REAL-TIME FPGA IMPLEMENTATION OF NONLINEAR SELF-INTERFERENCE CANCELLATION IN FULL-DUPLEX RADIO TRANSCEIVER

Master of Science Thesis

Examiners: professor Mikko Valkama, MSc Dani Korpi

Examiner and topic approved: Faculty Council of the Faculty of Computing and Electrical Engineering on 7th of December 2016
ABSTRACT

MAUNO PIILILÄ: Real-time FPGA implementation of nonlinear self-interference cancellation in full-duplex radio transceiver
Tampere University of Technology
Master of Science Thesis, 54 pages, 0 Appendix pages
December 2016
Master’s Degree Programme in Information Technology
Major: Degree Programme in Information Technology, MSc (Tech)
Examiners: professor Mikko Valkama, MSc Dani Korpi

Keywords: digital, full-duplex, FPGA, cancellation, real-time

In full-duplex wireless communications, transmission and reception of signals occur simultaneously on the same center carrier frequency. It is one technique to increase limited wireless capacity, and particularly the spectral efficiency. The challenge of full-duplex technique is to separate the weak received signal from the transmitted signal, which can be even 120 dB stronger signal. This is called self-interference. Current duplexing techniques used in existing two-way communication systems are either time or frequency division duplexing, where signals received and transmitted utilizes different carrier frequencies, or transmission and reception are done in different time slots. The usage of full-duplex communications may, in theory, double the spectral efficiency of a wireless link and thereon the corresponding wireless system. To cancel the self-interference, and to recover the received signal, the self-interference has to be modelled accurately. Particularly important is the ability to model the nonlinear nature of the self-interference, that is caused by power amplifiers.

In this Master’s Thesis work, one recently developed nonlinear self-interference model and corresponding nonlinear digital self-interference cancellation technique is explored. Specifically, a real-time FPGA implementation is pursued and developed for 20 MHz LTE-like channel bandwidth. This real-time Labview FPGA solution of digital self-interference canceller can, of the overall three-phase self-interference suppression solution, provide measured cancellation at maximum of about 34 dB. The other self-interference cancellation or isolation methods used in measurements are a passive circulator with 20 dB attenuation, and an active RF-canceller, that at measurements provided about 43 dB cancellation. Thus the total real-time cancellation of the self-interference in the overall demonstration system is maximally nearly 100 dB. The developed real-time FPGA implementation could possibly be used as a base for an ASIC circuit development. The resources available in the FPGA used, affects essentially into the structure of the canceller and to the performance of the cancellation.
MAUNO PIILILÄ: Oman lähettimen aiheuttaman häiriön digitaalinen vaimennus full-duplex radiolaitteessa: toimintaperiaate ja reaaliaikainen FPGA toteutus
Tampereen teknillinen yliopisto
Diplomityö, 54 sivua, 0 liitesivua
Joulukuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Tietoliikenteen digitaaliteknikka
Tarkastajat: professori Mikko Valkama, DI Dani Korpi
Avainsanat: Digitaalinen, Full-Duplex, FPGA, vaimennus, reaaliaikainen


Tässä diplomityössä on tutkittu hiljattain kehitettyä epälineaarista itseishäiriön mallia ja vastaavaa epälineaarista digitaalista itseishäiriön kumoamistekniikkaa. Erityisesti työ keskittyy reaaliaikaisen FPGA toteutuksen kehittämiseen itseishäiriön kumoamiseksi, ja tuloksena on kehitetty 20 Mhz kaistanleveyksisellä LTE signaalin kaltaisella kanavalla toimiva toteutus. Digitaalinen itseishäiriön reaaliaikainen Labview FPGA toteutus pystyy, viimeisenä osana kolmivaiheista itseishäiriön vaimennusta, mitatusti vaimentamaan häiriötä parhaimmillaan n. 34 dB. Mittauskerralla käytetyt muut vaimentimet ovat 20 dB vaimentava passiivinen sirkulaaattori ja aktiivinen RF-vaimentaja, joka mittauskerralla vaimensi n. 43 dB. Eli yhteensä reaaliaikainen konaisvaimennus on parhaimmillaan himman alle 100 dB. Kehitetty reaaliaikaisista FPGA toteutusta voi mahdollisesti käyttää pohjana vastaavan ASIC piirin suunnittelussa. Käytössä olleen FPGA piirin käytössä oleva tila vaikutti olennaisesti toteutuksen rakenteeseen ja vaimentajan kokoon sekä tehokkuuteen.
PREFACE

I started this project in fall 2015 and started writing process after the program was working in October 2016.

I would like to thank Mikko Valkama for an interesting project. Learning completely new kind of graphical programming language was interesting, but also very demanding for a person that is used to textual programming.

Also I would like to thank Dani Korpi for the guidance into digital cancelling and helping, especially, with the measurement results. Joose Tamminen and Matias Turunen deserves also some credit for building measurement setup and for some good ideas and points. My brother Reima Piililä has been a great help when spell checking the English, and forming the final structure of this document. Also acknowledgements for Mikko and Dani for the same reasons.

Special thank-you also to Markku Renfors, who directed me to Mikko. Without his knowledge about this project, I probably would not have been part of it.

Tampere, August 7, 2017

Mauno Piililä
# TABLE OF CONTENTS

1. INTRODUCTION........................................................................................................1
2. FULL-DUPLEX RADIO FUNDAMENTALS..................................................................3
   2.1 Full-Duplex Transceiver......................................................................................3
   2.2 Major Source of Nonlinear Distortion.................................................................4
   2.3 RF-Canceller........................................................................................................5
   2.4 Digital SI Cancellation..........................................................................................6
3. DSIC ARCHITECTURAL DETAILS AND DEVELOPMENT ENVIRONMENT..............7
   3.1 Digital SI Cancelling Algorithm........................................................................7
   3.2 Delay Search.......................................................................................................12
   3.3 Calculating Signal Power....................................................................................12
   3.4 Introduction to Labview Programming...............................................................14
      3.4.1 Virtual Instruments......................................................................................14
      3.4.2 A Normal Labview Example Program.........................................................15
      3.4.3 FPGA Version of the Sample Program.......................................................17
   3.5 Test Benches and Debugging.............................................................................19
4. DEVELOPED REAL-TIME IMPLEMENTATION OF DSIC.................................21
   4.1 Equipment Used to Implement the Real-Time DSIC..........................................21
   4.2 Transceiver Template Used..............................................................................22
   4.3 Implementation of the Digital Self-Interference Canceller...............................24
      4.3.1 Transceiver Functioning Logic......................................................................24
      4.3.2 Self-Interference Canceller Functioning in Broadly.................................24
      4.3.3 The Functioning of the Real-Time DSIC..................................................26
      4.3.4 Side Logic to the Real-Time DSIC.........................................................27
      4.3.5 The DSIC Core Block Diagrams...............................................................31
5. MEASUREMENTS AND RESULTS..........................................................................42
   5.1 Measurement Setup.............................................................................................42
   5.2 Using the Program...............................................................................................43
   5.3 Other Program Properties..................................................................................46
   5.4 Direct Cable Measurements...............................................................................47
   5.5 Measurements with RF Canceller and Antenna................................................48
6. CONCLUSIONS.........................................................................................................53
LIST OF FIGURES

Figure 1: Transmitter and receiver chains with SI cancellation..............................4

Figure 2: FET drain current in relation to gate voltage in black, blue is the linear/ideal case.................................................................................................................................5

Figure 3: Digital self-interference canceller structure................................................6

Figure 4: Samples of a control cluster on the left and an indicator cluster on the right, both consisting different types of controls and indicators. Context help shown describes the types of different controls of the control cluster.................................................................14

Figure 5: Front panel and block diagram of a mean calculation...............................16

Figure 6: A sum function front panel and block diagram.........................................16

Figure 7: FPGA compatible efficient sum function..................................................18

Figure 8: Typical test bench.......................................................................................19

Figure 9: Functioning of the canceller. Filtering SI and calculation of new coefficients. .................................................................................................................................25

Figure 10: The circuit for the reset signal and for the FIFO fetcher between FPGAs in the left side of the FPGA_canceller_MAIN.vi block diagram........................................28

Figure 11: Connections in and out of the DSIC in the right side of the FPGA_canceller_MAIN.vi........................................................................................................................29

Figure 12: The filter_base.vi includes the block to start the DSIC, the DSIC block and input and output power calculation blocks........................................................................29

Figure 13: The block diagram of activate_after_count.vi, that will trigger the cancellation process, when input stream is synchronized properly........................................30

Figure 14: The average power calculation as an IP Builder block..............................31

Figure 15: The SI cancellation core VI filter_core.vi, that does the SI cancellation.....32

Figure 16: The state_cycle.vi handles correct states for the output calculation and for the filter coefficient updater................................................................................33

Figure 17: Block diagram read_memory.vi, that handles the reading of the nonlinear basis functions from the block memory..............................................................33
Figure 18: The case structure for other states "1", "2" and "3".

Figure 19: The read_bases.vi inserts basis functions coming from the memory into an array to be used in calculations.

Figure 20: Convert BRAM element in format U64 into complex fixed-point number.

Figure 21: The read_Rx.vi reads the samples arriving from the receiver.

Figure 22: CANCEL SI block diagram calculate_output.vi.

Figure 23: Block of vector_multiply_sum_4x.vi to calculate sum of vector sums calculated in MULT + SUM block.

Figure 24: Multiplying and adding the results together in vector_multiply_sum.vi. It is MULT + SUM in previous figure.

Figure 25: Block diagram to fetch values from vectors at the position pointed by the signal "index" in single_multiply.vi.

Figure 26: Add pairs from array in add_array_pairs.vi.

Figure 27: Block diagram update_filter.vi to update filter coefficients.

Figure 28: The block diagram for update_taps.vi.

Figure 29: Calculating new value for one tap in update_tap.vi.

Figure 30: Measurement setup.

Figure 31: Adjusting program parameters for maximal cancellation of SI.

Figure 32: Picture from the left side of the front panel. There are settings for transceiver and DSIC.

Figure 33: The other half of the front panel, which includes measurement data and program state information.

Figure 34: The Transceiver and DSIC with direct cable.

Figure 35: Output power before and after the beginning of digital cancellation.

Figure 36: Frequency components of the signal at different stages of transceiver and cancellation chain.

Figure 37: Bandpass filter filters frequencies near the direct current offset and over the LTE band.
Figure 38: Frequency components of canceller stages after bandpass filtering and averaging

Figure 39: Received signal after RF- and digital cancellation

Figure 40: Performance of the cancellation stages with different transmit powers

Figure 41: Tap power on two separate measurements

Figure 42: Readjustment of 5 strongest and most important filter taps over time
# LIST OF ABBREVIATIONS AND TERMS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAbCC</td>
<td>Fixed-point word length representation structure, where AA is the word length, b is s (signed) or u (unsigned) and CC the integer part length with possible sign bit.</td>
</tr>
<tr>
<td>ADC</td>
<td>analog to digital converter</td>
</tr>
<tr>
<td>ASIC</td>
<td>application specific integrated circuit</td>
</tr>
<tr>
<td>BRAM</td>
<td>block random access memory</td>
</tr>
<tr>
<td>DSIC</td>
<td>digital self-interference canceller</td>
</tr>
<tr>
<td>FDD</td>
<td>frequency-division duplexing</td>
</tr>
<tr>
<td>FDX</td>
<td>full-duplex</td>
</tr>
<tr>
<td>FET</td>
<td>field-effect transistor</td>
</tr>
<tr>
<td>FIFO</td>
<td>first in and first out bus</td>
</tr>
<tr>
<td>FPGA</td>
<td>field-programmable gate array</td>
</tr>
<tr>
<td>GD</td>
<td>gradient de</td>
</tr>
<tr>
<td>I</td>
<td>in-phase</td>
</tr>
<tr>
<td>I/Q</td>
<td>in-phase and quadrature</td>
</tr>
<tr>
<td>LMS</td>
<td>Least mean squares</td>
</tr>
<tr>
<td>LTE</td>
<td>long term evolution</td>
</tr>
<tr>
<td>NI</td>
<td>National Instruments</td>
</tr>
<tr>
<td>P1dB</td>
<td>1 dB compression point</td>
</tr>
<tr>
<td>P2P</td>
<td>peer to peer</td>
</tr>
<tr>
<td>PA</td>
<td>power amplifier</td>
</tr>
<tr>
<td>PAPR</td>
<td>peak to average ratio</td>
</tr>
<tr>
<td>PC</td>
<td>personal computer</td>
</tr>
<tr>
<td>Q</td>
<td>quadrature</td>
</tr>
<tr>
<td>RF</td>
<td>radio frequency</td>
</tr>
<tr>
<td>SCTL</td>
<td>single cycle timed loop</td>
</tr>
<tr>
<td>SI</td>
<td>self-interference</td>
</tr>
<tr>
<td>SMA</td>
<td>SubMiniature version A</td>
</tr>
<tr>
<td>SoI</td>
<td>signal of interest</td>
</tr>
<tr>
<td>TDD</td>
<td>time-division duplexing</td>
</tr>
<tr>
<td>UUT</td>
<td>unit under test</td>
</tr>
<tr>
<td>VGA</td>
<td>variable gain amplifier</td>
</tr>
<tr>
<td>VI</td>
<td>virtual instrument</td>
</tr>
</tbody>
</table>

* convolution operation  
* complex conjugate  
\[ a(n) \] normalized received samples  
\[ a_i(n) \] in-phase component of \[ a(n) \]  
\[ a_q(n) \] quadrature component of \[ a(n) \]  
\[ b(n) \] orthogonalized nonlinear basis functions  
\[ \hat{b}(n) \] \( \hat{u}(p,n) \) as vector notation  
\[ D \] eigenvalues  
\[ e(n) \] error signal  
\[ E[] \] mean over time
\( e_{GD}(n) \) \( error \ in \ GD \ method \)
\( h(n) \) \( h_{pM}(p,n) \ as \ vector \ notation \)
\( h_{pM,RF}(t) \) \( impulse \ response \ of \ the \ channel \ at \ time \ instant \)
\( h_{pM}(p,n) \) \( the \ nth \ modelled \ memory \ coefficient \ of \ the \ pth \ ordered \ nonlinearity \ component \)
\( i \) \( index \ of \ a \ vector \)
\( i \) \( loop \ count \)
\( I_R \) \( received \ in-phase \ signal \ of \ interest \)
\( I_T \) \( transmitted \ in-phase \ signal \)
\( M_m \) \( model \ memory \ length \)
\( M_{post} \) \( memory \ length \ after \ current \ sample \)
\( M_{pre} \) \( memory \ length \ before \ current \ sample \)
\( n \) \( sample \ index \)
\( N \) \( number \ of \ filter \ co-efficients \)
\( N \) \( number \ of \ loops \)
\( N_s \) \( number \ of \ samples \)
\( p \) \( order \ of \ nonlinearity \ component \)
\( P(n) \) \( instant \ power \)
\( P_{avg} \) \( average \ power \)
\( P_{dBM} \) \( power \ in \ desibel \ scale \)
\( P_{odd} \) \( highest \ nonlinearity \ order \ of \ the \ model \)
\( Q \) \( eigenvectors \)
\( Q_T \) \( received \ quadrature \ signal \ of \ interest \)
\( Q_T \) \( transmitted \ quadrature \ signal \)
\( R \) \( resistance \)
\( R_{cs}(s) \) \( calculated \ cross-correlation \)
\( S \) \( transform \ matrix \)
\( S \) \( normalized \ transform \ matrix \)
\( t \) \( time \)
\( \tilde{u}(n) \) \( u(p,n) \ as \ vector \ notation \)
\( u(p,n) \) \( the \ nth \ modelled \ nonlinear \ basis \ function \ of \ the \ pth \ ordered \ nonlinearity \ component \)
\( \hat{u}(p,n) \) \( interpolated \ u(p,n) \)
\( V(n) \) \( instant \ voltage \)
\( V_{pp} \) \( peak \ to \ peak \ voltage \)
\( w_n \) \( complex \ weight \ of \ a \ tap \)
\( x_{pa}(n) \) \( PA \ input \ signal \)
\( x_{RF}(t) \) \( input \ signal \ to \ RF-canceller \)
\( x_{in}(n) \) \( original \ transmitted \ signal \)
\( x_{tx}(n) \) \( interpolated \ xtx(n) \)
\( y(n) \) \( sample \ received \)
\( y_{pa}(n) \) \( PA \ output \ at \ sample \ index \)
\( y_{pa}(n) \) \( the \ LMS \ calculated \ SI \)
\( y_{pa}(t) \) \( PA \ output \ at \ time \ instant \)
\( y_{RX+SI}(n) \) \( SoI \ and \ SI \ combined \)
\( y_{SI}(n) \) \( the \ SI \ signal \ received \)
\( z(n) \) \( noise \ at \ sample \ index \)
\( z(t) \) \( noise \ at \ time \ instant \)
\( z_{RF}(t) \) \( RF-canceller \ noise \)
\( \mu \) \( common \ update \ scalar \)
\[ \mu_i \quad \text{update scalar indexed by } i \]
\[ \Sigma \quad \text{covariance matrix} \]
1. INTRODUCTION

Amount of mobile devices is increasing in the world and also, majority of those require more bandwidth to move data over the air. This is because, it is more common for people to watch videos and other bandwidth demanding material with their mobile devices from the internet. That results an increasing need of more wireless bandwidth reserved for this data to be transmitted over the air. It has been predicted [8], that in year 2030 there will likely be 10,000 times more wireless traffic compared to year 2010. There are physical limits, that restricts how different frequencies can be used and how much it is possible to transmit data in those [18], consequently resulting the need to improve the efficiency of current technologies.

Receiving and transmitting signals at the same time at the same center-frequency is called full-duplex (FDX) transmission. Current wireless solutions accomplish two-way communications either by using different frequencies at the same time called frequency-division duplexing (FDD) or by fast switching between transmission and reception on half-duplex channel. The latter is called time-division duplexing (TDD). Currently, there exists no final commercial products that can communicate with FDX transmission over the same carrier frequency, with common antenna for both. This kind of technology, that is currently under development (for example [10] and [13]), is one possibility to improve efficiency of the current wireless technologies. The spectral efficiency of frequency utilization could be doubled this way, in theory. [13]

Problem with described FDD system is self-interference (SI) caused mostly by the coupling of the strong transmit signal to the receiver [2]. Transmitter and receiver are connected to the same antenna, or alternatively, their antennas are closely separated. This causes transmitted signal to overcome received, because received signal is highly attenuated during long travelled path from the other transmitter. The power difference can be as high as 120 dB [13]. Furthermore, the power amplifier (PA) of the transmitter and especially a low cost PA, induces major nonlinear signal components [12]. Due to the nonlinearity of signal, it causes suppression of SI to be a difficult task. To suppress SI caused by transmitter, three techniques are being developed to overcome this problem. First one is passive circulator component. [17] Second one cancels part of the transmitted analog signal that is conducted to antenna and receiver. This is called radio frequency (RF) canceller [9]. The last phase of cancellation affects signal, when it has been transformed into digital domain in the receiver. This phase is called digital nonlinear self-interference canceller [13].
The goal of this thesis project was to implement and document a real-time digital SI-canceller (DSIC) and to use it simultaneously with circulator and RF-canceller in measurements. This is because we wanted to examine how DSIC would work in real-time in varying environment. Moving antenna, or obstacles moving in the vicinity of antenna, alters signal paths [20], which affects also the digital cancellation. To get instantaneous results, a real-time solution was required. Equipment used for DSIC and transceiver included two FlexRIOs, one transceiver module, a case to connect these and personal computer (PC). Except for the PC, these were manufactured by National Instruments and are designed natively to be programmed with Labview, thus resulted it to be selected as the main programming language. Both FlexRIOs have a field programmable gate array (FPGA), that are well suited for high throughput digital signal processing, which was a requirement for the task. With the FPGAs, in certain development projects, it is possible to achieve higher throughput, than with traditional application specific integrated circuits (ASIC). ASIC solutions are built to perform better in already developed and optimized algorithm applications. Creating an ASIC solution is a slow development process, however, with FPGAs it is possible to create, and easily alter, specialised processing solutions during algorithm development phase. After the algorithm under development is working with FPGA, the next stage is the costly and slow development process of an ASIC implementation. Previous project [13] had made DSIC to work with a new algorithm. They run the algorithm with simulated or recorded signals on general purpose ASIC processors, but not in real-time, because of the poor performance of general ASIC processor (PC). This project demonstrates various logic blocks, from which the most are required to be implemented in implementations of DSIC on FPGA. Same blocks probably exists also in the development process of an ASIC circuit. Thus, the results can possibly be used to predict how much resources an equal sized ASIC design would require.

The remaining chapters are structured as follows:: At first, the chapter 2 focuses describing the components of transceiver chain, the source of the nonlinear SI and the SI cancellation stages. Chapter 3 first introduces to the algorithms implemented and Labview programming. In chapter 4, the equipment used to implement a real-time DSIC is presented, the modifications done into Labview transceiver programming template is described and the last subchapter 4.3 shows the developed implementation of the canceller on FPGA. In this chapter, changes to transceiver logic are described briefly and the host side logic (PC) is omitted. Next chapter 5 describes the measurement setup to measure performance of the implemented DSIC, using the program and the results of the measurements. Finally conclusions from the work have been derived in chapter 6.
2. FULL-DUPLEX RADIO FUNDAMENTALS

This chapter describes components of the equipment used to transmit and receive long term evolution (LTE) signal and to cancel the SI. First section 2.1 introduces the full-duplex transceiver blocks. Section 2.2 describes major source of nonlinear interference. Next section 2.3 briefly explains effects of RF-cancellation stage used in measurement setup. Digital SI cancellation stage is briefly explained in 2.4. More in-depth analysis of DSIC can be found in subchapter 3.1.

2.1 Full-Duplex Transceiver

LTE and many other of digital transmission techniques uses in-phase (I) and quadrature (Q) modulation (I/Q-modulation), where two data-streams are combined into same carrier frequency, but with a phase shift of 90°. [1] This way the spectral efficiency is maximized. Transmitter creates an analog signal from digital I/Q signals, modulates it to a carrier frequency and sends it to the antenna. Receiver does the same in reverse order. It demodulates analog signal from the antenna and converts it to digital domain.

Figure 1 represents general digital signal transceiver chain [16] (without controlling blocks for signal synchronization, LTE signal generation and decoding), with a common antenna for both, with a PA and added SI cancellation stages. At first, transmitter converts digital I₁ and Q₁ signals to analog baseband signals. This may induce out-of-band signal components, that are low-pass filtered before baseband signal is modulated with carrier frequency to passband. This weak signal is pre-amplified by variable gain amplifier (VGA) and band-pass filtered to remove frequency components that are no in the passband. Finally, signal is amplified by a PA to target transmitting power. Normally, this is directly conducted to the antenna, but in this current full-duplex in-band signalling solution, major part of the signal power must be directed to the RF-canceller [19]. Also a circulator is needed before the antenna. Circulator is a passive, in this case, three port device to provide isolation between signal transmitted to the antenna (from the first port to the second) and signal received from it (from the second port to the third). [17]
The circulator does not offer perfect isolation between transmitter and receiver, that is why further cancellation is needed. The RF-canceller cancels some of the SI caused by circulator leakage and reflections caused by antenna imperfections. Receiver sensitivity must be tuned according to weak power of signal of interest (SoI) received from the antenna. Without the circulator and the RF-canceller, the SI signal part would exceed receiver's perception range and therefore making acquisition of the SoI impossible. [13] Also, cancelling all SI at the analog RF-canceller would require expensive solutions, and possibly, not being feasible at all. [10] That is the reason why the rest of the SI have to be removed in digital domain.

After the RF-canceller, combination of the SI and the SoI is in a level, that the receiver can operate with. At first, in general LTE signal receiver structure, is band-pass filtering, that filters frequencies not in pass-band. Weak signal is then attenuated by a VGA and demodulated. Again, out-of-baseband frequencies are filtered and the resulting analog signal is converted to digital, in an analog to digital converter (ADC).

After the receiver chain, in the last stage, DSIC removes rest of the SI and leaves the received SoI ($I_R$ and $Q_R$). This process requires the signals transmitted ($I_T$ and $Q_T$) to operate.

### 2.2 Major Source of Nonlinear Distortion

In addition to huge power difference between signal transmitted and signal received, the PA causes also nonlinear distortion. [14] A common PA uses field-effect transistors (FET) to amplify weak powered signals. FET has three terminals called gate, source and drain. Voltage difference between gate and source affects how large current conducts from source to drain. [3] This can be used to amplify weak signal, because resistance between gate and source is so massive (around 100 MΩ or more), that nearly no current
flows through this channel, and therefore the input signal loses no power. Unfortunately, this relation between gate amplitude and drain current is not linear as seen in figure 2.

![Diagram](image)

**Figure 2:** FET drain current in relation to gate voltage in black, blue is the linear/ideal case.

FET has a nearly linear gate-voltage to drain-current region, that can be used to amplify signal, but greater voltages induce current to saturate. Also in the linear region near saturation region it is not completely linear and this creates nonlinear distortion to resulting signal. This nonlinear distortion causes SI cancellation to be a difficult task. Data sheets declares PAs linearity by informing the 1 dB compression point (P1dB), demonstrated in figure 2, in which the power of the PA output is 1dB less, than it would be if the PA would be ideal and thus nearly linear.

### 2.3 RF-Canceller

A relative simple SI modelling is needed at this stage, by the research work made. [5], [10], There are no active components between the PA output and the RF-canceller input. This means, that channel between those is very linear and the most of the SI is caused by attenuated circulator leakage, and a delayed reflection from the antenna. Those can be modelled with

\[
x_{RF}(t) = h_{PA-RF}(t) * y_{PA}(t) + z(t).
\]

(1)

Here \( t \) is time, \( y_{PA}(t) \) is the PA output, \( h_{PA-RF}(t) \) is impulse response of the channel, \( z(t) \) is noise, \( x_{RF}(t) \) is the input to the RF-canceller and * signify the convolution operation. When the receiver is tuned to the most sensitive state, a two tap filter will suffice [13], to attenuate SI in the RF-canceller to a level that does not exceed the upper power limit
of the range, where the receiver is tuned. Constantly exceeding this limit would mess the signal because of saturation, and retrieval of information from it would be impossible. Rest of the SI and delayed multipath components from environment, can be filtered in digital domain. A two tap RF-canceller has two vector modulators, which tune their complex coefficients (consisting amplitude and phase) to match delayed and attenuated versions of \( y_{RF}(t) \). [9] Idea is to create opposite signal that will negate most of the SI coming through the circulator and from the antenna. The signal at the RF-canceller output \( y_{RF}(t) \) can be represented with

\[
y_{RF}(t) = x_{RF}(t) - \sum_{n=1}^{N} w_n y_{PA}(t - \tau_n) + z_{RF}(t).
\]

In this, \( N \) is the number of taps in the RF-canceller, which in this work is 3, \( w_n \) is the complex weight of the tap, \( \tau_n \) is delay of the copy of the PA output signal and \( z_{RF}(t) \) is the noise caused by the RF-canceller. Adding a third vector modulator, improves cancellation by removing the third largest SI component. [19]

### 2.4 Digital SI Cancellation

To remove rest of the SI from signal coming from the receiver after RF-cancellation, digital SI cancellation is required. [13] The signal path to the DSIC goes through the transmitter, the circulator, the RF-canceller and the receiver. All of these, except the circulator, may induce nonlinear distortion to the signal. Figure 3 represents structure of the DSIC canceller. Green box in the figure represents the DSIC part, that has been implemented on FPGA. The SI is removed with a least mean squares (LMS) filter using pre-calculated nonlinear basis functions (red box in the picture) affecting the filter operation. Detailed modelling of the SI and the cancellation algorithm is deduced in chapter 3.1.

![Figure 3: Digital self-interference canceller structure.](image)

---

[1] This is a placeholder for the actual figure. Please provide the correct image or diagram.
3. DSIC ARCHITECTURAL DETAILS AND DEVELOPMENT ENVIRONMENT

This chapter deduces algorithms used in the implementation of the DSIC. First section 3.1 focuses to digital SI cancelling algorithm. Next two sections are assisting sections to understand codes produced, that are partly demonstrated in chapter 4 (DSIC FPGA implementation). First of these, 3.2, describes the method to solve the delay for transmitted signal from transmitter to receiver. The second 3.3 describes how signal power calculation is done efficiently between FPGA and host PC. Section 3.4 focuses into graphical Labview programming and the last, 3.5, describes important debug and test routines during development.

3.1 Digital SI Cancelling Algorithm

This algorithm, which derivation is shown here, has been invented into current form by D. Korpi et al. [13] The derivation begins from model of the PA and lead to the algorithm implemented into the FPGA.

In addition to huge power difference between transmitted and received signal, the transceiver induces also nonlinear signal components. This is why, distortion must be modelled accurately, because there are many active components, that may cause nonlinear interference, especially in low-cost highly non-ideal devices. [2] Majority of nonlinear distortion in the transceiver chain is caused by PA. Effects of the PA can typically be modelled using discrete time parallel Hammerstein model [7], expressed by formula

\[
y_{\text{PA}}(n) = \sum_{p=1}^{P_{\text{odd}}} \sum_{m=-M_p}^{M_p} h_{\text{PA}}(p,m)^* x_{\text{PA}}(n-m)|x_{\text{PA}}(n-m)|^{p-1},
\]

where \( n \) is the sample index, \( y_{\text{PA}}(n) \) PA output, \( P_{\text{odd}} \) (odd number) specifies the highest considered nonlinearity order of the model, \( M_p \) is model memory length before and after current sample, \( h_{\text{PA}}(p,n) \) is the \( n \)th modelled memory coefficient of the \( p \)th ordered nonlinearity component and \( x_{\text{PA}}(n) \) is the input signal to the PA. Symbol \(^*\) means complex conjugate of a number. Because the PA is causing the most of the nonlinear distortion, \( x_{\text{PA}}(n) \) can be approximated by original transmitted signal \( x_{\text{TX}}(n) \) and then \( h_{\text{PA}}(p,n) \) includes power and phase changes caused by the rest of the transmission chain. To cancel SI, the PA output \( y_{\text{PA}}(n) \) needs to be subtracted from the signal coming from the receiver \( y_{\text{rx+SI}}(n) \) to reveal SI \( y_{\text{TX}}(n) \) with added noise. The RF-cancelling stage is bypassed in
this review, because it should no have major influence to the DSIC functioning and DSIC can be used without RF-canceller. Formula

\[ y_{rx}(n) = y_{rx+SI}(n) - y_{pa}(n) + z(n) \]  

(4)

represents this operation. To simplify the structure of the canceller, it can operate without SoI from antenna, and this leaves only noise \( z(n) \) left at the output of the canceller. This noise is combined noise from transceiver chain, noise and signals from environment and from model mismatch of formula (3). This noise should be less than receivers noise floor in optimal case.

Calculation of formula (3), requires knowledge of the \( h_{pa}(p,n) \). This can be calculated iteratively from output signal with LMS. [6] The LMS is a computationally light algorithm suitable for low-cost mobile devices. It is a derivation from gradient descent (GD) method, where error signal between the modelled signal and the real signal received, updates memory coefficients of the filter. In this case, the error signal is

\[ e(n) = y_{si}(n) - \hat{y}_{pa}(n), \]  

(5)

when no the SoI exist and only the SI signal \( y_{si}(n) \) is received. The error signal \( e(n) \) is an approximation, that all nonlinear distortions is caused by the PA and \( \hat{y}_{pa}(n) \) is LMS calculated term from formula (3).

In the GD and in the LMS method, the idea is to find the coefficients \( h_{pa}(p,n) \), that minimizes the mean squared error

\[ e_{GD}(n) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N} e(n)^* e(n) = E[|e(n)|^2] \to 0. \]  

(6)

In this formula, \( e_{GD}(n) \) is the error to be minimized, \( E[\cdot] \) the mean over time. To simplify formulas, vector notation of \( h_{pa}(p,n) \) is

\[ \bar{h}(n) = \begin{bmatrix} h_{pa}(1, n-M_p), h_{pa}(3, n-M_p) ... h_{pa}(P_{odd}, n-M_p), \\ h_{pa}(1, n-M_p + 1), h_{pa}(3, n-M_p + 1) ... h_{pa}(P_{odd}, n+M_p + 1), \\ ... h_{pa}(1, n+M_p), h_{pa}(3, n+M_p) ... h_{pa}(P_{odd}, n+M_p) \end{bmatrix}^T \]  

(7)

and
\[ u(p,n) = x_{\text{in}}(n) | x_{\text{in}}(n) |^{p-1}, \]
\[
\bar{u}(n) = \begin{bmatrix}
  u(1, n-M_p), u(3, n-M_p), \ldots, u(P_{\text{odd}}, n-M_p), \\
  u(1, n-M_p+1), u(3, n-M_p+1), \ldots, u(P_{\text{odd}}, n+M_p+1), \\
  \ldots, u(1, n+M_p), u(3, n+M_p), \ldots, u(P_{\text{odd}}, n+M_p)
\end{bmatrix}^T.
\] (8)

Term \( u(p,n) \) is called nonlinear basis function of the \( n \)th nonlinear basis function of the \( p \)th ordered nonlinearity component and \( \bar{u}(n) \) vector holds those values on different orders and times. Formula (3) can then be written into
\[
y_{ph}(n) = \bar{h}(n)^H \bar{u}(n). \tag{9}\]

The GD method achieves the minimum by updating \( \bar{h} \) with gradient of
\[
\frac{\partial e_{\text{GD}}}{\partial \bar{h}_i} = \frac{\partial}{\partial \bar{h}_i} E[|e(n)|^2]
= E[-2e(n) \frac{\partial}{\partial \bar{h}_i} e(n)]
= -2E[e(n) \bar{h}_i(n)]. \tag{10}\]

Here \( i \) is the index of a vector. The LMS algorithm is a simplification of this. It uses only current time instant to update the coefficients, instead of averaging over time. New coefficients are calculated with
\[
\bar{h}_i(n+1) = \bar{h}_i(n) - \mu_i \frac{\partial e_{\text{GD}}}{\partial \bar{h}_i} \approx \bar{h}_i(n) + \mu_i e(n) \bar{u}_i(n). \tag{11}\]

This updates coefficients with a small update scalar \( \mu_i \) into direction of the gradient and it may be different for different derivates. These algorithms, however requires, that gradient over time is 0 for those to function correctly. This is true when basis functions are orthogonal to error signal, meaning for that
\[
\forall i, \lim_{n \to \infty} \frac{\partial e_{\text{GD}}(n)}{\partial \bar{h}_i} = 0. \tag{12}\]

It is possibly always not the case for raw basis functions and those are required to get orthogonalized, so that functions in (9) and (11) converges faster to a solution and minimizes the error signal. To achieve this, the basis functions must be decorrelated and orthogonalized by a whitening matrix proposed in [13]. For this process, input signal must be interpolated, because the nonlinear basis function calculation process induces frequencies higher than in the signal band. That would result in folding of signal.
For example, if the highest nonlinearity order is 11, signal has to be interpolated 10 times, so that higher frequencies would not fold. If current values of the nonlinear basis functions $\hat{u}(p,n)$, calculated from interpolated transmitted signal $\dot{x}(n)$ by

$$\hat{u}(p,n) = \dot{x}(n)|\dot{x}(n)|^{p-1}, \quad (13)$$

and those are put to vector $\bar{b}(n)$

$$\bar{b}(n) = [\hat{u}(1,n), \hat{u}(3,n), \hat{u}(5,n), \ldots, \hat{u}(P_{\text{odd}},n)]^T, \quad (14)$$

the covariance matrix $\Sigma$ of it is

$$\Sigma = E[\bar{b}(n)\bar{b}(n)^H], \quad (15)$$

and an eigendecomposition equation of it is

$$\Sigma = QDQ^H. \quad (16)$$

From (16) it is possible to calculate eigenvectors $Q$ and diagonal matrix $D$ of eigenvalues of the covariance matrix $\Sigma$. Values in $Q$ and $D$ should be in descending order (Matlab’s $\text{eig}$-function uses accenting order). After $Q$ and $D$ are solved, the transformation matrix $S$ can be calculated with

$$S = D^{-\frac{1}{2}}Q^H. \quad (17)$$

The square root is element wise. If signal model stays the same (LTE signal in this case), also statistical properties of the signal and $S$ stays the same over time. This provides the possibility to calculate $S$ beforehand from random LTE signal. Evaluating a matrix multiplication with the $S$ and the nonlinear basis functions by

$$\tilde{b}(n) = S \bar{b}(n), \quad (18)$$

results $\tilde{b}(n)$ to be orthogonalized and decorrelated and LMS algorithm will converge faster to a better solution. [13] To have a constant update factor for all the nonlinearity orders in formula (11), rows of $S$ can be normalized with standard deviation of the orthogonalized basis functions by

$$\hat{S}_i = \frac{S_i}{\sqrt{E[(\bar{b}_i - E[\bar{b}_i])^2]}}, \quad (19)$$

Here $S_i$ is a row vector of $S$, $\bar{b}_i$ is also a row vector from a matrix calculated with formula (18) and $\hat{S}_i$ row of resulting matrix $\hat{S}$, which replaces the $S$ in formula (18). This
means, that also variances are equal, and dynamic range of values are reduced, resulting
a fixed-point implementation easier to implement with FPGA.

If the LTE signal is fixed, the orthogonalized nonlinear basis functions can also be cal-
culated beforehand and no FPGA implementation for orthogonalization is required pro-
posed in [13]. By calculating basis functions

\[ \tilde{u}(n) = \begin{bmatrix} \tilde{b}(n-M_p) & \tilde{b}(n-M_p+1) & \ldots & \tilde{b}(n+M_p) \end{bmatrix}^T \]  

(20)

beforehand, formulas 5, 9 and 11 can be combined into following algorithm.

\begin{algorithm}
\begin{algorithmic}
\State \( \mu = 2^{-13} \)
\State \( \mathbf{h} = [0 \ 0 \ \ldots \ 0]^T \)
\State \( n = 0 \)
\While {Transmitting}
\State \( e(n) = y_{ls}(n) - \mathbf{h}(n)^T * \tilde{u}(n) \)
\State \( \mathbf{h}(n+1) = \mathbf{h}(n) + \mu \times e(n)^T \times \tilde{u}(n) \)
\State \( n = n + 1 \)
\EndWhile
\end{algorithmic}
\end{algorithm}

Algorithm 1: Pseudocode of the DSIC.

Algorithm 1 performs the digital cancellation. In original pseudocode [13], \( \mu \) was set in-
dependently for each coefficient calculation, but because of formula (19), it could be set
common for all. Pseudocode describes the current implementation by setting \( n \) to begin
from 0, and nonlinear basis functions before first sample to 0. After the calculation of
\( \tilde{u}(n) \) it has to be decimated into the original sample rate.

Along the development progress, different number of nonlinearity orders were tested,
which effects directly to the size of the LMS filter, because each order has its own
memory. Also different memory lengths were tested with these LMS filters. At the be-
\(
\text{beginning of development process the goal was set to get project working with 11 nonlinear-
\text{ity orders and filter memory target was more than 31 samples. 11 nonlinearity order means 6 LMS-filters because only odd nonlinearities affect signal. [13] This goal proved to be too much for current equipment. 7 nonlinearity orders and 27 memory length was the largest implementation of the algorithm that was able to fit into the larger of the FPGAs. This means 4 LMS filters in nonlinearity orders 1, 3, 5 and 7. In this version of the algorithm pre- and post-memory length were set to equal } M_p \text{ size of a value } 13. \text{ Then the total memory size is } M_p+1+M_p=27. \text{ This means, that the lengths of vectors } \mathbf{h} \text{ and } \tilde{u} \text{ are } 4 \times 27=108 \text{ complex values, that include all of the memories of different order. The implemented block diagrams for this algorithm are shown in the chapter 4.3.5.}
\)
3.2 Delay Search

Sample delay from transmitter to receiver is possible to acquire by doing cross-correlation between received and transmitted signal. Equation for cross-correlation is

$$R_{xy}(s) = \sum_{n=-\infty}^{\infty} x^*_n(n) y(n+s). \quad (21)$$

In the formula (21) $x_n(n)$ is the transmitted signal and $y(n)$ is received, $s$ is the delay, $n$ the sample index and $R_{xy}(s)$ is the cross-correlation calculated. With discrete size vectors, Labview’s cross-correlation [15] works as

$$x(j) = 0, j < 0 \lor N_x \leq j$$
$$y(j) = 0, j < 0 \lor M_y \leq j$$
$$0 \leq u \land u < M_y + N_x - 2$$

$$R_{xy}(u) = \sum_{n=0}^{N_x-1} x^*_n(n) y(n+u-N_x+1).$$

Here in this equation $M_y$ is the vector length of $x$ and $N_x$ the vector length of $y$. Delay can be acquired by searching the index of the maximum of the correlation by

$$\arg \max_s R_{xy}(s).$$

Combining formulas (21) and (22) results in formula

$$\sum_{n=0}^{N_x-1} x^*_n(n) y(n+s) = \sum_{n=0}^{N_x-1} x^*_n(n) y(n+u-N_x+1) \Rightarrow$$
$$s = u - N_x + 1.$$ 

In this work, it is enough to calculate correlation only for one of the I and Q branches, because delay is the same for both. This means that no complex conjugate is needed. The implementation of this is not covered by this document. This is here as a documentary purpose to understand the code created to find the delay.

3.3 Calculating Signal Power

Instant power is calculated [4] by

$$P(n) = \frac{|V(n)|^2}{R}. \quad (25)$$
The calculation of instant power \( P(n) \) requires knowledge of instant voltage \( V(n) \) and impedance \( R \) of an antenna in ohms. Average signal power of discrete signal can be calculated with formula

\[
P_{\text{avg}} = \langle P(n) \rangle = \lim_{N \to \infty} \frac{1}{2N+1} \sum_{n=-N}^{N} P(n),
\]

(26)

where \( P_{\text{avg}} \) is average power over time and \( N \) number of samples.

In the transceiver, voltage values \( a(n) \) transmitted or received are normalized to have values between \(-1 \rightarrow 1\). To get real voltage, this value must be scaled to correct range with \( V_{pp} \), which is peak to peak value of current voltage range by

\[
V(n) = \frac{V_{pp}}{2} a(n).
\]

(27)

Two separate signals I and Q exists, thus \( a(n) \) is complex value. Combining formulas (25), (26) and (27) results in equation

\[
P(n) = \left( a_i(n) + ia_Q(n) \right) \left( a_i(n) - ia_Q(n) \right) = a_i(n)^2 + a_Q(n)^2
\]

\[
\rightarrow P(n) = \frac{V_{pp}}{4R} \left( a_i(n)^2 + a_Q(n)^2 \right).
\]

(28)

Here \( a_i(n) \) is real and \( a_Q(n) \) the imaginary parts of \( a(n) \). Combining equations (26), (27), adding second signal, ignoring past samples and with only finite samples forms formula

\[
P_{\text{avg}} = \lim_{N \to \infty} \frac{1}{2N+1} \sum_{n=-N}^{N} \frac{V_{pp}}{4R} \left( a_i(n)^2 + a_Q(n)^2 \right)
\]

\[
\rightarrow P_{\text{avg}} = \frac{V_{pp}^2}{4NR} \sum_{n=0}^{N-1} a_i(n)^2 + a_Q(n)^2.
\]

(29)

Equation (29) is implemented partly on FPGA and partly on host. Lastly linear scale \( P_{\text{avg}} \) is converted to logarithmic dBm scale with

\[
P_{\text{dbm}} = 10 \log_{10} \left( 1000 P_{\text{linear}} \right).
\]

(30)

This is done at host PC, because implementing logarithm operation on FPGA is not resource efficient. The FPGA part of the implementation of the equation (29) can be found in chapter 4.3.5 in figure 14.
3.4 Introduction to Labview Programming

Most of the programming languages are based on structural textual programming, however, Labview is an exception to this. In Labview, programming is mostly done with function icons and connections between these. The functionality of the program is programmed inside virtual instruments (VI) described in the first subsection 3.4.1. Next subchapter 3.4.2 shows an example of a simple normal Labview program and the last 3.4.3 describes some of the major differences, that must be taken into consideration when programming FPGAs with Labview.

3.4.1 Virtual Instruments

Each VI created with Labview consists of two parts: a front panel and a block diagram. The front panel consists of inputs called as controls and outputs called as indicators. Controls, which accept input values, can be, e.g. buttons, input text or value boxes. Indicators can be a lamp, output text, value boxes, a graph, etc. but usually directly non-editable by user. Arrays can combine multiple components of the same type and a cluster various different types.

Figure 4: Samples of a control cluster on the left and an indicator cluster on the right, both consisting different types of controls and indicators. Context help shown describes the types of different controls of the control cluster.
Figure 4 represents a cluster of controls on the left and a cluster of indicators on the right. The right control cluster has different type value fields, a boolean button and an array of input values. The left has a graph indicator two value indicators, a boolean lamp indicator and a sub-cluster with two indicators.

The block diagram creates the functionality of the VI. In the block diagram, data flows usually from controls on the left to the indicators on the right, unless there is feed-back nodes from outputs to inputs. A feed-back node is a shift register, that holds value received at the input of it in a clock cycle, and forward the value to the output at next clock cycle. In a feed-back node, the input is at the right and the output at the left, and in feed-forward node it is the other way round. A VI can have sub VIs, which may also have sub VIs. This way complex structures of a program can be divided to smaller VIs. Sub VIs front panel can not be seen on the main VI front panel and only a VI icon with input and output terminals are visible in the block diagram of a VI.

An important matter to know, when using feed-back or forward nodes, is the execution mode of the sub VIs. If this matter is not known, it may result into very long lasting debugging condition. By default, normal VIs executed on PC are set to be in non-reentrant execution mode. In this mode, there exist only one instant of the VI in the memory. If sub VI is supposed to hold information from cycle to the next cycle and multiple instances of the sub VI exists in the block diagram to be executed on the same cycle, the program would not work as expected, because data stored to feed registers on one instance would affect others instances. This usually is not the expected way for program to work. Therefore, the execution mode has to be altered from the VI properties into “Preallocated clone re-entrant execution” -mode. This way each sub VI instance has its own memory space. When creating VIs to FPGA target, this mode is active by default for all VIs.

3.4.2 A Normal Labview Example Program

As a simple example how to program a function on Labview, in figure 5 mean calculation is performed in sample_mean.vi. It has a sub VI sample_sum.vi illustrated in figure 6. The sample_sum.vi calculates the sum of the array connected to the input of it, with three different ways, all producing equal results. As can be seen in the figure 5, front panel of the sample_sum.vi is not visible in the sample_mean.vi. However, the sample_sum.vi has a front panel and it is visible, when used independently. At the upper right corner of the front panel, an icon for the VI is shown, and to the right of it, a terminal connection panel can be seen. The icon of the sample_sum.vi is visible in the block diagram of the sample_mean.vi and input and output components of it are connected to the terminals of the sample_sum.vi. In the both figures, common Labview programming component are used. All calculations are done for 32 bit signed integers, except for the division. For the division, integers are cast to double precision floats.
After taking array size in the figure 5, there is a red coercion point at the divider terminal of the division component. This may induce slow down to the program execution and program may not function as expected. Therefore, type casting components should always be used to avoid coercion type casting, as is done before the dividend terminal in figure.

Figure 5: Front panel and block diagram of a mean calculation.

Figure 6: A sum function front panel and block diagram.
In the figure 6, three ways to calculate sum with Labview is shown. The first is the Labview’s ready made basic function to add array elements with each other. Both the for-loops do the calculation equally, however, using different techniques. In the upper for-loop, loop count $N$ has been set to the size of input array and the loop has a tunnelling input for it. In a tunnelling input, array stays as a whole, and therefore, array cells has to be indexed. Indexing is done here with the loop count $i$. Fetched value from the array is added to the sum of the last loop cycle executed. The sum is transferred to the next cycle with a shift register, that in the beginning of the loop is set to 0. Shift register in Labview always has an input terminal to current cycle and output terminal for a value transferred to the next cycle. After the last cycle, the value in shift register transfers forward in the diagram after the for-loop. The last for-loop has an auto-indexing input. It automatically sets the loop count $N$ to the size of the array and indexes and fetches the array cells from the array according to loop count $i$.

### 3.4.3 FPGA Version of the Sample Program

The first of these for-loops in figure 6, which was the ready made Labview component, is not available, when programming FPGAs. The other two will work, when not inside single cycle timed loop (SCTL), however, those may perform poorly in high throughput applications. SCTLs have a clock attached to it. In SCTL, in every clock cycle, signal travels through every combinatory logic between shift registers (or feed-back or feed-forward registers), which holds the value for the next clock cycle. Logic components induce delays to the signal, and if these combinatory paths are too long, the delay of the logic exceeds clock cycle time. The longest or slowest path is called as a critical path. In this situation the logic would not work as expected and the path must be divided with extra shift register. For-loops inside SCTL, that has shift registers are not supported by the Labview directly, as is with multidimensional arrays, and many other common coding structures and components.

Labview has a tool called IP Builder, that can create properly timed summing component. As an example, for a sum function to be build by the IP Builder, the block diagram is done equally from either of the for-loops in the figure 6. Then directives are made for the block diagram. The directives controls how the final component is being build. There are various options for directives that can be set. The main directive is the speed of the SCTL. Compiled component has to be able to function with that speed. In high throughput applications it is also required to set initiation interval. It means after how many clock cycles new input sample is targeted to arrive. If input sample speed is the same as the SCTL speed, the initiation interval would be 1 cycles. When the initiation interval is 1, and if it is intended to calculate the whole input array at the same clock cycle, and not just one cell of it, a directive “All elements” has to be set for the array. There are many other options, but in high throughput applications, these are the most important. When the directives are set, then those are tested if it is feasible to achieve
these requirements with a test build. If test completes successfully a component can be build. This compiled component works correctly also in simulation mode of the target FPGA. If test fails, the directives, the block diagram or target throughput has to be altered. In the case of this sum function, if the input array is set to size 108 and the directives mentioned, the IP Builder would create a component, that calculates new sum in each clock cycle but the delay to the output would be 85 samples. This is not an optimal solution for an efficient sum function. Following figure 7 represents an efficient sum function for an array with 13 complex numbers that is possible to compile to the FPGA.

Complex numbers are a value pair cluster of real and imaginary parts in an array. For the simplicity it is only 13 samples, but it represents a for-loop block and the principle, that can be used to form summing function for greater arrays. The idea of this VI is to divide array to pairs, sum those pairs and form pairs to sum from the sums. After each arithmetical operation, there is a shift register to shorten the logic critical path. The for-loop blocks sum \( N \) (even) cells and form \( N/2 \) results. These has to be calculated manually in FPGA applications, because dynamically sized loops and vectors are not supported directly in SCTL, unless programmed with the IP Builder.

![Image of circuit diagram](image)

*Figure 7: FPGA compatible efficient sum function.*

Because forming pairs from a table which size is odd the last cell from table is transferred forward. In the first loop, 12 first cells are summed, and an array of 6 sums are at the output of it. To index the first value to sum from the next pair, loop count is doubled and for the second pair 1 is added to this index. Next the complex number cluster is un-
bound. After unbinding, real and imaginary parts are summed separately and bound to form an array of complex numbers again. The blue coercion dot at the output of the sum function means, that fixed-point numbers are manually set to a specific bit length length in the sum function properties. Otherwise Labview would define it automatically, and it is not always the best solution. In between the for-loops, there is a shift register and in the second loop 3 pairs are summed. After the second loop, again there is a feed-forward register before the next sum operation. Because the array of sums after the second loop is odd, the first pair of sums is summed together, and the last sum is summed with the 13th cell of the VI input array. It is important to add equal amount of feed-forward (or feed-back or shift registers), to the path of the 13th complex, as there are in the path passing through the for-loops. Otherwise the sums are not synced properly into correct clock cycle and the VI would not work. After the last feed-forward nodes the last summation is done and the result is visible on the “Complex sum” -indicator. This VI has a calculatory delay of 3 clock cycles.

As a programming procedure, it would be wise to split the 2 delay feed-forward node in the VI block diagram, so that delay in each spot on vertical line, when inspecting the code, would have an equal delay. This would make the code easier to read and to debug during development. However, Labview raises an error on compilation, when feed-forward nodes are serialized without some other programming component between nodes. It would also be wise to create a hand shaking algorithm to handle situations, when there would not be valid samples in the input or output on each of the clock cycles, but these are just simple demonstrations of Labview programming.

3.5 Test Benches and Debugging

When programming FPGAs, test benches are important tools for validating the functioning of code blocks. By experience, testing code and debugging errors are the most time consuming process when programming FPGAs. Nearly all program blocks require a customized test bench as seen in figure 8 to verify correct functioning.

![Test bench diagram](image)

*Figure 8: Typical test bench.*
In a typical test bench, there are input signals and calculated output signals for unit under test (UUT), which are known. These are compared with the values generated by the UUT. The reference output signals can be pre- or post-calculated, or generated with a block functioning the same way as UUT. The pre-calculation can be used, if the input signal is known beforehand. If the input signal is not known, input and output signals can be recorded, reference output post-calculated with test-bench and verified with recorded output. Pre- and post-calculation can be done with different tools than Labview. This project has done many verifications with Matlab, which can also be used inside Labview code. It is possible also to create an equally functioning block with Labview code, that is not possible to compile to the FPGA, but can be used in simulation environment. Simulation mode can be enabled from the properties of the target board. By default, VIs are compiled and executed on target. Compilation, however, takes a lot of time and usually simulation mode is preferred way to test proper functioning of the program. If blocks are simple, outputs can be verified without the output test-bench.

The UUT must function properly also when no samples arrives, and it should wait in the hold state without memory corruption. When the reset signal arrives, block has to clear its memory units to prevent corruption of output after the reset. This is important task for test bench to verify. In this project, all of the previous methods are used and functioning of the different states verified.
4. DEVELOPED REAL-TIME IMPLEMENTATION OF DSIC

This chapter focuses to describe the DSIC implementation. Section 4.1 focuses on equipments used to implement the DSIC. Section 4.2 introduces transceiver code template created by National Instruments (NI) to work as a base for developers, and briefly describe how it has been modified for this project. The last section 4.3 describes the functioning of the FPGA DSIC and inspects the block diagrams created.

4.1 Equipment Used to Implement the Real-Time DSIC

DSIC and transceiver has been implemented with Labview FPGA programming environment into two FlexRIOs. FlexRIOs are equipment made by National Instruments for high throughput input, output and signal processing. These include a FPGA circuit and various modules can be attached into it, depending on what is being developed. FPGAs are better suited for the development of high performance signal processing algorithms than ASIC. ASICs are developed for a certain purpose for tested algorithms or tasks, and perform often poorly with new algorithms, that are under development for high-throughput signal processing. FPGA implementation operate usually with slower frequency than an ASIC solution, however, they offer the possibility to program specialized processors better suited for new algorithms under development. Based on FPGA prototyping, faster ASIC implementations can be developed.

Table 1: Resources available and their usage in FPGAs.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>PXIe-7962R (Virtex 5)</th>
<th>PXIe-7972R (Kintex 7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>7930/8160 (97.2%)</td>
<td>37654/50950 (73.9%)</td>
</tr>
<tr>
<td>Slice registers</td>
<td>24476/32640 (75.0%)</td>
<td>117538/407600 (28.8%)</td>
</tr>
<tr>
<td>Slice LUT</td>
<td>20184/32640 (61.8%)</td>
<td>119631/203800 (58.7%)</td>
</tr>
<tr>
<td>Block RAM</td>
<td>70/132 (53.0%)</td>
<td>178/445 (40.0%)</td>
</tr>
<tr>
<td>DSP48s</td>
<td>140/288 (48.6%)</td>
<td>220/840 (26.2%)</td>
</tr>
</tbody>
</table>

In this work, a NI 5791 transceiver module is attached to the FlexRIO NI PXIe-7962R, which has a FPGA, that has less capacity. That FPGA focuses on transmission of LTE signal and receiving the rest of that signal after circulator and RF-canceller. FlexRIO NI PXIe-7972R has more resources and it handles the digital cancelling and sends res-
ults and measurement data to personal computer (PC). Table 1 shows the resources available in FPGAs. The FlexRIOs are attached into NI PXIe-1073 chassis, which provides for the two FPGAs a way to communicate with each other and with the PC. The available space and the chassis bus speed of 250 MB/s restricted the implementation size of the DSIC the most. At the beginning of project, only one FPGA was used. However, during development with current implementation extent, it was clear that the project would not fit into only one FPGA, and two of those were required.

The target was to transmit 20 MHz wide LTE OFDM signal in 2.46 GHz carrier frequency. The peak to average ratio of the signal was 11.0 dB. Sampling frequency has to be greater than signal bandwidth, so that there is enough band for transition band of the low-pass filter at the receiver. Narrow transition band raises filter size by adding more coefficients into finite impulse response filter (FIR). Transceiver had a code template designed to be operated on 130 MHz frequency, so most suitable sampling frequency was one fifth of it, which equals to 26 MHz. This allows 6 MHz transition band. However, project ended using only ready made solution, built into the template by NI to handle in the low-pass filtering of the receiver, so accurate characteristics of this filter implementation are unknown.

Transmitter accepts by accuracy of 16 bits I and Q input signals, and digital output of receiver has the accuracy of 14 bits for input signal. In practice, creates receiver 16 bits wide numerical values, because it is common in computer technology to use bit lengths that are divisible by a byte (16 bit=2 bytes). These all together mean, that transmission of one signal stream in bus reserves

\[
\text{reservedSpeed} = \text{sizeOfComplex} \times \text{wordSize} \times \text{signalFrequency} \\
\text{reservedSpeed} = 2 \times 16 \text{bit} \times 26 \text{MHz} \\
\text{reservedSpeed} = 832 \text{bit} \times \text{MHz} \\
\text{reservedSpeed} = 832 \text{Mbit/s} = 104 \text{MB/s}. \\
\]

Therefore, only two signals in sample frequency fits to be transmitted at the same time in the chassis bus, which has the bandwidth of 250MB/s. This is why the two streams selected, was the stream between FPGAs and the stream from the DSIC FPGA into the PC. Therefore, the LTE signal had to be programmed into the beginning of the transmitter chain.

### 4.2 Transceiver Template Used

NI has made a Labview FPGA programming template for NI-579X transceiver called “Simple NI-579X Streaming”. This template offers a ready made implementation for a transceiver, that is divided into two pars: one for the host PC to control the transceiver and to the implementation of the FPGA transceiver. The host PC code includes user control interface, code for creating a signal to be transmitted, receiving data from the receiver and visual blocks to display calculated measurements from the received signal.
For example, the template can show the I/Q signal received and show the power spectrum. The FPGA part of this template controls the transceiver and handles signals from and to the PC. If no alteration is required into the FPGA template code, a precompiled binary driver comes with the template for different FlexRIOs. However, for the NI PXIe-7962R there was no precompiled binary and the project had to be moved to this target, also the code had been changed, so compilation was unavoidable.

The FPGA part of the template is split into two branches, to a transmitting branch and to a receiving branch, which are coded into same VI block diagram. Various values can be set for the transmitter branch: trigger for beginning of transmission, carrier frequency, sampling frequency, transmit power, phase and some other parameters not relevant for this work can be defined. All the same values can be determined for the receiver branch, however, instead of the transmit power, a maximum power level of the receiver input can be defined. The PC can work as a starting trigger for the both branches independently, but it is possible also to trigger both with the same trigger as is demanded in this solution. The carrier frequencies can be tuned from 200 MHz to 4.4 GHz. The maximum sampling frequency is 130 MHz. In the template, transmission power and the maximum power of the receiver input can be set from -32 dBm to 20 dBm, but in practice, there is an upper limit for transmitter, that changes with carrier frequency, and with 2.46 GHz that is around 8 dBm. The phases can be adjusted between -0.5 – 0.5 symbol width in the time domain, so that the receiver would get the signal as strong as possible, when the phase is correct.

In this project, the FPGA template was altered to work with the NI PXIe-7962R. And because of the chassis bus speed, it was impossible to have a third stream at the sample frequency from the PC, thus the LTE signal generation had to be programmed directly into the transceiver code. This solution stored a 10000 sample LTE signal into block random access memory (BRAM) of the FPGA. Instead into the PC, the template also had to be altered to send the received data into the other FPGA.

Changes to the template for the host PC were massive. Ready made implementation had separate implementations to control the transmitter and the receiver in two separate files, and therefore those had to be combined. Only the blocks controlling the parameters of the transceiver were left almost the same as they were in the original template, but otherwise, many new blocks had to be implemented. These included configuration of data stream between FPGA(s), adjusting the signal phase between the transmitter and the receiver and cleaning of data from buffers between program runs. Host templates were only simple examples how to control the transceiver, and lacked the proper cleaning of the streams. Because, there was data left in the buffers, it caused code to miss-function in certain situations, when operating the transceiver with the DSIC.
4.3 Implementation of the Digital Self-Interference Canceller

In addition to the core of the canceller, it requires several other blocks of code to handle the transmission and receiving of signals, stream control between the FPGAs, streams in and out of the canceller, reset logic, the DSIC start logic and signal power measurement blocks. First the functioning of the transceiver is described in 4.3.1. Then the actual canceller functioning is described broadly in 4.3.2. The 4.3.4 describes the circuits around the DSIC and the subchapter 4.3.5 describes the circuits of the actual DSIC core, that handles the cancellation according to algorithm represented on page 11. The focus is on the FPGA implementation of the canceller, the host PC side functions are not described in detail in this work.

4.3.1 Transceiver Functioning Logic

The transceiver transfers and receives with normalized signal voltage in the accuracy of 16s1 evenly on every fifth clock cycle. These values in both branches (transmitter and receiver branches) are scaled into proper range by the transmit power and the maximum power of the receiver input. In the transmitter, this means VGA power amplifier is set to amplify normalized LTE signal into certain maximal transmit power. In the receiver, the maximum input power has to be set properly into corresponding transmit power, to ensure that no saturation occurs in the received signal and the accuracy of the signal would be as close to the normalized range. If the receiver’s sensitivity is set into for too large power range, in relation to the transmit power, the accuracy of the received signal gets worse and cancellation results suffers. These both situations may prevent the SI cancellation totally.

For the peer to peer (P2P) FIFO bus between FPGAs, the received samples are converted into a single U32 (unsigned integer with 32 bits) values. Other possibilities are U8, U16, U64 and boolean, but U32 is convenient, because it can hold both of the I and Q branches of the complex signal. Complex signals are not directly supported on the Labview FPGA module. These samples are sent to the other FlexRIO in the chassis bus. The other FlexRIO reads and processes the samples from the bus with the speed of 32.5 MHz as long as there are samples.

4.3.2 Self-Interference Canceller Functioning in Broadly

In the beginning, all the LMS filter coefficients are set to zero. At first, after six input samples, the coefficients are updated with the first output sample and then cancellation of the seventh sample will be calculated with the updated coefficients. Therefore, iteration delay, or update delay, is 7.5 samples. Calculation of one sample is done in four clock cycles. This update delay matches 30 clock cycles of main-loop which operates on the speed of 130MHz. The delay from input to output is 4.25 samples, meaning 17 clock
cycles. The delay in the update branch is 3.25, thus the total delay being 7.5 samples. Because the total update delay is not an integer, it is actually 8 samples. This means, that the 2nd output sample will update the coefficients for the 9th input sample and so on. This process is shown in figure 9.

![Diagram](image)

**Figure 9: Functioning of the canceller. Filtering SI and calculation of new coefficients.**

The fixed-point numbers in the DSIC vary in their accuracy and range. In fixed-point number representation, reserved bits variable has a decimal point location after the first x number of bits. E.g. a 16 bit variable could be split to have 10 most significant bits for integers and 6 least significant bits for fractions. Fixed-point arithmetic operations require less logic usage from the FPGA, compared to floating point representation. Floating point numbers require less effort from designer, because these have a dynamic range. With the fixed-point numbers, optimal accuracy and dynamic range, has to be solved after each arithmetic operation in the algorithm to prevent the FPGA resources being wasted, overflows and loss of the accuracy. This is done by solving maximum values from the signals after each arithmetic operation and from the effect to the cancellation performance, when bits are removed. For solving the requirements, extensive Matlab simulations have been made. The simulations have been done with recorded input signal from receiver when the DSIC is not working. To illustrate word lengths, AAhhCC marking has been used, where AA is word length, b is signed (s) or unsigned (u) and CC the integer part length with possible sign bit. The real bit usage is double to the size of this word length, because numbers are complex numbers, that have a real and imaginary parts. As an example, 25s8 is a number, that has a decimal width of 25-8=17 bits and it is signed. Labview uses two’s complement with signed fixed-point numbers, thus the most significant bit is the sign bit.

Generating one output sample and updating coefficients both require 4 clock cycles. Since the main-loop operates on 130 MHz, program is able to function also in maximum of 32.5 MHz symbol speed. The faster symbol speed, than the required 26 MHz,
ensues from the bursting chassis bus, so the bus is not constantly streamed with data at the speed of 26 MHz. These bursts move a multiple data items in blocks, that are send at the speed of the bus. While these data blocks, other data streams are served on the bus and the DSIC must wait for the next burst. If the main-loop would operate on the symbol speed of 26 MHz, it is a possibility, that a bus buffer overrun occurs.

The DSIC has six states “0”, “1”, “2”, “3”, “hold” and “reset”. With the numbered stages there are always 27 calculations from vectors of length of 108 values. Therefore, 4 cycles are required for one full length vector arithmetic operation. At the first state “0” on the first clock cycle, arithmetic operation is done for the vector cells in range 1...27, on the second cycle state “1” calculates cells 28...54, etc. If there are no new samples arriving at the input, the DSIC will wait in the “hold” state. In the “hold” state all blocks will keep their current values unaltered. When the host sends a reset signal, all blocks will change to the “reset” state, which will clear all the data blocks for the cancellation to start from the beginning. These stages are common for the calculation of the output and also for the new coefficient calculation.

4.3.3 The Functioning of the Real-Time DSIC

The DSIC will wait in “hold” state always when the bus is empty. When the DSIC is started, the first state “0” operates with the first 27 values of vectors in calculations. At first, 27 nonlinear basis function values, that have the accuracy of 25s8, are multiplied by 27 first filter coefficients. Before this multiplication accuracy of 25s1 coefficient values are rounded to accuracy of 18s1. This saves digital signal processors (DSP) from the FPGA, that can process maximum of 25 bit times 18 bit multiplication. If larger bit values would be used, it would double the usage of the 4 DSPs. This multiplication results a full scale of 44s10 values. These basis function values are stored into 4 block random access memory (BRAM) memory blocks with each sized of 10000 samples. In the next block, these multiplication results are summed with each other and this sum is summed with the sums calculated also in the states “1”, “2” and “3”. The total sum is rounded into accuracy of 16s1 and subtracted from input signal sample coming from the receiver. The result of the subtraction is the attenuated signal, that in optimal case will converge into near zero amplitude. This value with accuracy of the 16s1 will be send to PC and to the update block of coefficients. The pipelining of this block results in a delay of 17 samples.

When implementing this cancellation block, it is not important at which order these four 27 length vector multiplications are calculated, because all are summed before the subtraction from the input signal. Relevant is, that the coefficients and the basis functions are aligned and timed correctly with the signal from the receiver.

The block updating the coefficients, receives first output sample after 17 clock cycles. That is why the basis functions has to be delayed equally, for that the coefficient update
stays synchronized with the input signal. This is done with pipelining buffer of 17 samples from the block reading values from BRAM. The states are pipelined equally. The update process consumes 10 clock cycles, which equals to 2.5 samples. Because of this delay, current implementation updates coefficients 3 cycles behind. This is a difference in the functioning of the implementation compared to the formula (11) or to the pseudo code 1 on page 11, where the coefficients are supposed to be updated to the last value of the coefficients. However, the difference, or possible error signal, is negligible to the size of the update step. During 4 clock cycles, 4*27=108 multiplications are calculated, where 25s8 basis functions are multiplied with the attenuated signal, that has the accuracy of 16x1. The results, at full accuracy, are 42s10 (the LMS error signal) for the coefficients. These are scaled by moving bits to the right 13 times, which equals multiplication with 2^{-13}. These results are rounded into accuracy of 25s1 and added to the coefficients, that has the same accuracy. Firstly updated coefficients affect the 9th calculated output sample of the DSIC.

When rounding values, normal rounding method is used, where values are rounded equally to the nearest number. It is mathematically the most accurate method compared to the other options, which are rounding always up or down, or truncate extra bits. The normal method is the most costly to implement as in circuit size, but the rounding error of the other methods might be cumulative and affect the results of the SI cancelling. However, this work has not studied this possibility.

For overflow situations, the saturation method is used. If calculation overflows from the dynamic range of values, the resulting value is set to its minimum or maximum of the range. It is probably more safe, than the spinning overflow method. In spinning overflow, bits not belonging to the range are truncated and only a small overflow can cause a large error to calculations. Spinning overflow would, however, require less space from the FPGA, because extra logic would not be needed as in the saturation method. Affects from using the spinning instead of the saturating overflow, has not been studied in this work.

### 4.3.4 Side Logic to the Real-Time DSIC

Next figure 10 represents the logic for fetching samples from the bus and reset circuit. It is the left side of the FPGA canceller MAIN.vi block diagram. When host sets “reset” signal high, if there is elements in point-to-point first in, first out bus (P2P FIFO), those are fetched out and wasted. Normally, if there are samples in the bus (P2P FIFO), those are read from the bus on every fourth clock cycle (at the speed 32.5 MHz) and “sample ready?” will then be true. There are three conditions that has to be met, before “reset done?” valued true signal can be read by the host: after the reset arrives (raising edge), 65 cycles has to be elapsed, the FIFO should be empty and the DSIC canceller has to be ready for input (“DC reset done?” in the picture). The first one of these conditions is
probably an obsolete code, used during the development and could be removed, because of the implementation of the “DC reset done?” signal.

The right side of the FPGA_canceller_MAIN.vi block diagram is shown in figure 11. Signals leaving the right side of the figure 10 are seen in the left side of the figure 11. The DSIC (filter_base.vi) gets the FIFO data and input valid from the figure 10 and “DC reset done?” is send back to the figure 10. The DSIC gets also various configuration parameters in, where the delay from the transmitter to the receiver in “Filter configuration” is the most important variable. The delay variable affects whether the output data from the DSIC is the unaltered signal coming from the input P2P FIFO (delay of -1 or other negative value), or the cancelled signal (a positive delay, less than 10000). The “Tap fetcher configuration” controls speed and mode, how the filter coefficient data is send through “Filter Tap FIFO” into the PC. The DSIC input and output average powers are send on every 128th clock cycle through “Power FIFO”. There are two signals to stop or disable the functionning of the canceller, some debug options and status information coming from the DSIC. There are some code that is left from the figures 10 and 11, but those are mainly simple information for debugging purposes.

Figure 10: The circuit for the reset signal and for the FIFO fetcher between FPGAs in the left side of the FPGA_canceller_MAIN.vi block diagram.
Figure 11: Connections in and out of the DSIC in the right side of the FPGA_canceler/Main.vi.

Figure 12 represents the block diagram for the “DC BASE” in the figure 11. The “DC START” handles the correct synchronization with the input signal and starts the cancellation process when “run?” signal has been given. The “run?” signal also selects the output mode. When “run?” is false, the input signal is the output signal of the DSIC and also the input valid is the output valid. When “run?” is true the output is changed to the cancelled signal and to the DSIC output valid. The “Stop?” signal prevents the output to be valid. This saves the bandwidth in the chassis bus, when no output signal is transmitted by the DSIC, an it can be given, for example, for faster tap transmission. There are two “POWER CALC.” blocks to calculate the signal powers: upper block for the output signal and lower for the input signal. Calculated value for the power is hold in previous value until a new one has been calculated.

To start the cancellation on the correct sample a synchronization circuit is required. Figure 13 represents this circuit. Before the host PC has found the correct delay from the transmitter to the receiver “Start delay” is set to negative number and the “Run?” will stay in false state. When the host PC has calculated the delay from the stream going past the DSIC filter in figure 12, it sends the positive delay to this synchronization circuit, which will trigger the “Run?” signal at the correct sample on the next period of the 10000 LTE samples. The “Run?” signal will stay true until next reset. This changes the
stream into the PC, into the output of the DSIC in the figure 12, and starts the cancellation process.

Figure 12: The filter_base.vi includes the block to start the DSIC, the DSIC block and input and output power calculation blocks.

We are transmitting and receiving 10000 samples with certain delay. When we know how many samples is the delay, we must start running after that delay in next 10000 samples.

Figure 13: The block diagram of activate_after_count.vi, that will trigger the cancellation process, when input stream is synchronized properly.
In the next figure 14, calculation of the average power is shown. The implementation has been made with Labview’s special tool called IP Builder. After creating directives that the coded block has to be met, from the upper block diagram in the figure 14, a block at the lower diagram has been built. The directives in this case were, that the block should operate on a speed of 130 MHz and the initiation interval should be 1. With the IP Builder, there is no need to handle the critical path and to pipeline the block diagram, because the IP Builder handles this automatically according to the directives. When “Reset” is low the power average is calculated as stated in the equation 29 in section on the page 12. The incoming current sample to this block is added to the sum of all values stored in the block, before scaling, and stored to the first element of a 64 length array. The samples in the array is moved one slot forward in the array and the last sample is fetched out of the array and subtracted from the total sum. Lastly, the average is scaled with 1/64 (bit shift with 2⁻⁶) to get the true average.

![Block Diagrams](image)

**Figure 14:** The average power calculation as an IP Builder block.

### 4.3.5 The DSIC Core Block Diagrams

The next figure 15 represents filter_core.vi, that is seen as “DC CORE” in the figure 12. In this VI the SI cancellation is done. After reset, it takes time to read the nonlinear
basis function values from the BRAM, before the cancellation can begin. Therefore, there is a counter of 64 clock cycles before the cancellation can be started. The counter in figure 10 could probably be removed and the start signal in there could possibly be taken from the “Reset done?” signal in the figure 15, which indicates, that the “READ BASE” block is ready to operate with the signal to be cancelled. The DSIC stays in the hold state while there are no input samples. The states of the canceller is handled by state_cycle.vi shown in the figure 16.

**Figure 15: The SI cancellation core VI filter_core.vi, that does the SI cancellation.**

When the “Sample ready?” signal arrives, the case structure starts to send the states “0”, “1”, “2” and “3” forward on consecutive clock cycles. From the state “3”, it will enter into the “hold” state, unless the “Sample ready?” signal is true on the next clock cycle. The circuit also handles the resetting of the output calculation and the coefficient (or tap, as in picture 15 updating by sending the “reset” state, when the “Reset?”” signal is active.
Figure 16: The state_cycle.vi handles correct states for the output calculation and for the filter coefficient updater.

Figure 17 represents the circuit read_memory.vi to read the nonlinear basis functions from the block memory. The case structure has the states “1”, “2”, “3”, “4”, “hold” and “reset”.

Figure 17: Block diagram read_memory.vi, that handles the reading of the nonlinear basis functions from the block memory.

In figure 17, the most complicated state “4” is shown, and in figure 18 the simplified case for the other numbered states. After the “Reset?” is set to low, the circuit reads the basis functions used to calculate the first output sample of the canceller. Before the first sample arrives to be cancelled (“Sample ready?” is set to true for the first time), 13 future nonlinear basis functions of each nonlinearity order must be in memory, because the filter memory is ±13 samples in addition to the current sample. These basis func-
tions are in BRAM blocks starting from memory index 0 for all orders. Because the
BRAM memory reading delay is 2 clock cycles, when reading address with index 14, 13
first samples of all orders has been read from the memory into the output of this block
diagram, and the “Reset done?” is triggered up. The “Sample ready?” output is high al-
ways when there are samples coming from the memory, but “Round completed?” is
only true, when the highest order (state “4”) has been read and the cancellation has been
started. The “Sample index” output signal is used for debugging purposes.

![Diagram 18](image)

*Figure 18: The case structure for other states "1", "2" and "3".*

In the figure 19, the samples read in the read_memory.vi (“READ MEM” in the picture)
are put into the first element of an array length 108 elements always, when sample is
ready from the block “READ MEM”. The signal “Round completed?” of the “READ
MEM” output controls, when the next basis functions are updated to the output “Bases”.
The read_bases.vi is seen in the figure 15 as the block “READ BASE”.

![Diagram 19](image)

*Figure 19: The read_bases.vi inserts basis functions coming from the memory into an
array to be used in calculations.*
The “U64 to CFXP” block converts the U64 samples into a complex fixed-point. The complex fixed-point is a cluster of two fixed-point numbers of accuracy of 25s8. The block diagram of this U64toCFXP.vi is shown in figure 20.

Figure 20: Convert BRAM element in format U64 into complex fixed-point number.

The input signal reading block is described in the next figure 21. It converts one U32 into two 16s1 fixed-point numbers. Then it combines the I and Q signals and a debugging index information as a cluster into the output “Rx”. The read_Rx.vi is seen as a “READ RX” block in the figure 15.

Figure 21: The read_Rx.vi reads the samples arriving from the receiver.

Signals coming from the “READ RX”, the “Rx”; and from the “READ BASE”, the “Bases”; are send to the “CANCEL SI D17” (delay 17 clock cycles) block in the figure 15. The signal “Next ready?” of the “READ BASE” controls the “STATE CYCLE” block, which handles the states of the “CANCEL SI” and the “UPDATE TAPS” blocks. The “CANCEL SI” block diagram is shown in figure 22.
Calculations are made with vectors of length of 108. To save the DSP blocks, vectors are divided into 4 vectors of length of 27. Therefore, the cost of 4 clock cycles and 4 times increased loop speed, only 27 complex multiplications can be made in “MULT + SUM 4x” on one clock cycle. This means, that $3 \times 27 \times 4 = 324$ (one complex calculation reserves 4 DSPs) out of the total of 840 can be saved. First vectors “Taps” and “Base in” are divided into length of 27 arrays to be multiplied on separate clock cycles and are combined into a single cluster. This cluster is send to “MULT SUM 4x”, which is shown in figure 23. At first, the multiplication and summing the multiplied results occurs in “MULT SUM” as in figure 24.

In the figure 24, the data stream is send to the “CMUL D4” with an index number for elements to be fetched from the input vectors. The complex number multiplication is shown in figure 25, where the basis functions are multiplied by the filter coefficients. Because of the optimal usage of DSP blocks, the filter coefficients have to be rounded into the accuracy of 18s1. The DSP48E block of NI PXIe-7962R handles calculations with input bit lengths of 25 for the other value, and 18 for the second to be multiplied. This way the total usage of DSP slices for one complex calculation will stay in four units. With either value having longer bit length the number of slices used would raise.

After the multiplication in figure 24, there is a series of sum operations to sum all multiplication results together. When there is more than one pair to be summed, the “SUM PAIRS” block is used, which is represented in figure 26. It adds element with index $2^i$ with element $2^i+1$. The $i$ is the current loop count.
Figure 23: Block of vector_multiply_sum_4x.vi to calculate sum of vector sums calculated in MULT + SUM block.

Figure 24: Multiplying and adding the results together in vector_multiply_sum.vi. It is MULT + SUM in previous figure.
Figure 25: Block diagram to fetch values from vectors at the position pointed by the signal "index" in single_multiply.vi

Figure 26: Add pairs from array in add_array_pairs.vi.

The add_array_pairs.vi is a simplification for the block diagram in figure 24, to make the code more readable. Labview FPGA does not directly allow variable size arrays, even thought the fixed size would probably be easily calculated on the precompilation time. To achieve better code to read and to better fit the code into A4 size, unrolled for-loops from the original vector_multiply_sum.vi are hidden into add_array_pairs.vi in the figure 24. To achieve compilable code, this subVI should be unrolled in the vector_multiply_sum.vi before compilation. Other option is to do an own subVI for each instance of add_array_pairs.vi in vector_multiply_sum.vi, where only the input array “Array in” size is fixed into the size of 2**”Number of pairs”.

Multiplication results of 27 values forms 13 pairs to sum, and one element remain odd. These 13 results forms 6 pairs to sums and one odd. Now the element left at the first stage is added to this other odd element. After the next sum of 3 pairs, again one odd remains and that is summed with the sum of the first odds. Lastly, the remain pair of sums are summed with each other.
After calculating the first sum of first 27 multiplications in the “MULT + SUM“ block in figure 23, the rest of the vectors are also calculated into single sum in the following clock cycles. Then the circuit after the “MULT + SUM” sums the results of the 4 previous summations together. When the state is “0”, input sum is added with a value of 0, because it is the first sum. On other numerical states, the input sum is summed to the current total sum. On the “hold” state, the current sum is summed with 0 and in the “reset” 0 is added with 0.

Using the IP builder tool, and to recode the code in figure 24 with that tool, would possibly result in more dynamical code to edit, if changes are required into code in the future. However, it would require time and debugging to do the changes. Because, this version is working, it is used until recoding is required.

After the cancelling signal has been calculated in the figure 22, it is subtracted from “Rx” signal to achieve the cancelled SI signal. This signal is send to the parent filter_core.vi, which converts fixed-point complex into U32 and sends it to the host PC through the parent VI filter_base.vi. The cancelled signal is also used in the filter update process, that occurs in the block “UPDATE FILTER” in the figure 15. Update_filter.vi block is shown in figure 27.

![Figure 27: Block diagram update_filter.vi to update filter coefficients.](image)

The basis functions and canceller output samples has to be in sync with the input samples arriving from the receiver. Therefore, there are 17 feed-forward registers in figure 15 to match the delay of the calculation of the cancelled signal. As when calculating the cancelled signal, the basis functions are divided into 4 length of 27 arrays to be cal-
culated on separate clock cycles. Because, the cancelled signal is only a one sample, a vector of length of 27 is formed from that sample, where each element has the same value. These vectors and the state information is combined into a single cluster and that is fed into the “UPDATE TAPS” block, which is shown in figure 28. That cluster and the index in the vectors are inputs for the “UPDATE TAP D4” block.

Figure 28: The block diagram for update_taps.vi.

Figure 29: Calculating new value for one tap in update_tap.vi.
Values are fetched from the vectors according to the element index. The basis functions in accuracy of 25s8 are multiplied with the cancelled signal in accuracy of 16s1. After that, the resulting error signal is scaled down by shifting bits to the right depending on the step size. The resulting small update factor is then added with coefficients having accuracy of 25s1. The updating value is very small, thus the accuracy of the filter coefficients have to be better, than when calculating the canceller output, where the coefficients are rounded into 18s1 precision. When new taps are calculated, those are tied together into a cluster in figure 28. After update, the length of 27 arrays are combined into a single 108 length array. A complex conjugate is taken before it is send to the “CAL-CEL SI D17” block in the figure 15, because complex conjugate of the coefficients are used, when calculating the cancelled signal.
5. MEASUREMENTS AND RESULTS

This chapter represents the equipment used, the usage of DSIC program and measurements taken. The main measurements were taken with equipment represented in 5.1. The next sub chapter 5.2 focuses into the usage of the program and 5.3 to acquiring information about real-time digital SI cancelling process. At first, simple direct cable measurements are introduced in 5.4 and 5.5 demonstrates the main measurements.

5.1 Measurement Setup

To get the wanted results, it requires several equipments to be attached into the measurement setup. Figure 1 in the chapter 2.1 presented the whole transmission chain. Figure 30 shows the real equipment setup.

![Figure 30: Measurement setup.](image)

In the right, there is a chassis of National Instruments (NI), PXIe-1073, which includes the transceiver NI 5791 + FlexRIO NI PXIe-7962R combination and FlexRIO NI PXIe-
7972R handles the DSIC operation. Transmitter is connected into a PA by Texas Instruments, CC2595. After the PA, signal is split into four outputs from which three is required for the RF-canceller and the last is attached into circulator JC-C2300T2500S6. The antenna port of the circulator is connected into a whip antenna and the output port back into the RF-canceller. From the RF-canceller, a SMA cable leads into the receiver. Functioning of this setup is better described in the section 2.1. The RF-canceller also needs a local oscillator Hewlett Packard E4437B, a low noise amplifier HD Communications Corporation HD24089 and a Hewlett Packard E3631A and two power supplies: Aim-Tti and PL303QMD-P. Further details about the RF-canceller can be read from the Master’s thesis of Tamminen. [19] Vector signal transceiver NI PXIe-5645R was used as a reference measurement equipment.

The DSIC can operate also independently with only a direct cable with attenuator between transmitter and receiver or with circulator and antenna. Direct cable method was used during development process and the resulting SI cancellation result without the other SI cancellation stages, is shown in subchapter 5.4.

5.2 Using the Program

Program works in three stages. At first, transmit power and receive reference levels has to be set accordingly by current attenuation and RF-cancelling. Next the right signal phase for the received signal has to be found for to acquire the maximal signal strength. The program calculates from the received signal travelling unaltered past the DSIC, if the delay is found and reports results to the user. And in the last stage the DSIC is started by the user. This process is shown in following figure 31.

![Figure 31: Adjusting program parameters for maximal cancellation of SI.](image)

Next figure 32 shows parameters possible to be set for the program. The other half on the front panel includes information about the transceiver and the status of DSIC is shown in figure 33.
Figure 32: Picture from the left side of the front panel. There are settings for transceiver and DSIC.
Figure 33: The other half of the front panel, which includes measurement data and program state information.

The search of the phase must be performed always, when the equipment or the transmit power is changed, because these affects the sample phase at the receiver. If the phase is affected largely by the signal path, it may also affect the sample delay. Normally this delay change does not affect functioning of the canceller, because the program calculates it. However, if delay is set manually, then it must be taken into consideration.
When the program is started, transceiver receives parameters set by the user and the transmission and reception starts simultaneously. Transmitter is programmed to send the 10000 sample LTE signal constantly, always starting from the first sample, after the last one has been send. The received signal is send to the host PC past by the DSIC, if the delay is not yet found and set. The host PC calculates cross-correlation between the signal received to the signal send. If host finds a positive spike in the correlation, which exceeds certain level in respect to the average random noise power of the correlation, the delay from the transmitter to the receiver is has been found and calculated. If the phase is not right enough, the delay will not be found and the phase has to be adjusted. It has to readjusted until the delay is found. This process does not require the program to be restarted.

When the correct phase and delay has been found, the program must be restarted for the DSIC to work. This is because, there is a bug somewhere in the code, that prevents cancellation from not to work, if the phase has been altered while running the program.

After setting the phase correctly and the program is restarted. The program first works as stated before. When the program has calculated the delay, it is send to the DSIC FPGA. When the DSIC receives the delay, it will wait the delay amount of samples from the beginning of the next LTE signal round. Now signal is synchronized and SI cancelling is started. After the start, the stream from this FPGA into the host PC switches to the SI cancelled signal. If the cancelling works, stream power received by the host, should start converging near to zero. The convergence is too fast for human to see it and it can be seen only in the records made.

5.3 Other Program Properties

The DSIC also calculates the signal powers before the digital cancelling and after it. This is done with a running average from 128 samples, as stated in formula (29) in chapter 3.3. These linear values are send to the host, which first scales those into the correct power range and then transforms them into logarithmic power decibel values as in formula (30). Difference between these powers gives information of how effectively the DSIC is working.

It is also possible to acquire information about the internal functioning of the DSIC. All the filter coefficients are memorized at the same clock cycle, and by default, send at the every 4th clock cycle to the host PC. This speed can be altered to be faster, however, then the sending of the canceller output signal must be stopped, because the chassis bus is not able to handle all the combined streams simultaneously. After each of these coefficients have been send to the host a new sample of those are taken and the transmission of those starts over. With the default speed, these snapshots are taken in every 4*108=432th clock cycle. It is also possible to switch this acquiring to only a single coefficient. Benefits of these methods are, that it is possible to monitor how coefficients
change over time while the environment changes near the antenna. The order in which these coefficients are transmitted is the same as in formula (7) in chapter 3.1.

5.4 Direct Cable Measurements

Direct cable was mainly used during the development phase. A SubMiniature version A (SMA) cable of length of 20 cm was connected from the transmitter directly into the receiver with no extra attenuation and antenna as seen in next picture.

![Image of the transceiver and DSIC with direct cable.](image)

Figure 34: The Transceiver and DSIC with direct cable.

With current canceller configuration, where the first 4 odd orders of the nonlinear basis functions with memory length of 27 is implemented, 39.5 dB attenuation is achieved. Figure 35 represents the output of the canceller, when it is off and after it is switched on at the sample index around $6 \times 10^5$. The blue line is an instant signal power. The strong spikes seen in the figure with a period of 10000 samples is probably a discontinuity point in the LTE signal. This has not been investigated, what the source is, properly.
Figure 35: Output power before and after the beginning of digital cancellation.

In the figure 35, the red line represents mean power, which when transmitting with 0 dBm power, and at the beginning, when no digital cancellation exists, it is signal peak to average ratio (PAPR) valued to -11.6 dB. When digital cancellation begins, after 200 000 samples the output mean power reaches -51.0 dB which means total of 39.4 dB attenuation when SI is cancelled only with the DSIC.

5.5 Measurements with RF Canceller and Antenna

The main set of measurements were conducted with the RF-canceller, the circulator and the antenna attached between the transmitter and the receiver, as described in chapter 5.1. There is another article from our team [11] describing these results also. Following picture shows different frequency components of the signal at different stages. Transmit power is set to 7dBm. With this power the overall efficiency of the canceller stages performs optimally and achieves the best combined cancellation.

In figure 36, real performance of the RF- and digital cancellation at different frequencies can be seen. The blue is the transmit signal at different frequencies, the black after the RF-canceller, the green after digital cancelling and the violet is the receivers noise floor. Without corrections, -61.2 dBm SI is left after the cancellation stages. However, there is a spike near 0 frequencies, that is possibly resulted by a direct current offset at the receiver. There are also other frequency components outside the LTE band and additive white noise. The source of the white noise is not known.
By applying a bandpass filter to the data after the both RF-cancellation and digital cancellation, the real LTE band signal cancellation strength can be acquired. The passband of the bandpass filter is between $0.3 - 10$ MHz for positive frequencies as stated in following figure 37.

The white noise left to the signal after the digital cancelling on the LTE band, can be filtered away by averaging the signal. Because consecutive 10000 samples signal blocks transmitted are equal with each other, by averaging signal with multiple sequential blocks, white noise can be attenuated and the real performance of the DSIC can be obtained, as seen in the next figure 38.
Following results have been calculated from the measured data with different the transmit powers and after applying the bandpass filtering to the data. This data in table 2 is better illustrated with following figures 39 and 40.

Table 2: Signal powers and performance of the DSIC in different stages.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>8.0</th>
<th>3.0</th>
<th>2.0</th>
<th>7.0</th>
<th>12.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit signal power to antenna (dBm)</td>
<td>-75.8</td>
<td>-68.5</td>
<td>-60.8</td>
<td>-55.2</td>
<td>-51.2</td>
</tr>
<tr>
<td>RF canceller output (dBm)</td>
<td>-85.4</td>
<td>-83.9</td>
<td>-80.4</td>
<td>-75.8</td>
<td>-67.7</td>
</tr>
<tr>
<td>SI after DSIC with the unknown noise (dBm)</td>
<td>-99.5</td>
<td>-97.6</td>
<td>-95.1</td>
<td>-89.9</td>
<td>-81.4</td>
</tr>
<tr>
<td>SI after DSIC without the unknown noise (dBm)</td>
<td>67.7</td>
<td>65.5</td>
<td>62.7</td>
<td>62.2</td>
<td>63.2</td>
</tr>
<tr>
<td>DSIC performance with the unknown noise (dB)</td>
<td>8.6</td>
<td>15.3</td>
<td>19.7</td>
<td>20.6</td>
<td>16.5</td>
</tr>
<tr>
<td>DSIC performance without the unknown noise (dB)</td>
<td>23.8</td>
<td>29.1</td>
<td>34.4</td>
<td>34.7</td>
<td>30.2</td>
</tr>
<tr>
<td>Unknown noise (dB)</td>
<td>14.2</td>
<td>13.8</td>
<td>14.7</td>
<td>14.1</td>
<td>13.8</td>
</tr>
</tbody>
</table>

Figure 38: Frequency components of canceller stages after bandpass filtering and averaging.

Figure 39: Received signal after RF- and digital cancellation.
below about 7-8 dBm. The next figure describes the different performance characters of the cancellers with differing transmit power.

![Graph showing performance of canceller stages with different transmit powers.](image)

*Figure 40: Performance of the cancellation stages with different transmit powers.*

As can be seen from the table 2 and from the figure 39 possibly the RF-canceller implementation causes nearly constant 14 dB extra noise regardless of the transmit power into the resulting signal after the DSIC.

From the figures 39 and 40 it can be deduced, that the DSIC cancels the best, when residual power is near noise floor. With the lower transmit powers the accuracy and the noise floor of the receiver limits the performance of the digital cancellation. When the average transmit power raises over 8 dBm, clipping in the NI 5791 causes distortion, that the RF-canceller or the DSIC can not attenuate.

Figure 41 represents the average coefficient powers over time from two different measurements. It can be seen that filter coefficients near zero delay are the most important and that the non-linearity order 1 is the most important, when attenuating the SI.

![Graph showing tap power on two separate measurements.](image)

*Figure 41: Tap power on two separate measurements.*
In figure 42 readjustment of the filter coefficients over time can be seen when fast rotating pendulum is being held near the antenna. This pendulum alters the environment and electrical characteristics near the antenna, and therefore, the DSIC has to readjust to this change. The current implementation of the DSIC updates the filter coefficients at the speed of 26 MHz, however, as seen from the figure 42 slower update speed could be used.

*Figure 42: Readjustment of 5 strongest and most important filter taps over time.*
6. CONCLUSIONS

This work demonstrated the structure of a real-time DSIC in the context of a complete in-band full-duplex radio demonstrator, as a Labview FPGA implementation. Labview is a graphical programming language, and learning that proved to be more difficult and slower, than learning a new textual based language for a person used to work with textual languages. Also, the lack of advanced programming structures, e.g. multidimensional arrays, when programming the FPGA with the Labview, made the coding more demanding for the developer. However, the developed real-time implementation worked as expected by the simulations, except for the unknown 14 dB white noise, that had to be averaged away from the output signal of the canceller. After the circulator, the RF-cancelling and averaging the output, maximum of about 34 dB digital attenuation is achieved, when the transmit power is 7 dBm and the LTE signal bandwidth is 20 MHz. This is a rather expected performance, if it is compared to a 46 dB of non-realtime results [13]. The difference of 12 dB can be explained by the longer memory length (79 compared to 27) and 5 more nonlinearity orders (orders 9, 11, 13, 15 and 17) in addition to 4 first odd orders of the implementation. When working alone with a direct cable between the transmitter and the receiver, DSIC was able to cancel 39 dB, thus adding the low-cost PA and other canceller stages affects the DSIC cancelling efficiency by about 5dB.

For the future development, the canceller algorithm demands a lot of resources from the FPGA. The FPGA used in this project, with low resources, made the implementation even more difficult task. Primary goal was to have the 6 first odd nonlinearity orders (orders: 1, 3, 5, 7, 9 and 11) and with filter memory length of 31 or more for all orders, but the resources available, limited the size of the canceller. With the 4 first odd orders and with the filter memory length of 27 samples and with the optimization of resource usage, the canceller fits to the FPGA used. To save the DSPs, the main-loop had to run 5 times the signal speed, that less DSPs could be used by reusing those at different clock cycles. The extra complexity and the higher main-loop speed, that this DSP saving process created, caused other FPGA resources to diminish (FPGA slices). Future implementations could find the optimal usage of both resources, which in this work are reported in the table 1 on the page 21. The next version of the canceller could also have a slower filter coefficient update rate, because changes in the environment are very slow compared to the speed of the signal as can seen in the figure 42 in the page 52. With slower speed, less pipelining registers are required or less DSPs could be used, and resources would be saved for the other tasks. Also lowering the main-loop clock speed from 130 MHz into something greater than 104 MHz, could save some of the pipelining
registers to be used elsewhere. Currently, the main-loop can handle streams at the sample rate of 32.5 MHz and the requirement is just a little bit faster, than the signal at the sample rate of 26 MHz. This is for that the data would not accrue into the chassis bus buffers.

The best solution to increase the size of the DSIC real-time implementation would be using a FPGA, that has more resources. With more resources available, the structure could be simplified. Having more resources would also make it possible to add higher nonlinearity order filter stages, and use more memory samples in the filter for the cancellation. Also, the pre-calculated nonlinear basis functions could be implemented and calculated in real-time for a random LTE signal, if a bigger FPGA is used. The current solution prevents constantly altering signals.
LÄHTEET


