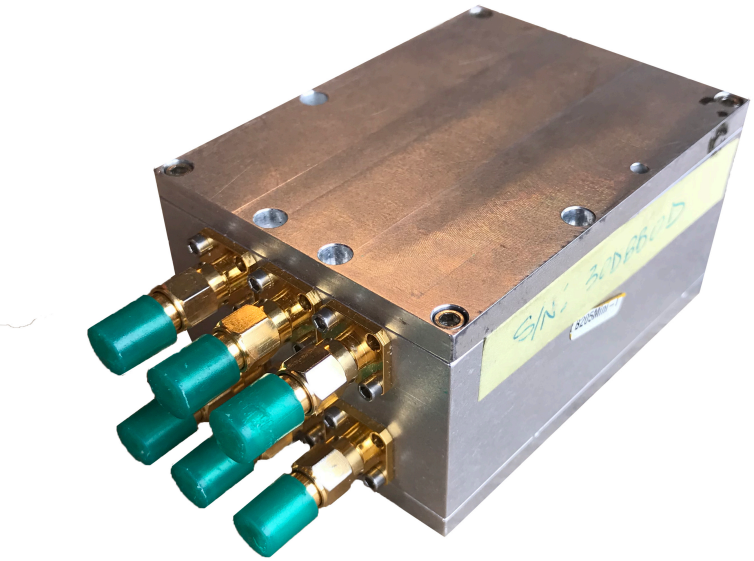
CHALMERS



Transmissionsmätning med USRP B205mini-i

Kandidatarbete EENX15-18-32

ARVID BJURKLINT, KLARA GRANBOM, JENS NILSSON,
TOBIAS SANDSTRÖM, ROBIN SUNDIN, JENS ÅKERLUND

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2018-05-14

Abstract

In this project, the possibility of implementing transmission measurements using a Software Defined Radio has been investigated, hoping to use this technology in the field of medical diagnostics in the future. Different methods have been used to investigate the ability to measure received phase and amplitude of a transmitted signal. Measured data could then be compared to the transmitted signal and used to recreate the structure of a test object for which the signal passed through.

Repeated measurements on the same experimental setup generated different results regarding phase and amplitude. Therefore, reference measurements had to be carried out in parallel to get comparable data between runs. A number of methods were evaluated but to draw any definite conclusions more tests are needed.

Sammandrag

I det här projektet har möjligheten att genomföra transmissionsmätningar med hjälp av en Software Defined Radio undersökts, med förhoppning om att i framtiden kunna använda denna teknik inom medicinsk diagnostik. Olika metoder har använts för att undersöka möjligheten att mäta mottagen fas och amplitud för en skickad signal. Uppmätt data skulle sedan kunna jämföras med den skickade signalen och användas för att återskapa strukturen av ett testobjekt för vilken signalen passerat genom.

Upprepade mätningar på samma mätuppställning genererade olika resultat gällande fas och amplitud. Därför behövdes referensmätningar genomföras parallellt för att få jämförbar data mellan körningar. Ett antal metoder utvärderades men mer mätningar krävs för att kunna dra några definitiva slutsatser.

Ordlista

ADC	Analog till digital konverterare (Analog to Digital Converter)
CW	Kontinuerlig sinussignal (Continuous Wave)
DAC	Digital till analog konverterare (Digital to Analog Converter)
DFT	Diskret fouriertransform (Discrete Fourier Transform)
DSP	Digital signalprocessor (Digital Signal Processor)
FFT	Snabb fouriertransform (Fast Fourier Transform)
FMCW	Frekvensmodulerad kontinuerlig sinussignal (Frequency Modulated Continuous Wave)
FPGA	På-plats-programmerbar grindmatris (Field-Programmable Gate Array)
GPIO	Generell ingång/utgång (General-Purpose Input/Output)
IF	Mellanfrekvens (Intermediate Frequency)
LO	Lokaloscillator (Local Oscillator)
RF	Radiofrekvens (Radio Frequency)
SB	Sidband (Side Band)
SDR	Mjukvarudefinierad radio (Software defined radio)
UHD	USRP-drivrutiner (USRP Hardware Drivers)
USB/LSB	Övre och undre sidband (Upper/Lower SideBand)
USRP	Universal Software Radio Peripheral
VNA	Nätverksanalysator (Vector Network Analyzer)
RX	Mottagarport (Receiver port)
TX	Sändarport (Transmitter port)
TRX	Mottagar- och sändarport (Transciever port)

Innehåll

1

Inledning

1.1 Bakgrund

Genom att skicka mikrovågor på olika frekvenser mellan antenner finns möjligheter att konstruera instrument med tillämpningar bland annat inom medicinsk diagnostik. Exempel på tillämpningar är detektion av stroke och bröstcancer [1], [2]. Diagnostiseringen utförs genom att mikrovågor sänds genom objektet som studeras för att därefter se hur mottagen signal förändras då den passerat genom eller reflekterats av objektet [3]. Genom att studera hur amplitud och fas förändras går det att dra slutsatser kring objektets materialegenskaper. Om det är en strokepatient vars hjärna undersöks går det potentiellt att avgöra om patienten har en hjärnblödning eller en blodpropp [3].

I dagsläget forskas det på att använda nätverksanalysatorer för att utföra mätningar. Den modell som kommer att jämföras med i denna rapport är *Rhode & Schwarz - ZNBT8* med 16 portar som har dimensionerna 463 mm $\times$ 240 mm $\times$ 612 mm och väger 31 kg [4]. Vid strokefall är det är viktigt att avgöra om det är en blodpropp eller blödning i hjärnan för att ge rätt behandling. Behandlingen behövs göras omgående då två miljoner hjärnceller dör per minut vid propputlöst stroke. Därför är det viktigt att kunna avgöra vilken typ av stroke en patient har så fort som möjligt. Då nätverksanalysatorn är stor och tung hade ett mer portabelt mätsystem varit bättre för sådan typ av diagnostik [5]. Med billigare mätsystem hade även ambulanser kunnat utrustas och fler patienter hade kunnat undersökas i ett tidigare skede.

I projektet undersöks det om en Software Defined Radio (SDR) kan användas för att utföra transmissionsmätningar och även undersöka hur resultaten står sig mot en nätverksanalysator från *Rhode & Schwartz*. I projektet kommer en USRP B205mini-i framtagen av *Ettus Research* [6] användas. Dimensionerna för B205mini-i är 83.3 x 50.8 x 8.4 mm och det väger 24 gram [7] vilket gör det mer portabelt än nätverksanalysatorn. B205mini-i är billig i jämförelse med en nätverksanalysator, men har endast två portar.

1.2 Syfte

Syftet med arbetet är att undersöka om det är möjligt att använda en Software Defined Radio (SDR), specifikt en B205mini-i, för att utföra transmissionsmätningar och mäta mottagen fas och amplitud. Om det är möjligt skulle B205mini-i potentiellt kunna användas för tillämpningar inom mikrovågsdiagnostik för medicinskt bruk.

För att begränsa projektets storlek har det valts att endast titta på transmission mellan två antenner, en sändar- och en mottagarantenn. Resultat som fås och mätmetoder som utvecklas kan sedan potentiellt överföras till system för mikrovågsdiagnostik som består av fler antenner.

Olika mjukvaror, GNURadio, LabView, `Matlab`, Python och C++, kommer att utforskas för att se vilken, eller vilka, som fungerar bäst för arbetet. Resultat som fås med B205mini-i kommer jämföras med en nätverksanalysator som idag används för mikrovågsdiagnostik på forskningsnivå. Rapporten kommer att dokumentera transmissionsmätningarnas utförande så att de enkelt kan upprepas av andra inom samma fält.

2

Teori

2.1 Mikrovågsteknik för medicinsk diagnostik

Möjligheten att använda mikrovågsteknik inom medicinsk diagnostik bygger på att det finns en dielektrisk skillnad mellan friska och sjuka vävnader. Ett system för mikrovågsdiagnostik består av antenner och mätelektronik för att avgöra spridningen av utsända mikrovågssignaler [1]. Hur en mikrovågssignal sprids beror på dielektriska egenskaper i det område som signalen transmitteras genom. Genom att undersöka spridningsdata kan det bland annat avgöras om sjuk vävnad fanns i området mellan antennerna.

2.2 Software Defined Radio

Mjukvarudefinerad radio, eng: Software Defined Radio (SDR), är en radio där komponenter som i vanliga radiosystem är implementerade med hårdvara istället är implementerade som mjukvara. Med ett sådant system är det möjligt att byta egenskaper, till exempel frekvensområde, genom att använda en dator eller ett inbyggt system istället för byte av komponenter i sändare och mottagare [8].

En SDR har minst en sändarport och/eller mottagarport och innehåller en analog till digital-omvandlare, eng: Analog to Digital Converter (ADC) och/eller digital till analog-omvandlare, eng: Digital to Analog Converter (DAC). I en SDR-mätuppställning kommunicerar SDR:n med en dator eller ett inbyggt system med hjälp av den digitala signalen. I en mätuppställning ingår vanligtvis någon typ av RF-front end, vilket är en krets som består av komponenter som bearbetar signalen på den ursprungliga frekvensen, samt ofta någon typ av antenn.

2.3 B205mini-i

Den SDR som används i projektet är av modellen B205mini-i, framtagen av *Ettus Research*. Se kretsschema för B205mini-i i figur 2.1. B205mini-i är ett kort som är mindre än och billigare i förhållande till utrustning som redan finns och används för transmissionsmätningar. Kortets pris samt storlek och därmed portabilitet kan

göra det möjligt att skapa portabel och billigare utrustning för mikrovågsbaserad medicinsk diagnostik.

B205mini-i har tre portar: Sänd- och mottagarport (TRX), mottagarport (RX) och en port för extern klocka och puls per sekund-synkronisering (REF).

B205mini-i-kortet kan operera på frekvenser mellan 70 MHz till 6 GHz, med en maximal bandbredd på 56 MHz. Kortet har en inkanal och en utkanal, eng: Single Input Single Output (SISO), som kan köras i full duplex-läge, det vill säga det kan sända och ta emot parallellt. Det finns ytterligare modeller i B200- och B210-serierna, däribland kort med flera in- och utkanaler, eng: Multiple Input Multiple Output (MIMO).

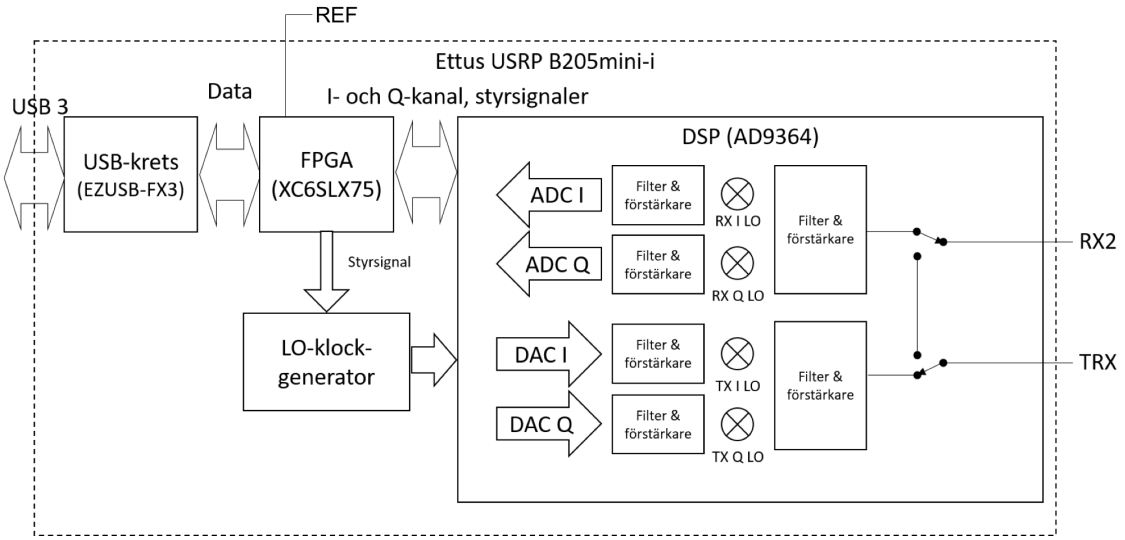
Kortets analoga-digitala gränssnitt sköts av dess digitala signalbehandlar-krets, eng: Digital Signal Processor (DSP), av typen AD9364 [9]. DSP-kretsen har i sin tur två gränssnitt, ett för mottagna signaler samt ett för att transmittera signaler.

Transmitterade signaler börjar som digitala signaler på två kanaler, kallade I- och Q-kanaler [10]. En av kanalerna är i fas, I-kanalen, eng: In-phase, och den andra, Q-kanalen är fasförskjuten 90 grader, eng: Quadrature. Kanalerna kan ses som en komplex representation där I-kanalen och Q-kanalen är realdelen respektive imaginärdelen. Dessa mellanfrekvenssignaler, eng: Intermediate Frequency (IF), konverteras sedan till analoga med en DAC, och mixas därefter upp till radiofrekvens-signaler (RF) som kan transmitteras med till exempel ett antennsystem.

De mottagna signalerna gör i stort sett samma resa som transmitterade. De mixas ner till mellanfrekvenser från RF, för att därefter konverteras till digitala signaler med en ADC, till en I- och en Q-kanal.

Med I- och Q-kanaler är det möjligt att definiera negativa frekvenser beroende på om Q-kanalen är 90 grader före eller 90 grader efter i fas i förhållande till I-kanalen.

B205mini-i-kortet har även en på-plats-programmerbar grindmatris, eng: Field-Programmable Gate Array (FPGA), som är kortets styrenhet. FPGA:n styr bland annat kortets DSP, switchen för att växla mellan mottagar- och sändarport, TRX-port, eller mottagarport, RX-port, samt dataströmmarna från och till datorn. FPGA-kretsen kan programmeras om vid behov för att ändra hur kortet styrs. Omprogrammering bör dock göras med försiktighet eftersom det kan skada kortet om exempelvis fel kommandon skickas till DSP:n.



Figur 2.1: Kretsschema för *Ettus Research* B205mini-i

2.4 Elektromagnetiska vågor

Den utsända signalen från ett SDR-kort utsätts för en mängd dissipativa effekter så som reflektioner vid gränsytan mellan medium och dämpning. Vid gränsytor mellan medium kan andelen strålning som transmitteras och reflekteras bestämmas med Fresnels ekvationer [11]. Ekvationerna tar hänsyn till egenskaper hos de medium som finns kring gränsytan. Dämpning vid propagation beror på vågens frekvens, mediumens relativa permittivitet, relativa permeabilitet och konduktivitet, som även kan förändra vågens fas. Genom att mäta dessa effekter är det möjligt att bestämma egenskaper eller förändringar i det medium som undersöks.

2.5 Decibel - dB

Decibel är en måttenhet som ofta kommer upp i samband med elektriska signaler, exempelvis för att beskriva förstärkning av en signal. Det är ett logaritmiskt mått och är definierat enligt

$$\mathrm{dB} = 10\log_{10}\left(\frac{\text{effekt}}{\text{referensvärde}}\right). \tag{2.1}$$

Lägg märke till att skalan utgår från effekt och inte amplitud av en signal. Effekten är dock proportionell mot amplituden av en signal i kvadrat, vilket ger

$$\mathrm{dB} = 10\log_{10}\left(\frac{(\text{amplitud})^2}{(\text{referensvärde})^2}\right) = 20\log_{10}\left(\frac{\text{amplitud}}{\text{referensvärde}}\right). \tag{2.2}$$

2.5.1 dBm

I rapporten anges effekt stundvis med måttet dBm. Det beskriver effekten i förhållande till en referens på en milliwatt, 1 mW, enligt

$$\text{dBm} = 10\log_{10}\left(\frac{\text{effekt (mW)}}{1\text{ (mW)}}\right). \tag{2.3}$$

2.6 Antennegenskaper

För att utföra mikrovågsdiagnostik krävs transmission av en signal med hjälp av ett antennsystem. Om ett brett frekvensområde ska kunna sändas och tas emot är det fördelaktigt med en bredbandig antenn och om en signal ska sändas eller tas emot i en riktning är en antenn med hög direktivitet önskvärt.

Direktivitet, D, är ett mått på hur starkt en antenn kan sända eller ta emot i en given riktning och är definierat enligt

$$D = G_D(\phi,\theta)\Big|_{max}, \tag{2.4}$$

där

$$G_D(\phi,\theta) = \frac{U(\phi,\theta)}{\frac{1}{4\pi}\int_0^{\pi}\int_0^{2\pi} U(\phi,\theta)\sin\theta d\phi d\theta} \quad [11] \tag{2.5}$$

är antennförstärkningen och $U(\phi,\theta)$ är strålningsintensiteten.

Bandbredden av en antenn beror bland annat på geometrin och strömfördelningen över själva antennen. Det är ett svårt problem att beräkna strömfördelningen, även för de enklaste typerna av antenner. I de flesta fall krävs det därför numeriska metoder för att beräkna strömfördelningen och därmed bandbredden [11].

Antennförstärkning kan variera stort vid olika frekvenser för en given antenn och således även direktiviteten vilket skulle kunna orsaka problem i form av väldigt låg antennförstärkning i vissa riktningar där den varit hög vid en annan frekvens.

I detta projekt har ingen större vikt lagts i att beräkna vare sig direktivitet eller bandbredd av antennsystem som använts, men kan vara av intresse vid eventuella efterföljande projekt.

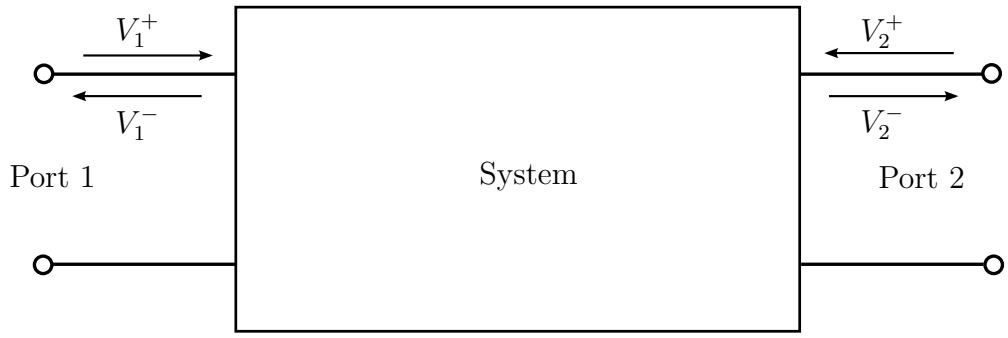
2.7 S-parametrar

I projektet läggs stor vikt vid att beräkna skillnader i fas och amplitud för signaler som propagerar respektive reflekteras i ett system. Ett mått på dessa skillnader är S-parametrar, även kallade spridningsparametrar, som i ett system med n stycken

portar definieras som

$$S_{ij} = \frac{V_i^-}{V_j^+}\Big|_{V_{k\neq i}^+=0} \quad i,j = 1,2,...,n-1,n \tag{2.6}$$

där V_i^- är spänningen som kommer ut från systemet från port i och V_j^+ är spänningen in i systemet på port j [12]. Genom att mäta de komplexa och frekvensberoende S-parametrarna för ett system kan information fås om hur systemet beter sig vid olika frekvenser. Av speciellt intresse i denna rapport är S-parametern S_{21} som är kvoten mellan signalen ut från systemet på port 2 och signalen in till systemet på port 1, se figur 2.2, som ger information om vad som händer med signalen efter transmission genom systemet.

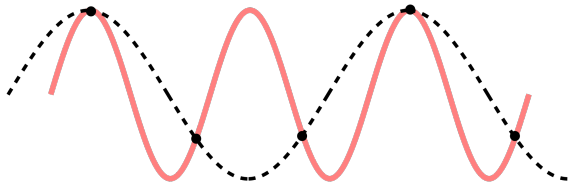


Figur 2.2: En tvåport, $n = 2$, där spridningsparametrarna beskrivs av en 2×2-matris enligt ekvation 2.6. Systemet är godtyckligt och kan till exempel vara en elektrisk krets eller ett antennsystem.

2.8 Digital signalbehandling

2.8.1 Sampling

Sampling handlar om att göra om en kontinuerlig signal till en diskret signal och används ofta vid analog till digital omvandling. Att spela in musik med en mikrofon och lagra musiken i en dator är ett exempel på sampling. De fysiska ljudvågorna får mikrofonens membran att vibrera och den skickar vidare denna analoga signal till datorn. Datorn i sin tur samplar signalen genom att upprepat mäta dess värde med en viss frekvens, samples per sekund, och sparar sedan dessa mätvärden som digital data. Ju högre samplingsfrekvens desto mer lik den analoga signalen blir den samplade digitala signalen. Nyquist-Shannons samplingsteorem säger att signalen måste samplas med minst dubbelt så hög frekvens som den högsta frekvensen som finns i signalen [13]. Används lägre samplingsfrekvens så kan fel som i figur 2.3 erhållas, där den samplade signalen har hälften så hög frekvens som originalsignalen [14].



Figur 2.3: Den dubbla frekvensen av sinusvågen är högre än samplingsfrekvensen vilket här gör att den samplade signalen får häften så hög frekvens som originalsignalen [14].

2.8.2 Fouriertransform

I rapporten används diskreta fouriertransformer för att föra över en signal från tids- till frekvensdomän. Detta för att enklare kunna analysera amplitud och fas för specifika frekvenser.

I tidsdomänen ses en signal som en funktion av tiden likt sinusvågen i figur 2.3. För signaler innehållandes flera frekvenser är det ofta av intresse att undersöka dess frekvensspektrum, det vill säga fördelningen av den ursprungliga signalen som funktion av frekvenser, med hjälp av fouriertransform. Det är många gånger fördelaktigt att arbeta i frekvensdomänen vid signalbehandling, exempelvis vid filtrering och brusreducering, då oönskade frekvenser enklare går att lokalisera.

Vid digital signalbehandling byggs signaler upp av en ändlig sekvens samples. För att undersöka frekvensspektrumet appliceras en diskret fouriertransform (DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi kn/N} \tag{2.7}$$

där både in- och utdata utgörs av N komplexa tal. Den transformerade datan är indelad i N diskreta bins. Varje bin innehåller datapunkter för ett visst frekvensintervall. Då antalet bins bestäms av antalet samplade punkter bestäms bredden av frekvensintervallen av samplingsfrekvensen. Högsta möjliga frekvens för vilken en DFT kan detektera frekvenser är halva samplingsfrekvensen, se avsnitt 2.8.1. DFT:n har en komplexitet $\mathcal{O}(N^2)$, något som enkelt verifieras genom att räkna antalet summationer i definitionen. En vanligt förekommande algoritm, kallad snabb fouriertransform (FFT), minskar antalet aritmetiska operationer till $\mathcal{O}(N \log N)$.

För mer ingående information om Fouriertransformen hänvisas läsaren till [13].

2.8.3 Signaltransmission

Digitala transmissionssystem bygger ofta på att en digitalt genererad signal vid låga frekvenser flyttas upp till högre frekvenser genom någon form av analog krets. Skälet till detta är att signalgenererande digital till analog konverterare (DAC) inte klarar högre frekvenser. För att nyttja en större del av spektrumet blandas därför

signalen med en lokaloscillator (LO) enligt heterodynprincipen, med hjälp av en blandare, eng: mixer. Frekvensen för LO kallas i rapporten för centerfrekvens och den lågfrekventa för mellanfrekvens, eng: Intermediate Frequency (IF). Matematiskt kan blandningen av signalen och LO beskrivas som att en sinussignal, $\mathrm{g}_{\mathrm{signal}}(t) = \cos(f_{\mathrm{signal}} 2\pi t)$ multipliceras med en LO som även den är en sinussignal, $\mathrm{g}_{\mathrm{LO}}(t) = \cos(f_{\mathrm{LO}} 2\pi t)$, där f_{signal} och f_{LO} är frekvenserna för signalen respektive lokaloscillator. Den resulterande produkten blir

$$\mathrm{g}_{\mathrm{produkt}}(t) = \frac{1}{2}\cos((f_{\mathrm{LO}} + f_{\mathrm{signal}})2\pi t) + \frac{1}{2}\cos((f_{\mathrm{LO}} - f_{\mathrm{signal}})2\pi t).$$

Principen är densamma för övriga signaler då de med hjälp av fourieranalys kan delas upp i en summa av sinussignaler enligt

$$\mathrm{g}_{\mathrm{signal}}(t) = \sum_i A_i \cos(f_i 2\pi t),$$

där f_i är de frekvenser som signalen består av och A_i är motsvarande amplituder. Motsvarande hetereodynmixad signal ges av

$$\mathrm{g}_{\mathrm{produkt}}(t) = \sum_i A_i(\frac{1}{2}\cos((f_{\mathrm{LO}} + f_{\mathrm{i}})2\pi t) + \frac{1}{2}\cos((f_{\mathrm{LO}} - f_{\mathrm{i}})2\pi t)).$$

Signalen blir alltså speglad runt centerfrekvensen, vilket innebär att samma information skickas på signalens undre och övre sidband, eng: upper/lower sideband (USB/LSB). För att minimera användandet av frekvensrymden är det bra om ett av sidbanden dämpas ut. Den effektiva bandbredden fördubblas då det oanvända sidbandet kan användas för annan överföring. Som nämnt i avsnitt 2.3 kan digitala signalsystem åstadkomma enkelsidiga transmissioner genom att använda två fasförskjutna lokaloscillatorer. Den komplexa utsignalen ges av

$$\begin{aligned}\mathrm{g}_{\mathrm{produkt}}(t) = \sum_i \mathrm{I}_i(t)(\frac{1}{2}\cos((f_{\mathrm{LO}} + f_{\mathrm{i}})2\pi t) + \frac{1}{2}\cos((f_{\mathrm{LO}} - f_{\mathrm{i}})2\pi t)) + \\ \sum_i \mathrm{Q}_i(t)(\frac{1}{2}\cos((f_{\mathrm{LO}} + f_{\mathrm{i}})2\pi t) + \frac{1}{2}\cos((f_{\mathrm{LO}} - f_{\mathrm{i}})2\pi t)),\end{aligned}$$

där fourierkoefficienterna ges av den komplexa signalen som ska skickas enligt

$$\mathrm{I}(t) = \mathrm{Re}(\mathrm{g}_{\mathrm{signal}}(t)),$$

$$\mathrm{Q}(t) = \mathrm{Im}(\mathrm{g}_{\mathrm{signal}}(t)).$$

2.8.4 Testsignaler för uppmätning av amplitud- och fasegenskaper

Den enklaste typen av signal är en kontinuerlig sinussignal, eng: Continuous Wave (CW). Matematiskt beskrivs signalstyrkan för CW

$$\mathrm{s}(t) = \cos(f_{\mathrm{signal}} 2\pi t).$$

Vid blandning med enbart en lokaloscillator skulle det resulterande spektrumet innehålla två frekvenser $f_{\text{LO}} \pm f_{\text{signal}}$. För att endast få med den positiva frekvensen kan den analytiska signalen beräknas enligt

$$\hat{s}(t) = \cos(f_{\text{signal}} 2\pi t - \frac{\pi}{2}) = \sin(f_{\text{signal}} 2\pi t),$$

och

$$s_{\text{analytisk}} = s(t) + j\hat{s}(t) = \cos(f_{\text{signal}} 2\pi t - \frac{\pi}{2}) + j \sin(f_{\text{signal}} 2\pi t).$$

För ett bredare spektrum kan en summa av CW användas, där antal signaler och dess frekvenser väljs efter ändamål. Dock minskar amplituden för varje frekvens med antalet, vilket kan vara en nackdel i brusiga förhållanden.

3

Metod

3.1 Val av mjukvara

För att utföra transmissionsmätningar behövdes en programvara

- som fungerar tillsammans med USRP B205mini-i-kortet,
- som separat kan hantera sändar- och mottagardel på kortet,
- där det går att specificera vilken typ av signal som skickas till sändarporten på kortet,
- som antingen kan utföra signalbehandling eller spara signalen till en fil så att signalbehandling kan utföras senare med annan programvara.

För frekvenssvep och fasmätning krävs mer av programvaran såsom synkronisering av sändare och mottagare men för enklare mätuppställningar var kraven ovan tillräckliga. I början av projektet valdes att tre programvaror skulle undersökas; GNU Radio, MATLAB och LabVIEW. Senare i projektet började även C++ och Python undersökas.

3.1.1 GNU Radio Companion

GNU Radio Companion är ett grafiskt program som bygger på att sätta ihop olika block till en flödeskarta. Varje block har en uppgift, till exempel att generera en sinussignal, och det finns specifika block som kan hantera sändar- och mottagardel på kortet. Signalbehandling kan utföras i realtid men det går även att spara data till en fil. Utöver block som generar signaler går det även att importera signaler från en fil, exempelvis är det möjligt att generera en signal i MATLAB och importera till GNU Radio Companion. När flödeskartan i GNU Radio Companion kompileras genereras en Pythonfil som körs av programmet. Se B för installationsguide till GNU Radio och GNU Radio Companion.

3.1.2 Python med UHD

Den Pythonfil som genereras av flödeskartan i GNU Radio Companion kan modifieras och ger mer flexibilitet än GNU Radio Companion. Till exempel går det att skriva loopar och ändra olika variabler under körning med hjälp av funktioner från UHD-biblioteket, kortets drivrutiner. Ett arbetssätt som användes under projektet var att bygga upp grunden av programmet i GNU Radio Companion för att sedan utöka det genom modifikationer av den Pythonkod som genereras från GNU Radio Companion. Python uppfyller alla punkter i avsnitt 3.1 ovan som behövs av en programvara för att utföra transmissionsmätningar. Utöver att spara signalen till en fil går det att utföra signalbehandling och presentera resultat.

3.1.3 MATLAB

MATLAB är ett scriptprogram som har stöd för en del SDR-modeller. Det har ej stöd för USRP B205mini-i-kortet och användes därför inte vid kommunikation med kortet. MATLAB har dock stöd för andra USRP-modeller från *Ettus Research*. I projektet har MATLAB använts för signalgenerering, signalbehandling samt för presentation av resultat.

3.1.4 LabVIEW

LabVIEW är likt GNU Radio Companion ett grafiskt program där olika block byggs ihop till ett flödesschema. Licensen som krävdes för att kommunicera med kortet saknades och LabVIEW användes därför inte i projektet.

3.1.5 C++ med UHD

C++ är ett programmeringsspråk på lägre nivå än GNU Radio Companion vilket ger mer flexibilitet i användningen av kortet. Det används tillsammans med UHD-biblioteket som innehåller funktioner för att styra kortet. Det uppfyller alla kraven i listan ovan och det går även att skriva loopar vilket gör att frekvenssvep kan genomföras.

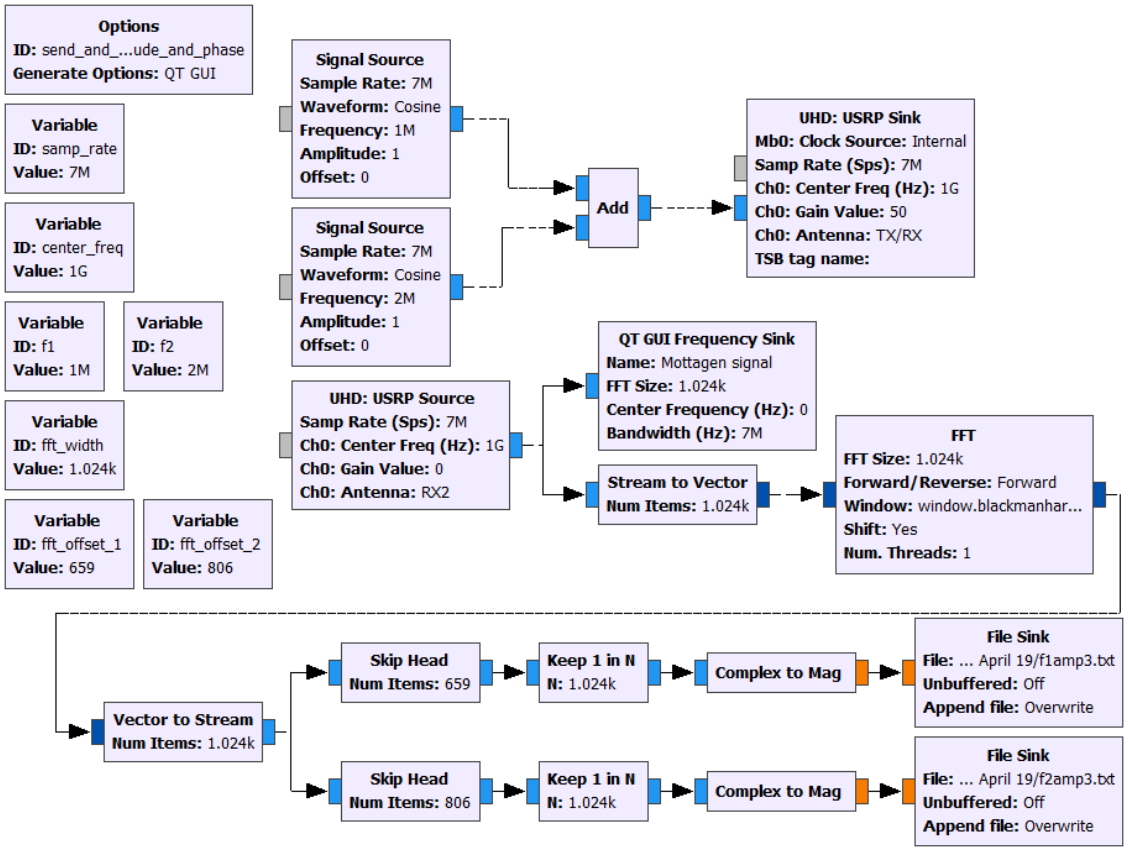
3.2 Tillämpning av mjukvara

Mjukvaran som initialt valdes att användas var GNU Radio Companion med viss modifikation av den genererade Pythonkoden.

3.2.1 GNU Radio Companion

För fas- och amplitudmätningar i GNU Radio Companion skrevs två olika program. Dels ett där databehandling skedde direkt i GNU Radio Companion med hjälp av olika block som finns i programmet samt ett enklare GNU Radio Companion-program där data sparades och behandlades i MATLAB.

3.2.1.1 Insamling och behandling av data i GNU Radio Companion



Figur 3.1: Flödesschema i GNU Radio Companion för amplitudmätning och databehandling.

I figur 3.1 adderas två signaler från `Signal Source`-block och ges som input till ett `UHD: USRP Sink`-block vilket sköter utsändning av signalen. Blocket `UHD: USRP Source` tar emot signalen. Signalen görs om till en vektor av längd 1024 i ett `Stream to Vector`-block för att sedan fouriertransformeras med FFT-fönster av längd 1024. Varje element i den fouriertransformerade vektorn svarar mot amplituden och fasen för en viss frekvens, nämligen

$$f_k \approx k\frac{\text{samplingsfrekvens}}{1024},$$

där k är index för motsvarande bin, se avsnitt 2.8.2. Vektorn görs därefter om till en ström igen. Två frekvenser undersöks och behöver plockas ut ur strömmen av fouriertransformerad data. Rätt bin, k, motsvarande de specifika frekvenserna ges av

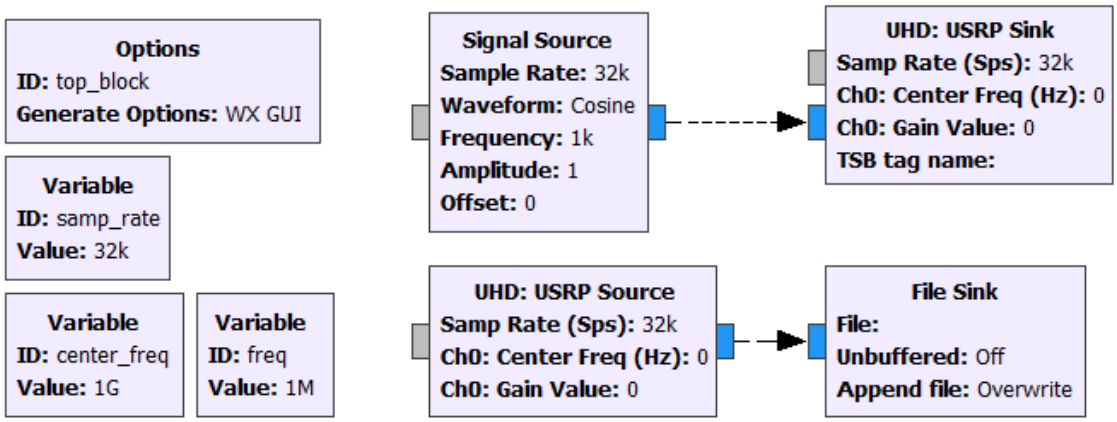
$$k \approx f \frac{1024}{\text{samplingsfrekvens}},$$

avrundat till närmaste heltal. `Skip Head`-blocket flyttar markören i programmet till rätt motsvarande element i strömmen. Varje vektor ut från `FFT`-blocket utgörs av 1024 punkter, varför `Keep 1 in N`-blocket med $N = 1024$ kontinuerligt plockar ut signalen vid önskade frekvenser. Slutligen beräknas amplituden (magnituden) av den komplexvärda datan och sparas till fil. Block för att plocka ut fas, `Complex to Arg`-block, finns också tillgängliga i GNU Radio Companion, och kan användas på motsvarande plats som `Complex to Mag`-blocken i presenterat program i figur 3.1

En viktig parameter i `UHD: USRP Source`- och `UHD: USRP Sink`-blocken är gain, det vill säga hur mycket signalen förstärks. Den kan anges i decibel eller som normaliserad. I projektet har främst gain angetts i dB och anpassats efter hur mycket dämpning som använts. Vid 30 dB dämpning sattes gainen på `UHD: USRP Sink`-blocket mellan 30 och 50 dB. Gainen på `UHD: USRP Source`-blocket sattes nästan uteslutande till 0 dB.

3.2.1.2 Databehandling i MATLAB

Istället för att genomföra databehandling i realtid kan GNU Radio Companion alternativt användas enbart för att sända och ta emot en signal. Den otransformerade signalen kan sedan sparas till fil och behandlas i annan programvara. De block som behövs i GNU Radio Companion kan då reduceras till ett `Signal Source`-block, ett `UHD: USRP Sink`-block och ett `UHD: USRP Source`-block för att sända och ta emot en signal, samt ett `File Sink`-block för lagring, se figur 3.2. Formatet på sparad data är 32-bitars flyttal där vartannat tal utgör real- respektive imaginärdel.

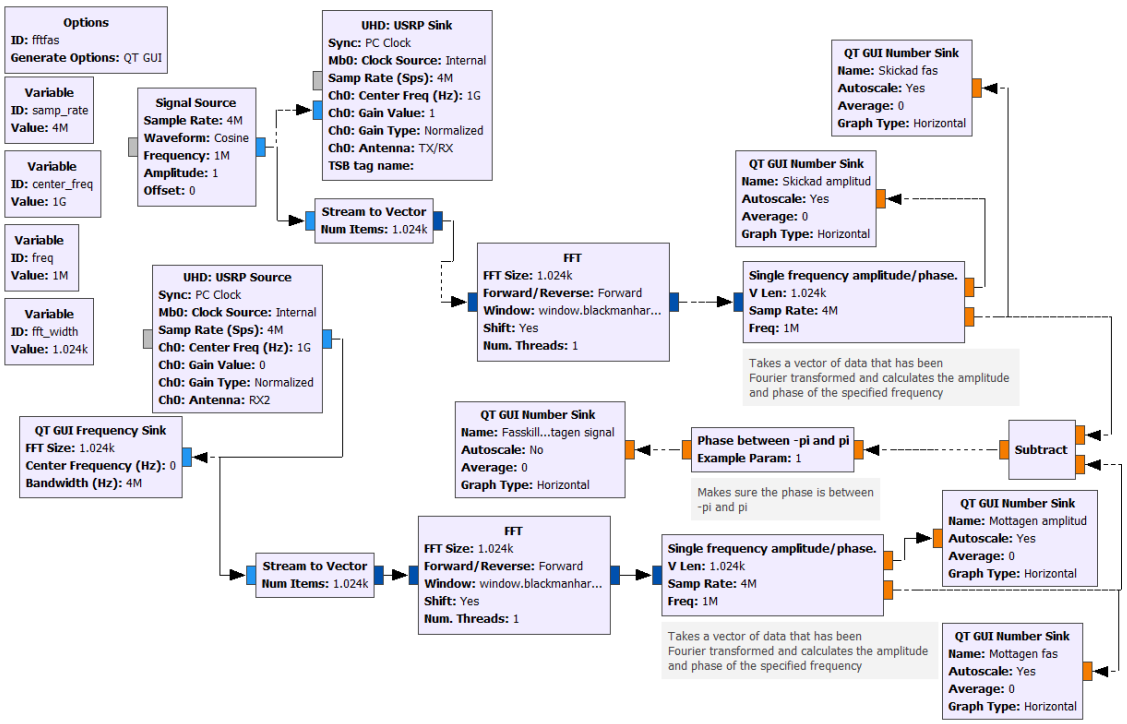


Figur 3.2: Flödesschema i GNU Radio för att spara signal till fil.

Datafilen öppnas i MATLAB med funktionerna `fopen()` och `fread()`, för den senare specificeras korrekt argument för motsvarande datatyp. I MATLAB implementerad FFT-funktion skiljer sig från GNU Radios implementation på det sätt att utdata är annorlunda ordnad. Det första elementet i utdatavektorn från FFT:n (första binen) innehåller lägst uppskattade frekvensintervall. Den mittersta binen svarar mot högst uppmätbara frekvensintervall, det vill säga vid halva samplingsfrekvensen. Efterkommande element utgörs av signalens spegling i det negativa frekvensområdet med start i minus halva samplingsfrekvensen. MATLAB har också funktioner som beräknar både fas och magnitud av komplexa tal.

3.3 Metoder för S-parametrar

Program för att mäta S-parametrar skrevs i GNU Radio Companion, se figur 3.3. Blocket `Single frequency amplitude/phase` är inte ett standardblock i GNU Radio Companion utan skrevs under projektet för att minska antalet block. Blocket kombinerar funktionen av flera andra tillgängliga block; `Vector to Stream`, `Skip Head`, `Keep 1 in N`, samt `Complex to Mag` och `Complex to Arg`, jämför med figur 3.1. Pythonkod för `Single frequency amplitude/phase`-blocket finns i appendix C.1.1. Även blocket `Phase between -pi and pi` skrevs under projektet, vilket ser till att fasskillnaden anges mellan $-\pi$ och π, för Pythonkod se avsnitt C.1.2.



Figur 3.3: Flödesschema i GNU Radio Companion som mäter amplitud hos skickad och mottagen signal samt fasskillnad mellan skickad och mottagen signal.

Den genererade signalen delas upp i två flöden varav ena sänds ut från kortets sän-

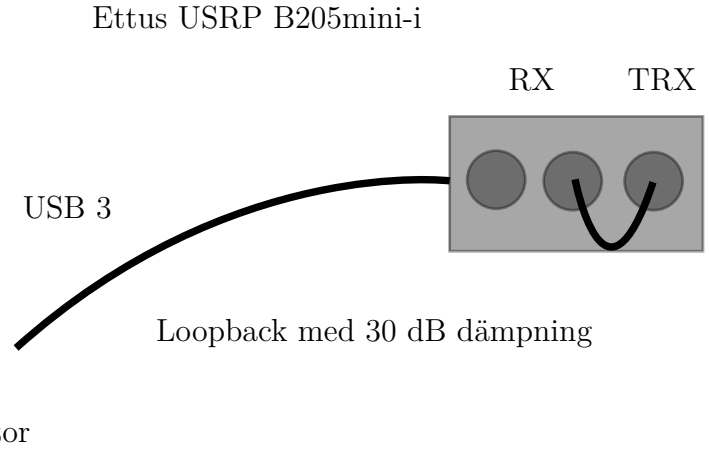
darport via blocket `UHD: USRP Sink` och den andra förs över till frekvensdomänen genom en `FFT`. Utsänd signal tas emot av blocket `UHD: USRP Source` och subtraheras, efter överföring till frekvensdomän, från skickad signal i `Subtract`-blocket. Skillnaden presenteras grafiskt genom ett `QT GUI Number Sink`-block. Övriga `QT GUI Number Sink`-block i programmet används för att ge grafisk information om så väl skickad som mottagen frekvens och fas. Blocket `QT GUI Frequency Sink` presenterar signalstyrka för frekvenser runt vald centerfrekvens för att bekräfta att mottagen signal stämmer överens med utsänd.

3.4 Experimentuppställningar

Initialt användes två experimentuppställningar beskrivna nedan. Den första användes för enkla transmissionsmätningar för att testa prestanda hos kortet, se avsnitt 3.4.1, och den andra för att mäta på ett testobjekt kallad fantom, se avsnitt 3.4.2.

3.4.1 Uppställning med loopback

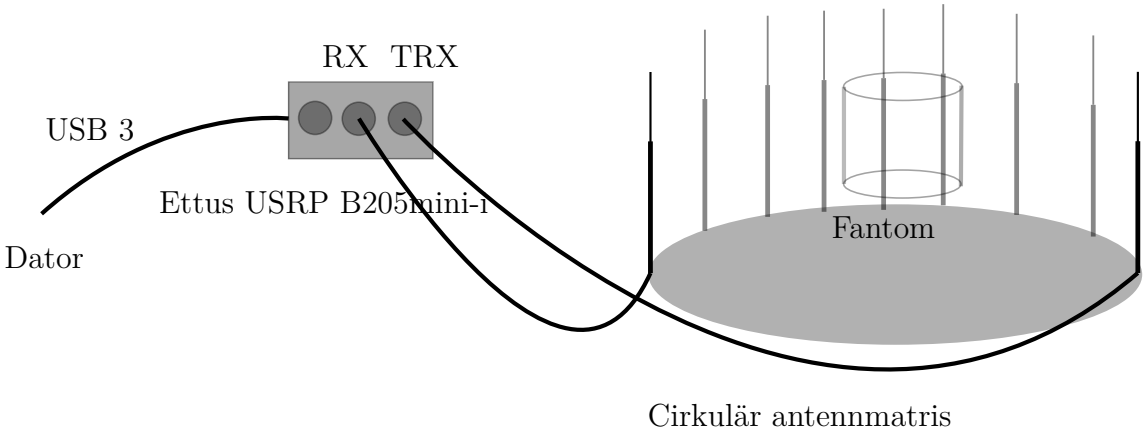
För att utföra enkla transmissionsmätningar kopplades B205mini-i med loopback, enligt figur 3.4. Loopback möjliggör att en signal kan skickas ut och mätas upp av samma kort. Fördelen med att använda loopback är att systemet blir mindre känsligt för yttre störningar såsom FM-radio, TV-sändningar och telefontrafik med mera. Dessutom kan frekvenser användas fritt utan att behöva ta hänsyn till vilka frekvenser som signaler får skickas ut på utan tillstånd. Loopback bestod av en SMA-kabel samt 30 dB dämpning. Dämpningen användes för att inte riskera att överbelasta kortet med för starka signaler.



Figur 3.4: Uppställning för att mäta amplitud- och fasegenskaper för loopback. B205mini-i-kortet kopplades till datorn via USB. Kortets sändarport (TX) och dess mottagarport (RX) var kopplade genom en SMA-kabel och 30 dB dämpare.

3.4.2 Uppställning med fantom

För att kunna mäta karakteristik för de undersökta systemen krävs någon form av kalibreringsmätning för att kompensera för externa faktorer såsom kablage, antenner etc. Dessutom var kalibreringsmätningar nödvändiga för att kunna säkerställa att det var fantomen som påverkar mätningarna. I figur 3.5 visas uppställningen som användes för att genomföra mätningar med fantom. Fantomen placerades i mitten av ett cirkulärt uppställt antennsystem, varefter mätning utfördes mellan två av antennerna så att signalen transmitterades rakt igenom fantomen. Avståndet mellan valda antenner var 15 cm, vilket motsvarade diametern på antennsystemet. Antennerna var monopoler gjorda av rigida koaxialkablar där isolering och yttre ledare skalats bort. Fantomen utgjordes av en plastflaska fylld med 88% glycerol och 12% vatten. Analogt genomfördes mätning i fri luft utan fantom.



Figur 3.5: Uppställning för att mäta amplitud- och fasegenskaper för fantom och fri luft. B2015-mini-i-kortet kopplades till datorn via USB. Kortets sändarport (TX) och dess mottagarport (RX) var kopplade till två motsatta antenner med SMA-kablar. En fantom i form av en plastflaska fylld med 88% glycerol och 12% vatten kunde placeras mellan antennerna.

4

Resultat

4.1 Variation i amplitud och fas

Det första som undersöktes i projektet var om mätdata var konsekvent mellan körningar. Genom upprepade mätningar på samma experimentuppställningar undersöktes variation av amplitud- och fasförändringar. Jämförelser av mätdata gjordes också för ändringar av variabler under körning.

4.1.1 Variation mellan körningar

En SMA-kabel kopplades mellan sändar- och mottagarport på kortet med en dämpning av 30 dB. Programmet i figur 3.3 kördes upprepade gånger för att ta reda på om amplitud och fasskillnad var konstant mellan körningar. Tabell 4.1 visar fasskillnad i radianer mellan mottagen och skickad signal samt mottagen amplitud vid olika körningar. Presenterade data visar att både amplitud och fasskillnad varierar mellan körningar, varför det inte är möjligt att göra meningsfulla jämförelser av resultat från olika körningar. Enhet för amplitud i GNU Radio Companion anges inte, varför de presenteras enhetslöst.

Tabell 4.1: Mottagen amplitud samt fasskillnad vid fem olika loopback-körningar med 30 dB dämpning.

Körning	Mottagen amplitud (-)	Fasskillnad (rad)
1	7.6	1.3
2	8.7	0.8
3	7.9	1.0
4	8.4	2.0
5	7.4	3.1

4.1.2 Variation vid byte av centerfrekvens under körning

För att genomföra frekvenssvep krävs byte av centerfrekvens under körning. Variation av amplitud och fasskillnad under byte av centerfrekvens undersöktes med hjälp av programmet i figur 3.3. Dock med några modifikationer av den genererade

Pythonkoden, se appendix C.1.3. Modifikationen utgörs av metoden `change_center_frequency_and_sleep` som skiftar centerfrekvensen fram och tillbaka mellan 1 och 1.01 GHz med ett tio sekunders intervall.

Resultaten i tabell 4.2 uppvisar variationer både hos amplitud och fasskillnad då centerfrekvensen ändras. Variationen leder till begränsningar för hur kortet kan användas, varför ett antal metoder utvärderades för att kunna genomföra konsekventa mätningar av amplitud och fasskillnad med varierande centerfrekvens.

Tabell 4.2: Mottagen amplitud samt fasskillnad vid byte mellan två centerfrekvenser, 1.00 GHz samt 1.01 GHz, vid en körning.

Fasskillnad (rad), 1.00 GHz	Fasskillnad (rad), 1.01 GHz	Amplitud (-), 1.00 GHz	Amplitud (-), 1.01 GHz
0.10	0.35	6.6	10.1
0.80	0.39	7.7	8.9
2.34	0.36	7.7	10.2
0.77	1.17	7.7	8.9
2.34	1.17	7.7	8.9
1.61	0.39	6.7	8.9
0.40	1.94	6.7	10.2

4.1.3 Uppmätt effekt och frekvens hos kortet

Amplituddata eller utsänd effekt i absoluta termer är inte tillgängliga vid mätningar med hjälp av B205mini-i-kortet. På tillhörande datablad för kortet anges endast att maximal utsänd effekt överstiger 10 dBm [7]. Effektmätningar genomfördes därför vid utvalda frekvenser med hjälp av ett oscilloskop. Kortet kopplades till *54845A Infiniium Oscilloscope*, ett oscilloskop från *Hewlett Packard*, via en koaxialkabel med SMA-kontakt. En sinussignal genererades med hjälp av GNU Radio Companion och terminerades i oscilloskopet av en last på 1 MΩ.

Uppmätt effekt fluktuerar något mellan körningar, se tabell 4.3. Att utsänd effekt ej är konsekvent för *Ettus Resarch* B200-serier är dock känt sedan tidigare och finns dokumenterat bland annat på *Ettus Mailing List* [15]. Grafisk representation av utsänd effekt finns på *Ettus Research Knowledge Base* för flera frekvenser, även för de som presenterats i tabell 4.3 nedan [16].

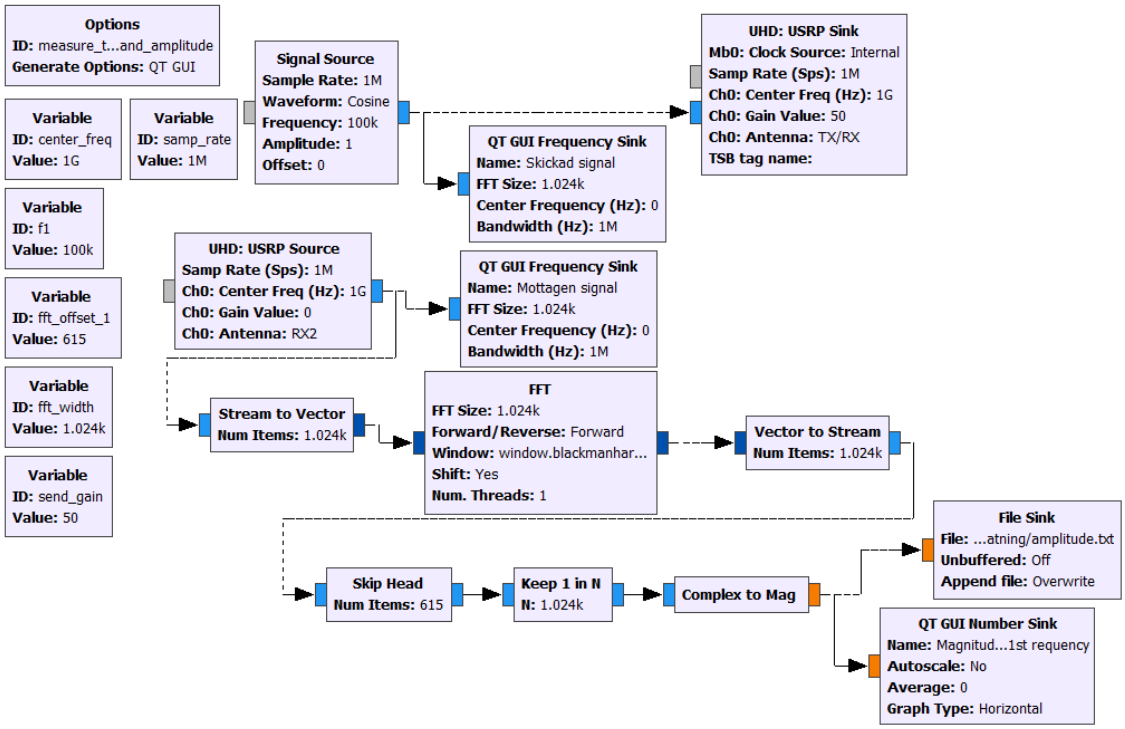
Tabell 4.3: Utsänd effekt för utvalda frekvenser.

Frekvens (MHz)	Gain (dB)	Mätning 1 Effekt (dBm)	Mätning 2 Effekt (dBm)
100	50	-13.8501	-13.4733
101	30	-30.7520	-31.7005
300	50	-16.5363	-16.3537
500	50	-20.2646	-20.3546

Värden uppmätta med oscilloskopet avviker något från presenterade värden på *Ettus Research Knowledge Base*, men generellt sett är värdena av samma storleksordning. Störst avvikelse observeras vid 101 Mhz, där oscilloskop och av *Ettus Research* presenterad data skiljer med ungefär 8 dBm. Då flertalet faktorer kan spela in till varför såpass stor skillnad inträffar vid den specifika frekvensen kommer inte vidare arbete läggas på att ta reda på bakomliggande orsaker, men resultatet är värt att notera.

4.1.4 Temperaturpåverkan på utskickad signal

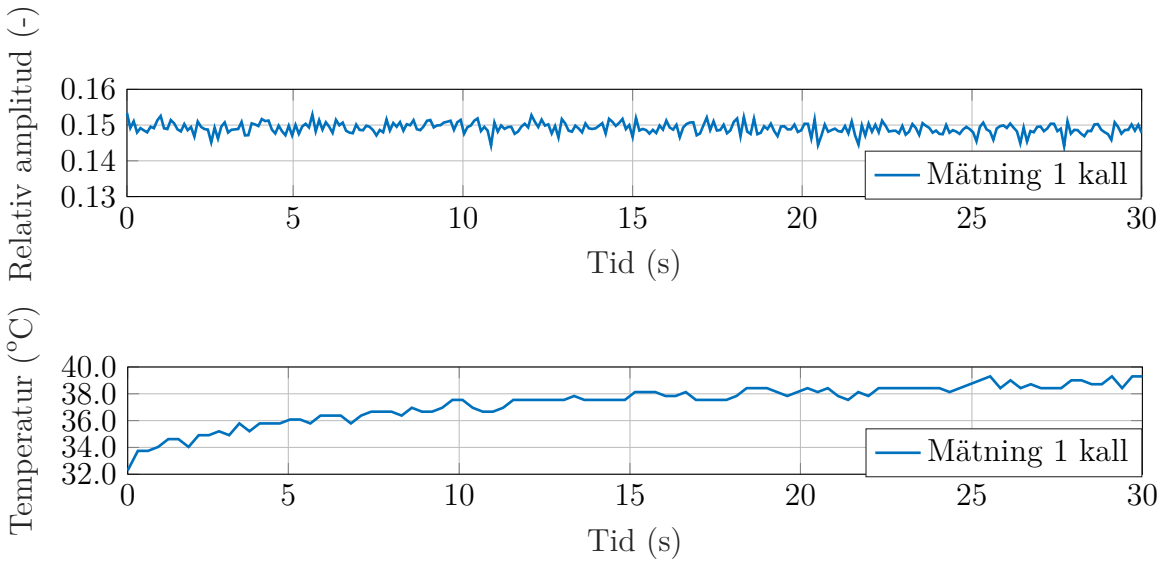
För att avgöra om den utsända amplituden hos B205mini-i är temperaturberoende, skrevs ett Pythonprogram baserat på flödeskartan i figur 4.1. I Pythonkoden lades en metod till som sparar temperaturdata hos kortet, se appendix C.1.4. En SMA-kabel med en dämpning av 30 dB kopplades mellan sändar- och mottagarporten på B205mini-i och den mottagna amplituden och temperaturen uppmättes. Amplituden och temperaturen jämfördes sedan över tid för att se om det fanns något samband.



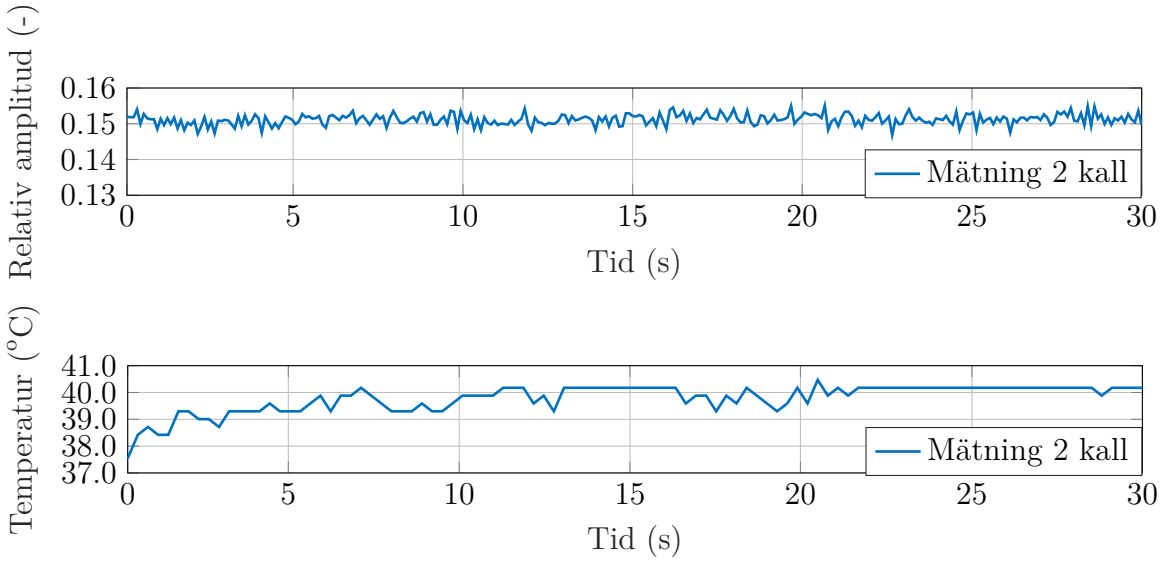
Figur 4.1: Flödesschema i GNU Radio Companion som skickar ut en enkel cosinussignal på 0.1 MHz och sparar amplituden hos den mottagna signalen för denna frekvens.

Först gjordes tre mätningar på rad då B205mini-i ej var uppvärmt och sedan ytterligare tre mätningar då kortet använts i ungefär en timme. I figurerna 4.2, 4.3 och 4.4 visas hur temperaturen och amplituden ändrades över tid för ouppvärmt kort. Mätningarna för då kortet använts i ungefär en timme visas i figurerna 4.5, 4.6 och 4.7. Det är tydligt att temperaturen ökar över tid men någon korrelation med amplitud

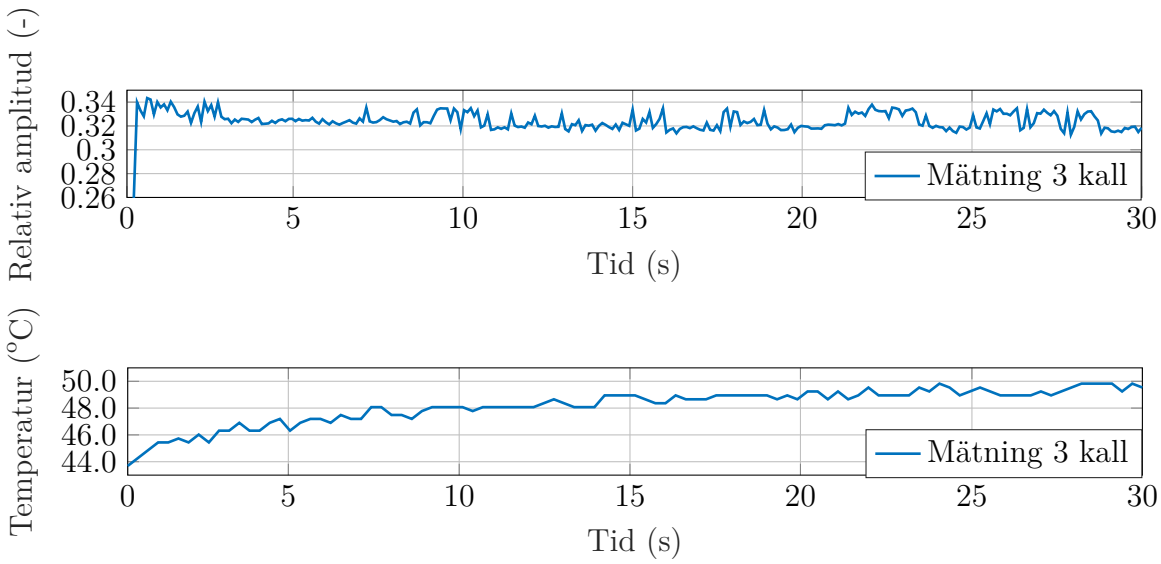
kan inte ses. Amplituden är inte konstant men det går inte att utröna potentiella trender. Mätning 3 för ouppvärmt kort sticker ut då amplituden är betydligt högre än i övriga mätningar. Amplituden går ned något från mätningens start till mätningen slut i mätning 3 för ouppvärmt kort. Inga skillnader mellan uppvärmt och ouppvärmt kort kan observeras.



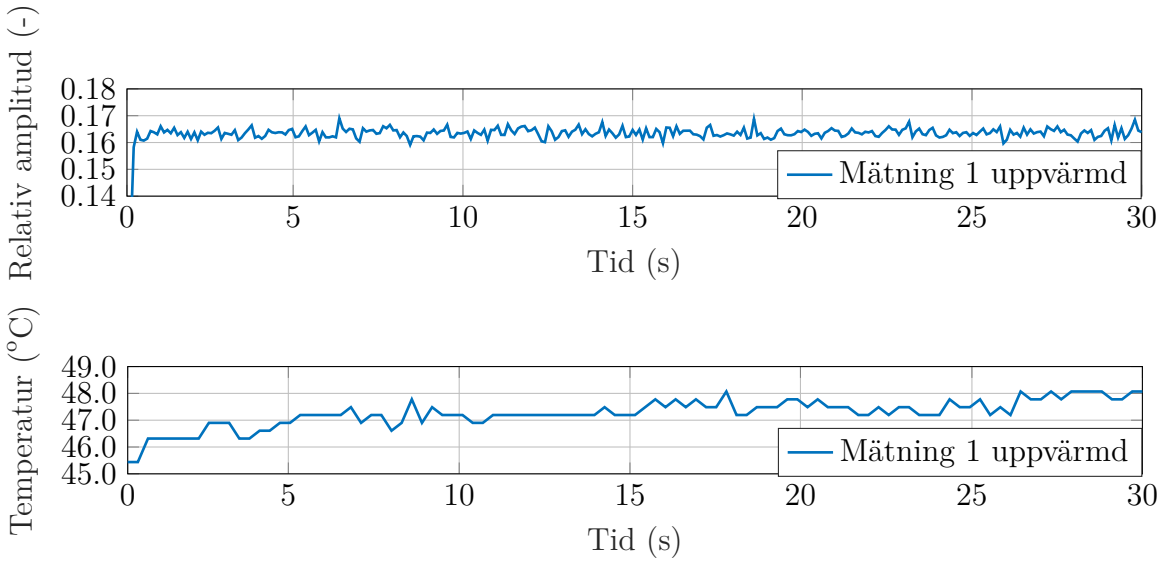
Figur 4.2: Hur temperatur och amplitud ändras under transmissionsmätning. Kortet var inte uppvärmt innan mätningen gjordes. Mätning 1.



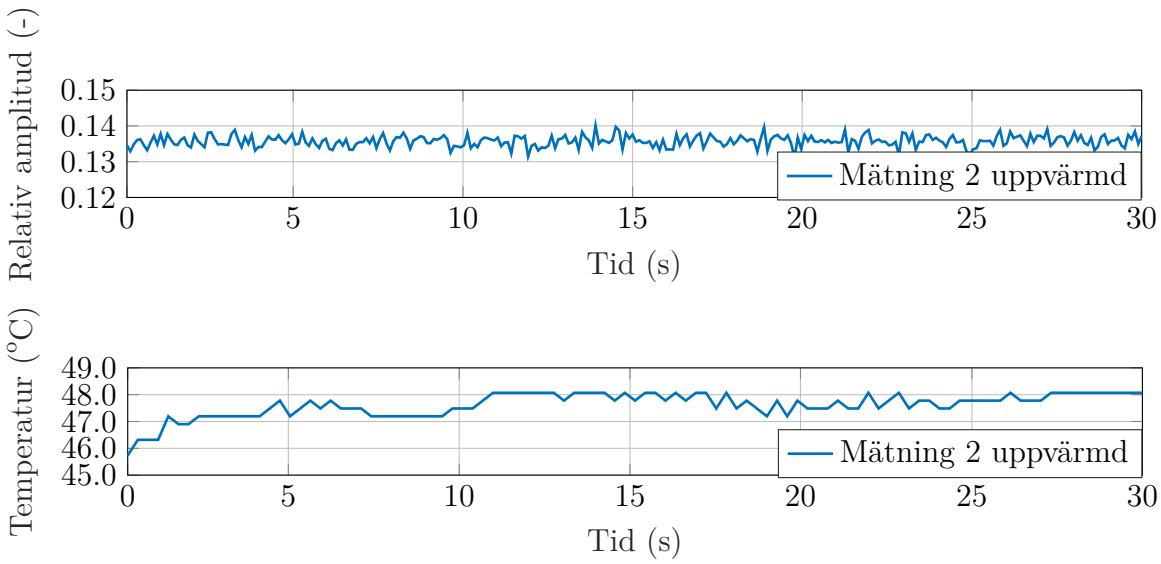
Figur 4.3: Hur temperatur och amplitud ändras under transmissionsmätning. Kortet var inte uppvärmt innan mätningen gjordes. Mätning 2.



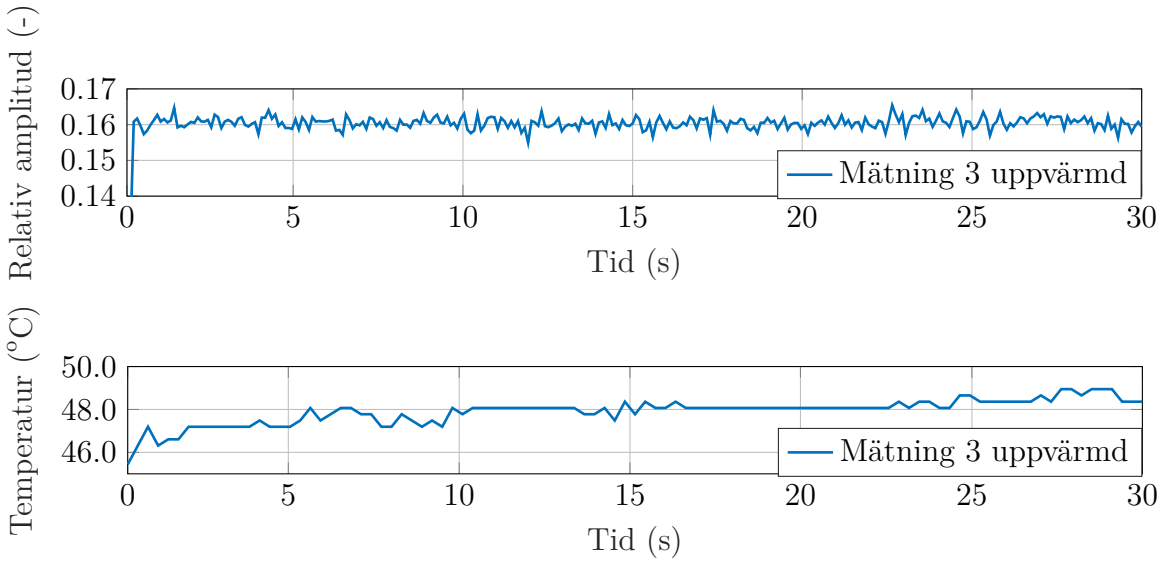
Figur 4.4: Hur temperatur och amplitud ändras under transmissionsmätning. Kortet var inte uppvärmt innan mätningen gjordes. Mätning 3.



Figur 4.5: Hur temperatur och amplitud ändras under transmissionsmätning. Kortet hade använts i ungefär en timme innan mätningen gjordes. Mätning 1.



Figur 4.6: Hur temperatur och amplitud ändras under transmissionsmätning. Kortet hade använts i ungefär en timme innan mätningen gjordes. Mätning 2.



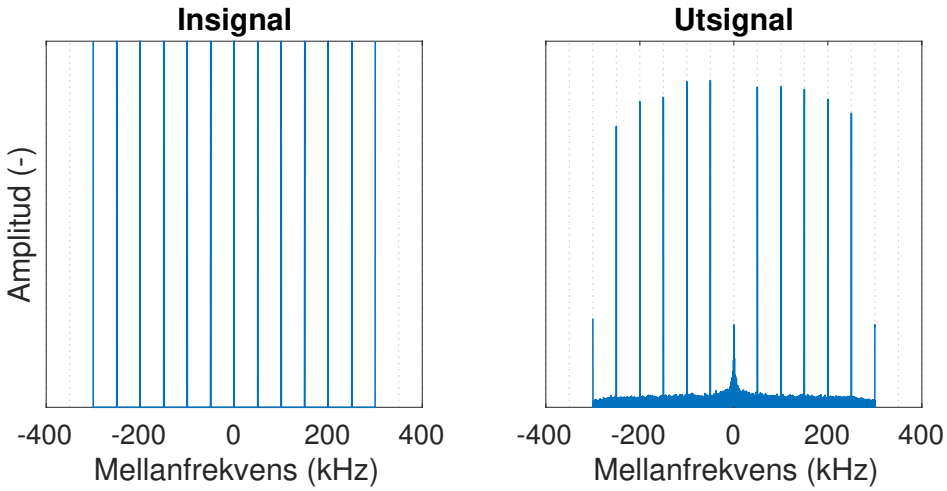
Figur 4.7: Hur temperatur och amplitud ändras under transmissionsmätning. Kortet hade använts i ungefär en timme innan mätningen gjordes. Mätning 3.

4.1.5 Amplitudvariation bland mellanfrekvenser

B205mini-i verkade inte kunna återge den genererade insignalen, bestående av en summa sinussignaler vid ekvidistanta frekvenser runt centerfrekvensen, vilken motsvarar mellanfrekvens 0 Hz. Som kan ses i högra spektrumet i figur 4.8 är centerfrekvensen (mitten av spektrumet) starkt dämpad i förhållande till övriga frekvenser.

Med undantag för centerfrekvensen ökar dämpningen med avstånd från mellanfrekvensen. Dämpningen ser inte heller ut att vara helt uniform.

Utsignalen uppmättes med loopback för att säkerställa att antennsystemet inte skulle bidra till distorsion av spektrumet.



Figur 4.8: Figuren visar hur spektrumet av signalen förändras vid transmission vid centerfrekvensen 1.5 GHz. Insignalen är signalen kortet samplar när det sänder och utsignalen är signalen uppmätt av kortet via loopback.

4.2 Fasmätning med tidskommandon

Vid konsekutiva mätningar av fas i GNU Radio Companion var den initiala fasskillnaden mellan utsänd och mottagen signal till synes slumpmässig mellan mätningar. Det skulle kunna vara ett tecken på att sändarporten (TRX-porten) och mottagarporten (RX-porten) på DSP:n inte börjar sampla samtidigt. En tänkt lösning var att använda tidskommandon som finns tillgängliga i C++-biblioteket UHD, som gör det möjligt att ange när kommandon ska utföras av B205mini-i-kortet med väldigt hög precision [17]. Ett program skrevs för att göra mätningar med tidskommandon i C++. Nedan följer ett exempel på kod med tidskommandon för när TX- och RX-strömmarna ska börja.

```
1  //Skapar USRP-objektet och kopplar det till kortet kopplat via USB
2  uhd::usrp::multi_usrp::sptr usrp_device = uhd::usrp::multi_usrp::make(
       uhd::device_addr_t());
3
4  //Använd kortets interna klockkälla
5  usrp_device->set_clock_source("internal");
6
7  //Sätter tiden på kortet till 0.0 sekunder
8  usrp_device->set_time_now(uhd::time_spec_t(0.0));
9
10 //Skapar metadata (som håller information om tiden) för TX-strömmen,
       anger sedan när strömmen ska börja
```

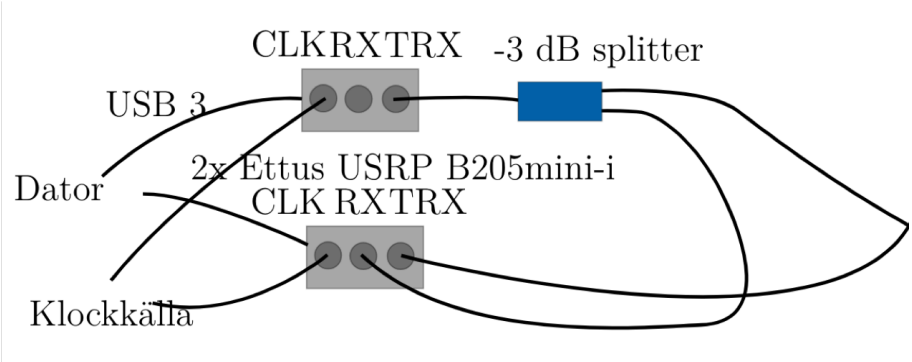
```
uhd::tx_metadata_t md;
md.has_time_spec = true;
uhd::time_spec_t tspec((double) start_time);
md.time_spec = tspec;

//Sätter upp RX-strömmen för kontinuerlig ström, anger sist när strö
    mmen ska börja
uhd::stream_cmd_t stream_cmd(uhd::stream_cmd_t::
    STREAM_MODE_START_CONTINUOUS);
stream_cmd.num_samps = 0;
stream_cmd.stream_now = false;
stream_cmd.time_spec = uhd::time_spec_t((double) start_time);
rx_stream->issue_stream_cmd(stream_cmd);
```

För att få en så tydlig signal som möjligt utfördes mätningen med loopback. Signalen som sändes var enbart en CW.

Mätningar visade samma resultat som tidigare försök till fasmätning med GNU Radio Companion. Fasskillnaden var slumpmässig mellan mätningar och inget mönster kunde observeras. Datan visade att tidskommandon inte var till hjälp för att bestämma initialfas. I efterhand hittades en förklaring till att fasskillnaden var slumpmässig: TX-mixern och RX-mixern har var sin lokaloscillator (LO) som blir tilldelade sin frekvens från en gemensam källa på DSP:n, men det finns inget som garanterar att initialfaserna för deras respektive LO är samma. Initialfaserna är därmed slumpmässiga mellan mätningar och tidskommandon förhindrar inte detta.

4.3 Fasmätning med två kort och extern klocka

Som konstaterades i avsnitt 4.2 har utsänd och mottagen signal olika och slumpmässiga initialfaser på grund av separata lokaloscillatorer för RX- respektive TX-mixer. En metod för att synkronisera RX- och TX-mixer var att använda två B205mini-i och en extern klocka enligt figur 4.9. Ett av korten användes som sändare och det andra som mottagare. På mottagarkortet utnyttjades möjligheten att byta mellan mottagning på RX- och TRX-portarna. Båda kopplades i loopback och den skickade signalen delades upp med en T-korsning. En av kablarna från T-korsningen var något längre än den andra för att en fasskillnad skulle kunna detekteras. B205mini-i har en intern switch som gör det möjligt att byta mellan RX- och TRX-portarna. Koden i avsnitt C.2.2 skrevs där det var möjligt att byta mellan de externa portarna. Signalen som skickades var en enkel sinussignal.

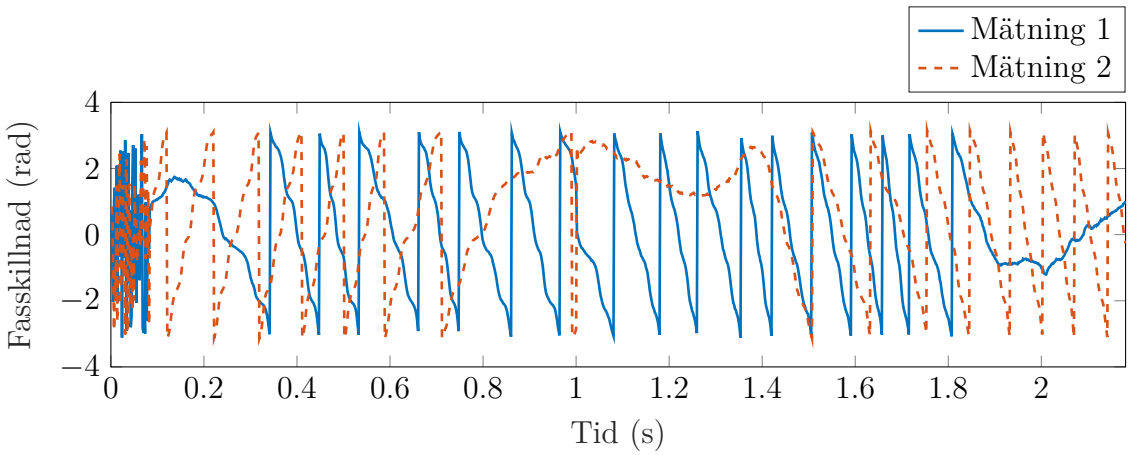


Figur 4.9: Uppställning för att mäta amplitud- och fasegenskaper med hjälp av extern klocka.

Försök att synkronisera de två korten gjordes med hjälp av en extern klocka som genererade en 10 MHz referenssignal. Den externa klockan utgjordes av en funktionsgenerator, en *Agilent 33210A* [18]. Referenssignalen delades upp till de två korten med en T-korsning. Den externa klockan kopplades till kortens referensport.

Kortet behöver en referenssignal på minst 10 dBm [6] och då referenssignalen från den externa klockan delades upp med en T-korsning gick det inte att säkerställa en mottagen effekt på 10 dBm på båda korten. Försöket upprepades med en referenssignal på över 20 dBm med förhoppningen att båda portarna skulle få åtminstone 10 dBm mottagen effekt.

Synkronisering av de två B205mini-i-korten fungerade inte då fasskillnad mellan mottagen och skickad signal förändrades över tid. Data från två mätningar kan ses i figur 4.10.



Figur 4.10: Två olika mätningar av fasskillnad mellan genererad och mottagen signal.

4.4 Amplitudmätning med införande av fantom under pågående körning

I avsnitt 3.4.2 presenterades en uppställning som mäter transmitterad signal genom en fantom placerad mellan två antenner. Som referens till mätningar genom fantom mättes också transmission i fri luft. För att undgå problemet med varierande utsänd amplitud mellan körningar genomfördes mätningar genom fantom och i fri luft under samma körning. Detta för att kunna se hur den transmitterade signalen förändrades i och med införande av fantom. Amplitud mättes först i fri luft under tio sekunder, och därefter genom fantomen i tio sekunder. Kod finns i appendix C.1.5. Mätningen upprepades för tre centerfrekvenser; 1, 1.5 och 2 GHz. På varje centerfrekvens skickades 4 mellanfrekvenser; 0.1, 0.2, 0.3 och 0.4 MHz.

I tabell 4.4 och 4.5 visas kvoten mellan uppmätt amplitud för transmission genom fantom och fri luft, då de båda amplituderna mättes i samma körning. Givet att insignalens amplitud är konstant går det att visa att denna kvot är $|S_{21}^{\text{fantom}}|/|S_{21}^{\text{fri luft}}|$.

Tabell 4.4: Amplitudkvot mellan fantom och fri luft (-) för tre olika centerfrekvenser och de fyra mellanfrekvenserna. Mätning 1.

Mellanfrekvens / Centerfrekvens	100 kHz	200 kHz	300 kHz	400 kHz
1 GHz	0.89	0.89	0.76	0.59
1.5 GHz	0.84	0.82	0.95	0.81
2 GHz	0.35	0.61	0.60	0.40

Tabell 4.5: Amplitudkvot mellan fantom och fri luft (-) för tre olika centerfrekvenser och de fyra mellanfrekvenserna. Mätning 2.

Mellanfrekvens / Centerfrekvens	100 kHz	200 kHz	300 kHz	400 kHz
1 GHz	0.74	0.71	0.75	0.96
1.5 GHz	0.93	0.90	1.05	1.34
2 GHz	0.30	0.29	0.34	0.41

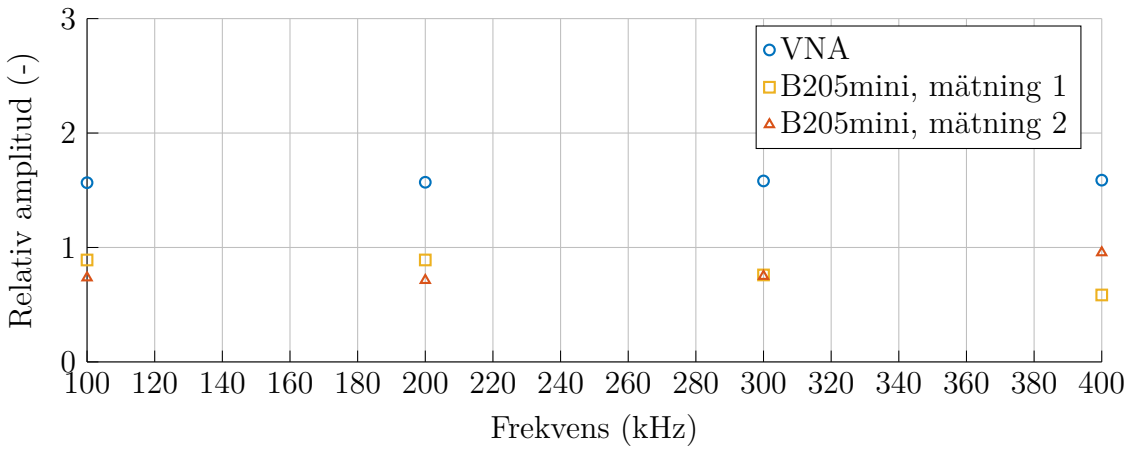
4.4.1 Jämförelse med nätverksanalysator

För prestandajämförelse genomfördes transmissionsmätningar med en *Rohde & Schwarz ZNBT8 VNA* [4], se tabell 4.6. Mätningar genomfördes på samma experimentuppställning som för resultaten presenterade i tabell 4.4 och 4.5.

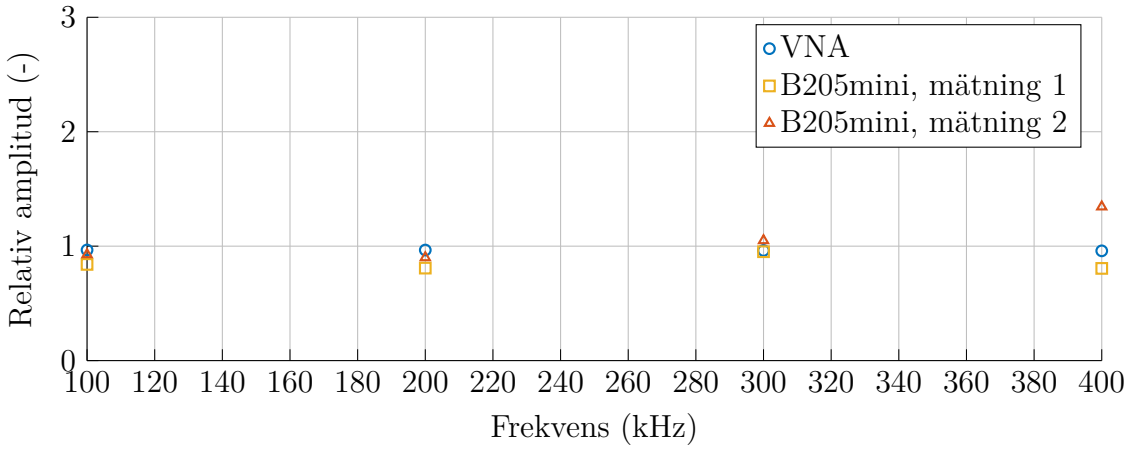
Tabell 4.6: Relativ amplitud (-) uppmätt med VNA

Mellanfrekvens / Centerfrekvens	100 kHz	200 kHz	300 kHz	400 kHz
1 GHz	1.5661	1.5699	1.5813	1.5883
1.5 GHz	0.9666	0.9660	0.9625	0.9589
2 GHz	0.4060	0.4075	0.4085	0.4099

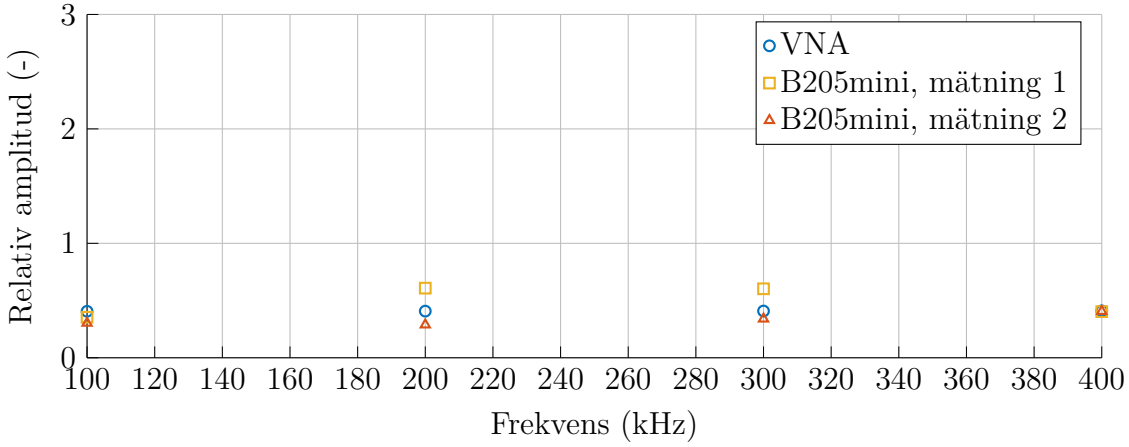
Uppmätta kvotvärden med B205mini-i varierar mellan mätningar och avviker också från resultat givna av VNA, se figurer 4.11, 4.12 och 4.13. Vid centerfrekvens 1 GHz uppvisar VNA:n en positiv förstärkning av amplituden för propagation genom fantom jämfört med fri luft, vilket inte är fallet för mätningarna gjorda med B205mini-i. Omvänt gäller för mätningar vid en centerfrekvens på 1.5 GHz, då istället en av mätningarna gjorda med B205mini-i uppvisar en förstärkning genom fantom för 300 kHz och 400 kHz till skillnad från VNA:n. Dock uppvisar den andra mätningen med B205mini-i, i likhet med VNA:n, dämpning av amplitud genom fantom jämfört med fri luft.



Figur 4.11: Relativa amplituder som kvoter av propagation genom fantom och luft vid centerfrekvens 1 GHz.



Figur 4.12: Relativa amplituder som kvoter av propagation genom fantom och luft vid centerfrekvens 1.5 GHz.



Figur 4.13: Relativa amplituder som kvoter av propagation genom fantom och luft vid centerfrekvens 2 GHz.

4.5 Mätningar relativt känd referens med hjälp av switch

För att få konsekventa resultat mellan körningar och vid byte av centerfrekvens utfördes mätningar relativt känd referens beskrivna i avsnitten nedan. Switchen som användes var *Mini-Circuit ZTVX-16-18-S* [19] som dels kunde styras manuellt genom en separat dator men även genom kommandon i ett program skrivet i C++.

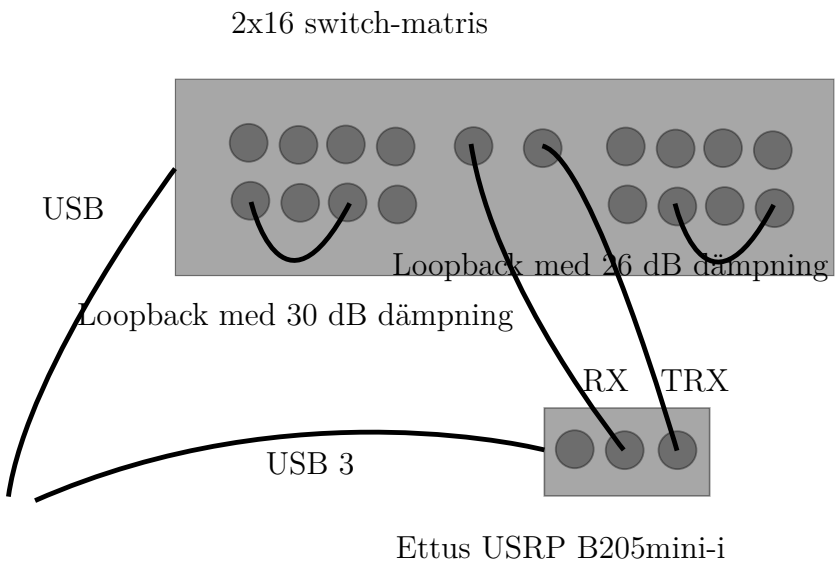
4.5.1 Mätning av två loopback med olika dämpning

Relativa mätningar utfördes med två stycken loopback-kopplingar som växlades mellan med hjälp av switchen, se figur 4.14.

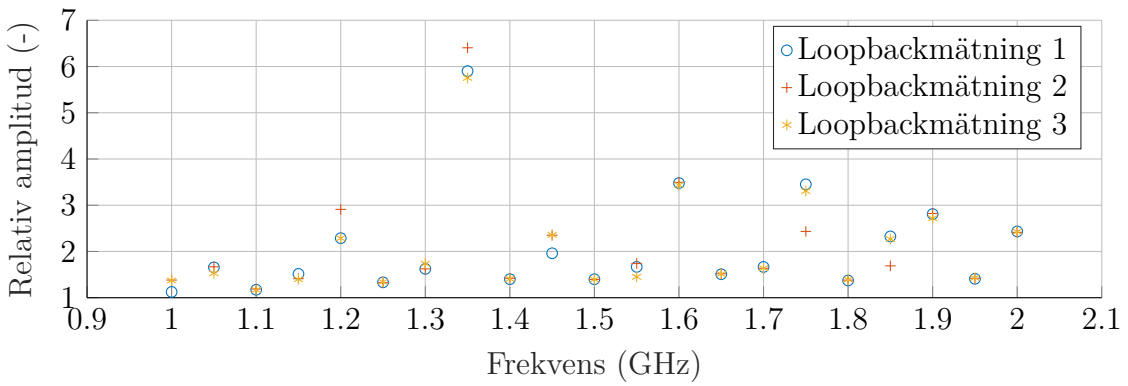
I figur 4.15 visas kvoten mellan mottagen amplitud för loopback med 26 dB dämpning och mottagen amplitud för loopback med 30 dB dämpning,

$$\frac{|S_{21}^{\text{26 dB loopback}}|}{|S_{21}^{\text{30 dB loopback}}|}.$$

Teoretiskt bör denna kvot vara 1.58 då det är 4 dB skillnad i dämpning, se avsnitt 2.5. Mätdatan i figur 4.15 avviker dock från 1.58 och varierar mellan frekvenser. Speciellt vid 1.35 GHz är mottagen amplitud för loopback med 26 dB betydligt högre än väntat jämfört med mätningar gjorda med 30 dB dämpning. Resultaten är förhållandevis konsekventa mellan mätningar, förutom vid några frekvenser, speciellt vid 1.35 GHz.



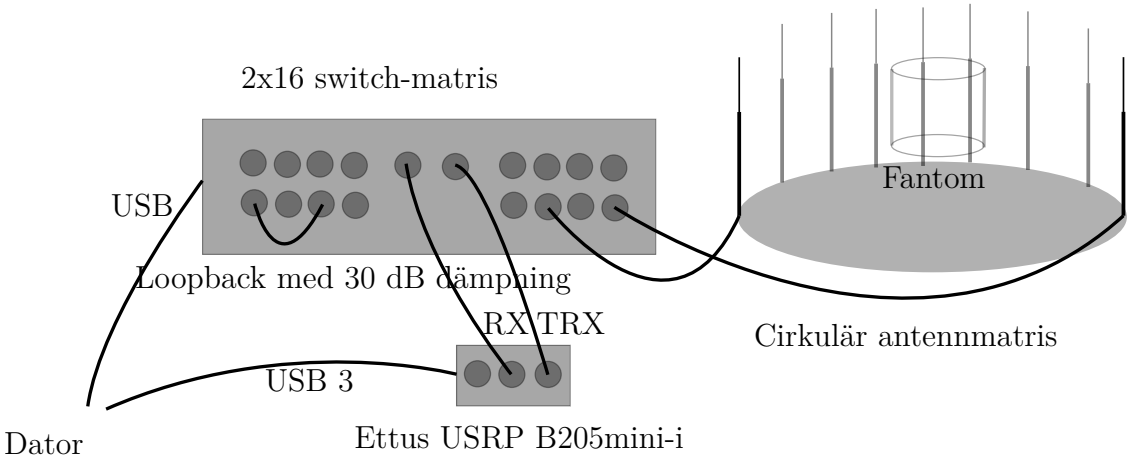
Figur 4.14: Uppställning för att mäta amplitud- och fasegenskaper mellan två loopbackkopplingar med hjälp av en switch. B205mini-i-kortet kopplades till datorn via USB. Kortets utkanal (TX) och dess inkanal (RX) var kopplade till switchen som programmerades till att byta mellan loopback-portarna.



Figur 4.15: Kvoten mellan uppmätt amplitud med 26 dB dämpning och 30 dB dämpning.

4.5.2 Mätning med fantom

Experimentuppställningen i figur 4.14 utökades för att genomföra mätning med fantom, se figur 4.16. Switchen anslöts mellan B205mini-i-kortet och antennerna. Med hjälp av switchen kunde signalen skickas och tas emot av antennerna eller skickas tillbaka till kortet genom en SMA-kabel med 30 dB dämpning. Sådan användning av switchen gjordes för att kunna få en referensmätning att jämföra resultat mellan körningar mot. Den mottagna amplituden från antennerna dividerades med den mottagna amplituden från loopback för att ta hänsyn till att amplituden varierade mellan körningar. På samma sätt och av samma anledning mättes fasskillnad mellan den skickade och mottagna signalen för både loopback och antenner. Switchen styrdes med ett C++-program, se appendix C.2.4, och ett flertal frekvenssvep mellan 1 och 2 GHz genomfördes med 50 MHz mellanrum. Testsignalen som användes var en summa av CW vid mellanfrekvenserna 0 Hz, ±50 kHz, ±100 kHz, ±150 kHz, ±200 kHz och ±250 kHz.



Figur 4.16: Uppställning för att mäta amplitud- och fasegenskaper för fantom med hjälp av en switch. B205mini-i-kortet kopplades till datorn via USB. Kortets utkanal (TX) och dess inkanal (RX) var kopplade till switchen som programmerades till att byta mellan loopback och antenn.

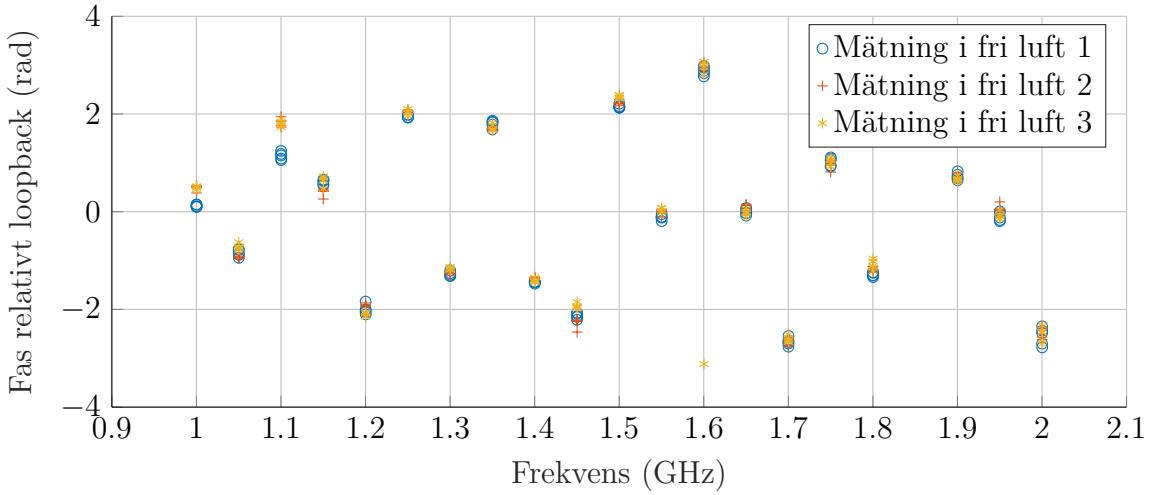
4.5.2.1 Fasskillnad mellan mottagen och skickad signal i fri luft och med fantom

För att kunna göra fasmätningar som är konsekventa mellan körningar beräknades fasskillnad mellan skickad och mottagen signal, Δfas, för mätning i fri luft relativt loopback, $(\Delta\text{fas}_{\text{fri luft}} + \text{fas}_{\text{initial}}) - (\Delta\text{fas}_{\text{loopback}} + \text{fas}_{\text{initial}})$. Slumpmässigheten i initialfas hos RX- och TX-mixern, se avsnitt 4.2, är gemensam för båda termerna och kommer subtraheras bort. Mätningar gjorda i fri luft relativt loopback visas i figur 4.17. Endast mätdata för de positiva mellanfrekvenserna visas. Inget samband kan observeras mellan fasskillnad och frekvens ur figuren.

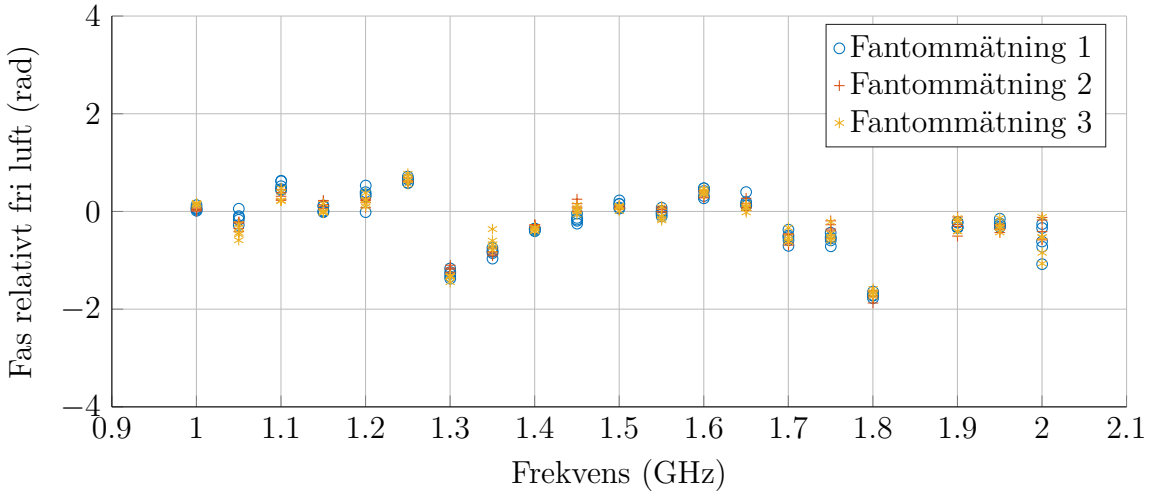
Mätningen upprepades med fantom med avsikt att jämföra skillnad i fas med mätning 1 från figur 4.17, enligt

$$\left[(\Delta\text{fas}_{\text{fri luft}} + \text{fas}^{(1)}_{\text{initial}}) - (\Delta\text{fas}^{(1)}_{\text{loopback}} + \text{fas}^{(1)}_{\text{initial}})\right] - \left[(\Delta\text{fas}_{\text{fantom}} + \text{fas}^{(2)}_{\text{initial}}) - (\Delta\text{fas}^{(2)}_{\text{loopback}} + \text{fas}^{(2)}_{\text{initial}})\right]. \quad (4.1)$$

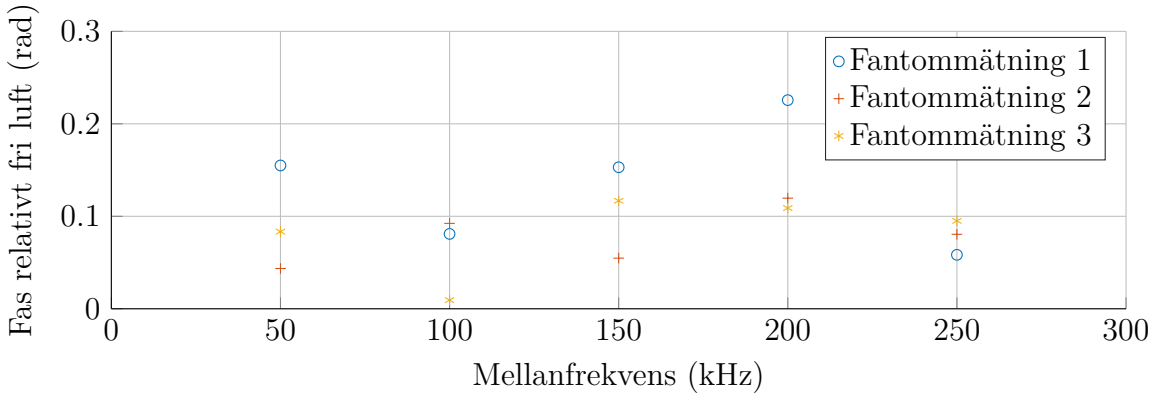
I figur 4.18 visas resultatet från ekvation (4.1). Båda mätningarna hade loopback med 30 dB dämpning som referens. Värdena ser ut att kunna utgöra en kontinuerlig mätserie bortsett från hoppet vid 1.3 GHz. Resultaten mellan mätningarna ligger relativt nära varandra. Förstoring av figur 4.18 vid centerfrekvens 1.5 GHz kan ses i figur 4.19.



Figur 4.17: Fas relativt loopback vid mätning i fri luft.



Figur 4.18: Fasskillnad mellan fri luft och fantom med loopback som referens.



Figur 4.19: Fasskillnad mellan fri luft och fantom med loopback som referens vid centerfrekvensen 1.5 GHz.

4.5.2.2 Kvot mellan amplitud för fantom och fri luft

För att mäta relativa amplituder användes samma uppställning som i avsnitt 4.5.2.1. Kvoten av amplitud i fri luft och loopback definieras då

$$\frac{\text{Amplitud fri luft}}{\text{Amplitud loopback}}.$$

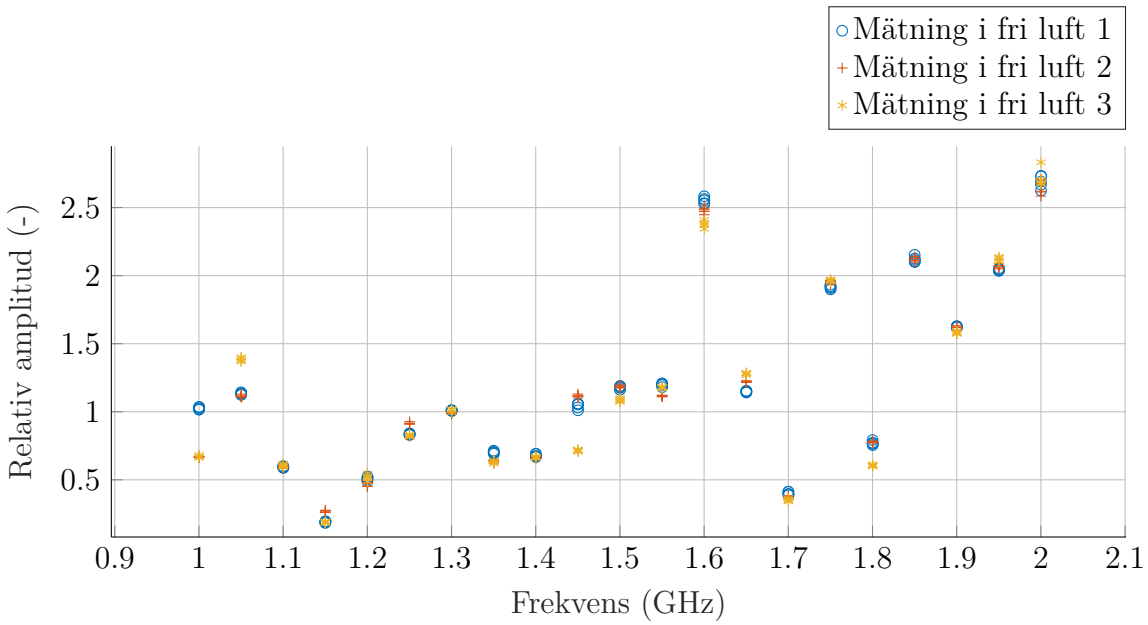
Resultaten som visas i figur 4.20 tycks tyda på att amplituden ökar för högre frekvenser.

För jämförelse av mätning med fantom och fri luft beräknades den relativa amplituden enligt

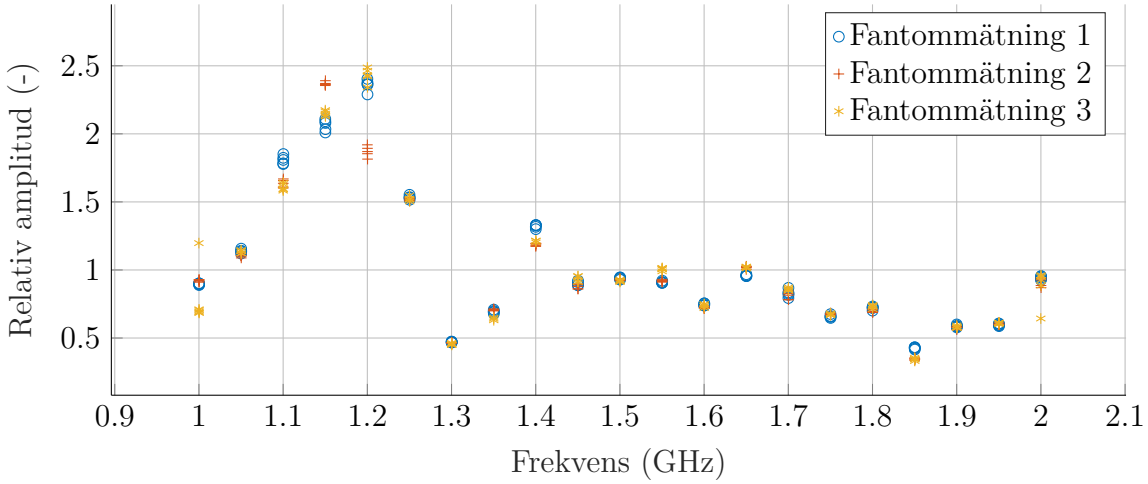
$$\frac{\text{Amplitud fantom/Amplitud loopback}_{\text{fantom}}}{\text{Amplitud fri luft/Amplitud loopback}_{\text{fri luft}}}.$$

Figur 4.21 visar hur den relativa amplituden varierar med olika frekvenser för både mätningar i fri luft och fantom. Tre olika amplitudmätningar med fantom genomfördes. Alla tre mätningar jämfördes med mätning 1 i fri luft.

Bortsett från vissa undantag uppvisar resultaten i 4.21 generella trender. För de lägre uppmätta frekvenserna transmitteras mer signal genom fantom än i fri luft, då kvoten är större än 1. Maximal förstärkning genom fantomen för de uppmätta frekvenserna infaller runt 1.2 GHz. Dämpning av signalen genom fantom jämfört med fri luft sker från frekvenser runt 1.45 GHz. Mätningarna är också tillsynes någorlunda konsekventa mellan körningar.



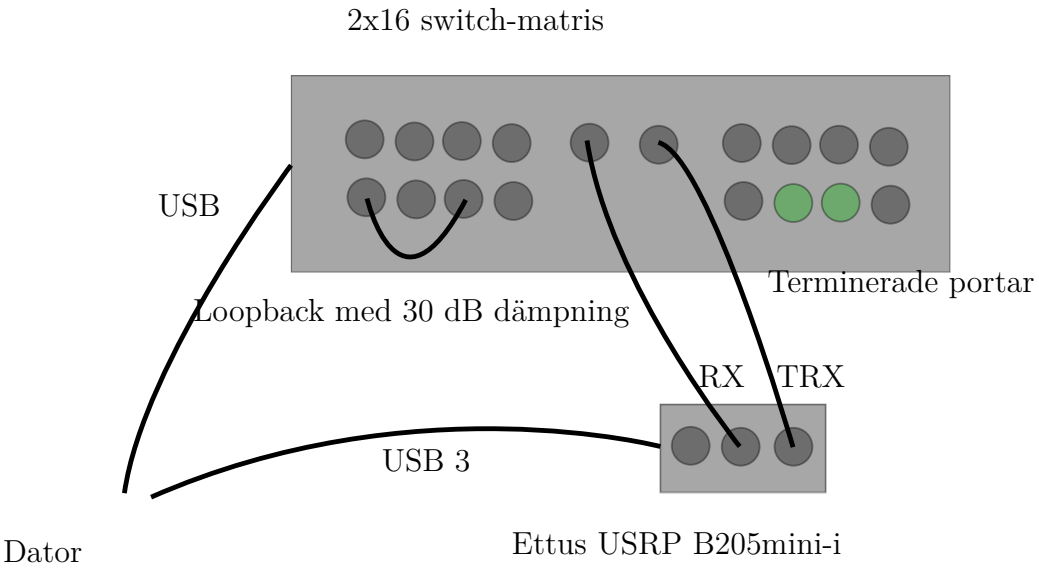
Figur 4.20: Amplitud vid mätning i fri luft dividerad med amplitud vid mätning med loopback.



Figur 4.21: Amplitud för olika mätningar med fantom i förhållande till en kalibreringsmätning i fri luft.

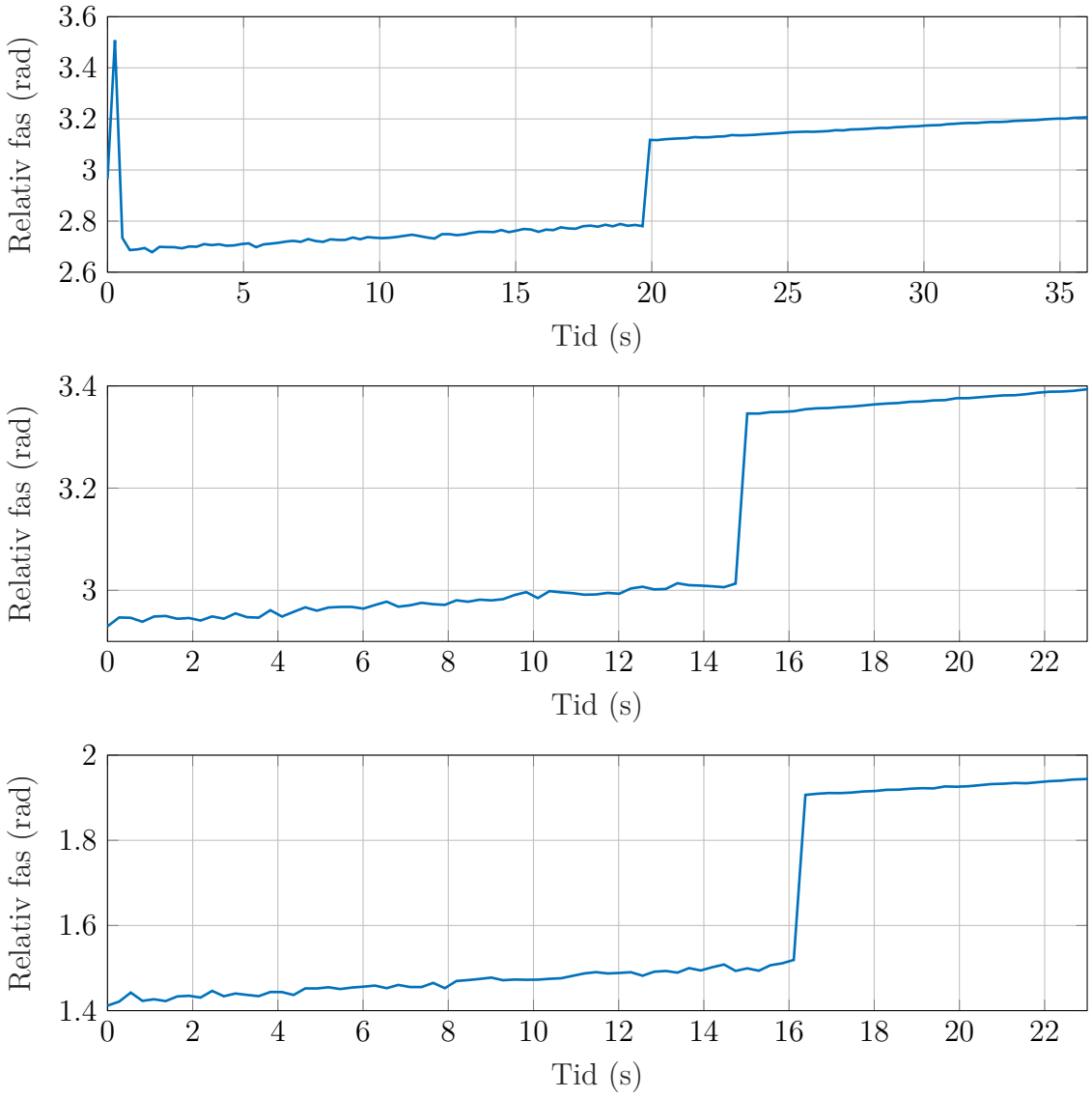
4.5.3 Mätning med crosstalk

Ytterligare en möjlig kalibrering vid fasmätningar är att utnyttja intern crosstalk som sker mellan mottagar- och sändarledning på samma kort. Med crosstalk menas den kapacitiva eller induktiva energiöverföring som sker mellan två separerade ledare internt i kortet. Den interna switchen för TRX-porten i B205mini-i kan inte sköta bytet med fabriksinställningar, istället utnyttjades den externa switchen. För att undersöka möjligheten att använda crosstalk för att bestämma fas användes en uppställning enligt figur 4.22. En testsignal bestående av en summa av CW med en centerfrekvens på 1.00 GHz skickades till switchen. Switchen var inledningsvis kopplad för att terminera signalen och låta B205mini-i-kortet mäta intern crosstalk. Uppställningen kopplades sedan över till en loopback-konfiguration för att göra det möjligt att mäta fasskillnad mellan loopback och crosstalk. Skillnaden bör vara samma oberoende initialfasen på kortet.



Figur 4.22: Uppställning för att mäta fasegenskaper för loopback med hjälp av crosstalk. B205mini-i-kortet kopplades till datorn via USB. Kortets utkanal (TX) och dess inkanal (RX) var kopplade till switchen som programmerades till att byta mellan loopback och terminering.

I figur 4.23 ses hur fasen ändrar sig när switchen konfigurerades om från resistiv terminering till loopback. Före konfigurationen borde inte mottagarporten registrera någon signal, varför registrerad signal således bör vara crosstalk från sändarsidan på samma kort. Uppmätt fasskillnad med crosstalk fluktuerar mer än fasskillnaden med loopback. Differensen i fasskillnad mellan crosstalk och loopback var vid övergången vid första mätningen 0.338 rad samt 0.333 rad och 0.388 rad i mätning 2 och 3.



Figur 4.23: Mätningar av fas med crosstalk för att sedan runt 15 s kopplas om till -30 dB loopback med hjälp av en switchmatris. En testsignal bestående av en summa CW med en centerfrekvens på 1.00 GHz användes.

5

Slutsatser och Diskussion

Separata kalibreringsmätningar visade sig inte fungera då B205mini-i-kortet inte är fabrikskalibrerat, vilket gör att den uppmätta amplituden är i godtyckliga enheter och kan variera på grund av olika anledningar så som temperatur och byte av centerfrekvens. Temperaturpåverkan undersöktes och resultat påvisar inte något direkt samband mellan amplitud och temperatur.

Initialfasen av signalen varierade mellan körningar då RX-mixern och TX-mixern har separata lokaloscillatorer (LO). Variationen gjorde att separat kalibreringsmätning för fas inte var möjligt. Resultat som styrker denna slutsats kan ses i avsnitt 4.1, vilket visar att amplitud och fas varierar kraftigt mellan mätningar.

För att försöka kompensera för amplitud och fasskillnad mellan mätningar gjordes mätningar i fri luft och med fantom under samma körning (utan loopback). Det gjordes också mätningar där amplitud och fasskillnad uppmättes relativt loopback under samma körning. Resultaten utan loopback varierar väldigt mycket mellan mätningar. En förklaring kan vara att positionen på fantomen var olika vid olika körningar. Då insättning av fantom tog ungefär tio sekunder är det också möjligt att amplitud på utsänd signal varierade under själva körningarna.

Resultat från mätningar med en extern klocksignal visade inte att korten blev synkroniserade. Istället driftade de i fas relativt varandra. Varför synkroniseringen inte lyckades fann vi inget svar på.

För att undersöka möjligheten med relativa loopback-mätningar utfördes en mätning med två olika loopback-kopplingar med hjälp av en switch. Den relativa amplituden mellan 26 dB- och 30 dB-dämpade loopback-konfigurationer var betydligt större för vissa frekvenser jämfört med andra. I teorin ska de enbart variera med 4 dB för samtliga frekvenser, vilket inte var fallet. Att tre upprepade mätningar på samma uppställning gav snarlika resultat tyder på någon störkälla. En möjlig förklaring skulle kunna vara crosstalk i B205mini-i eller i switchen. En annan möjlig förklaring är reflektioner som uppstår i utrustningen, till exempel i switchen vid kanalbyte.

Testsignalen som används vid mätningar kan också vara upphovet till fel i mätningarna. För samtliga mätningar användes en summa av sinussignaler som genererades med hjälp av inversa fouriertransformen. Frekvenserna fördelades symmetriskt-ekvidistant runt 0 Hz där varje positiv frekvens har sin spegling i det negativa frekvensspektrumet. De negativa frekvenserna kan ha placerats i en frekvensbin ± 1 fel. Vid mätningar framgick det att fasen driftade i tiden, exempelvis i figur 4.23,

vilket skulle kunna vara på grund av felplacering av frekvenser. Dessutom visade mätningar oväntade övertoner, bestående av summan av utskickade frekvenser. Övertonerna kan även de bidra till en driftande fas samt en felaktig amplitudangivelse. Det kan vara värt att generera testsignaler där de övre frekvenserna inte ligger nära en summa av de lägre.

Undersökning av crosstalk som möjlig referens vid fasmätningar genomfördes också. Då crosstalk innebär att den utskickade signalen kan plockas upp av mottagaren internt i kortet försvinner behovet av en extern loopback för att bestämma fas. Vid upprepade körningar visade det sig att den utskickade signalen går att mäta upp, även fast mottagar- och sändarporten var terminerade genom switchen. Fasskillnaden mellan crosstalk-signalen och loopback var ungefär densamma mellan körningarna, vilket tyder på att crosstalk troligtvis kan användas som faskaliberingskälla.

För att få någon form av indikation på hur bra resultat genererade med B205mini-i-kortet var gjordes mätningar med en nätverksanalysator (VNA) att jämföra mot. VNA:n kan mäta S-parametrarna, som kan användas för att få information om transmissionseffekter, med extremt hög precision. Vid vissa frekvenser gav mätningar med B205mini-i-kortet resultat nära mätningar med VNA:n, men vid andra frekvenser var resultaten mer olika. Om skillnaden i resultat beror på kortet eller andra omständigheter är okänt. Fantomens position var inte fixerad mellan mätningar, vilket är en möjlig förklaring till skillnaderna.

Baserat på framtagna resultat kan inga definitiva slutsatser dras om B205mini-i kan användas för mikrovågsbaserad medicinsk diagnostik. Fler mätningar krävs och felkällor behöver undersökas ytterligare.

5.1 Fortsatt studie

Relativt få mätningar har genomförts med respektive uppställning och program, varför fler mätningar skulle behövas för att säkerställa tillförlitligheten i resultaten. Påverkan av felkällor så som varierande temperatur och placering av fantom har inte heller utretts på ett tillfredsställande sätt, vilket försvårar analys av insamlade mätdata. Det faktum att fas och amplitud varierar mellan, och under körningar, gör det svårt att styrka riktigheten i presenterade mätningar, och mer data behövs för att styrka resultaten. Mer referensdata med tillförlitlig utrustning, exempelvis VNA, hade också behövts för att jämföra med resultat från mätningar med B205mini-i.

Tillgång till switch kom relativt sent i projektet, varför förhållandevis få mätningar har utförts. Mätningar med switch var lovande, men insamling av mer mätdata hade varit bra för att styrka resultaten. Med switch kunde även mätningar automatiseras, och större frekvenssvep genomföras utan avbrott för manuell konfiguration av experimentuppställning. Det vore därför en bra utgångspunkt för vidare studier att redan från början använda en switch vid experiment.

I teorin verkar extern klocka användbar för att synkronisera två B205mini-i, dock lyckades inte kommunikation mellan korten och klockan. Enligt dokumentation för

kortet ska det fungera, varför vidare undersökning är att rekommendera.

Ett alternativ till att använda en extern switch vore att ändra antennkonfigurationen på kortet. Detta kan göras genom att ändra GPIO-utgångarna i C-koden. Mottagarkretsen skulle på så vis kunna kopplas bort från RX-porten. Detta skulle resultera i att enbart intern crosstalk skulle plockas upp av kortet, vilket potentiellt möjliggör absoluta amplitud- och fasmätningar. *Ettus Research* avrådde oss i mailkorrespondens från omprogrammering av kortet på grund av potentiella risker med att förstöra mottagarkretsen, samt uppmärksammade oss på att garantin vid en sådan eventualitet inte gäller. Se appendix A för mailkonversation. Ett tillämpat flerantennsystem skulle kräva någon form av switch i vilket fall och en switchkonfiguration med dämpad loopback skulle kunna användas i syfte att få konsekventa mätningar.

Alla mätningar har genomförts med både sändar- och mottagardel samtidigt, varför utvärdering av komponenterna var för sig inte har varit möjlig. Det hade varit av intresse att testa dem separat för att se om kortet är bättre lämpat att användas enbart som sändare eller enbart som mottagare. Då datahastigheterna för datorn och USB-länken är begränsade sätter detta en gräns på hur bredbandiga mätningar som kan utföras. En möjlighet skulle vara att istället låta FPGA:n konstruera de testsignaler som skickas ut. Detta hade effektivt dubblat bandbredden (upp till begränsningen av kortet) då datorn inte behöver generera en signal som ska skickas ut.

Samtliga mätningar har gjorts med en eller flera adderade sinussignaler. En uppdelning i frekvensdomänen bidrar till att varje frekvens har en lägre amplitud, något som ger problem i brusiga förhållanden. Ett alternativ är att använda en frekvenssvepande sinussignal som enbart består av en CW som stegvis går igenom den tillgängliga bandbredden.

Med ett B210-kort från *Ettus Research* [20] hade kalibreringsmätning kunnat underlättas då det kortet har två sändar- och två mottagarportar. Kalibrering och faktisk mätning hade då kunnat genomföras parallellt istället för sekventiellt, som varit fallet för mätningar gjorda med B205mini-i. Mottagarna på B210 delar lokaloscillator (LO) och har därmed samma initialfas vilket gör att fasskillnaden mellan kalibrering och faktisk mätning bör vara överensstämmande vid upprepade mätningar. Undersökningar inom samma område har gjorts för B210 tidigare, se [21].

I genomförda mätningar samlades data in under fem sekunder per centerfrekvens. Dock undersöktes inte hur lång tid som behövdes för att samla in pålitlig data. Tid är en viktig faktor vid mikrovågsbaserad medicinsk diagnostik, varför vidare undersökning av tidsåtgång för mätningar hade varit intressant. För att optimera tidsåtgång är tidskommandon i UHD användbara.

6

Referenser

[1] Andreas Fhager, Stefan Candefjord och Mikael Elam. ”Microwave Diagnostics Ahead”. I: April (2018).

[2] Tomas Rydholm, Andreas Fhager och Mikael Persson. ”A First Evaluation of the Realistic Supelec-Breast Phantom”. I: 1 (2017).

[3] X. Zeng m. fl. ”Development of a Time Domain Microwave System for Medical Diagnostics”. I: *IEEE Transactions on Instrumentation and Measurement* 63.12 (2014), s. 2931–2939.

[4] Rohde & Schwarz. *R & S ® ZVL Vector Network Analyzer Specifications*. Tekn. rapport. 2017. URL: https://cdn.rohde-schwarz.com/pws/dl_downloads/dl_common_library/dl_brochures_and_datasheets/pdf_1/ZNBT_bro_en_3606-9727-12_v0500.pdf.

[5] Fredrik Hedlund. ”Medicinsk Vetenskap”. I: *1* (2013), s. 32–37.

[6] Ettus Research. *USRP B205mini-i*. 2018. URL: https://www.ettus.com/product/details/USRP-B205Mini-I.

[7] Ettus Research. *USRP B200mini Series*. 2018. URL: https://www.ettus.com/content/files/USRP_B200mini_Data_Sheet.pdf.

[8] José Raúl Machado-Fernández. ”Software Defined Radio: Basic Principles and Applications Software Defined Radio: Principios y aplicaciones básicas Software Defined Radio: Princípios e Aplicações básicas”. I: *Revista Facultad de Ingeniería (Fac. Ing.), Enero-Abril Revista Facultad de Ingeniería (Fac. Ing.)* 24.38 (2015), s. 79–96. ISSN: 2357-5328. URL: http://www.scielo.org.co/pdf/rfing/v24n38/v24n38a07.pdf.

[9] Analog Devices. *AD9364 Data Sheet*. 2014. URL: http://www.analog.com/media/en/technical-documentation/data-sheets/AD9364.pdf.

[10] National Instruments. *What is I/Q Data? - National Instruments*. 2016. URL: http://www.ni.com/tutorial/4805/en/ (hämtad 2018-05-04).

[11] David K. Cheng. *Field and Wave Electromagnetics*. Second. Pearson Education Limited, 2014.

[12] Av Jan Stake och Mattias Ingvarson. ”Högfrekvensteknik för E3/F3”. 2017.

[13] Gerald B. Folland. *Fourier Analysis and its Applications*. 1st. American Mathematical Society, 1992, s. 433.

[14] Wikimedia Commons. *Sampling one point five times per cycle leads to a skewed representation! [Image]*. 2012. URL: https://commons.wikimedia.org/wiki/File:CPT-sound-nyquist-thereom-1.5percycle.svg.

[15] Ettus Research. *Om uteffekt - Ettus Mailing list*. 2015. URL: http://lists.ettus.com/pipermail/usrp-users_lists.ettus.com/2015-March/013188.html.

[16] Ettus Research. *RF Performance Data*. 2016. URL: https://kb.ettus.com/images/a/ae/B200mini_B205_RF_Performance_Data_20160119.pdf.

[17] Ettus Research. *USRP Hardware Driver and USRP Manual: uhd::time_spec_t Class Reference*. URL: https://files.ettus.com/manual/classuhd_1_1time__spec__t.html (hämtad 2018-05-01).

[18] Agilent Technologies. *Agilent 33210A 10 MHz Function/Arbitrary Waveform Generator*. 2008. URL: http://sdpha2.ucsd.edu/Lab_Equip_Manuals/hp_33210a_Data_Sheet.pdf (hämtad 2018-05-11).

[19] Mini-Circuits. *Mini-Circuits*. URL: https://ww2.minicircuits.com/.

[20] Ettus Research. *USRP B210*. 2018. URL: https://www.ettus.com/product/details/UB210-KIT.

[21] Florian Schönberger. "Multichannel, LabView-controlled, Software Defined Radio Measurement System for Microwave Breast Tomography". Diss. 2017.

A

Mailkonversation med Ettus Research angående omprogrammering av switchar

From: ... on behalf of Ettus Research Support <support@ettus.com>
Sent: Saturday, April 14, 2018 12:28 AM CET
To: ...
Subject: Re: Reconfiguring internal switches

The risk is that you accidentally connect the RX2 port to the RX/TX port while it's transmitting, which will more-than-likely damage the RX chain. If you have high confidence that this won't happen, then you can do as you suggest, but if something is damaged as a result, then the warranty would not be honored.

-...
On Fri, Apr 13, 2018 at 8:42 AM CET, ... wrote:
Hello again,

Thank you for your reply! The problem for us is to synchronize/calibrate the TX-/RX phase offset in order to measure the propagation-induced phase offset of the signal transmitted from the same board. Since we are sweeping many different center frequencies, using a terminator would require us to manually switch the antennas after each center frequency re-tune. We are using more than 20 different center frequencies so we pretty much need to perform the switching programmatically. How risky would it be to do this using the internal switches?

Thanks

From: ... on behalf of Ettus Research Support <support@ettus.com>
Sent: Friday, April 13, 2018 3:43:38 AM CET
To: ...
Subject: Re: Reconfiguring internal switches

Just put a 50Ohm terminator on the RX2 antenna input–there'll be enough cross-talk from TX that way–no need to mess about with changing the switching logic and potentially damaging your unit.

-...
On Thu, Apr 12, 2018 at 8:23 PM CET, ... wrote:
Hello,

We are a group of students trying to make relative phase measurements using a USRP B205mini for transmitting and receiving a reference signal. We would like to programmatically disable the receiver antenna initially in order to only pick up the internal crosstalk from the transmitter. We hope to be able to establish the TX-RX LO phase offset, which differs each clock re-tune, using the crosstalk. Looking at the schematic [1], the RX switch can be disconnected from the RX2 port, however, only using the UHD set_antenna commands will not allow for this configuration. Looking at the RTL code, the switch configuration could be overridden by changing the GPIO pins,

6: cFE_SEL_RX_RX2 -> 0,
5: cFE_SEL_TRX_TX -> 1,
4: cFE_SEL_RX_TRX -> 1,
3: cFE_SEL_TRX_RX -> 0,

yielding a mask and value of 0b1111000 and 0b0110000 respectively for the crosstalk configuration. Could this potentially damage the board due to reflections? According to the datasheet of the switch used, the switch acts as a reflective load. As the reflections only should happen on the receiver side (as long as we configure the switches/pins correctly), we should be fine when using 30 dB attenuation between the RX and TX ports, but we would like to have this confirmed.

Thank you

B

Startguide för GNU Radio

B.1 Linux

Beroende på vilken Linuxdistribution som GNU Radio ska installeras på används terminalkommandon enligt tabell B.1.

Tabell B.1: Terminalkommandon för att installera GNU Radio i Linux.

Linuxdistribution	Installationskommando
Debian/Ubuntu etc	$ apt install gnuradio
Fedora	$ dnf install gnuradio
Red Hat Enterprise Linux/CentOS	$ yum install gnuradio
Archlinux	$ pacman -S gnuradio
Gentoo Linux	$ emerge net-wireless/gnuradio

För vissa distributioner behövs även USRP Hardware Driver (UHD) installeras. För bland annat Archlinux installeras UHD tillsammans med GNU Radio.

Den grafiska miljön GNU Radio Companion startas via terminalen genom kommandot ”$gnuradio-companion”.

B.2 macOS

För macOS finns det flera sätt att installera GNU Radio. Den som rekommenderas av Ettus Research görs med pakethanteraren MacPorts enligt följande steg:

1. Ladda hem XQuartz från `https://www.xquartz.org`.
2. Installera XCode och XCode Command Line Tools exempelvis genom Mac App Store.
3. Acceptera användarlicenser för XCode genom att köra ”$ sudo xcodebuild -license” i terminalen och följ instruktioner.
4. Ladda hem rätt version av MacPorts till ditt operativsystem från `https://www.macports.org/install.php`.

5. Installera UHD via MacPorts genom att skriva ”$ sudo port install uhd” i terminalen. Det går även att installera uhd-devel vilken är den senaste versionen av UHD som ständigt är under utveckling.
6. Installera GNU Radio genom att skriva ”$ sudo port install gnuradio” i terminalen.

Starta det grafiska gränssnittet för GNU Radio genom att skriva ”$ gnuradio-companion” i terminalen.

Observera att om en annan pakethanterare används (exempelvis Brew) kan det krävas att path-variabler exporteras om för att rätt subrutiner ska köras när GNU Radio startar.

B.3 Windows

För Windows existerar färdiga installationspaket som finns på `http://www.gcndevelopment.com/gnuradio/downloads.htm`. Dessa inkluderar nödvändiga program samt vanliga drivrutiner för olika SDR-kort.

För B205mini-i-kortet finns installationspaket för dess drivrutiner på `http://files.ettus.com/binaries/uhd/latest_release/`.

B.4 Vanliga fel och möjliga lösningar

Om pakethanterare används för att installera GNU Radio, kontrollera att det är den senaste versionen av GNU Radio som finns i pakethanteraren.

Ett ofta förekommande fel för macOS är att Python kraschar när Pythonkod från GNU Radio exekveras. En lösning på detta problem kan vara att istället installera utvecklingsversioner av program och drivrutiner, gnuradio-devel och uhd-devel.

Som nämnts i macOS-kapitlet är det lätt att få problem med GNU Radio ifall miljövariabler är felaktivt konfigurerade, något som ofta är fallet ifall två pakethanterare används. Säkerställ att alla mappar blivit borttagna vid avinstallation, speciellt de mappar och filer som listas vid exekvering av ”$ echo $path”.

Mer information och exempel finns på den officiella hemsidan för GNU Radio: `https://wiki.gnuradio.org/index.php/InstallingGR`. För dokumentation av drivrutiner för kort från Ettus Research se `http://files.ettus.com/manual/page_install.html`.

C

Kod

C.1 Pythonkod

C.1.1 Egenskrivet GNU Radio Companionblock som beräknar fas och amplitud för en viss frekvens

```
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first class it finds
to get ports and parameters of your block. The arguments to __init__ will
be the parameters. All of them are required to have default values!
"""

import numpy as np
from gnuradio import gr
import scipy


class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    """Takes a vector of data that has been Fourier transformed and calculates the amplitude and phase of the specified frequency"""

    def __init__(self, v_len=1024, samp_rate=32000, freq=2000):  # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='Single frequency amplitude/phase.',   # will show up in GRC
            in_sig=[(np.complex64, v_len)],
            out_sig=[np.float32, np.float32]
        )
        # if an attribute with the same name as a parameter is found,
        # a callback is registered (properties work, too).
        self.v_len = (v_len)
        self.samp_rate = int(samp_rate)
        self.freq = int(freq)
```

```
30
31      def work(self, input_items, output_items):
32          """Takes a vector of data that has been Fourier transformed and
        calculates the amplitude and phase of the specified frequency"""
33          def getBin(v_len, samp_rate, freq):
34              return int(np.round(
35                  (float(v_len)/2)+(float(v_len)/2)*(float(freq)/(float(
        samp_rate)/2))
36              ))
37
38
39          freq_bin = getBin(self.v_len, self.samp_rate, self.freq)
40
41          for index, work_item in enumerate(input_items[0]):
42              amp = np.absolute(work_item[freq_bin])
43              phase = np.angle(work_item[freq_bin])
44              output_items[0][index] = amp
45              output_items[1][index] = phase
46          return len(output_items[0])
```

Listing C.1: Pythonkod för ett egenskrivet GNU Radio Companionblock som beräknar fas och amplitud för en viss frekvens. Indata är en datavektor som har fouriertransformerats och därmed består av ett antal ”frekvenshinkar”.

C.1.2 Egenskrivet GNU Radio Companionblock som ser till att fasskillnaden ligger mellan $-\pi$ och π

```
1  """
2      Embedded Python Blocks:
3
4      Each time this file is saved, GRC will instantiate the first class
       it finds
5      to get ports and parameters of your block. The arguments to
       __init__  will
6      be the parameters. All of them are required to have default values!
7      """
8
9  import numpy as np
10 from gnuradio import gr
11 import scipy
12
13
14 class blk(gr.sync_block):  # other base classes are basic_block,
       decim_block, interp_block
15     """Make sure the phase is between -pi and pi"""
16
17     def __init__(self, example_param=1.0):  # only default arguments
       here
18         """arguments to this function show up as parameters in GRC"""
19         gr.sync_block.__init__(
20                                 self,
21                                 name='Phase between -pi and pi',   #
       will show up in GRC
22                                 in_sig=[np.float32],
```

```
                                    out_sig=[np.float32]
                                    )
                                    # if an attribute with the same name as
    a parameter is found,
                                    # a callback is registered (properties
    work, too).
        self.example_param = example_param

    def work(self, input_items, output_items):
        """Make sure the phase is between -pi and pi"""
        output_items[0][:] = (scipy.pi + input_items[0]) % (2 * scipy.
    pi) - scipy.pi #* self.example_param
        return len(output_items[0])
```

Listing C.2: Pythonkod för ett egenskrivet GNU Radio Companionblock som ser till att indatan, fasen i det här fallet, ligger mellan -pi och pi.

C.1.3 Växla centerfrekvens och presentera fasskillnad mellan skickad och mottagen signal

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Fftfas
# Generated: Wed Apr 11 14:04:51 2018
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from PyQt4 import Qt
from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import fft
from gnuradio import gr
from gnuradio import qtgui
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.fft import window
from gnuradio.filter import firdes
from optparse import OptionParser
import epy_block_0
import epy_block_0_0
import epy_block_0_0_1
import sip
import sys
```

```
36 import time
37
38 # This line has been added from the grc-generated Python file.
39 import thread
40
41 from gnuradio import qtgui
42
43
44 class fftfas(gr.top_block, Qt.QWidget):
45
46     def __init__(self):
47         gr.top_block.__init__(self, "Phase difference")
48         Qt.QWidget.__init__(self)
49         self.setWindowTitle("Phase difference")
50         qtgui.util.check_set_qss()
51         try:
52             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
53         except:
54             pass
55         self.top_scroll_layout = Qt.QVBoxLayout()
56         self.setLayout(self.top_scroll_layout)
57         self.top_scroll = Qt.QScrollArea()
58         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
59         self.top_scroll_layout.addWidget(self.top_scroll)
60         self.top_scroll.setWidgetResizable(True)
61         self.top_widget = Qt.QWidget()
62         self.top_scroll.setWidget(self.top_widget)
63         self.top_layout = Qt.QVBoxLayout(self.top_widget)
64         self.top_grid_layout = Qt.QGridLayout()
65         self.top_layout.addLayout(self.top_grid_layout)
66
67         self.settings = Qt.QSettings("GNU Radio", "fftfas")
68         self.restoreGeometry(self.settings.value("geometry").
    toByteArray())
69
70
71         ##################################################
72         # Variables
73         ##################################################
74         self.samp_rate = samp_rate = 4e6
75         self.rel_freq = rel_freq = 1e6
76         self.fft_width = fft_width = 1024
77         self.skip = skip = (fft_width/2+ 1  + int(round(rel_freq*
    fft_width/samp_rate)))
78         self.center_freq = center_freq = 1e9
79
80         ##################################################
81         # Blocks
82         ##################################################
83         self.uhd_usrp_source_0 = uhd.usrp_source(
84             ",".join(("", "")),
85             uhd.stream_args(
86                 cpu_format="fc32",
87                 channels=range(1),
88             ),
89         )
```

```
90          self.uhd_usrp_source_0.set_clock_source('internal', 0)
91          self.uhd_usrp_source_0.set_samp_rate(samp_rate)
92          self.uhd_usrp_source_0.set_time_now(uhd.time_spec(time.time()),
     uhd.ALL_MBOARDS)
93          self.uhd_usrp_source_0.set_center_freq(center_freq, 0)
94          self.uhd_usrp_source_0.set_normalized_gain(0, 0)
95          self.uhd_usrp_source_0.set_antenna('RX2', 0)
96          self.uhd_usrp_sink_0 = uhd.usrp_sink(
97              ",".join(("", "")),
98              uhd.stream_args(
99                  cpu_format="fc32",
100                 channels=range(1),
101             ),
102         )
103         self.uhd_usrp_sink_0.set_clock_source('internal', 0)
104         self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
105         self.uhd_usrp_sink_0.set_time_now(uhd.time_spec(time.time()),
    uhd.ALL_MBOARDS)
106         self.uhd_usrp_sink_0.set_center_freq(center_freq, 0)
107         self.uhd_usrp_sink_0.set_normalized_gain(1, 0)
108         self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
109         self.qtgui_number_sink_1_1_0 = qtgui.number_sink(
110             gr.sizeof_float,
111             0,
112             qtgui.NUM_GRAPH_HORIZ,
113             1
114         )
115         self.qtgui_number_sink_1_1_0.set_update_time(0.10)
116         self.qtgui_number_sink_1_1_0.set_title('Mottagen fas')
117
118         labels = ['', '', '', '', '',
119                   '', '', '', '', '']
120         units = ['', '', '', '', '',
121                  '', '', '', '', '']
122         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
123                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
124         factor = [1, 1, 1, 1, 1,
125                   1, 1, 1, 1, 1]
126         for i in xrange(1):
127             self.qtgui_number_sink_1_1_0.set_min(i, -1)
128             self.qtgui_number_sink_1_1_0.set_max(i, 1)
129             self.qtgui_number_sink_1_1_0.set_color(i, colors[i][0],
    colors[i][1])
130             if len(labels[i]) == 0:
131                 self.qtgui_number_sink_1_1_0.set_label(i, "Data {0}".
    format(i))
132             else:
133                 self.qtgui_number_sink_1_1_0.set_label(i, labels[i])
134             self.qtgui_number_sink_1_1_0.set_unit(i, units[i])
135             self.qtgui_number_sink_1_1_0.set_factor(i, factor[i])
136
137         self.qtgui_number_sink_1_1_0.enable_autoscale(True)
138         self._qtgui_number_sink_1_1_0_win = sip.wrapinstance(self.
    qtgui_number_sink_1_1_0.pyqwidget(), Qt.QWidget)
```

```
139         self.top_layout.addWidget(self._qtgui_number_sink_1_1_0_win)
140         self.qtgui_number_sink_1_1 = qtgui.number_sink(
141             gr.sizeof_float,
142             0,
143             qtgui.NUM_GRAPH_HORIZ,
144             1
145         )
146         self.qtgui_number_sink_1_1.set_update_time(0.10)
147         self.qtgui_number_sink_1_1.set_title('Mottagen amplitud')
148 
149         labels = ['', '', '', '', '',
150                   '', '', '', '', '']
151         units = ['', '', '', '', '',
152                  '', '', '', '', '']
153         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
154                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
155         factor = [1, 1, 1, 1, 1,
156                   1, 1, 1, 1, 1]
157         for i in xrange(1):
158             self.qtgui_number_sink_1_1.set_min(i, -1)
159             self.qtgui_number_sink_1_1.set_max(i, 1)
160             self.qtgui_number_sink_1_1.set_color(i, colors[i][0],
    colors[i][1])
161             if len(labels[i]) == 0:
162                 self.qtgui_number_sink_1_1.set_label(i, "Data {0}".
    format(i))
163             else:
164                 self.qtgui_number_sink_1_1.set_label(i, labels[i])
165             self.qtgui_number_sink_1_1.set_unit(i, units[i])
166             self.qtgui_number_sink_1_1.set_factor(i, factor[i])
167 
168         self.qtgui_number_sink_1_1.enable_autoscale(True)
169         self._qtgui_number_sink_1_1_win = sip.wrapinstance(self.
    qtgui_number_sink_1_1.pyqwidget(), Qt.QWidget)
170         self.top_layout.addWidget(self._qtgui_number_sink_1_1_win)
171         self.qtgui_number_sink_1_0 = qtgui.number_sink(
172             gr.sizeof_float,
173             0,
174             qtgui.NUM_GRAPH_HORIZ,
175             1
176         )
177         self.qtgui_number_sink_1_0.set_update_time(0.10)
178         self.qtgui_number_sink_1_0.set_title('Skickad fas')
179 
180         labels = ['', '', '', '', '',
181                   '', '', '', '', '']
182         units = ['', '', '', '', '',
183                  '', '', '', '', '']
184         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
185                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
186         factor = [1, 1, 1, 1, 1,
187                   1, 1, 1, 1, 1]
```

```
188         for i in xrange(1):
189             self.qtgui_number_sink_1_0.set_min(i, -1)
190             self.qtgui_number_sink_1_0.set_max(i, 1)
191             self.qtgui_number_sink_1_0.set_color(i, colors[i][0],
    colors[i][1])
192             if len(labels[i]) == 0:
193                 self.qtgui_number_sink_1_0.set_label(i, "Data {0}".
    format(i))
194             else:
195                 self.qtgui_number_sink_1_0.set_label(i, labels[i])
196             self.qtgui_number_sink_1_0.set_unit(i, units[i])
197             self.qtgui_number_sink_1_0.set_factor(i, factor[i])
198 
199         self.qtgui_number_sink_1_0.enable_autoscale(True)
200         self._qtgui_number_sink_1_0_win = sip.wrapinstance(self.
    qtgui_number_sink_1_0.pyqwidget(), Qt.QWidget)
201         self.top_layout.addWidget(self._qtgui_number_sink_1_0_win)
202         self.qtgui_number_sink_1 = qtgui.number_sink(
203             gr.sizeof_float,
204             0,
205             qtgui.NUM_GRAPH_HORIZ,
206             1
207         )
208         self.qtgui_number_sink_1.set_update_time(0.10)
209         self.qtgui_number_sink_1.set_title('Skickad amplitud')
210 
211         labels = ['', '', '', '', '',
212                   '', '', '', '', '']
213         units = ['', '', '', '', '',
214                  '', '', '', '', '']
215         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
216                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
217         factor = [1, 1, 1, 1, 1,
218                   1, 1, 1, 1, 1]
219         for i in xrange(1):
220             self.qtgui_number_sink_1.set_min(i, -1)
221             self.qtgui_number_sink_1.set_max(i, 1)
222             self.qtgui_number_sink_1.set_color(i, colors[i][0], colors[
    i][1])
223             if len(labels[i]) == 0:
224                 self.qtgui_number_sink_1.set_label(i, "Data {0}".format
    (i))
225             else:
226                 self.qtgui_number_sink_1.set_label(i, labels[i])
227             self.qtgui_number_sink_1.set_unit(i, units[i])
228             self.qtgui_number_sink_1.set_factor(i, factor[i])
229 
230         self.qtgui_number_sink_1.enable_autoscale(True)
231         self._qtgui_number_sink_1_win = sip.wrapinstance(self.
    qtgui_number_sink_1.pyqwidget(), Qt.QWidget)
232         self.top_layout.addWidget(self._qtgui_number_sink_1_win)
233         self.qtgui_number_sink_0 = qtgui.number_sink(
234             gr.sizeof_float,
235             0,
```

```
236             qtgui.NUM_GRAPH_HORIZ,
237             1
238         )
239         self.qtgui_number_sink_0.set_update_time(0.10)
240         self.qtgui_number_sink_0.set_title('Fasskillnad mellan skickad
    och mottagen signal')
241 
242         labels = ['', '', '', '', '',
243                   '', '', '', '', '']
244         units = ['', '', '', '', '',
245                  '', '', '', '', '']
246         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
247                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
248         factor = [1, 1, 1, 1, 1,
249                   1, 1, 1, 1, 1]
250         for i in xrange(1):
251             self.qtgui_number_sink_0.set_min(i, -3.141)
252             self.qtgui_number_sink_0.set_max(i, 3.141)
253             self.qtgui_number_sink_0.set_color(i, colors[i][0], colors[
    i][1])
254             if len(labels[i]) == 0:
255                 self.qtgui_number_sink_0.set_label(i, "Data {0}".format
    (i))
256             else:
257                 self.qtgui_number_sink_0.set_label(i, labels[i])
258             self.qtgui_number_sink_0.set_unit(i, units[i])
259             self.qtgui_number_sink_0.set_factor(i, factor[i])
260 
261         self.qtgui_number_sink_0.enable_autoscale(False)
262         self._qtgui_number_sink_0_win = sip.wrapinstance(self.
    qtgui_number_sink_0.pyqwidget(), Qt.QWidget)
263         self.top_layout.addWidget(self._qtgui_number_sink_0_win)
264         self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
265             1024, #size
266             firdes.WIN_BLACKMAN_hARRIS, #wintype
267             0, #fc
268             samp_rate, #bw
269             "", #name
270             1 #number of inputs
271         )
272         self.qtgui_freq_sink_x_0.set_update_time(0.10)
273         self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
274         self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
275         self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE,
     0.0, 0, "")
276         self.qtgui_freq_sink_x_0.enable_autoscale(False)
277         self.qtgui_freq_sink_x_0.enable_grid(False)
278         self.qtgui_freq_sink_x_0.set_fft_average(1.0)
279         self.qtgui_freq_sink_x_0.enable_axis_labels(True)
280         self.qtgui_freq_sink_x_0.enable_control_panel(False)
281 
282         if not True:
283           self.qtgui_freq_sink_x_0.disable_legend()
284 
```

```
285         if "complex" == "float" or "complex" == "msg_float":
286           self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
287 
288         labels = ['', '', '', '', '',
289                   '', '', '', '', '']
290         widths = [1, 1, 1, 1, 1,
291                   1, 1, 1, 1, 1]
292         colors = ["blue", "red", "green", "black", "cyan",
293                   "magenta", "yellow", "dark red", "dark green", "dark
    blue"]
294         alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
295                   1.0, 1.0, 1.0, 1.0, 1.0]
296         for i in xrange(1):
297             if len(labels[i]) == 0:
298                 self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".
    format(i))
299             else:
300                 self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
301             self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
302             self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
303             self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
304 
305         self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.
    qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
306         self.top_layout.addWidget(self._qtgui_freq_sink_x_0_win)
307         self.fft_vxx_0_0 = fft.fft_vcc(fft_width, True, (window.
    blackmanharris(1024)), True, 1)
308         self.fft_vxx_0 = fft.fft_vcc(fft_width, True, (window.
    blackmanharris(1024)), True, 1)
309         self.epy_block_0_0_1 = epy_block_0_0_1.blk(v_len=1024,
    samp_rate=samp_rate, freq=rel_freq)
310         self.epy_block_0_0 = epy_block_0_0.blk(v_len=1024, samp_rate=
    samp_rate, freq=rel_freq)
311         self.epy_block_0 = epy_block_0.blk(example_param=1.0)
312         self.blocks_sub_xx_0 = blocks.sub_ff(1)
313         self.blocks_stream_to_vector_0_0 = blocks.stream_to_vector(gr.
    sizeof_gr_complex*1, 1024)
314         self.blocks_stream_to_vector_0 = blocks.stream_to_vector(gr.
    sizeof_gr_complex*1, 1024)
315         self.analog_sig_source_x_0 = analog.sig_source_c(samp_rate,
    analog.GR_COS_WAVE, rel_freq, 1, 0)
316 
317 
318 
319         ##################################################
320         # Connections
321         ##################################################
322         self.connect((self.analog_sig_source_x_0, 0), (self.
    blocks_stream_to_vector_0_0, 0))
323         self.connect((self.analog_sig_source_x_0, 0), (self.
    uhd_usrp_sink_0, 0))
324         self.connect((self.blocks_stream_to_vector_0, 0), (self.
    fft_vxx_0, 0))
325         self.connect((self.blocks_stream_to_vector_0_0, 0), (self.
    fft_vxx_0_0, 0))
326         self.connect((self.blocks_sub_xx_0, 0), (self.epy_block_0, 0))
```

```
327         self.connect((self.epy_block_0, 0), (self.qtgui_number_sink_0,
    0))
328         self.connect((self.epy_block_0_0, 1), (self.blocks_sub_xx_0, 1)
    )
329         self.connect((self.epy_block_0_0, 0), (self.
    qtgui_number_sink_1_1, 0))
330         self.connect((self.epy_block_0_0, 1), (self.
    qtgui_number_sink_1_1_0, 0))
331         self.connect((self.epy_block_0_0_1, 1), (self.blocks_sub_xx_0,
    0))
332         self.connect((self.epy_block_0_0_1, 0), (self.
    qtgui_number_sink_1, 0))
333         self.connect((self.epy_block_0_0_1, 1), (self.
    qtgui_number_sink_1_0, 0))
334         self.connect((self.fft_vxx_0, 0), (self.epy_block_0_0, 0))
335         self.connect((self.fft_vxx_0_0, 0), (self.epy_block_0_0_1, 0))
336         self.connect((self.uhd_usrp_source_0, 0), (self.
    blocks_stream_to_vector_0, 0))
337         self.connect((self.uhd_usrp_source_0, 0), (self.
    qtgui_freq_sink_x_0, 0))
338 
339     def closeEvent(self, event):
340         self.settings = Qt.QSettings("GNU Radio", "fftfas")
341         self.settings.setValue("geometry", self.saveGeometry())
342         event.accept()
343 
344     def get_samp_rate(self):
345         return self.samp_rate
346 
347     def set_samp_rate(self, samp_rate):
348         self.samp_rate = samp_rate
349         self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
350         self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
351         self.set_skip((self.fft_width/2+ 1  + int(round(self.rel_freq*
    self.fft_width/self.samp_rate))))
352         self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
353         self.epy_block_0_0_1.samp_rate = self.samp_rate
354         self.epy_block_0_0.samp_rate = self.samp_rate
355         self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
356 
357     def get_rel_freq(self):
358         return self.rel_freq
359 
360     def set_rel_freq(self, rel_freq):
361         self.rel_freq = rel_freq
362         self.set_skip((self.fft_width/2+ 1  + int(round(self.rel_freq*
    self.fft_width/self.samp_rate))))
363         self.epy_block_0_0_1.freq = self.rel_freq
364         self.epy_block_0_0.freq = self.rel_freq
365         self.analog_sig_source_x_0.set_frequency(self.rel_freq)
366 
367     def get_fft_width(self):
368         return self.fft_width
369 
370     def set_fft_width(self, fft_width):
371         self.fft_width = fft_width
```

```
372         self.set_skip((self.fft_width/2+ 1  + int(round(self.rel_freq*
    self.fft_width/self.samp_rate))))
373 
374     def get_skip(self):
375         return self.skip
376 
377     def set_skip(self, skip):
378         self.skip = skip
379 
380     def get_center_freq(self):
381         return self.center_freq
382 
383     def set_center_freq(self, center_freq):
384         self.center_freq = center_freq
385         self.uhd_usrp_source_0.set_center_freq(self.center_freq, 0)
386         self.uhd_usrp_sink_0.set_center_freq(self.center_freq, 0)
387 
388 
389 def main(top_block_cls=fftfas, options=None):
390 
391     from distutils.version import StrictVersion
392     if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
393         style = gr.prefs().get_string('qtgui', 'style', 'raster')
394         Qt.QApplication.setGraphicsSystem(style)
395     qapp = Qt.QApplication(sys.argv)
396 
397     tb = top_block_cls()
398     tb.start()
399     tb.show()
400 
401     # This part has been added from the grc-generated Python file.
402     #-------------------------------------------------------------
403     def change_center_frequency_and_sleep():
404 
405         # Wait for user input
406         raw_input('Press enter when you want to continue: ')
407         print('Starting')
408 
409         start_sleep = 10
410         time.sleep(start_sleep)
411 
412         sleep_time = 10
413         first_center_freq = 1*10**9
414         second_center_freq = 1.01*10**9
415 
416         while (1):
417 
418             tb.set_center_freq(first_center_freq)
419             print('\n Current center frequency: {}'.format(tb.
    get_center_freq()))
420             time.sleep(sleep_time)
421 
422             tb.set_center_freq(second_center_freq)
423             print('\n Current center frequency: {}'.format(tb.
    get_center_freq()))
424             time.sleep(sleep_time)
```

```
425
426     thread.start_new_thread(change_center_frequency_and_sleep, ())
427
428     # End of addition
429
430     def quitting():
431         tb.stop()
432         tb.wait()
433     qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
434     qapp.exec_()
435
436
437 if __name__ == '__main__':
438     main()
```

Listing C.3: Pythonkod för en modifierad GNU Radio Companion flödeskarta, se 3.3. Utöver att göra det som flödeskartan visar så byter den centerfrekvens fram och tillbaka mellan 1 och 1.01 GH. Den stannar 10 sekunder på varje centerfrekvens. De bitar som har ändrats från den autogenererade koden är utmärkta.

C.1.4 Spara temperaturdata hos kortet

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  ##################################################
4  # GNU Radio Python Flow Graph
5  # Title: Measure Temperature And Amplitude
6  # Generated: Wed Apr 25 16:58:31 2018
7  ##################################################
8
9  if __name__ == '__main__':
10     import ctypes
11     import sys
12     if sys.platform.startswith('linux'):
13         try:
14             x11 = ctypes.cdll.LoadLibrary('libX11.so')
15             x11.XInitThreads()
16         except:
17             print "Warning: failed to XInitThreads()"
18
19 from PyQt4 import Qt
20 from gnuradio import analog
21 from gnuradio import blocks
22 from gnuradio import eng_notation
23 from gnuradio import fft
24 from gnuradio import gr
25 from gnuradio import qtgui
26 from gnuradio import uhd
27 from gnuradio.eng_option import eng_option
28 from gnuradio.fft import window
29 from gnuradio.filter import firdes
30 from optparse import OptionParser
31 import sip
32 import sys
33 import time
```

```
34
35 # This line has been added from the grc-generated Python file.
36 import thread
37
38 from gnuradio import qtgui
39
40
41 class measure_temperature_and_amplitude(gr.top_block, Qt.QWidget):
42
43     def __init__(self):
44         gr.top_block.__init__(self, "Measure Temperature And Amplitude"
    )
45         Qt.QWidget.__init__(self)
46         self.setWindowTitle("Measure Temperature And Amplitude")
47         qtgui.util.check_set_qss()
48         try:
49             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
50         except:
51             pass
52         self.top_scroll_layout = Qt.QVBoxLayout()
53         self.setLayout(self.top_scroll_layout)
54         self.top_scroll = Qt.QScrollArea()
55         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
56         self.top_scroll_layout.addWidget(self.top_scroll)
57         self.top_scroll.setWidgetResizable(True)
58         self.top_widget = Qt.QWidget()
59         self.top_scroll.setWidget(self.top_widget)
60         self.top_layout = Qt.QVBoxLayout(self.top_widget)
61         self.top_grid_layout = Qt.QGridLayout()
62         self.top_layout.addLayout(self.top_grid_layout)
63
64         self.settings = Qt.QSettings("GNU Radio", "
    measure_temperature_and_amplitude")
65         self.restoreGeometry(self.settings.value("geometry").
    toByteArray())
66
67
68         ##################################################
69         # Variables
70         ##################################################
71         self.samp_rate = samp_rate = 1e6
72         self.fft_width = fft_width = 1024
73         self.f1 = f1 = 0.1e6
74         self.send_gain = send_gain = 50
75         self.fft_offset_1 = fft_offset_1 = int(round(f1*fft_width/
    samp_rate) + fft_width/2 +1)
76         self.center_freq = center_freq = 1e9
77
78         ##################################################
79         # Blocks
80         ##################################################
81         self.uhd_usrp_source_0 = uhd.usrp_source(
82             ",".join(("", "")),
83             uhd.stream_args(
84                 cpu_format="fc32",
85                 channels=range(1),
```

```
            ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(center_freq, 0)
        self.uhd_usrp_source_0.set_gain(0, 0)
        self.uhd_usrp_source_0.set_antenna('RX2', 0)
        self.uhd_usrp_sink_0 = uhd.usrp_sink(
            ",".join(("", "")),
            uhd.stream_args(
                cpu_format="fc32",
                channels=range(1),
            ),
        )
        self.uhd_usrp_sink_0.set_clock_source('internal', 0)
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_center_freq(center_freq, 0)
        self.uhd_usrp_sink_0.set_gain(send_gain, 0)
        self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
        self.qtgui_number_sink_0 = qtgui.number_sink(
            gr.sizeof_float,
            0,
            qtgui.NUM_GRAPH_HORIZ,
            1
        )
        self.qtgui_number_sink_0.set_update_time(0.10)
        self.qtgui_number_sink_0.set_title('Magnitude of 1st requency')

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        units = ['', '', '', '', '',
                 '', '', '', '', '']
        colors = [("black", "black"), ("black", "black"), ("black", "black"), ("black", "black"), ("black", "black"),
                  ("black", "black"), ("black", "black"), ("black", "black"), ("black", "black"), ("black", "black")]
        factor = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        for i in xrange(1):
            self.qtgui_number_sink_0.set_min(i, -1)
            self.qtgui_number_sink_0.set_max(i, 1)
            self.qtgui_number_sink_0.set_color(i, colors[i][0], colors[i][1])
            if len(labels[i]) == 0:
                self.qtgui_number_sink_0.set_label(i, "Data {0}".format(i))
            else:
                self.qtgui_number_sink_0.set_label(i, labels[i])
            self.qtgui_number_sink_0.set_unit(i, units[i])
            self.qtgui_number_sink_0.set_factor(i, factor[i])

        self.qtgui_number_sink_0.enable_autoscale(False)
        self._qtgui_number_sink_0_win = sip.wrapinstance(self.qtgui_number_sink_0.pyqwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_number_sink_0_win)
        self.qtgui_freq_sink_x_0_0 = qtgui.freq_sink_c(
            fft_width, #size
```

```
137             firdes.WIN_BLACKMAN_hARRIS, #wintype
138             0, #fc
139             samp_rate, #bw
140             'Skickad signal', #name
141             1 #number of inputs
142         )
143         self.qtgui_freq_sink_x_0_0.set_update_time(0.10)
144         self.qtgui_freq_sink_x_0_0.set_y_axis(-140, 10)
145         self.qtgui_freq_sink_x_0_0.set_y_label('Relative Gain', 'dB')
146         self.qtgui_freq_sink_x_0_0.set_trigger_mode(qtgui.
    TRIG_MODE_FREE, 0.0, 0, "")
147         self.qtgui_freq_sink_x_0_0.enable_autoscale(False)
148         self.qtgui_freq_sink_x_0_0.enable_grid(False)
149         self.qtgui_freq_sink_x_0_0.set_fft_average(1.0)
150         self.qtgui_freq_sink_x_0_0.enable_axis_labels(True)
151         self.qtgui_freq_sink_x_0_0.enable_control_panel(True)
152 
153         if not True:
154           self.qtgui_freq_sink_x_0_0.disable_legend()
155 
156         if "complex" == "float" or "complex" == "msg_float":
157           self.qtgui_freq_sink_x_0_0.set_plot_pos_half(not True)
158 
159         labels = ['', '', '', '', '',
160                   '', '', '', '', '']
161         widths = [1, 1, 1, 1, 1,
162                   1, 1, 1, 1, 1]
163         colors = ["blue", "red", "green", "black", "cyan",
164                   "magenta", "yellow", "dark red", "dark green", "dark
    blue"]
165         alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
166                   1.0, 1.0, 1.0, 1.0, 1.0]
167         for i in xrange(1):
168             if len(labels[i]) == 0:
169                 self.qtgui_freq_sink_x_0_0.set_line_label(i, "Data {0}"
    .format(i))
170             else:
171                 self.qtgui_freq_sink_x_0_0.set_line_label(i, labels[i])
172             self.qtgui_freq_sink_x_0_0.set_line_width(i, widths[i])
173             self.qtgui_freq_sink_x_0_0.set_line_color(i, colors[i])
174             self.qtgui_freq_sink_x_0_0.set_line_alpha(i, alphas[i])
175 
176         self._qtgui_freq_sink_x_0_0_win = sip.wrapinstance(self.
    qtgui_freq_sink_x_0_0.pyqwidget(), Qt.QWidget)
177         self.top_layout.addWidget(self._qtgui_freq_sink_x_0_0_win)
178         self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
179             fft_width, #size
180             firdes.WIN_BLACKMAN_hARRIS, #wintype
181             0, #fc
182             samp_rate, #bw
183             'Mottagen signal', #name
184             1 #number of inputs
185         )
186         self.qtgui_freq_sink_x_0.set_update_time(0.10)
187         self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
188         self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
```

```
189         self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE,
    0.0, 0, "")
190         self.qtgui_freq_sink_x_0.enable_autoscale(False)
191         self.qtgui_freq_sink_x_0.enable_grid(False)
192         self.qtgui_freq_sink_x_0.set_fft_average(1.0)
193         self.qtgui_freq_sink_x_0.enable_axis_labels(True)
194         self.qtgui_freq_sink_x_0.enable_control_panel(True)
195 
196         if not True:
197           self.qtgui_freq_sink_x_0.disable_legend()
198 
199         if "complex" == "float" or "complex" == "msg_float":
200           self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
201 
202         labels = ['', '', '', '', '',
203                   '', '', '', '', '']
204         widths = [1, 1, 1, 1, 1,
205                   1, 1, 1, 1, 1]
206         colors = ["blue", "red", "green", "black", "cyan",
207                   "magenta", "yellow", "dark red", "dark green", "dark
    blue"]
208         alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
209                   1.0, 1.0, 1.0, 1.0, 1.0]
210         for i in xrange(1):
211             if len(labels[i]) == 0:
212                 self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".
    format(i))
213             else:
214                 self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
215             self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
216             self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
217             self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
218 
219         self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.
    qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
220         self.top_layout.addWidget(self._qtgui_freq_sink_x_0_win)
221         self.fft_vxx_0 = fft.fft_vcc(fft_width, True, (window.
    blackmanharris(fft_width)), True, 1)
222         self.blocks_vector_to_stream_0 = blocks.vector_to_stream(gr.
    sizeof_gr_complex*1, fft_width)
223         self.blocks_stream_to_vector_0 = blocks.stream_to_vector(gr.
    sizeof_gr_complex*1, fft_width)
224         self.blocks_skiphead_0 = blocks.skiphead(gr.sizeof_gr_complex
    *1, fft_offset_1)
225         self.blocks_keep_one_in_n_0 = blocks.keep_one_in_n(gr.
    sizeof_gr_complex*1, fft_width)
226         self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_float*1, '
    /Users/arvidbjurklint/Google Drive/Kandidatarbete-Software defined
    radio/Experimentdata/Temperaturmatning/amplitude.txt', False)
227         self.blocks_file_sink_1.set_unbuffered(False)
228         self.blocks_complex_to_mag_0 = blocks.complex_to_mag(1)
229         self.analog_sig_source_x_0 = analog.sig_source_c(samp_rate,
    analog.GR_COS_WAVE, f1, 1, 0)
230 
231 
232 
```

```
233         ##################################################
234         # Connections
235         ##################################################
236         self.connect((self.analog_sig_source_x_0, 0), (self.
    qtgui_freq_sink_x_0_0, 0))
237         self.connect((self.analog_sig_source_x_0, 0), (self.
    uhd_usrp_sink_0, 0))
238         self.connect((self.blocks_complex_to_mag_0, 0), (self.
    blocks_file_sink_1, 0))
239         self.connect((self.blocks_complex_to_mag_0, 0), (self.
    qtgui_number_sink_0, 0))
240         self.connect((self.blocks_keep_one_in_n_0, 0), (self.
    blocks_complex_to_mag_0, 0))
241         self.connect((self.blocks_skiphead_0, 0), (self.
    blocks_keep_one_in_n_0, 0))
242         self.connect((self.blocks_stream_to_vector_0, 0), (self.
    fft_vxx_0, 0))
243         self.connect((self.blocks_vector_to_stream_0, 0), (self.
    blocks_skiphead_0, 0))
244         self.connect((self.fft_vxx_0, 0), (self.
    blocks_vector_to_stream_0, 0))
245         self.connect((self.uhd_usrp_source_0, 0), (self.
    blocks_stream_to_vector_0, 0))
246         self.connect((self.uhd_usrp_source_0, 0), (self.
    qtgui_freq_sink_x_0, 0))
247 
248     def closeEvent(self, event):
249         self.settings = Qt.QSettings("GNU Radio", "
    measure_temperature_and_amplitude")
250         self.settings.setValue("geometry", self.saveGeometry())
251         event.accept()
252 
253     def get_samp_rate(self):
254         return self.samp_rate
255 
256     def set_samp_rate(self, samp_rate):
257         self.samp_rate = samp_rate
258         self.set_fft_offset_1(int(round(self.f1*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
259         self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
260         self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
261         self.qtgui_freq_sink_x_0_0.set_frequency_range(0, self.
    samp_rate)
262         self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
263         self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
264 
265     def get_fft_width(self):
266         return self.fft_width
267 
268     def set_fft_width(self, fft_width):
269         self.fft_width = fft_width
270         self.set_fft_offset_1(int(round(self.f1*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
271         self.blocks_keep_one_in_n_0.set_n(self.fft_width)
272 
273     def get_f1(self):
```

```
274         return self.f1
275
276     def set_f1(self, f1):
277         self.f1 = f1
278         self.set_fft_offset_1(int(round(self.f1*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
279         self.analog_sig_source_x_0.set_frequency(self.f1)
280
281     def get_send_gain(self):
282         return self.send_gain
283
284     def set_send_gain(self, send_gain):
285         self.send_gain = send_gain
286         self.uhd_usrp_sink_0.set_gain(self.send_gain, 0)
287
288
289     def get_fft_offset_1(self):
290         return self.fft_offset_1
291
292     def set_fft_offset_1(self, fft_offset_1):
293         self.fft_offset_1 = fft_offset_1
294
295     def get_center_freq(self):
296         return self.center_freq
297
298     def set_center_freq(self, center_freq):
299         self.center_freq = center_freq
300         self.uhd_usrp_source_0.set_center_freq(self.center_freq, 0)
301         self.uhd_usrp_sink_0.set_center_freq(self.center_freq, 0)
302
303
304 def main(top_block_cls=measure_temperature_and_amplitude, options=None)
    :
305
306     from distutils.version import StrictVersion
307     if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
308         style = gr.prefs().get_string('qtgui', 'style', 'raster')
309         Qt.QApplication.setGraphicsSystem(style)
310     qapp = Qt.QApplication(sys.argv)
311
312     tb = top_block_cls()
313     tb.start()
314     tb.show()
315
316     # This part has been added from the grc-generated Python file.
317     #-------------------------------------------------------------
318     def measure_and_save_temp():
319
320         i_temp = 0
321         temp_loops = 100
322         temp_file = open('/Users/arvidbjurklint/Google Drive/
    Kandidatarbete-Software defined radio/Experimentdata/
    Temperaturmatning/tempdata.txt', 'w')
323
324         while(i_temp <= temp_loops):
325             temp_file.write(str(tb.uhd_usrp_sink_0.get_sensor('temp')))
```

```
326             temp_file.write('\n')
327             time.sleep(0.3)
328             i_temp += 1
329 
330         # Close amplitude data file and temperature data file.
331         tb.blocks_file_sink_1.close()
332         temp_file.close()
333 
334         print('\n temp sweep done')
335 
336     thread.start_new_thread(measure_and_save_temp, ())
337 
338     # End of addition
339 
340     def quitting():
341         tb.stop()
342         tb.wait()
343     qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
344     qapp.exec_()
345 
346 
347 if __name__ == '__main__':
348     main()
```

Listing C.4: Pythonkod för en modifierad GNU Radio Companion flödeskarta, se 4.1. Utöver att göra det som flödeskartan visar så sparar den temperaturdatan hos kortet. De bitar som har ändrats från den autogenererade koden är utmärkta.

C.1.5 Skicka och ta emot amplituddata för fyra frekvenser för jämförelse med och utan fantom.

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  ##################################################
4  # GNU Radio Python Flow Graph
5  # Title: Send And Receive Amplitude And Phase
6  # Generated: Wed Apr 11 16:21:36 2018
7  ##################################################
8 
9  if __name__ == '__main__':
10     import ctypes
11     import sys
12     if sys.platform.startswith('linux'):
13         try:
14             x11 = ctypes.cdll.LoadLibrary('libX11.so')
15             x11.XInitThreads()
16         except:
17             print "Warning: failed to XInitThreads()"
18 
19 from PyQt4 import Qt
20 from gnuradio import analog
21 from gnuradio import blocks
22 from gnuradio import eng_notation
23 from gnuradio import fft
```

```
24 from gnuradio import gr
25 from gnuradio import qtgui
26 from gnuradio import uhd
27 from gnuradio.eng_option import eng_option
28 from gnuradio.fft import window
29 from gnuradio.filter import firdes
30 from optparse import OptionParser
31 import sip
32 import sys
33 import time
34 import thread
35 from gnuradio import qtgui
36
37
38 class send_and_receive_amplitude_and_phase(gr.top_block, Qt.QWidget):
39
40     def __init__(self):
41         gr.top_block.__init__(self, "Send And Receive Amplitude And
    Phase")
42         Qt.QWidget.__init__(self)
43         self.setWindowTitle("Send And Receive Amplitude And Phase")
44         qtgui.util.check_set_qss()
45         try:
46             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
47         except:
48             pass
49         self.top_scroll_layout = Qt.QVBoxLayout()
50         self.setLayout(self.top_scroll_layout)
51         self.top_scroll = Qt.QScrollArea()
52         self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
53         self.top_scroll_layout.addWidget(self.top_scroll)
54         self.top_scroll.setWidgetResizable(True)
55         self.top_widget = Qt.QWidget()
56         self.top_scroll.setWidget(self.top_widget)
57         self.top_layout = Qt.QVBoxLayout(self.top_widget)
58         self.top_grid_layout = Qt.QGridLayout()
59         self.top_layout.addLayout(self.top_grid_layout)
60
61         self.settings = Qt.QSettings("GNU Radio", "
    send_and_receive_amplitude_and_phase")
62         self.restoreGeometry(self.settings.value("geometry").
    toByteArray())
63
64
65         ##################################################
66         # Variables
67         ##################################################
68         self.samp_rate = samp_rate = 1e6
69         self.fft_width = fft_width = 4096
70         self.f4 = f4 = 0.4e6
71         self.f3 = f3 = 0.3e6
72         self.f2 = f2 = 0.2e6
73         self.f1 = f1 = 0.1e6
74         self.fft_offset_4 = fft_offset_4 = int(round(f4*fft_width/
    samp_rate) + fft_width/2 +1)
75         self.fft_offset_3 = fft_offset_3 = int(round(f3*fft_width/
```

```
    samp_rate) + fft_width/2 +1)
76          self.fft_offset_2 = fft_offset_2 = int(round(f2*fft_width/
    samp_rate) + fft_width/2 +1)
77          self.fft_offset_1 = fft_offset_1 = int(round(f1*fft_width/
    samp_rate) + fft_width/2 +1)
78          self.center_freq = center_freq = 1e9
79          self.send_gain = send_gain = 40
80
81          ##################################################
82          # Blocks
83          ##################################################
84          self.tab = Qt.QTabWidget()
85          self.tab_widget_0 = Qt.QWidget()
86          self.tab_layout_0 = Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom,
    self.tab_widget_0)
87          self.tab_grid_layout_0 = Qt.QGridLayout()
88          self.tab_layout_0.addLayout(self.tab_grid_layout_0)
89          self.tab.addTab(self.tab_widget_0, 'Magnitude')
90          self.tab_widget_1 = Qt.QWidget()
91          self.tab_layout_1 = Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom,
    self.tab_widget_1)
92          self.tab_grid_layout_1 = Qt.QGridLayout()
93          self.tab_layout_1.addLayout(self.tab_grid_layout_1)
94          self.tab.addTab(self.tab_widget_1, 'Phase')
95          self.top_layout.addWidget(self.tab)
96          self.uhd_usrp_source_0 = uhd.usrp_source(
97              ",".join(("", "")),
98              uhd.stream_args(
99                  cpu_format="fc32",
100                 channels=range(1),
101             ),
102         )
103         self.uhd_usrp_source_0.set_samp_rate(samp_rate)
104         self.uhd_usrp_source_0.set_center_freq(center_freq, 0)
105         self.uhd_usrp_source_0.set_gain(0, 0)
106         self.uhd_usrp_source_0.set_antenna('RX2', 0)
107         self.uhd_usrp_sink_0 = uhd.usrp_sink(
108             ",".join(("", "")),
109             uhd.stream_args(
110                 cpu_format="fc32",
111                 channels=range(1),
112             ),
113         )
114         self.uhd_usrp_sink_0.set_clock_source('internal', 0)
115         self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
116         self.uhd_usrp_sink_0.set_center_freq(center_freq, 0)
117         self.uhd_usrp_sink_0.set_gain(send_gain, 0)
118         self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
119         self.qtgui_number_sink_0_0_1 = qtgui.number_sink(
120             gr.sizeof_float,
121             0,
122             qtgui.NUM_GRAPH_HORIZ,
123             1
124         )
125         self.qtgui_number_sink_0_0_1.set_update_time(0.1)
126         self.qtgui_number_sink_0_0_1.set_title('Magnitude of 4th
```

```
    frequency')
127
128         labels = ['', '', '', '', '',
129                   '', '', '', '', '']
130         units = ['', '', '', '', '',
131                  '', '', '', '', '']
132         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
133                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
134         factor = [1, 1, 1, 1, 1,
135                   1, 1, 1, 1, 1]
136         for i in xrange(1):
137             self.qtgui_number_sink_0_0_1.set_min(i, -1)
138             self.qtgui_number_sink_0_0_1.set_max(i, 1)
139             self.qtgui_number_sink_0_0_1.set_color(i, colors[i][0],
    colors[i][1])
140             if len(labels[i]) == 0:
141                 self.qtgui_number_sink_0_0_1.set_label(i, "Data {0}".
    format(i))
142             else:
143                 self.qtgui_number_sink_0_0_1.set_label(i, labels[i])
144             self.qtgui_number_sink_0_0_1.set_unit(i, units[i])
145             self.qtgui_number_sink_0_0_1.set_factor(i, factor[i])
146
147         self.qtgui_number_sink_0_0_1.enable_autoscale(False)
148         self._qtgui_number_sink_0_0_1_win = sip.wrapinstance(self.
    qtgui_number_sink_0_0_1.pyqwidget(), Qt.QWidget)
149         self.tab_layout_0.addWidget(self._qtgui_number_sink_0_0_1_win)
150         self.qtgui_number_sink_0_0_0 = qtgui.number_sink(
151             gr.sizeof_float,
152             0,
153             qtgui.NUM_GRAPH_HORIZ,
154             1
155         )
156         self.qtgui_number_sink_0_0_0.set_update_time(0.1)
157         self.qtgui_number_sink_0_0_0.set_title('Magnitude of 3rd
    frequency')
158
159         labels = ['', '', '', '', '',
160                   '', '', '', '', '']
161         units = ['', '', '', '', '',
162                  '', '', '', '', '']
163         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
164                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
165         factor = [1, 1, 1, 1, 1,
166                   1, 1, 1, 1, 1]
167         for i in xrange(1):
168             self.qtgui_number_sink_0_0_0.set_min(i, -1)
169             self.qtgui_number_sink_0_0_0.set_max(i, 1)
170             self.qtgui_number_sink_0_0_0.set_color(i, colors[i][0],
    colors[i][1])
171             if len(labels[i]) == 0:
172                 self.qtgui_number_sink_0_0_0.set_label(i, "Data {0}".
```

```
    format(i))
173             else:
174                 self.qtgui_number_sink_0_0_0.set_label(i, labels[i])
175             self.qtgui_number_sink_0_0_0.set_unit(i, units[i])
176             self.qtgui_number_sink_0_0_0.set_factor(i, factor[i])
177 
178         self.qtgui_number_sink_0_0_0.enable_autoscale(False)
179         self._qtgui_number_sink_0_0_0_win = sip.wrapinstance(self.
    qtgui_number_sink_0_0_0.pyqwidget(), Qt.QWidget)
180         self.tab_layout_0.addWidget(self._qtgui_number_sink_0_0_0_win)
181         self.qtgui_number_sink_0_0 = qtgui.number_sink(
182             gr.sizeof_float,
183             0,
184             qtgui.NUM_GRAPH_HORIZ,
185             1
186         )
187         self.qtgui_number_sink_0_0.set_update_time(0.1)
188         self.qtgui_number_sink_0_0.set_title('Magnitude of 2nd
    frequency')
189 
190         labels = ['', '', '', '', '',
191                   '', '', '', '', '']
192         units = ['', '', '', '', '',
193                  '', '', '', '', '']
194         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
195                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
196         factor = [1, 1, 1, 1, 1,
197                   1, 1, 1, 1, 1]
198         for i in xrange(1):
199             self.qtgui_number_sink_0_0.set_min(i, -1)
200             self.qtgui_number_sink_0_0.set_max(i, 1)
201             self.qtgui_number_sink_0_0.set_color(i, colors[i][0],
    colors[i][1])
202             if len(labels[i]) == 0:
203                 self.qtgui_number_sink_0_0.set_label(i, "Data {0}".
    format(i))
204             else:
205                 self.qtgui_number_sink_0_0.set_label(i, labels[i])
206             self.qtgui_number_sink_0_0.set_unit(i, units[i])
207             self.qtgui_number_sink_0_0.set_factor(i, factor[i])
208 
209         self.qtgui_number_sink_0_0.enable_autoscale(False)
210         self._qtgui_number_sink_0_0_win = sip.wrapinstance(self.
    qtgui_number_sink_0_0.pyqwidget(), Qt.QWidget)
211         self.tab_layout_0.addWidget(self._qtgui_number_sink_0_0_win)
212         self.qtgui_number_sink_0 = qtgui.number_sink(
213             gr.sizeof_float,
214             0,
215             qtgui.NUM_GRAPH_HORIZ,
216             1
217         )
218         self.qtgui_number_sink_0.set_update_time(0.10)
219         self.qtgui_number_sink_0.set_title('Magnitude of 1st requency')
220 
```

```
221         labels = ['', '', '', '', '',
222                   '', '', '', '', '']
223         units = ['', '', '', '', '',
224                  '', '', '', '', '']
225         colors = [("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black"),
226                   ("black", "black"), ("black", "black"), ("black", "
    black"), ("black", "black"), ("black", "black")]
227         factor = [1, 1, 1, 1, 1,
228                   1, 1, 1, 1, 1]
229         for i in xrange(1):
230             self.qtgui_number_sink_0.set_min(i, -1)
231             self.qtgui_number_sink_0.set_max(i, 1)
232             self.qtgui_number_sink_0.set_color(i, colors[i][0], colors[
    i][1])
233             if len(labels[i]) == 0:
234                 self.qtgui_number_sink_0.set_label(i, "Data {0}".format
    (i))
235             else:
236                 self.qtgui_number_sink_0.set_label(i, labels[i])
237             self.qtgui_number_sink_0.set_unit(i, units[i])
238             self.qtgui_number_sink_0.set_factor(i, factor[i])
239 
240         self.qtgui_number_sink_0.enable_autoscale(False)
241         self._qtgui_number_sink_0_win = sip.wrapinstance(self.
    qtgui_number_sink_0.pyqwidget(), Qt.QWidget)
242         self.tab_layout_0.addWidget(self._qtgui_number_sink_0_win)
243         self.qtgui_freq_sink_x_0_0 = qtgui.freq_sink_c(
244             fft_width, #size
245             firdes.WIN_BLACKMAN_hARRIS, #wintype
246             0, #fc
247             samp_rate, #bw
248             'Skickad signal', #name
249             1 #number of inputs
250         )
251         self.qtgui_freq_sink_x_0_0.set_update_time(0.10)
252         self.qtgui_freq_sink_x_0_0.set_y_axis(-140, 10)
253         self.qtgui_freq_sink_x_0_0.set_y_label('Relative Gain', 'dB')
254         self.qtgui_freq_sink_x_0_0.set_trigger_mode(qtgui.
    TRIG_MODE_FREE, 0.0, 0, "")
255         self.qtgui_freq_sink_x_0_0.enable_autoscale(False)
256         self.qtgui_freq_sink_x_0_0.enable_grid(False)
257         self.qtgui_freq_sink_x_0_0.set_fft_average(1.0)
258         self.qtgui_freq_sink_x_0_0.enable_axis_labels(True)
259         self.qtgui_freq_sink_x_0_0.enable_control_panel(True)
260 
261         if not True:
262           self.qtgui_freq_sink_x_0_0.disable_legend()
263 
264         if "complex" == "float" or "complex" == "msg_float":
265           self.qtgui_freq_sink_x_0_0.set_plot_pos_half(not True)
266 
267         labels = ['', '', '', '', '',
268                   '', '', '', '', '']
269         widths = [1, 1, 1, 1, 1,
270                   1, 1, 1, 1, 1]
```

```
271         colors = ["blue", "red", "green", "black", "cyan",
272                   "magenta", "yellow", "dark red", "dark green", "dark
    blue"]
273         alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
274                   1.0, 1.0, 1.0, 1.0, 1.0]
275         for i in xrange(1):
276             if len(labels[i]) == 0:
277                 self.qtgui_freq_sink_x_0_0.set_line_label(i, "Data {0}"
    .format(i))
278             else:
279                 self.qtgui_freq_sink_x_0_0.set_line_label(i, labels[i])
280             self.qtgui_freq_sink_x_0_0.set_line_width(i, widths[i])
281             self.qtgui_freq_sink_x_0_0.set_line_color(i, colors[i])
282             self.qtgui_freq_sink_x_0_0.set_line_alpha(i, alphas[i])
283 
284         self._qtgui_freq_sink_x_0_0_win = sip.wrapinstance(self.
    qtgui_freq_sink_x_0_0.pyqwidget(), Qt.QWidget)
285         self.top_layout.addWidget(self._qtgui_freq_sink_x_0_0_win)
286         self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
287             fft_width, #size
288             firdes.WIN_BLACKMAN_hARRIS, #wintype
289             0, #fc
290             samp_rate, #bw
291             'Mottagen signal', #name
292             1 #number of inputs
293         )
294         self.qtgui_freq_sink_x_0.set_update_time(0.10)
295         self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
296         self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
297         self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE,
     0.0, 0, "")
298         self.qtgui_freq_sink_x_0.enable_autoscale(False)
299         self.qtgui_freq_sink_x_0.enable_grid(False)
300         self.qtgui_freq_sink_x_0.set_fft_average(1.0)
301         self.qtgui_freq_sink_x_0.enable_axis_labels(True)
302         self.qtgui_freq_sink_x_0.enable_control_panel(True)
303 
304         if not True:
305           self.qtgui_freq_sink_x_0.disable_legend()
306 
307         if "complex" == "float" or "complex" == "msg_float":
308           self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)
309 
310         labels = ['', '', '', '', '',
311                   '', '', '', '', '']
312         widths = [1, 1, 1, 1, 1,
313                   1, 1, 1, 1, 1]
314         colors = ["blue", "red", "green", "black", "cyan",
315                   "magenta", "yellow", "dark red", "dark green", "dark
    blue"]
316         alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
317                   1.0, 1.0, 1.0, 1.0, 1.0]
318         for i in xrange(1):
319             if len(labels[i]) == 0:
320                 self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".
    format(i))
```

```
321             else:
322                 self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
323             self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
324             self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
325             self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
326 
327         self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.
    qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
328         self.top_layout.addWidget(self._qtgui_freq_sink_x_0_win)
329         self.fft_vxx_0 = fft.fft_vcc(fft_width, True, (window.
    blackmanharris(fft_width)), True, 1)
330         self.blocks_vector_to_stream_0 = blocks.vector_to_stream(gr.
    sizeof_gr_complex*1, fft_width)
331         self.blocks_stream_to_vector_0 = blocks.stream_to_vector(gr.
    sizeof_gr_complex*1, fft_width)
332         self.blocks_skiphead_0_0_1 = blocks.skiphead(gr.
    sizeof_gr_complex*1, fft_offset_4)
333         self.blocks_skiphead_0_0_0 = blocks.skiphead(gr.
    sizeof_gr_complex*1, fft_offset_3)
334         self.blocks_skiphead_0_0 = blocks.skiphead(gr.sizeof_gr_complex
    *1, fft_offset_2)
335         self.blocks_skiphead_0 = blocks.skiphead(gr.sizeof_gr_complex
    *1, fft_offset_1)
336         self.blocks_keep_one_in_n_0_0_1 = blocks.keep_one_in_n(gr.
    sizeof_gr_complex*1, fft_width)
337         self.blocks_keep_one_in_n_0_0_0 = blocks.keep_one_in_n(gr.
    sizeof_gr_complex*1, fft_width)
338         self.blocks_keep_one_in_n_0_0 = blocks.keep_one_in_n(gr.
    sizeof_gr_complex*1, fft_width)
339         self.blocks_keep_one_in_n_0 = blocks.keep_one_in_n(gr.
    sizeof_gr_complex*1, fft_width)
340         self.blocks_file_sink_f4fas = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f4fas.txt', False)
341         self.blocks_file_sink_f4fas.set_unbuffered(False)
342         self.blocks_file_sink_f3fas = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f3fas.txt', False)
343         self.blocks_file_sink_f3fas.set_unbuffered(False)
344         self.blocks_file_sink_f2fas = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f2fas.txt', False)
345         self.blocks_file_sink_f2fas.set_unbuffered(False)
346         self.blocks_file_sink_f4amp = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f4amp.txt', False)
347         self.blocks_file_sink_f4amp.set_unbuffered(False)
348         self.blocks_file_sink_f3amp = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f3amp.txt', False)
349         self.blocks_file_sink_f3amp.set_unbuffered(False)
```

```
350         self.blocks_file_sink_f2amp = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f2amp.txt', False)
351         self.blocks_file_sink_f2amp.set_unbuffered(False)
352         self.blocks_file_sink_f1amp = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f1amp.txt', False)
353         self.blocks_file_sink_f1amp.set_unbuffered(False)
354         self.blocks_file_sink_f1fas = blocks.file_sink(gr.sizeof_float
    *1, '/Users/arvidbjurklint/Google Drive/Kandidatarbete-Software
    defined radio/GNUradio-projekt/send and receive amplitude GRC/data/
    f1fas.txt', False)
355         self.blocks_file_sink_f1fas.set_unbuffered(False)
356         self.blocks_complex_to_mag_0_0_1 = blocks.complex_to_mag(1)
357         self.blocks_complex_to_mag_0_0_0 = blocks.complex_to_mag(1)
358         self.blocks_complex_to_mag_0_0 = blocks.complex_to_mag(1)
359         self.blocks_complex_to_mag_0 = blocks.complex_to_mag(1)
360         self.blocks_complex_to_arg_2 = blocks.complex_to_arg(1)
361         self.blocks_complex_to_arg_1 = blocks.complex_to_arg(1)
362         self.blocks_complex_to_arg_0_0 = blocks.complex_to_arg(1)
363         self.blocks_complex_to_arg_0 = blocks.complex_to_arg(1)
364         self.blocks_add_xx_0_1 = blocks.add_vcc(1)
365         self.blocks_add_xx_0_0 = blocks.add_vcc(1)
366         self.blocks_add_xx_0 = blocks.add_vcc(1)
367         self.analog_sig_source_x_0_1_0 = analog.sig_source_c(samp_rate,
     analog.GR_COS_WAVE, f4, 1, 0)
368         self.analog_sig_source_x_0_1 = analog.sig_source_c(samp_rate,
    analog.GR_COS_WAVE, f3, 1, 0)
369         self.analog_sig_source_x_0_0 = analog.sig_source_c(samp_rate,
    analog.GR_COS_WAVE, f2, 1, 0)
370         self.analog_sig_source_x_0 = analog.sig_source_c(samp_rate,
    analog.GR_COS_WAVE, f1, 1, 0)
371
372
373
374         ##################################################
375         # Connections
376         ##################################################
377         self.connect((self.analog_sig_source_x_0, 0), (self.
    blocks_add_xx_0, 0))
378         self.connect((self.analog_sig_source_x_0_0, 0), (self.
    blocks_add_xx_0, 1))
379         self.connect((self.analog_sig_source_x_0_1, 0), (self.
    blocks_add_xx_0_0, 0))
380         self.connect((self.analog_sig_source_x_0_1_0, 0), (self.
    blocks_add_xx_0_0, 1))
381         self.connect((self.blocks_add_xx_0, 0), (self.blocks_add_xx_0_1
    , 0))
382         self.connect((self.blocks_add_xx_0_0, 0), (self.
    blocks_add_xx_0_1, 1))
383         self.connect((self.blocks_add_xx_0_1, 0), (self.
    qtgui_freq_sink_x_0_0, 0))
384         self.connect((self.blocks_add_xx_0_1, 0), (self.uhd_usrp_sink_0
    , 0))
```

```
385         self.connect((self.blocks_complex_to_arg_0, 0), (self.
    blocks_file_sink_f1fas, 0))
386         self.connect((self.blocks_complex_to_arg_0_0, 0), (self.
    blocks_file_sink_f2fas, 0))
387         self.connect((self.blocks_complex_to_arg_1, 0), (self.
    blocks_file_sink_f4fas, 0))
388         self.connect((self.blocks_complex_to_arg_2, 0), (self.
    blocks_file_sink_f3fas, 0))
389         self.connect((self.blocks_complex_to_mag_0, 0), (self.
    blocks_file_sink_f1amp, 0))
390         self.connect((self.blocks_complex_to_mag_0, 0), (self.
    qtgui_number_sink_0, 0))
391         self.connect((self.blocks_complex_to_mag_0_0, 0), (self.
    blocks_file_sink_f2amp, 0))
392         self.connect((self.blocks_complex_to_mag_0_0, 0), (self.
    qtgui_number_sink_0_0, 0))
393         self.connect((self.blocks_complex_to_mag_0_0_0, 0), (self.
    blocks_file_sink_f3amp, 0))
394         self.connect((self.blocks_complex_to_mag_0_0_0, 0), (self.
    qtgui_number_sink_0_0_0, 0))
395         self.connect((self.blocks_complex_to_mag_0_0_1, 0), (self.
    blocks_file_sink_f4amp, 0))
396         self.connect((self.blocks_complex_to_mag_0_0_1, 0), (self.
    qtgui_number_sink_0_0_1, 0))
397         self.connect((self.blocks_keep_one_in_n_0, 0), (self.
    blocks_complex_to_arg_0, 0))
398         self.connect((self.blocks_keep_one_in_n_0, 0), (self.
    blocks_complex_to_mag_0, 0))
399         self.connect((self.blocks_keep_one_in_n_0_0, 0), (self.
    blocks_complex_to_arg_0_0, 0))
400         self.connect((self.blocks_keep_one_in_n_0_0, 0), (self.
    blocks_complex_to_mag_0_0, 0))
401         self.connect((self.blocks_keep_one_in_n_0_0_0, 0), (self.
    blocks_complex_to_arg_2, 0))
402         self.connect((self.blocks_keep_one_in_n_0_0_0, 0), (self.
    blocks_complex_to_mag_0_0_0, 0))
403         self.connect((self.blocks_keep_one_in_n_0_0_1, 0), (self.
    blocks_complex_to_arg_1, 0))
404         self.connect((self.blocks_keep_one_in_n_0_0_1, 0), (self.
    blocks_complex_to_mag_0_0_1, 0))
405         self.connect((self.blocks_skiphead_0, 0), (self.
    blocks_keep_one_in_n_0, 0))
406         self.connect((self.blocks_skiphead_0_0, 0), (self.
    blocks_keep_one_in_n_0_0, 0))
407         self.connect((self.blocks_skiphead_0_0_0, 0), (self.
    blocks_keep_one_in_n_0_0_0, 0))
408         self.connect((self.blocks_skiphead_0_0_1, 0), (self.
    blocks_keep_one_in_n_0_0_1, 0))
409         self.connect((self.blocks_stream_to_vector_0, 0), (self.
    fft_vxx_0, 0))
410         self.connect((self.blocks_vector_to_stream_0, 0), (self.
    blocks_skiphead_0, 0))
411         self.connect((self.blocks_vector_to_stream_0, 0), (self.
    blocks_skiphead_0_0, 0))
412         self.connect((self.blocks_vector_to_stream_0, 0), (self.
    blocks_skiphead_0_0_0, 0))
```

```
413         self.connect((self.blocks_vector_to_stream_0, 0), (self.
    blocks_skiphead_0_0_1, 0))
414         self.connect((self.fft_vxx_0, 0), (self.
    blocks_vector_to_stream_0, 0))
415         self.connect((self.uhd_usrp_source_0, 0), (self.
    blocks_stream_to_vector_0, 0))
416         self.connect((self.uhd_usrp_source_0, 0), (self.
    qtgui_freq_sink_x_0, 0))
417 
418     def closeEvent(self, event):
419         self.settings = Qt.QSettings("GNU Radio", "
    send_and_receive_amplitude_and_phase")
420         self.settings.setValue("geometry", self.saveGeometry())
421         event.accept()
422 
423     def get_samp_rate(self):
424         return self.samp_rate
425 
426     def set_samp_rate(self, samp_rate):
427         self.samp_rate = samp_rate
428         self.set_fft_offset_4(int(round(self.f4*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
429         self.set_fft_offset_3(int(round(self.f3*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
430         self.set_fft_offset_2(int(round(self.f2*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
431         self.set_fft_offset_1(int(round(self.f1*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
432         self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
433         self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
434         self.qtgui_freq_sink_x_0_0.set_frequency_range(0, self.
    samp_rate)
435         self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
436         self.analog_sig_source_x_0_1_0.set_sampling_freq(self.samp_rate
    )
437         self.analog_sig_source_x_0_1.set_sampling_freq(self.samp_rate)
438         self.analog_sig_source_x_0_0.set_sampling_freq(self.samp_rate)
439         self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
440 
441     def get_fft_width(self):
442         return self.fft_width
443 
444     def set_fft_width(self, fft_width):
445         self.fft_width = fft_width
446         self.set_fft_offset_4(int(round(self.f4*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
447         self.set_fft_offset_3(int(round(self.f3*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
448         self.set_fft_offset_2(int(round(self.f2*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
449         self.set_fft_offset_1(int(round(self.f1*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
450         self.blocks_keep_one_in_n_0_0_1.set_n(self.fft_width)
451         self.blocks_keep_one_in_n_0_0_0.set_n(self.fft_width)
452         self.blocks_keep_one_in_n_0_0.set_n(self.fft_width)
453         self.blocks_keep_one_in_n_0.set_n(self.fft_width)
```

```
454
455     def get_f4(self):
456         return self.f4
457
458     def set_f4(self, f4):
459         self.f4 = f4
460         self.set_fft_offset_4(int(round(self.f4*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
461         self.analog_sig_source_x_0_1_0.set_frequency(self.f4)
462
463     def get_f3(self):
464         return self.f3
465
466     def set_f3(self, f3):
467         self.f3 = f3
468         self.set_fft_offset_3(int(round(self.f3*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
469         self.analog_sig_source_x_0_1.set_frequency(self.f3)
470
471     def get_f2(self):
472         return self.f2
473
474     def set_f2(self, f2):
475         self.f2 = f2
476         self.set_fft_offset_2(int(round(self.f2*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
477         self.analog_sig_source_x_0_0.set_frequency(self.f2)
478
479     def get_f1(self):
480         return self.f1
481
482     def set_f1(self, f1):
483         self.f1 = f1
484         self.set_fft_offset_1(int(round(self.f1*self.fft_width/self.
    samp_rate) + self.fft_width/2 +1))
485         self.analog_sig_source_x_0.set_frequency(self.f1)
486
487     def get_fft_offset_4(self):
488         return self.fft_offset_4
489
490     def set_fft_offset_4(self, fft_offset_4):
491         self.fft_offset_4 = fft_offset_4
492
493     def get_fft_offset_3(self):
494         return self.fft_offset_3
495
496     def set_fft_offset_3(self, fft_offset_3):
497         self.fft_offset_3 = fft_offset_3
498
499     def get_fft_offset_2(self):
500         return self.fft_offset_2
501
502     def set_fft_offset_2(self, fft_offset_2):
503         self.fft_offset_2 = fft_offset_2
504
505     def get_fft_offset_1(self):
```

```
506         return self.fft_offset_1
507
508     def set_fft_offset_1(self, fft_offset_1):
509         self.fft_offset_1 = fft_offset_1
510
511     def get_center_freq(self):
512         return self.center_freq
513
514     def set_center_freq(self, center_freq):
515         self.center_freq = center_freq
516         self.uhd_usrp_source_0.set_center_freq(self.center_freq, 0)
517         self.uhd_usrp_sink_0.set_center_freq(self.center_freq, 0)
518
519
520 def main(top_block_cls=send_and_receive_amplitude_and_phase, options=
    None):
521
522     from distutils.version import StrictVersion
523     if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
524         style = gr.prefs().get_string('qtgui', 'style', 'raster')
525         Qt.QApplication.setGraphicsSystem(style)
526     qapp = Qt.QApplication(sys.argv)
527
528     tb = top_block_cls()
529     tb.start()
530     tb.show()
531
532     # This part has been added from the grc-generated Python file.
533     #-------------------------------------------------------------
534     def measure_ref_and_phantom():
535
536         # Variables
537         folderPath = '/Users/arvidbjurklint/Google Drive/Kandidatarbete
    -Software defined radio/Experimentdata/Data April 27/'
538         name_of_run = 'ref_far_2'
539         center_freq = 2e9                    # 1, 1.5, 2
540         tb.set_center_freq(center_freq)
541         file_name_extension = str(int(center_freq))
542         seconds_per_measurement = 10
543
544         # Reference measurement
545         raw_input('Press enter when you want to start measuring the
    reference')
546         print('Starting')
547
548         # Open files
549         tb.blocks_file_sink_f1amp.open(folderPath + 'f1amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
550         tb.blocks_file_sink_f2amp.open(folderPath + 'f2amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
551         tb.blocks_file_sink_f3amp.open(folderPath + 'f3amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
552         tb.blocks_file_sink_f4amp.open(folderPath + 'f4amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
553         tb.blocks_file_sink_f1fas.open(folderPath + 'f1fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
```

```
554         tb.blocks_file_sink_f2fas.open(folderPath + 'f2fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
555         tb.blocks_file_sink_f3fas.open(folderPath + 'f3fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
556         tb.blocks_file_sink_f4fas.open(folderPath + 'f4fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
557
558         time.sleep(seconds_per_measurement)
559
560         # Close files
561         tb.blocks_file_sink_f1amp.close()
562         tb.blocks_file_sink_f2amp.close()
563         tb.blocks_file_sink_f3amp.close()
564         tb.blocks_file_sink_f4amp.close()
565         tb.blocks_file_sink_f1fas.close()
566         tb.blocks_file_sink_f2fas.close()
567         tb.blocks_file_sink_f3fas.close()
568         tb.blocks_file_sink_f4fas.close()
569
570
571         # Phantom measurement
572         # Wait for user input
573         raw_input('Press enter when you are ready for the phantom
    measurement')
574         print('Starting')
575
576         name_of_run = 'phantom_far_2'
577
578         # Open files
579         tb.blocks_file_sink_f1amp.open(folderPath + 'f1amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
580         tb.blocks_file_sink_f2amp.open(folderPath + 'f2amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
581         tb.blocks_file_sink_f3amp.open(folderPath + 'f3amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
582         tb.blocks_file_sink_f4amp.open(folderPath + 'f4amp_' +
    file_name_extension + '_' + name_of_run + '.txt')
583         tb.blocks_file_sink_f1fas.open(folderPath + 'f1fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
584         tb.blocks_file_sink_f2fas.open(folderPath + 'f2fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
585         tb.blocks_file_sink_f3fas.open(folderPath + 'f3fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
586         tb.blocks_file_sink_f4fas.open(folderPath + 'f4fas_' +
    file_name_extension + '_' + name_of_run + '.txt')
587
588         time.sleep(seconds_per_measurement)
589
590         # Close files
591         tb.blocks_file_sink_f1amp.close()
592         tb.blocks_file_sink_f2amp.close()
593         tb.blocks_file_sink_f3amp.close()
594         tb.blocks_file_sink_f4amp.close()
595         tb.blocks_file_sink_f1fas.close()
596         tb.blocks_file_sink_f2fas.close()
597         tb.blocks_file_sink_f3fas.close()
```

```
598             tb.blocks_file_sink_f4fas.close()
599
600             print('Done')
601
602
603
604         thread.start_new_thread(measure_ref_and_phantom, ())
605         # End of addition
606         #-----------------------------------------------------------------
607
608         def quitting():
609             tb.stop()
610             tb.wait()
611         qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
612         qapp.exec_()
613
614
615 if __name__ == '__main__':
616     main()
```

Listing C.5: Pythonkod som skickar ut fyra frekvenser, tar emot dessa och tar ut amplituden för varje frekvens. För varje körning görs detta två gånger, en för mätning utan fantom och en för mätning med.

C.2 C++-kod

C.2.1 Egenskriven C++-kod för datainsamling med ett kort

```
1  // This program is free software: you can redistribute it and/or modify
2  // it under the terms of the GNU General Public License as published by
3  // the Free Software Foundation, either version 3 of the License, or
4  // (at your option) any later version.
5  //
6  // This program is distributed in the hope that it will be useful,
7  // but WITHOUT ANY WARRANTY; without even the implied warranty of
8  // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
9  // GNU General Public License for more details.
10 //
11 // You should have received a copy of the GNU General Public License
12 // along with this program.  If not, see <http://www.gnu.org/licenses
       />.
13 //
14
15 #include <uhd/utils/thread_priority.hpp>
16 #include <uhd/utils/safe_main.hpp>
17 #include <uhd/usrp/multi_usrp.hpp>
18 #include <boost/program_options.hpp>
19 #include <iostream>
20 #include <thread>
21
22 #include "util.hpp"
23
24 namespace po = boost::program_options;
```

```
25
26 /*
    **********************************************************************
27  * Main function
28  **********************************************************************
    */
29 int UHD_SAFE_MAIN(int argc, char *argv[]){
30     uhd::set_thread_priority_safe();
31
32     //transmit variables to be set by po
33     double freq, tx_gain, tx_bw;
34
35     //receive variables to be set by po
36     std::string rx_args, file, type, rx_ant, rx_subdev, rx_channels;
37     size_t total_num_samps, spb;
38     double samp_rate, rx_freq, rx_gain, rx_bw;
39
40     //setup the program options
41     po::options_description desc("Allowed options");
42     desc.add_options()
43         ("help", "help message")
44         ("samp-rate", po::value<double>(&samp_rate), "rate of transmit
    outgoing samples")
45         ("freq", po::value<double>(&freq), "receive RF center frequency
     in Hz")
46         ("tx-gain", po::value<double>(&tx_gain), "gain for the transmit
     RF chain")
47         ("rx-gain", po::value<double>(&rx_gain), "gain for the receive
    RF chain")
48         ("tx-bw", po::value<double>(&tx_bw), "analog transmit filter
    bandwidth in Hz")
49         ("rx-bw", po::value<double>(&rx_bw), "analog receive filter
    bandwidth in Hz")
50     ;
51     po::variables_map vm;
52     po::store(po::parse_command_line(argc, argv, desc), vm);
53     po::notify(vm);
54
55     //print the help message
56     if (vm.count("help")){
57         std::cout << boost::format("UHD TXRX Loopback to File %s") %
    desc << std::endl;
58         return ~0;
59     }
60
61     //create a usrp device
62     uhd::usrp::multi_usrp::sptr usrp_device = uhd::usrp::multi_usrp::
    make(uhd::device_addr_t());
63
64     //Lock mboard clocks
65     usrp_device->set_clock_source("internal");
66
67     usrp_device->set_tx_antenna("TX/RX");
68     // Switch intial antenna config to tx/rx.
69     usrp_device->set_rx_antenna("RX2");
```

```
70
71      std::this_thread::sleep_for(std::chrono::seconds(1));
72
73      std::cout << boost::format("Setting TX Rate: %f Msps...") % (
    samp_rate/1e6) << std::endl;
74      usrp_device->set_tx_rate(samp_rate);
75      std::cout << boost::format("Actual TX Rate: %f Msps...") % (
    usrp_device->get_tx_rate()/1e6) << std::endl << std::endl;
76
77      std::cout << boost::format("Setting RX Rate: %f Msps...") % (
    samp_rate/1e6) << std::endl;
78      usrp_device->set_rx_rate(samp_rate);
79      std::cout << boost::format("Actual RX Rate: %f Msps...") % (
    usrp_device->get_rx_rate()/1e6) << std::endl << std::endl;
80
81      //set the transmit center frequency
82      if (not vm.count("freq")){
83          std::cerr << "Please specify the transmit center frequency with
     --freq" << std::endl;
84          return ~0;
85      }
86
87      std::cout << boost::format("Setting TX Freq: %f MHz...") % (freq/1
    e6) << std::endl;
88      uhd::tune_request_t tx_tune_request(freq);
89      usrp_device->set_tx_freq(tx_tune_request);
90      std::cout << boost::format("Actual TX Freq: %f MHz...") % (
    usrp_device->get_tx_freq()/1e6) << std::endl << std::endl;
91
92      //set the rf gain
93      if (vm.count("tx-gain")){
94          std::cout << boost::format("Setting TX Gain: %f dB...") %
    tx_gain << std::endl;
95          usrp_device->set_tx_gain(tx_gain);
96          std::cout << boost::format("Actual TX Gain: %f dB...") %
    usrp_device->get_tx_gain() << std::endl << std::endl;
97      }
98
99      std::cout << boost::format("Setting RX Freq: %f MHz...") % (freq/1
    e6) << std::endl;
100     uhd::tune_request_t rx_tune_request(freq);
101     usrp_device->set_rx_freq(rx_tune_request);
102     std::cout << boost::format("Actual RX Freq: %f MHz...") % (
    usrp_device->get_rx_freq()/1e6) << std::endl << std::endl;
103
104     //set the receive rf gain
105     if (vm.count("rx-gain")){
106         std::cout << boost::format("Setting RX Gain: %f dB...") %
    rx_gain << std::endl;
107         usrp_device->set_rx_gain(rx_gain);
108         std::cout << boost::format("Actual RX Gain: %f dB...") %
    usrp_device->get_rx_gain() << std::endl << std::endl;
109     }
110
111
112     //Check Ref and LO Lock detect
```

```
113     std::vector<std::string> sensor_names;
114     sensor_names = usrp_device->get_tx_sensor_names(0);
115     if (std::find(sensor_names.begin(), sensor_names.end(), "lo_locked"
        ) != sensor_names.end()) {
116         uhd::sensor_value_t lo_locked = usrp_device->get_tx_sensor("
        lo_locked",0);
117         std::cout << boost::format("Checking lock: %s ...") % lo_locked
        .to_pp_string() << std::endl;
118         UHD_ASSERT_THROW(lo_locked.to_bool());
119     }
120
121     std::cout << "Clock rate is " << usrp_device->get_master_clock_rate
        () << std::endl;
122
123
124     std::cout << "Sleeping for 5 seconds" << std::endl << std::flush;
125     std::this_thread::sleep_for(std::chrono::seconds(5));
126
127     //reset usrp time to prepare for transmit/receive
128     std::cout << boost::format("Setting device timestamp to 0...") <<
        std::endl;
129     usrp_device->set_time_now(uhd::time_spec_t(0.0));
130
131     std::cout << "Press Ctrl + C to stop streaming..." << std::endl;
132
133     //start transmit worker thread
134     boost::thread_group transmit_thread;
135     std::string in_file_path = "infile.bin";
136     std::string out_file_path = "outfile.bin";
137     transmit_thread.create_thread(boost::bind(&send_from_file,
        usrp_device, 3.0f, in_file_path));
138
139     //recv to file
140     recv_to_file(usrp_device, 2.9f, out_file_path);
141
142     //clean up transmit worker
143     stop_signal_called = true;
144     transmit_thread.join_all();
145
146     //finished
147     std::cout << std::endl << "Done!" << std::endl << std::endl << std
        ::flush;
148     return EXIT_SUCCESS;
149 }
```

Listing C.6: one_usrp.cpp, egenskriven C++-kod för datainsamling med ett kort

C.2.2 Egenskriven C++-kod för datainsamling med två kort

```
1 // This program is free software: you can redistribute it and/or modify
2 // it under the terms of the GNU General Public License as published by
3 // the Free Software Foundation, either version 3 of the License, or
4 // (at your option) any later version.
5 //
6 // This program is distributed in the hope that it will be useful,
```

```
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
#include <uhd/utils/thread_priority.hpp>
#include <uhd/utils/safe_main.hpp>
#include <uhd/usrp/multi_usrp.hpp>
#include <boost/program_options.hpp>
#include <iostream>
#include <thread>

#include "util.hpp"

namespace po = boost::program_options;

/*
    **********************************************************************
 * Main function
 **********************************************************************/
int UHD_SAFE_MAIN(int argc, char *argv[]){
    uhd::set_thread_priority_safe();

    //transmit variables to be set by po
    double freq, tx_gain, tx_bw;

    //receive variables to be set by po
    std::string tx_serial, rx_serial;
    size_t total_num_samps, spb;
    double samp_rate, rx_freq, rx_gain, rx_bw;

    //setup the program options
    po::options_description desc("Allowed options");
    desc.add_options()
        ("help", "help message")
        ("samp-rate", po::value<double>(&samp_rate), "rate of transmit outgoing samples")
        ("freq", po::value<double>(&freq), "receive RF center frequency in Hz")
        ("tx-gain", po::value<double>(&tx_gain), "gain for the transmit RF chain")
        ("rx-gain", po::value<double>(&rx_gain), "gain for the receive RF chain")
        ("tx-bw", po::value<double>(&tx_bw), "analog transmit filter bandwidth in Hz")
        ("rx-bw", po::value<double>(&rx_bw), "analog receive filter bandwidth in Hz")
        ("tx-serial", po::value<std::string>(&tx_serial), "tx card serial number (12345678)")
        ("rx-serial", po::value<std::string>(&rx_serial), "rx card serial number (12345678)")
```

```
51      ;
52      po::variables_map vm;
53      po::store(po::parse_command_line(argc, argv, desc), vm);
54      po::notify(vm);
55
56      //print the help message
57      if (vm.count("help")){
58          std::cout << boost::format("UHD TXRX Loopback to File %s") %
    desc << std::endl;
59          return ~0;
60      }
61
62      if (not vm.count("tx-serial")) {
63          std::cerr << "Please specify the tx serial" << std::endl;
64          return ~0;
65      }
66
67      if (not vm.count("rx-serial")) {
68          std::cerr << "Please specify the rx serial" << std::endl;
69          return ~0;
70      }
71
72      // Create tx device
73      uhd::device_addr_t tx_hint;
74      tx_hint["serial"] = tx_serial;
75      uhd::usrp::multi_usrp::sptr tx_usrp_device = uhd::usrp::multi_usrp
    ::make(tx_hint);
76
77      // Create rx device
78      uhd::device_addr_t rx_hint;
79      rx_hint["serial"] = rx_serial;
80      uhd::usrp::multi_usrp::sptr rx_usrp_device = uhd::usrp::multi_usrp
    ::make(rx_hint);
81
82      //Lock mboard clocks
83      tx_usrp_device->set_clock_source("external");
84      rx_usrp_device->set_clock_source("external");
85
86      tx_usrp_device->set_tx_antenna("TX/RX");
87      // Switch intial antenna config to tx/rx.
88      rx_usrp_device->set_rx_antenna("RX2");
89
90      std::this_thread::sleep_for(std::chrono::seconds(1));
91
92      std::cout << boost::format("Setting TX Rate: %f Msps...") % (
    samp_rate/1e6) << std::endl;
93      tx_usrp_device->set_tx_rate(samp_rate);
94      std::cout << boost::format("Actual TX Rate: %f Msps...") % (
    tx_usrp_device->get_tx_rate()/1e6) << std::endl << std::endl;
95
96      std::cout << boost::format("Setting RX Rate: %f Msps...") % (
    samp_rate/1e6) << std::endl;
97      rx_usrp_device->set_rx_rate(samp_rate);
98      std::cout << boost::format("Actual RX Rate: %f Msps...") % (
    rx_usrp_device->get_rx_rate()/1e6) << std::endl << std::endl;
99
```

```
100     //set the transmit center frequency
101     if (not vm.count("freq")){
102         std::cerr << "Please specify the transmit center frequency with
        --freq" << std::endl;
103         return ~0;
104     }
105
106     std::cout << boost::format("Setting TX Freq: %f MHz...") % (freq/1
        e6) << std::endl;
107     uhd::tune_request_t tx_tune_request(freq);
108     tx_usrp_device->set_tx_freq(tx_tune_request);
109     std::cout << boost::format("Actual TX Freq: %f MHz...") % (
        tx_usrp_device->get_tx_freq()/1e6) << std::endl << std::endl;
110
111     //set the rf gain
112     if (vm.count("tx-gain")){
113         std::cout << boost::format("Setting TX Gain: %f dB...") %
        tx_gain << std::endl;
114         tx_usrp_device->set_tx_gain(tx_gain);
115         std::cout << boost::format("Actual TX Gain: %f dB...") %
        tx_usrp_device->get_tx_gain() << std::endl << std::endl;
116     }
117
118     std::cout << boost::format("Setting RX Freq: %f MHz...") % (freq/1
        e6) << std::endl;
119     uhd::tune_request_t rx_tune_request(freq);
120     rx_usrp_device->set_rx_freq(rx_tune_request);
121     std::cout << boost::format("Actual RX Freq: %f MHz...") % (
        rx_usrp_device->get_rx_freq()/1e6) << std::endl << std::endl;
122
123     //set the receive rf gain
124     if (vm.count("rx-gain")){
125         std::cout << boost::format("Setting RX Gain: %f dB...") %
        rx_gain << std::endl;
126         rx_usrp_device->set_rx_gain(rx_gain);
127         std::cout << boost::format("Actual RX Gain: %f dB...") %
        rx_usrp_device->get_rx_gain() << std::endl << std::endl;
128     }
129
130     //Check Ref and LO Lock detect
131     std::vector<std::string> sensor_names;
132     // TODO: Add rx lock detection
133     sensor_names = tx_usrp_device->get_tx_sensor_names(0);
134     if (std::find(sensor_names.begin(), sensor_names.end(), "lo_locked"
        ) != sensor_names.end()) {
135         uhd::sensor_value_t lo_locked = tx_usrp_device->get_tx_sensor("
        lo_locked", 0);
136         std::cout << boost::format("Checking lock: %s ...") % lo_locked
        .to_pp_string() << std::endl;
137         UHD_ASSERT_THROW(lo_locked.to_bool());
138     }
139
140     std::cout << "Clock rate is " << tx_usrp_device->
        get_master_clock_rate() << std::endl;
141
142     std::cout << "Sleeping for 5 seconds" << std::endl << std::flush;
```

```
143     std::this_thread::sleep_for(std::chrono::seconds(5));
144
145     //reset usrp time to prepare for transmit/receive
146     // TODO: Add lag compensation.
147     std::cout << boost::format("Setting device timestamp to 0...") <<
        std::endl;
148     tx_usrp_device->set_time_now(uhd::time_spec_t(0.0));
149     rx_usrp_device->set_time_now(uhd::time_spec_t(0.0));
150
151     std::cout << "Press Ctrl + C to stop streaming..." << std::endl;
152
153     //start transmit worker thread
154     boost::thread_group transmit_thread;
155     std::string in_file_path = "infile.bin";
156     std::string out_file_path = "outfile.bin";
157     transmit_thread.create_thread(boost::bind(&send_from_file,
        tx_usrp_device, 4.0f, in_file_path));
158
159     boost::thread_group switch_thread;
160     switch_thread.create_thread(boost::bind(&switch_antenna,
        rx_usrp_device, 5.0f));
161
162     //recv to file
163     recv_to_file(rx_usrp_device, 3.0f, out_file_path);
164
165     //clean up transmit worker
166     stop_signal_called = true;
167     transmit_thread.join_all();
168
169     //finished
170     std::cout << std::endl << "Done!" << std::endl << std::endl;
171     return EXIT_SUCCESS;
172 }
```

Listing C.7: two_usrp.cpp, egenskriven C++-kod för datainsamling med två kort

C.2.3 Egenskrivna hjälpfunktioner

```
1  static bool stop_signal_called = false;
2
3  void init_matrix();
4
5  void send_scpi(std::string command_string);
6
7  void switch_matrix(int port1, int port2, float delay);
8
9  void switch_antenna(uhd::usrp::multi_usrp::sptr usrp_device, float time
       );
10
11 void send_from_file(uhd::usrp::multi_usrp::sptr usrp_device, double
       start_time, const std::string &file);
12
13 void recv_to_file(uhd::usrp::multi_usrp::sptr usrp_device, double
       start_time, const std::string &file);
```

Listing C.8: util.hpp, egenskrivna hjälpfunktioner

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.
//

#include <uhd/types/tune_request.hpp>
#include <uhd/utils/thread_priority.hpp>
#include <uhd/utils/safe_main.hpp>
#include <uhd/utils/static.hpp>
#include <uhd/usrp/multi_usrp.hpp>
#include <uhd/exception.hpp>
#include <boost/thread/thread.hpp>
#include <boost/program_options.hpp>
#include <boost/math/special_functions/round.hpp>
#include <boost/format.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/algorithm/string.hpp>
#include <algorithm>    // std::fill
#include <vector>       // std::vector
#include <boost/filesystem.hpp>
#include <iostream>
#include <fstream>
#include <thread>
#include <chrono>
#include <csignal>

#include "util.hpp"
#include <hidapi.h>

#include <stdio.h>
#include <stdlib.h>

void switch_antenna(uhd::usrp::multi_usrp::sptr usrp_device, float time) {
    std::this_thread::sleep_for(std::chrono::microseconds((long) time*1000000));
    //usrp_device->set_command_time(uhd::time_spec_t(time));
    usrp_device->set_rx_antenna("TX/RX");
    //usrp_device->clear_command_time();
}

int res;
unsigned char buffer[64];
hid_device *handle;
#define MAX_STR 255
wchar_t wstr[MAX_STR];
```

```
54
55 void init_matrix() {
56     handle = hid_open(0x20ce, 0x0022, NULL);
57
58     // Read the Manufacturer String
59   res = hid_get_manufacturer_string(handle, wstr, MAX_STR);
60   wprintf(L"Manufacturer String: %s\n", wstr);
61
62   // Read the Product String
63   res = hid_get_product_string(handle, wstr, MAX_STR);
64   wprintf(L"Product String: %s\n", wstr);
65
66   // Read the Serial Number String
67   res = hid_get_serial_number_string(handle, wstr, MAX_STR);
68   wprintf(L"Serial Number String: (%d) %s\n", wstr[0], wstr);
69 }
70
71 void send_scpi(std::string command_string) {
72     buffer[0] = 1;
73     size_t string_length = command_string.length();
74     for (int i = 0; i < string_length; i++) {
75         buffer[i+1] = command_string[i];
76     }
77     std::cout << "Writing buffer: " << buffer << std::endl;
78     res = hid_write(handle, buffer, string_length+1);
79 }
80
81 void switch_matrix(int port1, int port2, float delay) {
82     std::this_thread::sleep_for(std::chrono::microseconds((long) delay
    *1000000));
83     // SCPI interrupt code
84     std::string command_string = boost::str(boost::format(":PATH:A1:N%i
    ") % port1);
85     send_scpi(command_string);
86
87     command_string = boost::str(boost::format(":PATH:A2:N%i") % port2);
88     send_scpi(command_string);
89 }
90
91 void send_from_file(
92     uhd::usrp::multi_usrp::sptr usrp_device,
93     double start_time,
94     const std::string &file
95 ){
96     // Samples per buffer.
97     int spb = 5000;
98     uhd::tx_metadata_t md;
99
100    md.start_of_burst = true;
101    md.end_of_burst = false;
102
103    // Start stream at given time, compensate with zero padding
    duration.
104    // spb (samples) / tx_rate (samples/second) = time compensation (
    seconds).
105    md.has_time_spec = true;
```

```
106     uhd::time_spec_t tspec(start_time - ((double) spb)/((double)
    usrp_device->get_tx_rate()));
107     md.time_spec = tspec;
108 
109     // Zeropadding buffer.
110     std::vector<std::complex<float> > zeropadding(spb);
111     for (int i = 0; i < spb; i++) {
112         zeropadding.push_back(std::complex<float>(0.0, 0.0));
113     }
114 
115     uhd::stream_args_t stream_args("fc32", "sc16");
116 
117     std::vector<size_t> channel_nums;
118     channel_nums.push_back(0);
119 
120     stream_args.channels = channel_nums;
121     uhd::tx_streamer::sptr tx_stream = usrp_device->get_tx_stream(
    stream_args);
122 
123     int num = 20;
124     std::cout << "Sending initial zero padding at time " << (start_time
    - ((double) spb)/((double) usrp_device->get_tx_rate())) << "s." <<
     std::endl;
125     for (int i = 0; i < num; i++) {
126         tx_stream->send(&zeropadding.front(), spb, md, 0.1);
127         md.has_time_spec = false;
128     }
129 
130     // TODO: Implement memory caching of input file.
131     std::ifstream infile(file.c_str(), std::ifstream::binary);
132     int num_iter = 10;
133     for (int file_iter = 0; file_iter < num_iter; file_iter++) {
134 
135         std::vector<std::complex<float> > buff(spb);
136         bool eof = false;
137 
138         while(not eof){
139 
140             infile.read((char*)&buff.front(), buff.size()*sizeof(std::
    complex<float>));
141             size_t num_tx_samps = size_t(infile.gcount()/sizeof(std::
    complex<float>));
142             md.end_of_burst = infile.eof() and file_iter == num_iter -
    1;
143             eof = infile.eof();
144             tx_stream->send(&buff.front(), num_tx_samps, md, 30);
145 
146         }
147 
148         infile.clear();
149         infile.seekg(0, std::ios::beg);
150     }
151 
152     infile.close();
153     stop_signal_called = true;
154 }
```

```
155
156 void recv_to_file(
157     uhd::usrp::multi_usrp::sptr usrp_device,
158     double start_time,
159     const std::string &file
160 ){
161     unsigned long long num_total_samps = 0;
162     int spb = 10000;
163     //create a receive streamer
164     uhd::stream_args_t stream_args("fc32", "sc16");
165
166     std::vector<size_t> channel_nums;
167     channel_nums.push_back(boost::lexical_cast<size_t>(0));
168     stream_args.channels = channel_nums;
169     uhd::rx_streamer::sptr rx_stream = usrp_device->get_rx_stream(
    stream_args);
170
171     // Prepare buffers for received samples and metadata
172     uhd::rx_metadata_t md;
173     std::vector<std::complex<float> > buff(spb);
174     std::ofstream outfile;
175     outfile.open(file.c_str(), std::ofstream::binary);
176
177     bool overflow_message = true;
178     bool first_message = true;
179     float timeout = start_time + 0.3f; //expected settling time +
    padding for first recv
180
181     //setup streaming
182     uhd::stream_cmd_t stream_cmd(uhd::stream_cmd_t::
    STREAM_MODE_START_CONTINUOUS);
183     stream_cmd.num_samps = 0;
184     stream_cmd.stream_now = false;
185     stream_cmd.time_spec = uhd::time_spec_t(start_time);
186
187     rx_stream->issue_stream_cmd(stream_cmd);
188     std::cout << "Starting receiver at " << start_time << " s." << std
    ::endl;
189
190     while(not stop_signal_called){
191         // blocking
192         size_t num_rx_samps = rx_stream->recv(&buff.front(), buff.size
    (), md, timeout);
193         timeout = 0.1f; //small timeout for subsequent recv
194
195         if (md.error_code == uhd::rx_metadata_t::ERROR_CODE_TIMEOUT) {
196             std::cout << boost::format("Timeout while streaming") <<
    std::endl;
197             break;
198         }
199
200         if (md.error_code == uhd::rx_metadata_t::ERROR_CODE_OVERFLOW){
201             if (overflow_message){
202                 overflow_message = false;
203                 std::cerr << boost::format(
204                     "Got an overflow indication. Please consider the
```

```
    following:\n"
205                     "  Your write medium must sustain a rate of %fMB/s
    .\n"
206                     "  Dropped samples will not be written to the file
    .\n"
207                     "  Please modify this example for your purposes.\n"
208                     "  This message will not appear again.\n"
209                 ) % (usrp_device->get_rx_rate()*sizeof(std::complex<
    float>)/1e6);
210             }
211             continue;
212         }
213         if (md.error_code != uhd::rx_metadata_t::ERROR_CODE_NONE){
214             throw std::runtime_error(str(boost::format(
215                 "Receiver error %s"
216             ) % md.strerror()));
217         }
218
219         num_total_samps += num_rx_samps;
220
221         if (outfile.is_open()) {
222             outfile.write((const char*) &buff.front(), num_rx_samps*
    sizeof(std::complex<float>));
223         } else {
224             std::cout << "outfile not open!";
225         }
226
227     }
228     // Shut down receiver
229     stream_cmd.stream_mode = uhd::stream_cmd_t::
    STREAM_MODE_STOP_CONTINUOUS;
230     rx_stream->issue_stream_cmd(stream_cmd);
231
232     std::cout << std::flush;
233
234     // Close file
235     outfile.close();
236 }
```

Listing C.9: util.cpp, egenskrivna hjälpfunktioner

C.2.4 Egenskriven C++-kod för datainsamling med switch-matris

```
1  // This program is free software: you can redistribute it and/or modify
2  // it under the terms of the GNU General Public License as published by
3  // the Free Software Foundation, either version 3 of the License, or
4  // (at your option) any later version.
5  //
6  // This program is distributed in the hope that it will be useful,
7  // but WITHOUT ANY WARRANTY; without even the implied warranty of
8  // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
9  // GNU General Public License for more details.
10 //
11 // You should have received a copy of the GNU General Public License
```

```
12 // along with this program.  If not, see <http://www.gnu.org/licenses
       />.
13 //
14
15 #include <uhd/utils/thread_priority.hpp>
16 #include <uhd/utils/safe_main.hpp>
17 #include <uhd/usrp/multi_usrp.hpp>
18 #include <boost/program_options.hpp>
19 #include <iostream>
20 #include <thread>
21
22 #include "util.hpp"
23
24 namespace po = boost::program_options;
25
26 /*
       ***********************************************************************
27  * Main function
28  ***********************************************************************
       */
29 int UHD_SAFE_MAIN(int argc, char *argv[]){
30     uhd::set_thread_priority_safe();
31
32     //transmit variables to be set by po
33     double start_freq, end_freq, tx_gain, tx_bw, step;
34
35     //receive variables to be set by po
36     std::string rx_args, file, type, rx_ant, rx_subdev, rx_channels;
37     size_t total_num_samps, spb;
38     double samp_rate, rx_freq, rx_gain, rx_bw;
39
40     //setup the program options
41     po::options_description desc("Allowed options");
42     desc.add_options()
43         ("help", "help message")
44         ("samp-rate", po::value<double>(&samp_rate), "rate of transmit
       outgoing samples")
45         ("start-freq", po::value<double>(&start_freq), "starting center
        frequency in Hz")
46         ("end-freq", po::value<double>(&start_freq), "ending center
       frequency in Hz")
47         ("step", po::value<double>(&step), "frequency step for sweep in
        Hz")
48         ("tx-gain", po::value<double>(&tx_gain), "gain for the transmit
        RF chain")
49         ("rx-gain", po::value<double>(&rx_gain), "gain for the receive
       RF chain")
50         ("tx-bw", po::value<double>(&tx_bw), "analog transmit filter
       bandwidth in Hz")
51         ("rx-bw", po::value<double>(&rx_bw), "analog receive filter
       bandwidth in Hz")
52     ;
53     po::variables_map vm;
54     po::store(po::parse_command_line(argc, argv, desc), vm);
55     po::notify(vm);
```

```
56
57      //print the help message
58      if (vm.count("help")){
59          std::cout << boost::format("UHD TXRX Loopback to File %s") %
        desc << std::endl;
60          return ~0;
61      }
62
63      //create a usrp device
64      uhd::usrp::multi_usrp::sptr usrp_device = uhd::usrp::multi_usrp::
        make(uhd::device_addr_t());
65
66      //Lock mboard clocks
67      usrp_device->set_clock_source("internal");
68
69      usrp_device->set_tx_antenna("TX/RX");
70      // Switch intial antenna config to tx/rx.
71      usrp_device->set_rx_antenna("RX2");
72
73      std::this_thread::sleep_for(std::chrono::seconds(1));
74
75      std::cout << boost::format("Setting TX Rate: %f Msps...") % (
        samp_rate/1e6) << std::endl;
76      usrp_device->set_tx_rate(samp_rate);
77      std::cout << boost::format("Actual TX Rate: %f Msps...") % (
        usrp_device->get_tx_rate()/1e6) << std::endl << std::endl;
78
79      std::cout << boost::format("Setting RX Rate: %f Msps...") % (
        samp_rate/1e6) << std::endl;
80      usrp_device->set_rx_rate(samp_rate);
81      std::cout << boost::format("Actual RX Rate: %f Msps...") % (
        usrp_device->get_rx_rate()/1e6) << std::endl << std::endl;
82
83      //set the transmit center frequency
84      if (not vm.count("start-freq")){
85          std::cerr << "Please specify the transmit starting center
        frequency with --start-freq" << std::endl;
86          return ~0;
87      }
88      if (not vm.count("end-freq")){
89          std::cerr << "Please specify the transmit ending center
        frequency with --end-freq" << std::endl;
90          return ~0;
91      }
92
93      //set the receive rf gain
94      if (vm.count("rx-gain")){
95          std::cout << boost::format("Setting RX Gain: %f dB...") %
        rx_gain << std::endl;
96          usrp_device->set_rx_gain(rx_gain);
97          std::cout << boost::format("Actual RX Gain: %f dB...") %
        usrp_device->get_rx_gain() << std::endl << std::endl;
98      }
99
100     //set the rf gain
101     if (vm.count("tx-gain")){
```

```
102         std::cout << boost::format("Setting TX Gain: %f dB...") %
    tx_gain << std::endl;
103         usrp_device->set_tx_gain(tx_gain);
104         std::cout << boost::format("Actual TX Gain: %f dB...") %
    usrp_device->get_tx_gain() << std::endl << std::endl;
105     }
106
107     boost::thread_group transmit_thread;
108     boost::thread_group switch_thread;
109     std::string in_file_path;
110     std::string out_file_path;
111
112     int loopback_1 = 1;
113     int loopback_2 = 2;
114
115     int test_1 = 4;
116     int test_2 = 5;
117
118     //sweep through the frequencies
119     for(double freq = start_freq; freq <= end_freq; freq += step){
120
121         std::cout << "Resetting switch" << std::endl;
122         switch_matrix(loopback_1, loopback_2, 0.0f);
123
124         std::cout << boost::format("Setting TX Freq: %f MHz...") % (
    freq/1e6) << std::endl;
125         uhd::tune_request_t tx_tune_request(freq);
126         usrp_device->set_tx_freq(tx_tune_request);
127         std::cout << boost::format("Actual TX Freq: %f MHz...") % (
    usrp_device->get_tx_freq()/1e6) << std::endl << std::endl;
128
129         std::cout << boost::format("Setting RX Freq: %f MHz...") % (
    freq/1e6) << std::endl;
130         uhd::tune_request_t rx_tune_request(freq);
131         usrp_device->set_rx_freq(rx_tune_request);
132         std::cout << boost::format("Actual RX Freq: %f MHz...") % (
    usrp_device->get_rx_freq()/1e6) << std::endl << std::endl;
133
134         //Check Ref and LO Lock detect
135         std::vector<std::string> sensor_names;
136         sensor_names = usrp_device->get_tx_sensor_names(0);
137         if (std::find(sensor_names.begin(), sensor_names.end(), "
    lo_locked") != sensor_names.end()) {
138             uhd::sensor_value_t lo_locked = usrp_device->get_tx_sensor(
    "lo_locked",0);
139             std::cout << boost::format("Checking lock: %s ...") %
    lo_locked.to_pp_string() << std::endl;
140             UHD_ASSERT_THROW(lo_locked.to_bool());
141         }
142
143         std::cout << "Clock rate is " << usrp_device->
    get_master_clock_rate() << std::endl;
144
145         std::cout << "Sleeping for 1 seconds" << std::endl << std::
    flush;
146         std::this_thread::sleep_for(std::chrono::seconds(1));
```

```
147
148         //reset usrp time to prepare for transmit/receive
149         std::cout << boost::format("Setting device timestamp to 0...")
    << std::endl;
150         usrp_device->set_time_now(uhd::time_spec_t(0.0));
151
152         std::cout << "Press Ctrl + C to stop streaming..." << std::endl
    ;
153
154         //start transmit worker thread
155         in_file_path = "infile.bin";
156         out_file_path = boost::str(boost::format("sweep-measurements/
    outfile%.1f.bin")%freq);
157         transmit_thread.create_thread(boost::bind(&send_from_file,
    usrp_device, 3.0f, in_file_path));
158
159         switch_thread.create_thread(boost::bind(&switch_matrix, test_1,
     test_2, 4.0f));
160
161         //recv to file
162         recv_to_file(usrp_device, 2.9f, out_file_path);
163     }
164
165     //clean up transmit worker
166     stop_signal_called = true;
167     transmit_thread.join_all();
168
169     //finished
170     std::cout << std::endl << "Done!" << std::endl << std::endl << std
    ::flush;
171     return EXIT_SUCCESS;
172 }
```

Listing C.10: switch_usrp.cpp, egenskriven C++-kod för datainsamling med switchmatris

C.2.5 Egenskriven C++-kod för att skicka SCPI-kommandon till switch

```
1  // This program is free software: you can redistribute it and/or modify
2  // it under the terms of the GNU General Public License as published by
3  // the Free Software Foundation, either version 3 of the License, or
4  // (at your option) any later version.
5  //
6  // This program is distributed in the hope that it will be useful,
7  // but WITHOUT ANY WARRANTY; without even the implied warranty of
8  // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
9  // GNU General Public License for more details.
10 //
11 // You should have received a copy of the GNU General Public License
12 // along with this program.  If not, see <http://www.gnu.org/licenses
       />.
13 //
14
15 #include <uhd/utils/thread_priority.hpp>
```

```
16 #include <uhd/utils/safe_main.hpp>
17 #include <uhd/usrp/multi_usrp.hpp>
18 #include <boost/program_options.hpp>
19 #include <iostream>
20 #include <thread>
21
22 #include "util.hpp"
23
24 namespace po = boost::program_options;
25
26 /*
    **********************************************************************
27  * Main function
28  **********************************************************************
    */
29 int UHD_SAFE_MAIN(int argc, char *argv[]){
30     std::cout << "Initilizing switch matrix" << std::endl << std::flush
    ;
31     init_matrix();
32     std::string command_string = (std::string) argv[1];
33     std::cout << "Sending scpi command: " << command_string << std::
    endl;
34     send_scpi(command_string);
35     std::cout << std::endl << "Done!" << std::endl << std::endl << std
    ::flush;
36     return EXIT_SUCCESS;
37 }
```

Listing C.11: switch_scpi.cpp, Egenskriven C++-kod för att skicka SCPI-kommandon till switch