            index.(obname).skip=zeros(NumOfSeg,1);
        end
        if (kTocNewObjectList)
            objOrderList{q}=obname;
        else
            if ~ismember(obname,objOrderList)
                objOrderList{end+1}=obname;
            end
        end
        %Get the raw data Index
        rawdataindex=fread(fid,1,'uint32',kTocEndian);

        if rawdataindex==0
            if segCnt==0
                e=errordlg(sprintf('Seqment %.0f within ''%s'' has ''↩
                    rawdataindex'' value of 0 (%s.m).',segCnt,...
                    TDMSFileNameShort,mfilename),'Incorrect ''rawdataindex'''↩
                        );
                uiwait(e)
            end
            if kTocRawData
                if (kTocNewObjectList)
                    ccnt=index.(obname).rawdatacount+1;
                else
                    ccnt=index.(obname).rawdatacount;
                end
                index.(obname).rawdatacount=ccnt;
                index.(obname).datastartindex(ccnt)=SegInfo.DataStartPosn(↩
                    segCnt);
                index.(obname).arrayDim(ccnt)=index.(obname).arrayDim(ccnt-1)↩
                    ;
                index.(obname).nValues(ccnt)=index.(obname).nValues(ccnt-1);
                index.(obname).byteSize(ccnt)=index.(obname).byteSize(ccnt-1)↩
                    ;
                index.(obname).index(ccnt)=segCnt;
                SegInfo.NumChan(segCnt)=SegInfo.NumChan(segCnt)+1;
            end
        elseif rawdataindex+1==2^32
            %Objects raw data index matches previous index - no changes.  The↩
                 root object will always have an
```

```
                %'FFFFFFFF' entry
                if strcmpi(index.(obname).long_name,'Root')
                    index.(obname).rawdataindex=0;
                else
                    %Need to account for the case where an object (besides the '↩
                        root') is added that has no data but reports
                    %using previous.
                    if newob
                        index.(obname).rawdataindex=0;
                    else
                        if kTocRawData
                            if (kTocNewObjectList)
                                ccnt=index.(obname).rawdatacount+1;
                            else
                                ccnt=index.(obname).rawdatacount;
                            end
                            index.(obname).rawdatacount=ccnt;
                            index.(obname).datastartindex(ccnt)=SegInfo.↩
                                DataStartPosn(segCnt);
                            index.(obname).arrayDim(ccnt)=index.(obname).arrayDim↩
                                (ccnt-1);
                            index.(obname).nValues(ccnt)=index.(obname).nValues(↩
                                ccnt-1);
                            index.(obname).byteSize(ccnt)=index.(obname).byteSize↩
                                (ccnt-1);
                            index.(obname).index(ccnt)=segCnt;
                            SegInfo.NumChan(segCnt)=SegInfo.NumChan(segCnt)+1;
                        end
                    end
                end
            else
                %Get new object information
                if (kTocNewObjectList)
                    ccnt=index.(obname).rawdatacount+1;
                else
                    ccnt=index.(obname).rawdatacount;
                    if (ccnt==0)
                        ccnt=1;
                    end
                end
                index.(obname).rawdatacount=ccnt;
                index.(obname).datastartindex(ccnt)=SegInfo.DataStartPosn(segCnt)↩
                    ;
                %index(end).lenOfIndexInfo=fread(fid,1,'uint32');

                index.(obname).dataType=fread(fid,1,'uint32',kTocEndian);
                if (index.(obname).dataType~=32)
                    index.(obname).datasize=getDataSize(index.(obname).dataType);
                end
                index.(obname).arrayDim(ccnt)=fread(fid,1,'uint32',kTocEndian);
                index.(obname).nValues(ccnt)=fread(fid,1,'uint64',kTocEndian);
                index.(obname).index(ccnt)=segCnt;
                SegInfo.NumChan(segCnt)=SegInfo.NumChan(segCnt)+1;
                if index.(obname).dataType==32
                    %Datatype is a string
                    index.(obname).byteSize(ccnt)=fread(fid,1,'uint64',kTocEndian↩
                        );
                else
                    index.(obname).byteSize(ccnt)=0;
                end

            end

            %Get the properties
            numProps=fread(fid,1,'uint32',kTocEndian);
            if numProps>0

                if isfield(index.(obname),'PropertyInfo')
                    PropertyInfo=index.(obname).PropertyInfo;
                else
```

```
            clear PropertyInfo
        end
        for p=1:numProps
            propNameLength=fread(fid,1,'uint32',kTocEndian);
            switch 1
                case 1
                    PropName=fread(fid,propNameLength,'*uint8','l')';
                    PropName=native2unicode(PropName,'UTF-8');
                case 2
                    PropName=fread(fid,propNameLength,'uint8=>char','l')↩
                        ';
                otherwise
            end
            propsDataType=fread(fid,1,'uint32',kTocEndian);

            %Create property's name.  Use a generic field name to avoid ↩
                issues with strings that are too long and/or
            %characters that cannot be used in MATLAB variable names.  ↩
                The actual property name is retained for the
            %final output structure.
            if exist('PropertyInfo','var')
                %Check to see if the property already exists for this ↩
                    object.  Need to get the existing 'PropertyInfo'
                %structure for this object.  The 'PropertyInfo' structure↩
                     is not necessarily the same for every
                %object in the data file.
                PropIndex=find(strcmpi({PropertyInfo.Name},PropName));
                if isempty(PropIndex)
                    %Is does not exist, so create the generic name field ↩
                        name
                    propExists=false;
                    PropIndex=numel(PropertyInfo)+1;
                    propsName=sprintf('Property%.0f',PropIndex);
                    PropertyInfo(PropIndex).Name=PropName;
                    PropertyInfo(PropIndex).FieldName=propsName;
                else
                    %Assign the generic field name
                    propExists=true;
                    propsName=PropertyInfo(PropIndex).FieldName;
                end
            else
                %No properties exist for this object, so create the first↩
                     one using a generic name field name.
                propExists=false;
                PropIndex=p;
                propsName=sprintf('Property%.0f',PropIndex);
                PropertyInfo(PropIndex).Name=PropName;
                PropertyInfo(PropIndex).FieldName=propsName;
            end

            dataExists=isfield(index.(obname),'data');

            %Get the number of samples already found and in the object
            if dataExists
                nsamps=index.(obname).nsamples+1;
            else
                nsamps=0;
            end

            if propsDataType==32 %String data type
                PropertyInfo(PropIndex).DataType='String';
                propsValueLength=fread(fid,1,'uint32',kTocEndian);
                propsValue=convertToText(fread(fid,propsValueLength,'↩
                    uint8=>char',kTocEndian))';
                if propExists
                    if isfield(index.(obname).(propsName),'cnt')
                        cnt=index.(obname).(propsName).cnt+1;
                    else
                        cnt=1;
                    end
```

```
                index.(obname).(propsName).cnt=cnt;
                index.(obname).(propsName).value{cnt}=propsValue;
                index.(obname).(propsName).samples(cnt)=nsamps;
            else
                if strcmp(index.(obname).long_name,'Root')
                    %Header data
                    index.(obname).(propsName).name=index.(obname).↩
                        long_name;
                    index.(obname).(propsName).value={propsValue};
                    index.(obname).(propsName).cnt=1;
                else
                    index.(obname).(propsName).name=PropertyInfo(↩
                        PropIndex).Name;
                    index.(obname).(propsName).datatype=PropertyInfo(↩
                        PropIndex).DataType;
                    index.(obname).(propsName).cnt=1;
                    index.(obname).(propsName).value=cell(nsamps,1);↩
                            %Pre-allocation
                    index.(obname).(propsName).samples=zeros(nsamps↩
                        ,1); %Pre-allocation
                    if iscell(propsValue)
                        index.(obname).(propsName).value(1)=↩
                            propsValue;
                    else
                        index.(obname).(propsName).value(1)={↩
                            propsValue};
                    end
                    index.(obname).(propsName).samples(1)=nsamps;
                end
            end
        else %Numeric data type
            if propsDataType==68 %Timestamp
                PropertyInfo(PropIndex).DataType='Time';
                %Timestamp data type

                if kTocBigEndian
                    tsec=fread(fid,1,'uint64','b')+fread(fid,1,'↩
                        uint64','b')/2^64;  %time since Jan-1-1904 in↩
                         seconds
                else
                    tsec=fread(fid,1,'uint64','l')/2^64+fread(fid,1,'↩
                        uint64','l');  %time since Jan-1-1904 in ↩
                        seconds
                end
                %tsec=fread(fid,1,'uint64',kTocEndian)/2^64+fread(fid↩
                    ,1,'uint64',kTocEndian);   %time since Jan-1-1904↩
                     in seconds
                %R. Seltzer: Not sure why '5/24' (5 hours) is ↩
                    subtracted from the time value.  That's how it ↩
                    was
                %coded in the original function I downloaded from ↩
                    MATLAB Central.  But I found it to be 1 hour too
                %much.  So, I changed it to '4/24'.
                %propsValue=tsec/86400+695422-5/24; %/864000 convert ↩
                    to days; +695422 days from Jan-0-0000 to Jan↩
                    -1-1904
                propsValue=tsec/86400+695422-4/24;  %/864000 convert ↩
                    to days; +695422 days from Jan-0-0000 to Jan↩
                    -1-1904
            else  %Numeric
                PropertyInfo(PropIndex).DataType='Numeric';
                matType=LV2MatlabDataType(propsDataType);
                if strcmp(matType,'Undefined')
                    e=errordlg(sprintf('No MATLAB data type defined ↩
                        for a ''Property Data Type'' value of ''%.0f'↩
                        '.',...
                        propsDataType),'Undefined Property Data Type'↩
                            );
                    uiwait(e)
                    fclose(fid);
```

```
                            return
                        end
                        if strcmp(matType,'uint8=>char')
                            propsValue=convertToText(fread(fid,1,'uint8',↩
                                kTocEndian));
                        else
                            propsValue=fread(fid,1,matType,kTocEndian);
                        end
                    end
                    if propExists
                        cnt=index.(obname).(propsName).cnt+1;
                        index.(obname).(propsName).cnt=cnt;
                        index.(obname).(propsName).value(cnt)=propsValue;
                        index.(obname).(propsName).samples(cnt)=nsamps;
                    else
                        index.(obname).(propsName).name=PropertyInfo(↩
                            PropIndex).Name;
                        index.(obname).(propsName).datatype=PropertyInfo(↩
                            PropIndex).DataType;
                        index.(obname).(propsName).cnt=1;
                        index.(obname).(propsName).value=NaN(nsamps,1);     ↩
                                %Pre-allocation
                        index.(obname).(propsName).samples=zeros(nsamps,1); ↩
                                %Pre-allocation
                        index.(obname).(propsName).value(1)=propsValue;
                        index.(obname).(propsName).samples(1)=nsamps;
                    end
                end

            end %'end' for the 'Property' loop
            index.(obname).PropertyInfo=PropertyInfo;

        end

    end %'end' for the 'Objects' loop
end




%Address Decimation and Interleaving
if (kTocRawData)

    singleSegDataSize=0;
    rawDataBytes=0;
    chanBytes=0;
    rawDataOffset=0;

    for kk=1:numel(objOrderList)
        obname=objOrderList{kk};
        ccnt=hasRawDataInSeg(segCnt,index.(obname));
        if ccnt>0   %If segement has raw data
            index.(obname).rawdataoffset(ccnt)=rawDataOffset;
            if index.(obname).dataType==32 %Datatype is a string
                rawDataBytes=index.(obname).byteSize(ccnt);
            else
                rawDataBytes=index.(obname).nValues(ccnt)*index.(obname).↩
                    datasize;
                if kTocInterleavedData
                    chanBytes=index.(obname).datasize;
                else
                    chanBytes=rawDataBytes;
                end

            end
            rawDataOffset=rawDataOffset+chanBytes;
            singleSegDataSize=singleSegDataSize+rawDataBytes;
        end
```

```
        end



        %Calculate the offset for each segment.  The offset is the amount
        %of bytes for one non appened (non optimized) segement
%         for kk=1:numel(objOrderList)
%             obname=objOrderList{kk};
%             if hasRawDataInSeg(segCnt,index.(obname))   %If segement has raw ↩
    data
%                 index.(obname).rawdataoffset(ccnt)=rawdataoffset;      %This ↩
    will set the offset correctly for dec data.
%
%                 %Update the singleSegDataSize value for the next object
%                 if index.(obname).dataType==32 %Datatype is a string
%                     singleSegDataSize=singleSegDataSize+index.(obname).byteSize↩
    (ccnt);
%                 else
%                     singleSegDataSize=singleSegDataSize+index.(obname).nValues(↩
    ccnt)*index.(obname).datasize;
%                 end
%
%                 if kTocInterleavedData
%                     rawdataoffset=
%                 else
%                     rawdataoffset=singleSegDataSize;
%                 end
%             end
%         end
%




        %As part of LV's optimization, it will append back to back
        %segements into one segement (when nothing changes in meta data).
        if (singleSegDataSize~=SegInfo.DataLength(segCnt))         %Multiple ↩
            appended raw segement (offset did not grow to be greater than the ↩
            DataLength)
            numAppSegs=floor(SegInfo.DataLength(segCnt)/singleSegDataSize);  %↩
                Number of appened (optimized segements)
            for kk=1:numel(objOrderList)   %Loop through all the objects
                obname=objOrderList{kk};
                ccnt=hasRawDataInSeg(segCnt,index.(obname));
                if ccnt>0   %If segement has raw data
                    if kTocInterleavedData
                        if index.(obname).dataType==32 %Datatype is a string
                            error('Interleaved string channels are not supported.↩
                                ')
                        end
                        index.(obname).multiplier(ccnt)=numAppSegs*index.(obname)↩
                            .nValues(ccnt);
                        index.(obname).skip(ccnt)=singleSegDataSize/index.(obname↩
                            ).nValues(ccnt)-index.(obname).datasize;
                        index.(obname).nValues(ccnt)=1;
                    else  %Decimated Data
                        index.(obname).multiplier(ccnt)=numAppSegs;
                        if index.(obname).dataType==32 %Datatype is a string
                            index.(obname).skip(ccnt)=singleSegDataSize-index.(↩
                                obname).byteSize(ccnt);
                        else
                            index.(obname).skip(ccnt)=singleSegDataSize-index.(↩
                                obname).nValues(ccnt)*index.(obname).datasize;
                        end
                    end
                end

            end
```

```
        else   %If single segement
            if kTocInterleavedData
                for kk=1:numel(objOrderList)   %Loop through all the objects
                    obname=objOrderList{kk};
                    ccnt=hasRawDataInSeg(segCnt,index.(obname));
                    if ccnt>0  %If segement has raw data
                        if (index.(obname).index(ccnt)==segCnt)   %If the object ↩
                            has rawdata in the current segment
                            if index.(obname).dataType==32 %Datatype is a string
                               error('Interleaved string channels are not ↩
                                   supported.')
                            end
                            index.(obname).multiplier(ccnt)=index.(obname).↩
                                nValues(ccnt);
                            index.(obname).skip(ccnt)=singleSegDataSize/index.(↩
                                obname).nValues(ccnt)-index.(obname).datasize;
                            index.(obname).nValues(ccnt)=1;
                        end
                    end
                end
            end
        end



%         if (offset~=SegInfo.DataLength(segCnt))         %If the offset grew to ↩
    be greater than the DataLength (from Meta Data)
%             multiplier=floor(SegInfo.DataLength(segCnt)/offset);
%             for kk=1:numel(objOrderList)   %Loop through the objects
%                 obname=objOrderList{kk};
%                 ccnt=index.(obname).rawdatacount;
%                 if ccnt>0
%                     index.(obname).multiplier(segCnt)=multiplier;
%                     if index.(obname).dataType==32 %Datatype is a string
%                         index.(obname).skip(ccnt)=offset-index.(obname).↩
    byteSize(ccnt);
%                     else
%                         index.(obname).skip(ccnt)=offset-index.(obname).nValues↩
    (ccnt)*index.(obname).datasize;
%                     end
%                 end
%             end
%
%         end

        aa=1;
%         %Now adjust multiplier and skip if the data is interleaved
%         if kTocInterleavedData
%             if muliplier>0  %Muliple raw data segements appened
%
%             else   %Single raw segement
%
%             end
%         end

    end

end
%clean up the index if it has to much data
fnm=fieldnames(index);
for kk=1:numel(fnm)
    ccnt=index.(fnm{kk}).rawdatacount+1;

    index.(fnm{kk}).datastartindex(ccnt:end)=[];
    index.(fnm{kk}).arrayDim(ccnt:end)=[];
    index.(fnm{kk}).nValues(ccnt:end)=[];
    index.(fnm{kk}).byteSize(ccnt:end)=[];
    index.(fnm{kk}).index(ccnt:end)=[];
    index.(fnm{kk}).rawdataoffset(ccnt:end)=[];
    index.(fnm{kk}).multiplier(ccnt:end)=[];
```

```
    index.(fnm{kk}).skip(ccnt:end)=[];

end
end


%%
function ob=getData(fid,index,SegInfo)
%Using the file id (fid) and the index database, get the raw data.
%Returns the data in the ob structure.  The fields in the structure are the
%generic object names:
%   ob.Object3.data - raw data
%   ob.Object3.nsamples - number of samples
%
%   Note that not all of the objects in the index may not be in ob as it
%   contains only those objects which have raw data.

ob=[];
fnm=fieldnames(index);   %Get the object names
for kk=1:length(fnm)    %Loop through objects
    id=index.(fnm{kk});
    nsamples=sum(id.nValues.*id.multiplier);
    if id.rawdatacount>0  %Only work with channels with raw data
        cname=id.name;
        ob.(cname).nsamples=0;

        %Initialize the data matrix
        if id.dataType==32
            ob.(cname).data=cell(nsamples,1);
        else
            ob.(cname).data=zeros(nsamples,1);
        end

        for rr=1:id.rawdatacount
            %Loop through each of the segments and read the raw data

            %Move to the raw data start position
            fseek(fid,id.datastartindex(rr)+id.rawdataoffset(rr),'bof');

            nvals=id.nValues(rr);

            segmentNum=index.(cname).index(rr);
            segInterleaved=SegInfo.SegInterleaved(segmentNum);

            if SegInfo.SegBigEndian(segmentNum)
                kTocEndian='b';
            else
                kTocEndian='l';
            end

            numChan=SegInfo.NumChan(segmentNum);

            if nvals>0  %If there is data in this segement

                switch id.dataType

                    case 32     %String
                        %From the National Instruments web page (http://zone.ni.↩
                            com/devzone/cda/tut/p/id/5696) under the
                        %'Raw Data' description on page 4:
                        %String type channels are preprocessed for fast random ↩
                            access. All strings are concatenated to a
                        %contiguous piece of memory. The offset of the first ↩
                            character of each string in this contiguous piece
                        %of memory is stored to an array of unsigned 32-bit ↩
                            integers. This array of offset values is stored
                        %first, followed by the concatenated string values. This ↩
                            layout allows client applications to access
                        %any string value from anywhere in the file by ↩
                            repositioning the file pointer a maximum of three ↩
```

```
                    times
                %and without reading any data that is not needed by the ↩
                    client.
                data=cell(1,nvals*id.multiplier(rr));   %Pre-allocation
                for mm=1:id.multiplier(rr)
                    StrOffsetArray=fread(fid,nvals,'uint32','l');
                    for dcnt=1:nvals
                        if dcnt==1
                            StrLength=StrOffsetArray(dcnt);
                        else
                            StrLength=StrOffsetArray(dcnt)-StrOffsetArray↩
                                (dcnt-1);
                        end
                        data{1,dcnt+(mm-1)*nvals}=char(convertToText(↩
                            fread(fid,StrLength,'uint8=>char','l'))');
                    end
                    if (id.multiplier(rr)>1)&&(id.skip(rr)>0)
                        fseek(fid,id.skip(rr),'cof');
                    end
                end
                cnt=nvals*id.multiplier(rr);

            case 68     %Timestamp
                %data=NaN(1,nvals); %Pre-allocation
                data=NaN(1,nvals*id.multiplier(rr));
                for mm=1:id.multiplier(rr)
                    dn=fread(fid,2*nvals,'uint64',kTocEndian);
                    tsec=dn(1:2:end)/2^64+dn(2:2:end);
                    data((mm-1)*nvals+1:(mm)*nvals)=tsec↩
                        /86400+695422-4/24;
                    fseek(fid,id.skip(rr),'cof');
                end
                %{
                for dcnt=1:nvals
                    tsec=fread(fid,1,'uint64')/2^64+fread(fid,1,'uint64')↩
                        ;   %time since Jan-1-1904 in seconds
                    %R. Seltzer: Not sure why '5/24' (5 hours) is ↩
                        subtracted from the time value.  That's how it ↩
                        was
                    %coded in the original function I downloaded from ↩
                        MATLAB Central.  But I found it to be 1 hour too
                    %much.  So, I changed it to '4/24'.
                    data(1,dcnt)=tsec/86400+695422-5/24;    %/864000 ↩
                        convert to days; +695422 days from Jan-0-0000 to ↩
                        Jan-1-1904
                    data(1,dcnt)=tsec/86400+695422-4/24;    %/864000 ↩
                        convert to days; +695422 days from Jan-0-0000 to ↩
                        Jan-1-1904
                end
                %}
                cnt=nvals*id.multiplier(rr);

            otherwise   %Numeric
                matType=LV2MatlabDataType(id.dataType);
                if strcmp(matType,'Undefined')  %Bad Data types catch
                    e=errordlg(sprintf('No MATLAB data type defined for a↩
                         ''Raw Data Type'' value of ''%.0f''.',...
                        id.dataType),'Undefined Raw Data Type');
                    uiwait(e)
                    fclose(fid);
                    return
                end
                if (id.skip(rr)>0)
                    ntype=sprintf('%d*%s',nvals,matType);
                    [data,cnt]=fread(fid,nvals*id.multiplier(rr),ntype,id↩
                        .skip(rr),kTocEndian);
                    if strcmp(matType,'uint8=>char')
                        data=convertToText(data);
                    end
                else
```

```
                            % Added by Haench start
                            if  (id.dataType == 524300) || (id.dataType == ↩
                                1048589) % complex CDB data
                                [data,cnt]=fread(fid,2*nvals*id.multiplier(rr),↩
                                    matType,kTocEndian);
                                data= data(1:2:end)+1i*data(2:2:end);
                                cnt = cnt/2;
                            else
                                [data,cnt]=fread(fid,nvals*id.multiplier(rr),↩
                                    matType,kTocEndian);
                            end
                            % Haench end
                            % Original: [data,cnt]=fread(fid,nvals*id.multiplier(↩
                                rr),matType,kTocEndian);
                        end
                end

                %Update the sample counter
                if isfield(ob.(cname),'nsamples')
                    ssamples=ob.(cname).nsamples;
                else
                    ssamples=0;
                end
                if (cnt>0)
                    ob.(cname).data(ssamples+1:ssamples+cnt,1)=data;
                    ob.(cname).nsamples=ssamples+cnt;
                end
            end
        end

    end

end
end



function [DataStructure,GroupNames]=postProcess(ob,index)
%Re-organize the 'ob' structure into a more user friendly format for output.


DataStructure.Root=[];
DataStructure.MeasuredData.Name=[];
DataStructure.MeasuredData.Data=[];

obFieldNames=fieldnames(index);

cntData=1;

for i=1:numel(obFieldNames)

    cname=obFieldNames{i};

    if strcmp(index.(cname).long_name,'Root')

        DataStructure.Root.Name=index.(cname).long_name;

        %Assign all the 'Property' values
        if isfield(index.(cname),'PropertyInfo')
            for p=1:numel(index.(cname).PropertyInfo)
                cfield=index.(cname).PropertyInfo(p).FieldName;
                if isfield(index.(cname).(cfield),'datatype')
                    DataType=index.(cname).(cfield).datatype;
                else
                    %ASSUME a 'string' data type
                    DataType='String';
                end
                DataStructure.Root.Property(p).Name=index.(cname).PropertyInfo(p)↩
                    .Name;
```

```
            switch DataType
                case 'String'
                    if iscell(index.(cname).(cfield).value)
                        Value=index.(cname).(cfield).value';
                    else
                        Value=cellstr(index.(cname).(cfield).value);
                    end

                case 'Time'
                    clear Value
                    if index.(cname).(cfield).cnt==1
                        if iscell(index.(cname).(cfield).value)
                            Value=datestr(cell2mat(index.(cname).(cfield).↩
                                value),'dd-mmm-yyyy HH:MM:SS');
                        else
                            Value=datestr(index.(cname).(cfield).value,'dd-↩
                                mmm-yyyy HH:MM:SS');
                        end
                    else
                        Value=cell(index.(cname).(cfield).cnt,1);
                        for c=1:index.(cname).(cfield).cnt
                            if iscell(index.(cname).(cfield).value)
                                Value(c)={datestr(cell2mat(index.(cname).(↩
                                    cfield).value),'dd-mmm-yyyy HH:MM:SS')};
                            else
                                Value(c)={datestr(index.(cname).(cfield).↩
                                    value,'dd-mmm-yyyy HH:MM:SS')};
                            end
                        end
                    end

                case 'Numeric'
                    if isfield(index.(cname).(cfield),'cnt')
                        Value=NaN(index.(cname).(cfield).cnt,1);
                    else
                        if iscell(index.(cname).(cfield).value)
                            Value=NaN(numel(cell2mat(index.(cname).(cfield).↩
                                value)),1);
                        else
                            Value=NaN(numel(index.(cname).(cfield).value),1);
                        end
                    end
                    for c=1:numel(Value)
                        if iscell(index.(cname).(cfield).value)
                            Value(c)=index.(cname).(cfield).value{c};
                        else
                            Value(c)=index.(cname).(cfield).value(c);
                        end
                    end
                otherwise
                    e=errordlg(sprintf(['No format defined for Data Type ''%s↩
                        '' in the private function ''postProcess'' '...
                        'within %s.m.'],index.(cname).(cfield).datatype,↩
                            mfilename),'Undefined Property Data Type');
                    uiwait(e)
                    return
            end
            if isempty(Value)
                DataStructure.Root.Property(p).Value=[];
            else
                DataStructure.Root.Property(p).Value=Value;
            end
        end
    end


end

DataStructure.MeasuredData(cntData).Name=index.(cname).long_name;
```

```
%Should only need the 'ShortName' for debugging the function
%DataStructure.MeasuredData(cntData).ShortName=cname;
if (isfield(ob,cname))
    if isfield(ob.(cname),'data')
        DataStructure.MeasuredData(cntData).Data=ob.(cname).data;
        %The following field is redundant because the information can be ↩
            obtained from the size of the 'Data' field.
        DataStructure.MeasuredData(cntData).Total_Samples=ob.(cname).nsamples↩
            ;
    else
        DataStructure.MeasuredData(cntData).Data=[];
        DataStructure.MeasuredData(cntData).Total_Samples=0;
    end
else
    DataStructure.MeasuredData(cntData).Data=[];
    DataStructure.MeasuredData(cntData).Total_Samples=0;
end

%Assign all the 'Property' values
if isfield(index.(cname),'PropertyInfo')
    for p=1:numel(index.(cname).PropertyInfo)
        cfield=index.(cname).PropertyInfo(p).FieldName;
        DataStructure.MeasuredData(cntData).Property(p).Name=index.(cname).(↩
            cfield).name;

        if strcmpi(DataStructure.MeasuredData(cntData).Property(p).Name,'Root↩
            ')
            Value=index.(cname).(cfield).value;
        else

            switch index.(cname).(cfield).datatype
                case 'String'
                    clear Value
                    if index.(cname).(cfield).cnt==1
                        if iscell(index.(cname).(cfield).value)
                            Value=char(index.(cname).(cfield).value);
                        else
                            Value=index.(cname).(cfield).value;
                        end
                    else
                        Value=cell(index.(cname).(cfield).cnt,1);
                        for c=1:index.(cname).(cfield).cnt
                            if iscell(index.(cname).(cfield).value)
                                Value(c)=index.(cname).(cfield).value;
                            else
                                Value(c)={index.(cname).(cfield).value};
                            end
                        end
                    end

                case 'Time'
                    clear Value
                    if index.(cname).(cfield).cnt==1
                        if iscell(index.(cname).(cfield).value)
                            Value=datestr(cell2mat(index.(cname).(cfield).↩
                                value),'dd-mmm-yyyy HH:MM:SS');
                        else
                            Value=datestr(index.(cname).(cfield).value,'dd-↩
                                mmm-yyyy HH:MM:SS');
                        end
                    else
                        Value=cell(index.(cname).(cfield).cnt,1);
                        for c=1:index.(cname).(cfield).cnt
                            if iscell(index.(cname).(cfield).value)
                                Value(c)={datestr(cell2mat(index.(cname).(↩
                                    cfield).value),'dd-mmm-yyyy HH:MM:SS')};
                            else
                                Value(c)={datestr(index.(cname).(cfield).↩
                                    value,'dd-mmm-yyyy HH:MM:SS')};
                            end
```

```
                        end
                    end

                case 'Numeric'
                    if isfield(index.(cname).(cfield),'cnt')
                        Value=NaN(index.(cname).(cfield).cnt,1);
                    else
                        if iscell(index.(cname).(cfield).value)
                            Value=NaN(numel(cell2mat(index.(cname).(cfield).↩
                                value)),1);
                        else
                            Value=NaN(numel(index.(cname).(cfield).value),1);
                        end
                    end
                    for c=1:numel(Value)
                        if iscell(index.(cname).(cfield).value)
                            Value(c)=index.(cname).(cfield).value{c};
                        else
                            Value(c)=index.(cname).(cfield).value(c);
                        end
                    end

                otherwise
                    e=errordlg(sprintf(['No format defined for Data Type ''%s↩
                        '' in the private function ''postProcess'' '...
                        'within %s.m.'],index.(cname).(cfield).datatype,↩
                            mfilename),'Undefined Property Data Type');
                    uiwait(e)
                    return
            end
        end
        if isempty(Value)
            DataStructure.MeasuredData(cntData).Property(p).Value=[];
        else
            DataStructure.MeasuredData(cntData).Property(p).Value=Value;
        end
    end
else
    DataStructure.MeasuredData(cntData).Property=[];
end

cntData = cntData + 1;
end %'end' for the 'groups/channels' loop

%Extract the Group names
GroupIndices=false(numel(DataStructure.MeasuredData),1);
for d=1:numel(DataStructure.MeasuredData)

    if ~strcmpi(DataStructure.MeasuredData(d).Name,'Root')
        if (DataStructure.MeasuredData(d).Total_Samples==0)
            fs=strfind(DataStructure.MeasuredData(d).Name,'/');
            if (isempty(fs))
                GroupIndices(d)=true;
            end
        end
    end

end
if any(GroupIndices)
    GroupNames=sort({DataStructure.MeasuredData(GroupIndices).Name})';
else
    GroupNames=[];
end

end


function SegInfo=addChanCount(SegInfo,channelinfo)
%This function determines the number of channels in each segement by
%looping through the channels in channelinfo.  It looks at their index,
```

```
%which conatins the segments numbers in which that channel has data.
%It then increments the NumChan field in the segment info database
%(SegInfo).

fnm=fieldnames(channelinfo);   %Get the channels
for ccnt=1:numel(fnm)           %Loop through the channels
    channelName=fnm{ccnt};
    ind=channelinfo.(channelName).index;   %Get the index
    for icnt=1:numel(ind)
        segNum=ind(icnt);
        SegInfo.NumChan(segNum)=SegInfo.NumChan(segNum)+1;
    end
end
end

function sz=getDataSize(LVType)
%Get the number of bytes for each LV data type. See LV2MatlabDataType.
switch(LVType)
    case 0
        sz=0;
    case {1,5,33}
        sz=1;
    case 68
        sz=16;
    case {8,10}
        sz=8;
    case {3,7,9}
        sz=4;
    case {2,6}
        sz=2;
    case 32
        e=errordlg('Do not call the getDataSize function for strings.  Their size↩
            is written in the data file','Error');
        uiwait(e)
        sz=NaN;
    case 11
        sz=10;
    % Added by Haench for tdsTypeComplexSingleFloat=0x08000c,↩
        tdsTypeComplexDoubleFloat=0x10000d,
    case 524300
        sz=8;
    case 1048589
        sz=16;
    % end add haench
    otherwise
        error('LVData type %d is not defined',LVType)
end
end



function matType=LV2MatlabDataType(LVType)
%Cross Refernce Labview TDMS Data type to MATLAB
switch LVType
    case 0   %tdsTypeVoid
        matType='';
    case 1   %tdsTypeI8
        matType='int8';
    case 2   %tdsTypeI16
        matType='int16';
    case 3   %tdsTypeI32
        matType='int32';
    case 4   %tdsTypeI64
        matType='int64';
    case 5   %tdsTypeU8
        matType='uint8';
    case 6   %tdsTypeU16
        matType='uint16';
    case 7   %tdsTypeU32
```

```
        matType='uint32';
    case 8   %tdsTypeU64
        matType='uint64';
    case 9  %tdsTypeSingleFloat
        matType='single';
    case 10  %tdsTypeDoubleFloat
        matType='double';
    case 11  %tdsTypeExtendedFloat
        matType='10*char';
    case 25 %tdsTypeSingleFloat with units
        matType='Undefined';
    case 26 %tdsTypeDoubleFloat with units
        matType='Undefined';
    case 27 %tdsTypeextendedFloat with units
        matType='Undefined';
    case 32  %tdsTypeString
        matType='uint8=>char';
    case 33  %tdsTypeBoolean
        matType='bit1';
    case 68  %tdsTypeTimeStamp
        matType='2*int64';
    % Added by Haench for tdsTypeComplexSingleFloat=0x08000c,↩
        tdsTypeComplexDoubleFloat=0x10000d,
   case 524300
        matType='single';
    case 1048589
        matType='double';
    % end add haench
    otherwise
        matType='Undefined';
end

end

function ccnt=hasRawDataInSeg(segCnt,ob)
%Function to check and see if a channel has raw data in a particulr
%segment.  If it does, return the index of index (ccnt).  If not present,
%return ccn=0

indexList=ob.index;
ccnt=find(indexList==segCnt);
if isempty(ccnt)
    ccnt=0;
else
    ccnt=ccnt(1);
end
end

function text=convertToText(bytes)
%Convert numeric bytes to the character encoding localy set in MATLAB (TDMS uses ↩
    UTF-8)
text=native2unicode(bytes,'UTF-8');
end
```

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS