Diplomarbeit

Development and Setting Up of a 4×4 Real-time MIMO Testbed

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter Leitung von

DI Ernst Aschbacher und Prof. Dr. Markus Rupp
E389
Institut für Nachrichtentechnik und Hochfrequenztechnik

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Sebastian Caban
Matrikelnummer: 0025464
Adresse: Zanaschkagasse 12/31/30, 1120 Wien

Wien, Juni 2004

I hereby certify that the work reported in this diploma thesis is my own, and the work done by other authors is appropriately cited.

Sebastian Caban
Vienna, June 1, 2004

Abstract

Research on multiple antenna systems is mainly theoretical. A lot of work has been done to prove their operation on a mathematical basis — but now it is time for an experimental setup to demonstrate in *real-time operation* whether they work in an realistic scenario.

For this purpose a testbed has been developed and set up which can serve up to four transmit and four receive antennas simultaneously. It consists primarily of two PCs, one containing the transmit hardware and the other containing all the receive hardware. The user who wants to transmit data interfaces this system via a network connection by the use of simple MATLAB commands. The developed interface is *very* easy to use, powerful, and platform independent. Furthermore, the user does not need to perform any hardware programming.

Using this powerful testbed, it is very easy to add a real-time transmission to existing MATLAB simulations by just simply inserting a few MATLAB commands. Transmissions via air or channel emulators will enable new possibilities of DSP code development and testing of, e.g., UMTS, OFDM, and MIMO systems.

The testbed can be extended by DSP/FPGA modules. It is possible to use these modules in a separate PC to perform testing of coding techniques in a real-time hardware environment. They also interface the transmit and receive hardware via a network connection and simple MATLAB commands.

First impressive results using the developed MIMO testbed have already been achieved by Christian Mehlführer. His work shows exactly matching with existing simulation results.

Zusammenfassung

In den letzten Jahren wurden Mehrantennensysteme vor allem theoretisch erforscht. Es wurde sehr viel unternommen, um ihr Funktionieren auf einer mathematischen Grundlage zu zeigen — doch nun ist es daran in *Echtzeit* zu zeigen, ob all dies auch in einem realistischen Szenario funktioniert oder nicht.

Zu diesem Zweck wurde ein Testsystem entwickelt und in Betrieb genommen, mit dem bis zu vier Sende- und Empfangsantennen gleichzeitig synchron bedient werden können. Das System besteht im Wesentlichen aus zwei PCs. Einer beherbergt die Hardeware zum Senden, der andere die zum Empfangen. Der Anwender, der seine Daten übertragen möchte, kommuniziert mit diesem System über eine Netzwerkverbindung und wenige einfache MATLAB Kommandos. Die hierfür entwickelte Schnittstelle ist *sehr* einfach zu verwenden, sehr leistungsfähig und plattformunabhängig. Des weiteren benötigt der Benutzer keine Kentnisse über Hardwareprogrammierung.

Aufbauend auf dieser Grundlage ist es vergleichsweise einfach, eine bestehende MATLAB Simulation um eine Echtzeitübertragung zu erweitern. Schon ein paar einfache MATLAB Befehle genügen für eine Übertragung über Luft oder Kanalemulatoren. Dies eröffnet neue Möglichkeiten zur DSP Codeentwicklung und -testung, z.B. für UMTS, OFDM und MIMO Systeme.

Darüber hinaus ist es möglich das Testsystem um DSP/FPGA Module zu erweitern. Dies ermöglicht es einem Benutzer Detektionsverfahren, Kodierer und Detektoren in einer Echtzeitumgebung zu testen. Mit dem MIMO Testsystem kommuniziert der Benutzer weiterhin mittels einiger einfacher MATLAB Befehle über eine Netztwerkverbindung.

Erste beeindruckende Resultate mit diesem Testsystem wurden auch schon von Christian Mehlführer in seiner Arbeit vorgestellt. Sie zeigen eine exakte Übereinstimmung mit bereits vorhandenen Simulationsergebnissen.

Contents

1 Introduction **1**

2 The MIMO Testbed **3**

2.1 System Overview 3

2.1.1 The MATLAB INTERFACE 5

2.2 Setting Up the MIMO System 7

2.2.1 Setting Up the MIMO Servers 7

2.2.2 Setting Up the External Hardware 9

2.3 Transmitting Data via the MATLAB INTERFACE 11

2.3.1 Creating the Complex Baseband Data Samples 11

2.3.2 Triggering the Operation of Pollux 13

2.3.3 The Automatic Digital Up-conversion 14

2.3.4 The Analog Intermediate Frequency Signal 16

2.3.5 Example: Transmitting a 16 QAM 18

2.3.6 Example: Signal for a Multitone Measurement 19

2.3.7 Transmitting Multiple Blocks of Data 20

2.4 Receiving Data via the MATLAB INTERFACE 22

2.4.1 Handshaking of Pollux and Castor 23

2.4.2 The Automatic Digital Down-conversion 27

2.4.3 The Complex Baseband Data Samples 30

2.4.4 Example: 16 QAM 32

2.5 "xvTXDataOptions" 34

2.5.1 Operation Mode 35

2.5.2 TX Number of Repetitions 36

2.5.3 TX Center Frequency 37

2.5.4 TX Interpolation Factor 37

2.5.5 TX External Clock 37

2.5.6 TX Delay Between Blocks 38
2.5.6.1 Using the C++ Sleep(...) Function 38
2.5.6.2 Using the CPU's High Performance Counter . 40
2.5.7 RX Number of Samples 41
2.5.8 RX Number of Channels 41
2.5.9 RX Number of Repetitions 41
2.5.10 RX Center Frequency 42
2.5.11 RX Interpolation Factor 42
2.5.12 RX Filter 42
2.5.13 Flags 43
2.5.13.0 Flag 1: TX Continuous Mode 43
2.5.13.1 Flag 2: RX Continuous Mode 44
2.5.13.2 Flag 4: TX do not Delete .mat File 44
2.5.13.3 Flag 8: RX Use .dat File 45
2.5.13.4 Flag 16: Create .received File 46
2.5.13.5 Flag 32: TX force HDD operation 47
2.5.13.6 Flag 64: RX Force HDD Operation 47
2.5.13.7 Flag 128: Sync via File 47
2.5.13.8 Flag 256: RX Check CW Data 47
2.5.14 The Maximum Number of Samples per Block 48
2.6 Interfacing the MIMO Testbed via FTP 48

3 Extending the Testbed 49

Conclusion 52

A Picture of the ICS-564 board 53

B Picture of the ICS-554 board 54

C Picture of the SMT-365 board 55

List of Abbreviations 57

Bibliography 58

Chapter 1

Introduction

All over the world, a lot of research has already been performed to investigate the behavior and possibilities of multiple antenna systems but unfortunately, most MIMO[1] algorithms are tested only theoretically. A lot of symbolical calculations and numerical simulations (e.g. in MATLAB[2]) describe the anticipated behavior of multiple antenna transmissions. But actually it is not exactly known, whether all these results will stand the test of a real air transmission.

MATLAB can be used to perfectly simulate the behavior of digital transmitters and receivers. Every transmission needs a channel and, up to now, there is no way to reflect a real world scenario perfectly in a simulation. A lot of approaches exist to describe extremely simplified versions of the real world in form of channel models. Due to a lack of computing power and knowledge, these models are quite simple while still covering all the important aspects of a transmission. They are continuously optimized but based on their assumptions they remain models, not describing exact transmission behavior.

In order to obtain realistic behavior, the physical channel should be used instead. Therefore, the intention of this diploma thesis was to develop and set up a MIMO testbed which is able to carry out transmissions through a physical channel.

A user, investigating coding and decoding schemes, is usually not able to take care of hardware developing and setting up due to a lack of time and specialized knowledge; therefore, he must be able to use the testbed easily with nearly no effort directly out of an environment he knows well, e.g.

[1] MIMO = Multiple Input Multiple Output, more than one antenna is used
[2] MATLAB is a trademark of MathWorks Inc.

MATLAB. Multi user operation, LAN operation, and extensive data storage are important requirements to achieve this goal.

Chapter 2 of this thesis describes the setup of the developed MIMO testbed (see Figure 1.1). An introduction into the parts of the testbed is followed by a detailed description on how to use it. Code examples and a list of all operation modes round up this chapter.

Chapter 3 describes an extension of the testbed by DSP[3]+FPGA[4] boards. Using these boards enhances the possibilities of the MIMO testbed significantly by introducing a real-time environment for DSP code development and testing.

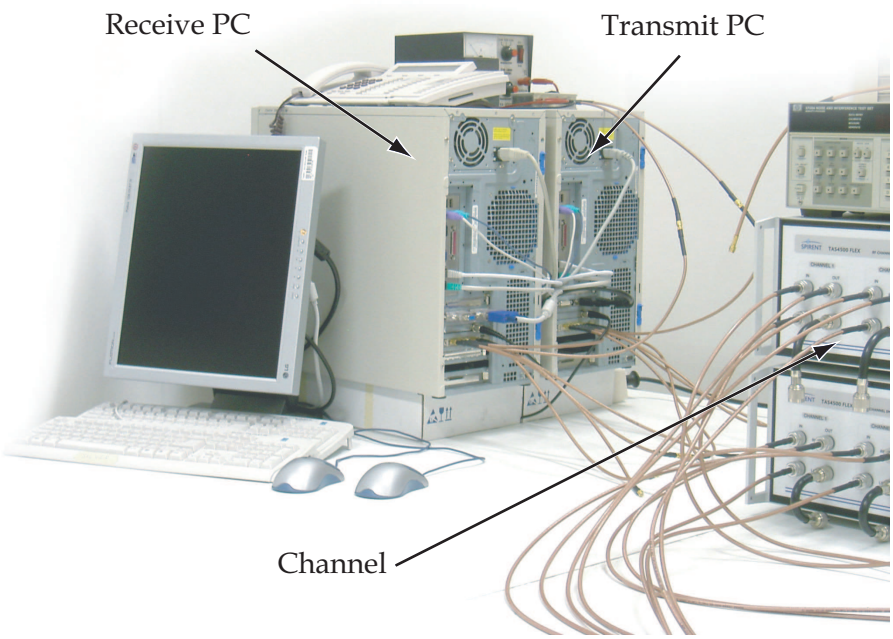

Figure 1.1: Picture of the MIMO testbed.

[3]DSP = Digital Signal Processor
[4]FPGA = Field Programmable Gate Array, a user programmable logic device

Chapter 2

The MIMO Testbed

The developed MIMO testbed is optimized to quickly transmit and receive data over an RF channel directly out of MATLAB with minimum effort. It can be used to implement, test, and optimize signal processing algorithms for MIMO transmission using any modulation scheme. Detailed knowledge of the hardware and hardware programming is therefore not needed. This reduces code development time dramatically and enables multiple users to operate the same *very* expensive hardware.

This chapter first presents a short overview about how the testbed is built up. Since the overall system is very complex, only the basic hardware information which is needed to understand how the overall system works is given. Next, it is explained how to interface this testbed by using MATLAB in order to transmit and receive data. Some code examples and a summary of all the different opportunities of this testbed end this chapter.

2.1 System Overview

The main task of the MIMO testbed is to transmit and receive user-data over an RF channel. This user-data is generated by a user PC which is connected to the testbed via a LAN[1] connection. The whole system is made up of four main parts as shown in Figure 2.1.1.

Furthermore, the overall testbed can be extended by DSP/FPGA boards located in the "user PC" (see Figure 2.1.1) to implement and test signal detection algorithms in an realistic real-time scenario. This extension of the testbed will be described in Chapter 3.

[1]LAN = Local Area Network, connection between computers

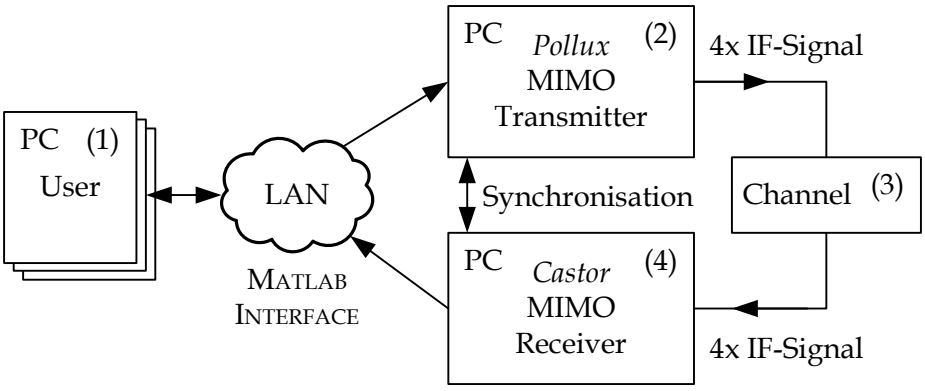

Figure 2.1.1: The MIMO testbed.

The MIMO testbed consists of[2]:

1. one or more **user PCs** where the baseband signals are generated. The so called MATLAB INTERFACE (Section 2.1.1) is used to transmit this data to the other parts of the system via a LAN connection and by the use of MATLAB commands.

2. a server PC, named **Pollux**, which gets the complex baseband data samples from the user PC and automatically up-converts them to band-pass IF samples. After a digital to analog conversion the PC finally transmits the IF signal over the channel (see Section 2.3.3). This PC also does all the handshaking needed for the MATLAB INTERFACE.

3. a **channel** which consists of IF to RF up-converters, RF to IF down-converters, and RF channel emulators or a real air interface. These parts were developed by Robert Langwieser [4] and Lukas Mayer [5] in their diploma theses for an IF of 70 MHz and an RF of 2.45 GHz. Figure 2.1.2 shows this channel which is not part of this diploma thesis and is therefore considered as a working black box.

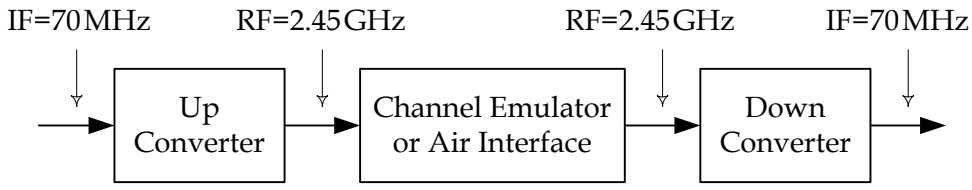

Figure 2.1.2: RF channel for the MIMO testbed.

[2]The item numbers correspond to the numbers in Figure 2.1.1

4. a server PC, named **Castor**, which receives the bandpass IF signal from the channel, converts it to the digital domain and automatically down-converts the resulting digital samples from IF to baseband. Finally the PC stores the complex baseband data samples on the internal hard-disk so that the user can read them via the LAN (see Section 2.4.2).

While maintaining considerable performance and flexibility, the system was designed to provide a *very* easy to use interface. To achieve this main objective, MATLAB (in combination with C++) has been used as programming language for the server PCs and for the user interface.

2.1.1 The MATLAB INTERFACE

The above mentioned MATLAB INTERFACE is the basic idea of the system. Using this interface, the user is able to transmit and receive data directly delivered from MATLAB from anywhere in the LAN *without* hardware programming and/or debugging.

The interface is completely implemented in MATLAB because MATLAB is platform independent[3], powerful, easy to use, and probably the most used programming language for development and testing of signal processing schemes in engineering. Furthermore, this interface uses ordinary files to communicate and handshake, not TCP/IP commands. This enables batch processing, reduces code development time, and gives the possibility to interface the MIMO System even through a firewall by using Secure-FTP (Figure 2.1.3):

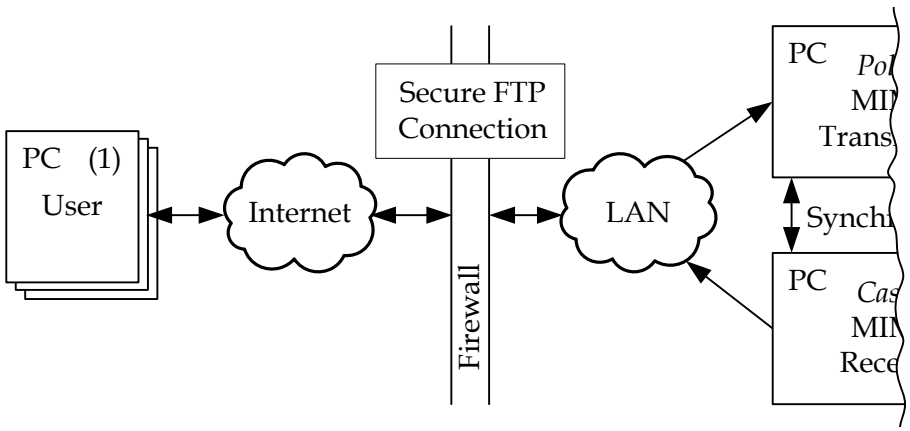

Figure 2.1.3: Interfacing via Secure-FTP.

[3]MATLAB is available for the following platforms: All Windows versions, Linux, Macintosh OSX, Solaris, AIX, Digital UNIX, HP-UX 10, HP-UX 20, and IRIX/IRIX64

Without going into further details of the hardware implementation, the operation of the MATLAB INTERFACE can be simplified to the following main sequences[4] (see Figure 2.1.4):

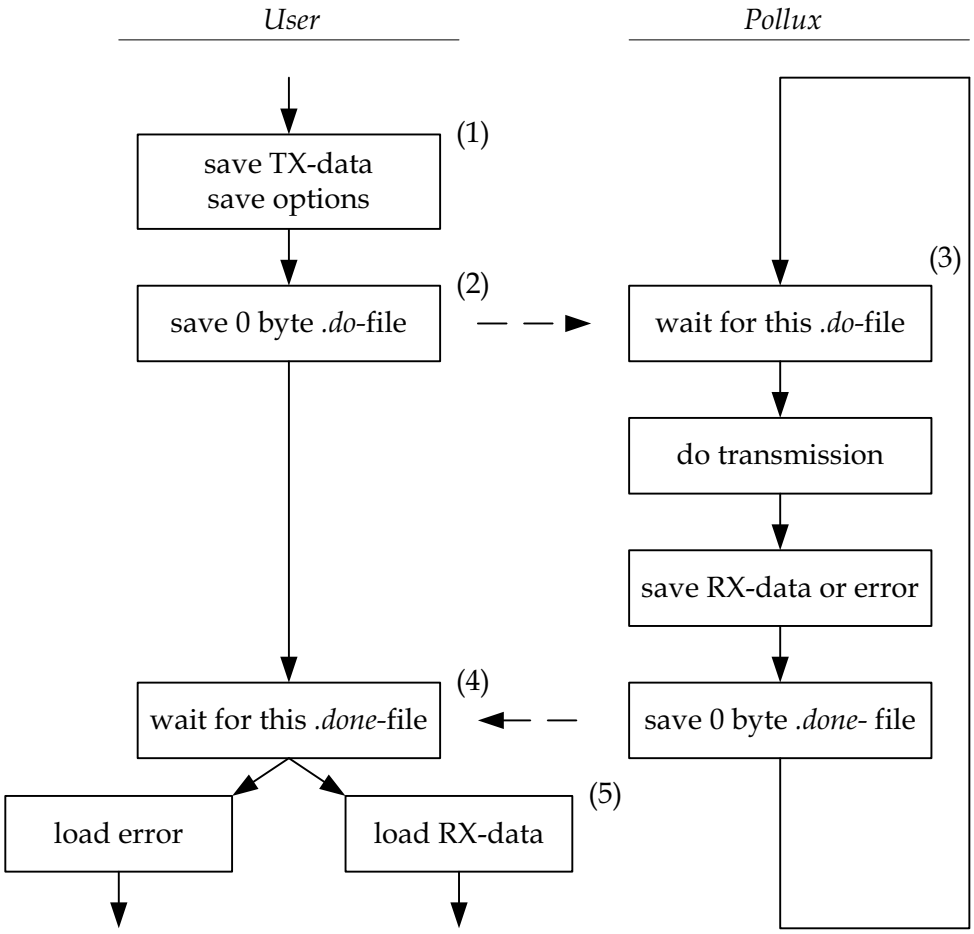

Figure 2.1.4: The MATLAB INTERFACE, handshaking.

1. The user saves the generated complex baseband data samples and an array of options (desired center frequency of the bandpass signal, etc.) using the MATLAB `save(...)` command to a dedicated folder (see Section 2.3).

[4]The item numpers correspond to the numbers in Figure 2.1.4

2. *After* completing the `save(...)` command the user saves a 0 byte large `.do`-file to the same folder. Because network file operations are always processed one after the other, it is guaranteed that the complex baseband data samples are completely saved when the `.do`-file is created.

3. This `.do`-file triggers the operation of Pollux (and Pollux triggers the operation of Castor if needed). The overall signal transmission is carried out automatically by the server software of these PCs (see Section 2.3.3).

4. The user waits (polling operation) for a 0 byte large `.done`-file in a dedicated network folder that indicates that Pollux has completed operation.

5. Now the user checks whether an error file has been written by Pollux. If such a file exists, the user loads this file containing the error messages. If not, the user loads the received complex baseband data samples using the MATLAB `load(...)` command (see Section 2.4).

This procedure can now be used to operate the whole system from a single PC via a LAN connection using MATLAB. The user is able to:

1. transmit data over a channel (without receiving anything),

2. receive data from a channel (without transmitting anything),

3. transmit data over a channel and receive it,

4. and hibernate[5] Pollux and Castor if not needed any more.

These four operation modes are much related to each other and their implementation is very similar. A detailed description and some examples can be obtained from Section 2.5.1.

2.2 Setting Up the MIMO System

2.2.1 Setting Up the MIMO Servers

Before data can be transmitted, the two host PCs have to be set up properly. As mentioned before, the transmitting PC is called Pollux and the receiving PC is called Castor.

[5]The Windows hibernate feature saves everything in memory on the hard disk, turns off the monitor and hard disk, and then turns off the computer. When the computer is restarted, the desktop is restored exactly.

The following steps have to be performed:

1. Castor and Pollux have to be powered on.
2. If clock synchronization between Pollux and Castor is needed, the external frequency divider has to be turned on. See Figure 2.2.2 for further information on how to connect this device.
3. On Pollux, the server program has to be started by double clicking at the "MIMO TRANSMITTER" icon on the desktop if it is not already running.
4. On Castor, the server program has to be started by double clicking at the "MIMO RECEIVER" icon on the desktop if it is not already running (see Figure 2.2.1).

The two server programs of Pollux and Castor look and behave very similar. User interaction is *never* required; therefore, the server programs show only a status screen like in the following Figure 2.2.1:

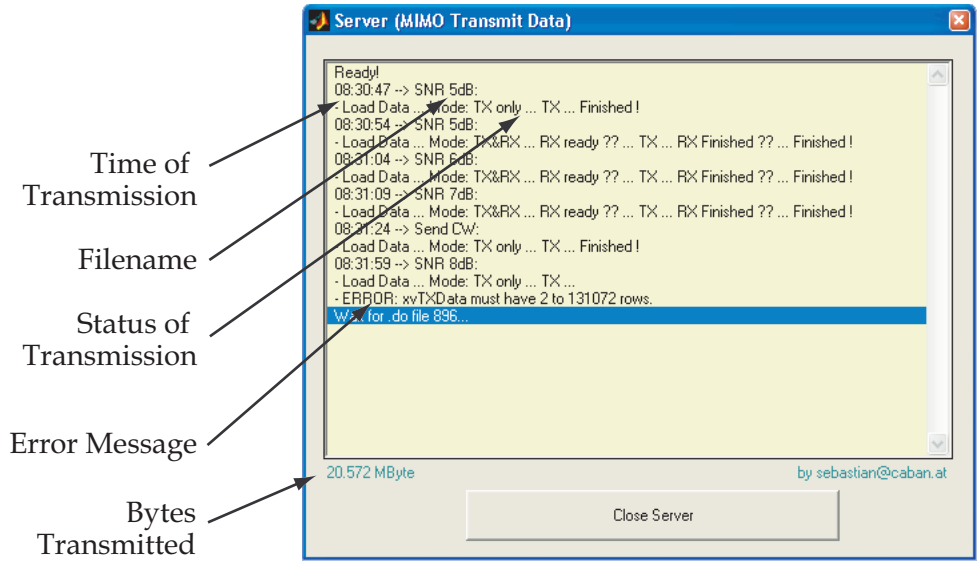

Figure 2.2.1: Server program on Pollux.

If idle, the server displays "Waiting for .do file" and increments a number next to this message. If a transmission is triggered, first the current time is displayed and then, next to it, the filename which activated this operation. The next line shows actual status information (the meaning is described in Section 2.4.1), and if an error occurs, a message is displayed.

2.2.2 Setting Up the External Hardware

Pollux converts the digital complex baseband data samples, made available by the user PC, to analog IF signals. Therefore, some external hardware has to be connected to make use of these signals. Furthermore, this hardware also has to be connected to Castor and external synchronization is needed (see Figure 2.2.2).

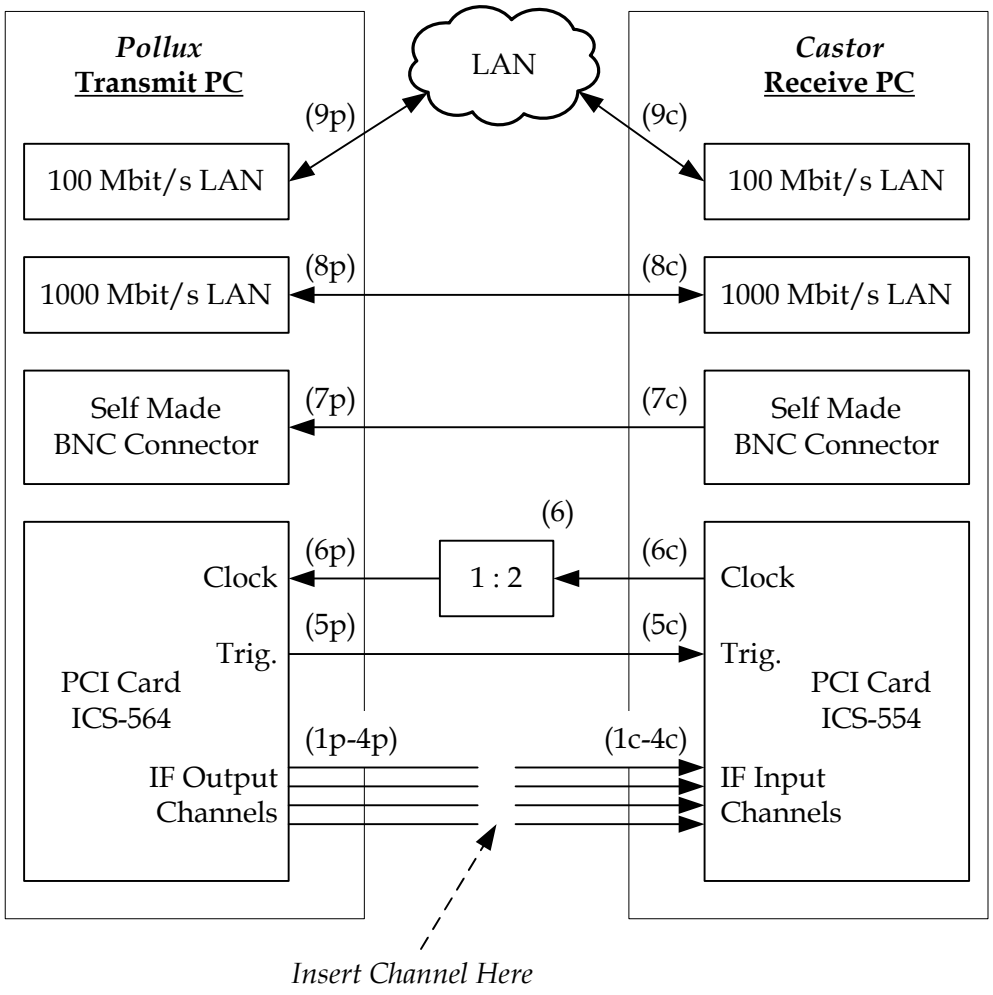

Figure 2.2.2: Connections on Castor and Pollux.

- The four analog IF outputs of Pollux (1p), (2p), (3p), and (4p) have to be connected to external RF hardware. Figure 2.2.3 shows these SMA connectors located on the back of Pollux. They are situated on the ICS-564 board [10] (see Appendix A). Full scale output is $1.2\,V_{pp}$

on a 50 Ω load. Thus the achievable mean output power in operation depends on the crest factor of the signal. The IF output signal is not filtered as shown in Figure 2.3.4.

- The four receive IF signals (1c), (2c), (3c), and (4c) have to be connected to the ICS-554 board [13] (see Appendix B) on the back of Castor (see Figure 2.2.3). The maximum input voltage is $1.2\,V_{pp}$ on a 50 Ω load. Thus back to back operation of the ICS-564 and the ICS-554 board is possible for testing purposes.

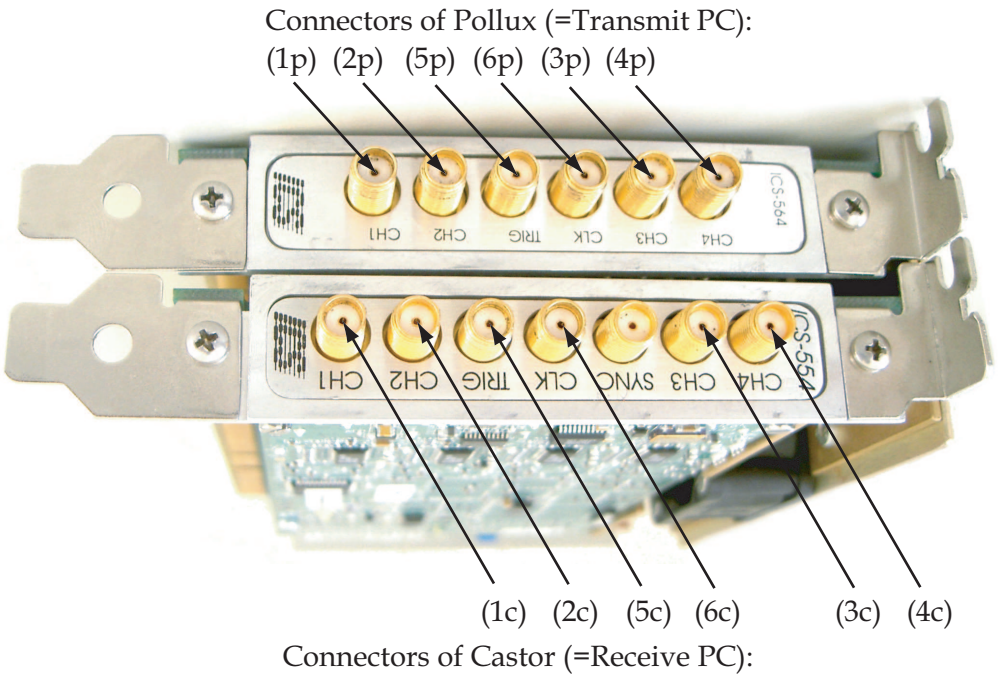

Figure 2.2.3: External hardware connectors of Pollux and Castor. (The orientation matches the actual orientation in the PC.)

Apart from the fact that the IF signals have to be connected, some other external connections are necessary for a proper operation of the system:

- Reception has to be triggered with the transmission of data. Therefore, the BNC connector (7c), the only BNC connector on the back of Pollux, has to be connected to the BNC connector (7p) to ensure that a transmission only starts if the receiver is ready. Also the SMA connector (5p) has to be connected with the SMA connector (5c) so that the reception is triggered exactly when the transmission starts. Further information on the triggering procedure is given in Section 2.4.1.

- If needed, the internal clocks of Pollux and Castor can be synchronized. This can be achieved by connecting the 100 MHz LVTTL[6] clock output of Castor (6c) to the 50 MHz LVTTL clock input of Pollux (6p) via an external frequency divider (6). Subsection 2.5.5 describes how to enable this synchronization in the MATLAB INTERFACE.

  Furthermore, it is also possible to synchronize Pollux at connector (6p) with an externally generated 50 MHz LVTTL clock. Synchronizing Castor with an external clock is not possible without modifying the hardware since connector (6c) is configured as an output. See [13] for further details.

- At last, Pollux (8p) and Castor (8c) have to be connected back to back with a crossed network cable to provide a fast 1,000 MBit/s connection, needed for internal data exchange. The LAN connection to the users is provided via (9c) and (9p) at a speed of 100 MBit/s.

The above described external cable connections are not always needed. For example, triggering can be done by the use of files. It is also possible to omit external clock synchronization if not required. In order to make use of these possibilities, the user has to set flags in the MATLAB INTERFACE as described in Section 2.5.

2.3 Transmitting Data via the MATLAB INTERFACE

As previously outlined in Section 2.1.1, interfacing the transmit PC is rather simple. In order to transmit (not receive) data over one or more of the four possible intermediate frequency channels, the user *only* has to

1. create complex baseband data samples in MATLAB as described in the following Section 2.3.1,

2. save these samples, and trigger the operation of Pollux (the transmitting PC) as described in Section 2.3.2.

2.3.1 Creating the Complex Baseband Data Samples

To allow maximum flexibility, the user does not provide data bits or symbols to the MATLAB INTERFACE but 14 bit complex baseband data samples.

[6] LVTTL = Low-Voltage Transistor-Transistor Logic, logic signal level standard

Therefore, the interface is independent of the modulation scheme and signal filtering. By generating baseband data samples in MATLAB, the user is able to create a wide range of baseband (and as a consequence IF) signals.

To fulfill the MATLAB INTERFACE specification, the user has to create a matrix which contains all the complex baseband data samples. This matrix must be named `xvTXData`[7]. Besides this, the real and the imaginary part of the samples have to be smaller than $2^{13}-1$ and larger than -2^{13} before converting them to the prescribed `int16` data type. An overflow is not checked in the MATLAB INTERFACE in order to maximize the data throughput. If needed, this check can be easily implemented in MATLAB before transmitting the data to the MATLAB INTERFACE.

The following code sample shows how to create a simple baseband signal: a sine wave. A complete copy of the MATLAB source code can be obtained from `\\Pollux\MIMO_TX_Data\samples\xmTX_Sine.m`.

```
xvTXChannels = 4;     % maximum 4
xvTXBlockLen = 2^16;  % maximum 2^17
xvTXData     = sin((1:xvTXBlockLen)*pi/4)).'*ones(1,xvTXChannels);
xvTXData     = int16(xvTXData*8176);
```

Executing this code, the following data matrix is created:

```
xvTXData =
       5781        5781        5781        5781
       8176        8176        8176        8176
       5781        5781        5781        5781
          0           0           0           0
      -5781       -5781       -5781       -5781
      -8176       -8176       -8176       -8176
        ...         ...         ...         ...
```

Each column corresponds to one of the four possible IF output channels. Column 1 corresponds to channel 1, column 2 corresponds to channel 2, and so on. This assignment is fixed and cannot be changed. Thus, for example, it is not possible to transmit data *only* on channel 3. As an alternative, the cable connections can be rewired to transmit data only on channel one (which is possible by generating just one column), or dummy data can be transmitted on channel one and two (and three columns are generated).

[7] As a programming convention, to prevent accidental use of MATLAB functions, constants always begin with `xc`, variables with `xv`, and m-files with `xm`.

The number of columns and, as a consequence, the **number of channels** can be set to 1, 2, 3, or 4. If not all four channels are needed, the amount of memory required to store the baseband data samples is also reduced by decreasing the number of columns.

The rows of `xvTXData` represent consecutive complex baseband data samples. All samples in one row are up-converted and transmitted simultaneously[8] on different channels as described later on.

If just transmitting *one* block, the maximum number of rows, and therefore, the **maximum number of data samples in one data block**, is limited to 2^{17} ($2^{17} = 131072$) by the transmitter FIFOs. This value does not depend on the number of channels transmitted simultaneously (=the number of columns). Only half of this value is available if multiple blocks are transmitted one after the other because the internal FIFOs must be able to store the next block, while still transmitting the previous one. More information on how to transmit more than one block of data will be given in Section 2.3.7.

2.3.2 Triggering the Operation of Pollux

After the complex baseband data samples (`xvTXData`) have been generated, they can be easily transmitted to Pollux by using the MATLAB INTERFACE. A vector named `xvTXDataOptions` specifies all user configurable options, e.g. the center frequency of the bandpass signal. These options are described in Section 2.5.

In order to transmit the data samples and options, the user has to save them to a `.mat`-file. Afterwards, he has to create a 0 byte large `.do`-file which triggers the operation of the transmitting PC, called Pollux. As described in detail in Section 2.1.1, this file also ensures that all data samples have been saved completely to the `.mat`-file before triggering Pollux.

```
xcTXDir         = '\\Pollux\MIMO_TXData\';
xvFileName      = 'Example Sine Wave';
xvTXDataOptions = [1, inf , 70.0, 4, 0, 0, 0, 0, 0, 0, 0, 0]

save([xcTXDir xvFileName '.mat'],'xvTXData','xvTXDataOptions');
xvFID=fopen([xcTXDir xvFileName '.do'],'w'); fclose(xvFID);
```

After executing these five lines of MATLAB code, Pollux creates the analog IF signal. However, there is only one block of baseband data which will be processed by Pollux. A possibility to transmit more data blocks one after the other is described in Section 2.3.7.

[8] Due to a firmware error, the up-conversion is simultaneous but not synchronous. The maximum error is ± half of a baseband sample length.

2.3.3 The Automatic Digital Up-conversion

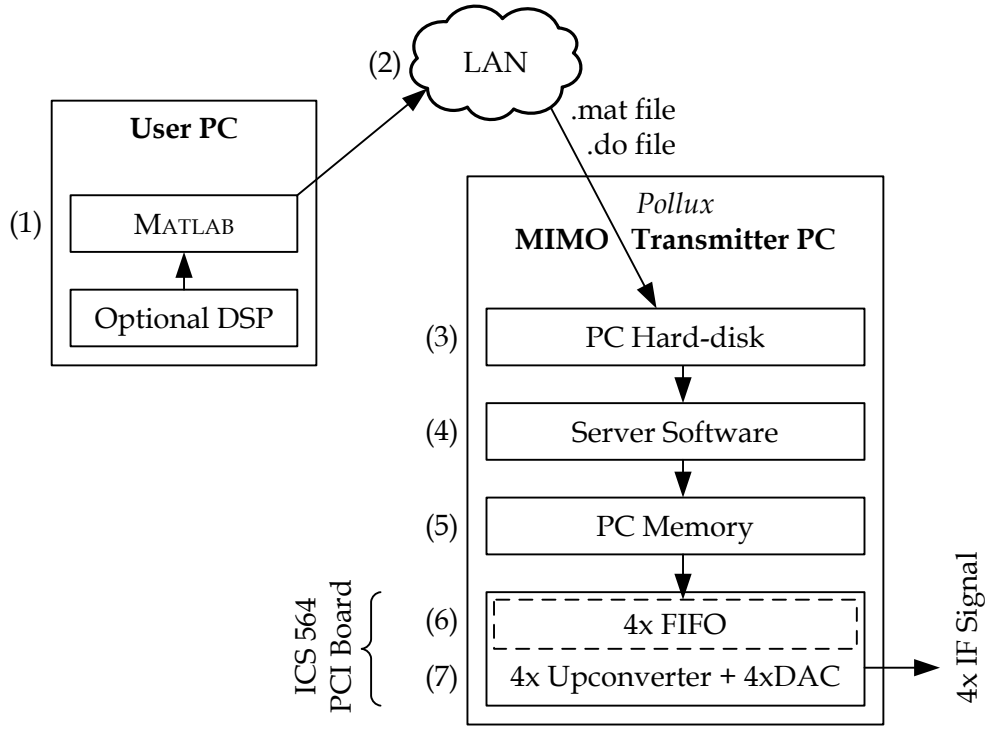

Figure 2.3.1: Simplified data flow for a transmission.

Triggered by a `.do`-file, Pollux starts its operation automatically. The up-conversion process can be simplified to the following steps (see Figure 2.3.1):

1. As described in the previous subsection, the user at first creates the digital baseband data samples and transmit options on his own PC. To do so, he can either use MATLAB and/or some optional hardware.

2. Then he stores this data via a LAN connection...

3. ...using the MATLAB `save(...)` command, on the internal hard-disk of Pollux, followed by a `.do`-file. As a huge number of tests showed, this process of saving data does not affect the performance of Pollux in any way. Therefore, it is possible to store multiple jobs on Pollux's hard-disk even while it is in operation and transmitting data.

4. If idle, the server program running on Pollux polls for new `.do`-files available and thus, for new data to be transmitted. After setting up the ICS-564 board [10] to fit the needs of the next transmission...

5. ... the data samples are first moved to the internal memory of Pollux. This is necessary because there is no direct connection between the hard-disk and the up-converter board.

6. Next, the data samples are transferred into the internal FIFOs of the ICS-564 board (IDT72V72100 [8]) where they stay ready until the transmission is triggered. These four FIFOs (2^{17} samples per channel), in connection with the internal memory, limit the maximum bock size of a transmission as previously described.

7. The rest of the signal processing happens in real-time in an Analog Devices chip (AD9857 [7]) located on the up-converter board (see Appendix A). In this chip, the data, taken out of the FIFOs, is processed as shown in Figure 2.3.2:

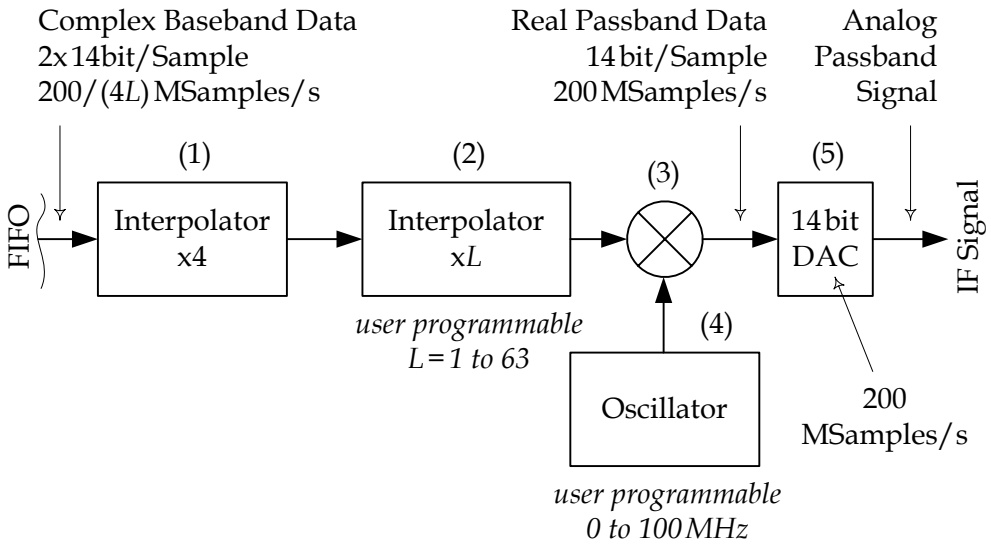

Figure 2.3.2: Simplified data flow in the ICS-564 board.

Since all four data streams (all four transmit channels) are equally processed and synchronously transmitted, only the data flow of one channel will be described in the following (see Figure 2.3.2):

1. The data samples taken out of the FIFO consist of 14 bits for the real part and 14 bits for the imaginary part. Interpolation is achieved by a fixed $\times 4$ interpolation filter and...

2. ... a user programmable ($\times 1$ to $\times 63$) (see Section 2.5.4) interpolation filter. Thus the overall interpolation can be set from 4 to 252, in steps

of 4, by the user. The output sample rate of this filtering process is always fixed to 200 MSamples/s.

3. In the next stage, the signal is multiplied with a complex sine wave in order to shift the signal to the IF domain.

4. This carrier is generated by a digital oscillator which is user programmable (see Section 2.5.3) in the range from 0 to 100 MHz. The theoretical resolution is smaller than 0.1 Hz, while the accuracy is limited by the internal crystal oscillator to a much larger value.

5. In the last stage the digital 14 bit signal is converted to the analog domain at 200 MSamples/s. The output of the converter is not filtered as described in Section 2.3.4. Thus higher order harmonics will appear in the spectrum at multiples of 200 MHz ± center frequency.

The overall process is a lot more complex than described here but to maintain simplicity, only necessary information to understand the overall system was listed. Further information on the up-conversion process can be obtained from the manuals of the ICS-564 board [10, 12], its firmware [11], the FIFOs [8], and the four digital up-converters [6] located on this board.

Since the up-conversion process is carried out completely automatically by the MATLAB INTERFACE, a user does not need detailed hardware knowledge of it in order to use the MIMO testbed.

2.3.4 The Analog Intermediate Frequency Signal

In the previous example, four lines of MATLAB code were needed to generate the baseband signal samples, and with five more code lines this signal was transmitted on all four IF channels.

To transmit data `(1)` in an endless loop `(inf)` at a center frequency of 70 MHz `(70)` with an overall interpolation factor of 4 `(4)` the following options must be set in the MATLAB INTERFACE (see Section 2.5 for more details):

```
xvTXDataOptions=[1,inf,70,4,0,0,0,0,0,0,0,0,0]
```

The next Figure 2.3.3 shows the measured[9] output spectrum for the generated sine wave.

[9] An Advantest R3271 spectrum analyzer was used to obtain the measurement results via a GPIB connection.

Since in the above example the complex baseband data samples were defined as xvTXData$[n] = \sin(2\pi\theta_o n)$ with $\theta_o = 1/8$, the bandpass signal consists of two spectral lines at 70 MHz $\pm$ $\theta_o \cdot 1/4 \cdot$ 200 MHz = 70 MHz $\pm$ 6.25 MHz. (200 MHz is the sampling frequency of the ADC and $1/4$ denotes the overall interpolation factor.)

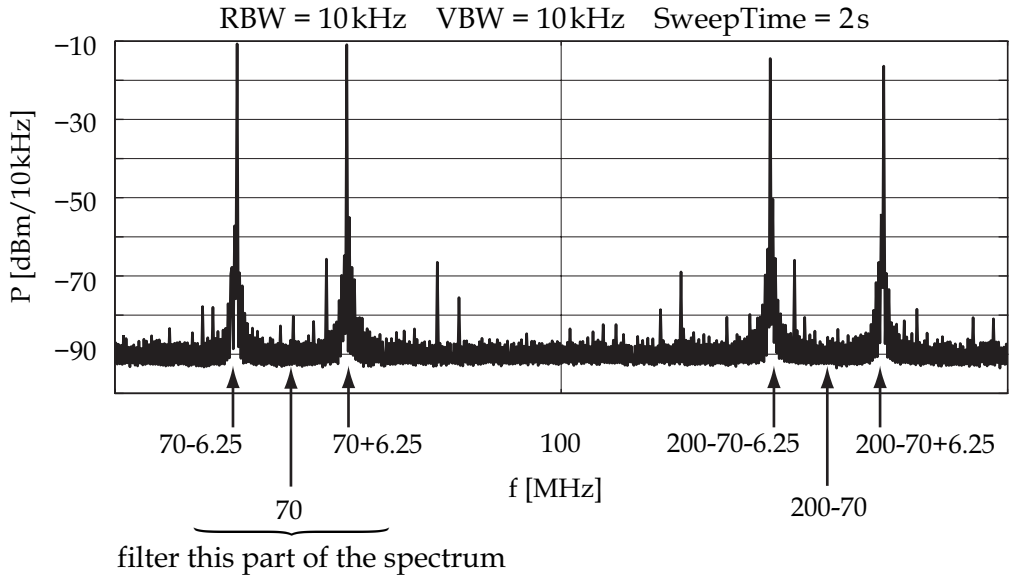

Figure 2.3.3: Measured IF spectrum of channel one.
`xvTXDataOptions = [1,inf,70.0,32,0,0,0,0,0,0,0,0,0]`

Due to the unfiltered output of the digital to analog conversion, higher order harmonics appear in the spectrum at multiples of 200 MHz $\pm$ center frequency. They can be easily filtered with an analog low-pass filter. Lukas Mayer built five of them with a cut-off frequency of 75 MHz in his diploma thesis [5]. They are boxed, have two SMA connectors, and operate for any IF lower than 75 MHz.

The **maximum bandwidth** of the IF signal is determined by the maximum sample rate of the baseband signal (200 MHz/4) minus some margin because of the internal filtering (80%). Although it is possible to transmit signals with 40 MHz bandwidth, it is not possible to receive them (see Section 2.4.3) due to limitations of the receiver. Reception can only be performed with an eight times lower maximum sample rate (6.25 MHz) which leads to a theoretically possible reception bandwidth of 6.25 MHz minus some margin. A detailed description will be given in Section 2.4.3.

Due to imperfect interpolation filters (especially if using higher interpolation factors) there are some undesired spectral components, in addition to

the higher order harmonics generated by the digital to analog conversion. They can be easily filtered with an IF filter. Furthermore, the baseband signal is hard limited at the beginning and at the end of each block (equal to a rectangular window function). This may also lead to undesired spectral lines.

2.3.5 Example: Transmitting a 16 QAM

To demonstrate how to use the MATLAB INTERFACE to generate a more complex signal, the following example *completely* shows how to create a 16 QAM signal. As in the previous example, the IF signal is transmitted at a center frequency of 70 MHz in an endless loop.

The following code example is working without any additions. It can be also found on Pollux at `\\Pollux\MIMO_TXData\samples\xmTX_16QAM.m`.

```
xvTXData = [     randsrc(5000, 1, [-3 1 1 3]/3)...
             + j*randsrc(5000, 1, [-3 1 1 3]/3) ];
xvFilter = rcosine (1, 4, 'fir/sqrt', 0.22, 40);
xvTXData = rcosflt (xvTXData, 1, 4, 'filter', xvFilter);
xvTXData = int16   (xvTXData*8176);

xcTXDir         = '\\Pollux\MIMO_TXData\';
xvFileName      = 'Test 16 QAM';
xvTXDataOptions = [1, inf , 70.0, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0]

save([xcTXDir xvFileName '.mat'],'xvTXData','xvTXDataOptions');
xvFID=fopen([xcTXDir xvFileName '.do'],'w'); fclose(xvFID);
```

First, the MATLAB code creates a sequence of 5000 random 16 QAM symbols. Next, these samples are filtered by a root raised cosine filter with an interpolation rate of 4 and a roll off factor of 0.22. At last, the data matrix plus the options are saved, and the transmission is triggered by creating a `.do`-file.

The resulting QAM bandwidth of 1.90625 MHz is therefore the symbol rate of 6.25 MHz (200 MHz/32 overall internal interpolation) divided by a factor of 4 (root raised cosine filter interpolation) times 1.22 (for the 0.22 roll of factor). The overall interpolation factor of 32 was chosen because it is the minimum possible value for receiving this data after a channel, as shown in the next Section 2.4.

Figure 2.3.4 shows the measured spectrum of the IF signal. In addition to the higher harmonics generated by digital to analog conversion, there are also some undesired spectral components smaller -85 dBm/10 kHz outside the area shown. They are results of imperfect internal interpolation and can be neglected. The 70 MHz carrier, which can be seen clearly, is a result of the transient response of the root raised cosine filter.

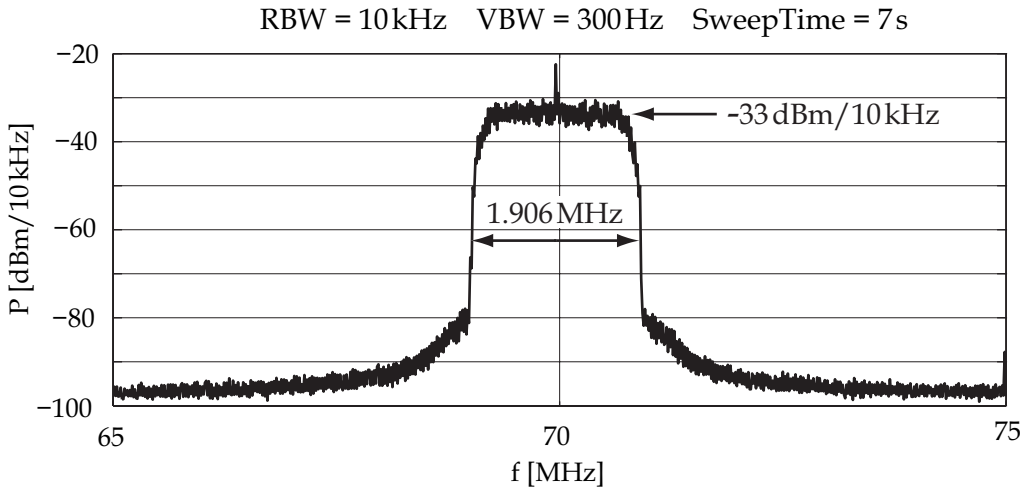

Figure 2.3.4: Measured IF spectrum of a 16 QAM signal.
`xvTXDataOptions = [1,inf,70.0,32,0,0,0,0,0,0,0,0,0]`

Because the maximum output signal level was used in this example, the **maximum output power** for a 16 QAM can be determined by simply integrating the spectral power density. Using a rectangular approximation of -33 dBm/10kHz in a 1.906 MHz range, an IF signal power of approximately -10 dBm is obtained.

2.3.6 Example: Signal for a Multitone Measurement

Not only modulation schemes can be tested using the MATLAB INTERFACE but it can be also used in some completely different applications, e.g. arbitrary signal generation. The following MATLAB code creates a set of equally spaced spectral lines. Up-converted to the IF, the resulting signal can be used for multitone measurements:

```
xvTXData     = 0.1*(ones(1,9)*sin([1:9]'*[1:2^17]*pi/20))'-0.05;
xvTXData     = int16(xvTXData*8176);
```

The measured output spectrum is shown in the Figure 2.3.5 below. This signal has already been successfully used to implement multitone measurements on an IF to RF up-converter. The complete example can be found in `\samples\xmTX_Multitone.m`.

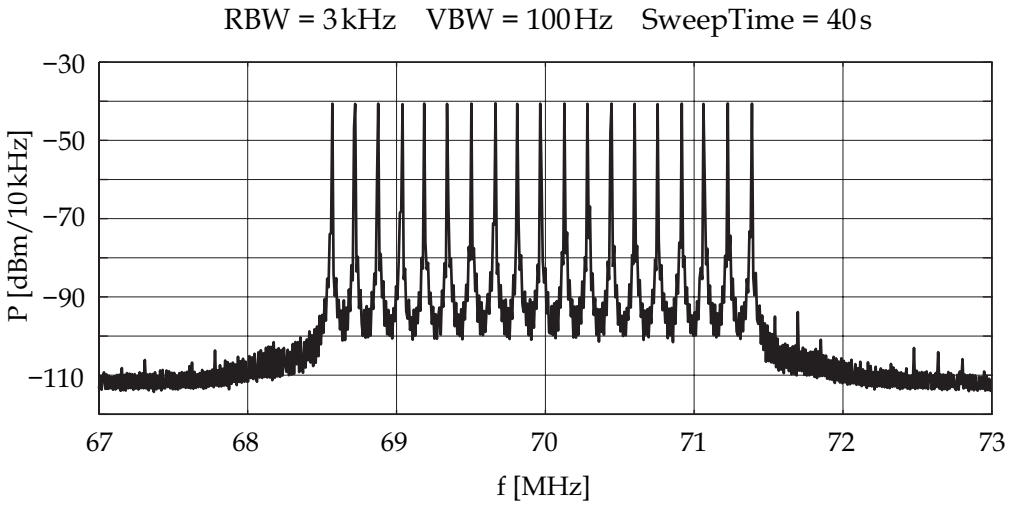

Figure 2.3.5: Measured IF spectrum for a multitone measurement.
`xvTXDataOptions = [1,inf,70.0,32,0,0,0,0,0,0,0,0,0]`

2.3.7 Transmitting Multiple Blocks of Data

As previously described, the size of a transmit block is limited by the internal FIFOs of the up-converter boards. But using the MATLAB INTERFACE multiple *equally sized* blocks of data can be transmitted one after the other in one operation:

```
xcTXDir         = '\\Pollux\MIMO_TXData\';
xvFileName      = 'Test Multiple Blocks';
xvTXDataOptions = [1, 1, 70.0, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0]

xvTXData = [create your data for block 1]
save([xcTXDir xvFileName '_1.mat'],'xvTXData','xvTXDataOptions');

xvTXData = [create your data for block 2]
save([xcTXDir xvFileName '_2.mat'],'xvTXData');

xvTXData = [create your data for block 3]
save([xcTXDir xvFileName '_3.mat'],'xvTXData');

xvFID=fopen([xcTXDir xvFileName '.do'],'w'); fclose(xvFID);
```

Now, not only one `.mat`-file is created but a set of multiple `_i.mat`-files, where `i` denotes the number of the transmit blocks. This number must begin with 1 and is increased by one for every subsequent block. After saving all these `_i.mat`-files, still only one `.do`-file is needed to trigger the operation of the transmit PC.

- Under special conditions, it is possible to transmit multiple blocks of data one after the other with no space in between. This is equal to transmitting a continuous signal and limits the maximum signal length to the free memory of Pollux (e.g. 600 MByte) divided by the number of channels (typically 4) and the bytes per complex sample (4). The resulting signal length in this case is 37.5 million samples. See Section 2.5.13.0 for more information on how to set the needed *'TX continuous mode'* flag in the MATLAB INTERFACE.

- Furthermore, it is possible to repeat all these blocks in a loop with no space in between to further enhance the signal length. The number of repetitions is user defined. See Section 2.5.2 for further information on the *'TX repeat how often'* parameter.

- If a continuous wave is not needed, it is possible to transmit blocks of maximum length with small spaces in between, one after the other, directly from the hard-disk. Although the maximum block length is limited to 2^{17}, the number of blocks is only limited by the free hard-disk space. Assuming 25 GBytes free, a block length of 2^{17} samples, and 4 transmit channels $25 \cdot 10^9/(2^{17} \cdot 4 \cdot 4) \approx 11920$ blocks can be transmitted in a row. Furthermore, repetition of this data is also possible. See Section 2.5.13.5 for more information on how to transmit only via the hard-disk by using the *'TX force HDD operation'* flag.

The maximum block length of the whole transmission system is furthermore limited by the maximum block length of the receiver. The above described limitations only apply to the transmitter, they do not exactly match those of the receiver. See Section 2.4.3 for further information.

The `\samples\xmTXMultiple_FMradio.m` example uses this ability to transmit one block after the other to create an ordinary FM radio transmit signal from a music file. Proof of operation was achieved by receiving this signal with an FM radio at 89.9 MHz. For further information please examine the example.

2.4 Receiving Data via the MATLAB INTERFACE

Receiving data using the MATLAB INTERFACE is as simple as transmitting. In order to transmit and receive data over a channel, the user has to perform the following steps:

1. Generate complex baseband data samples (`xvTXData`) to be transmitted:

```
xvTXData = [     randsrc(5000, 1, [-3 1 1 3]/3)...
             + j*randsrc(5000, 1, [-3 1 1 3]/3) ];
xvFilter = rcosine (1, 4, 'fir/sqrt', 0.22, 40);
xvTXData = rcosflt (xvTXData, 1, 4, 'filter', xvFilter);
xvTXData = int16   (xvTXData*8176);
```

2. Delete a possible `.done`-file of a previous transmission:

```
xcTXDir         = '\\Pollux\MIMO_TXData\';
xvFileName      = 'Test 16 QAM';

if size(dir([xcTXDir xvFileName '.done']),1)~=0
   delete  ([xcTXDir xvFileName '.done']);
   end;
```

3. Save the data plus some options[10] (`xvTXDataOptions`) to a `.mat`-file followed by a `.do`-file to trigger the operation of Pollux:

```
xvTXDataOptions = [3, 1, 70.0, 32, 1, 0, 0, 4, 0, 0, 0, 0, 0]

save([xcTXDir xvFileName '.mat'],'xvTXData','xvTXDataOptions');
xvFID=fopen([xcTXDir xvFileName '.do'],'w'); fclose(xvFID);
```

4. Wait for a `.done`-file (polling operation) on Pollux. This `.done`-file indicates that the transmission has been completed and the received data is available:

```
while size(dir([xcTXDir xvFileName '.done']),1)==0
   pause(0.2);
   end;
```

[10] Please note that the fist number in `xvTXDataOptions` is now "3" instead of "1" which means to "transmit and receive data" instead of "only transmit data". More information about the options can be obtained from Section 2.5.1.

For longer transmissions, this waiting period can be used to carry out some other calculations. This possibility to interleave other MATLAB code with the transmission of data is the reason, why the MATLAB INTERFACE is not implemented using a single `.m` MATLAB function.

5. Check whether an error has occurred by looking if a `_err.mat`-file exists. In case of an error, this error is loaded and trapped:

```
if length(dir([xcTXDir xvFileName '_err.mat']))~=0
   load      ([xcTXDir xvFileName '_err.mat']);
   error(xvError);
   end;
```

6. Finally the received complex baseband data samples are be obtained by simply loading them from the hard-disk of Castor:

```
xcRXDir  = '\\Castor\MIMO_RXData\';
load([xcRXDir xvFileName '.mat']);
xvRXData = double(xvRXData);
```

   If multiple blocks are received, an `i_.mat`-file (`i` denotes the block number) is created for each block as described in Section 2.5.9.

   The final conversion to complex double values is not necessary but the received data samples are of type '`int32 complex`' and MATLAB cannot perform further calculations with them if they are not converted.

The above described source code is complete and can be obtained from `\\Pollux\MIMO_TXData\samples\xmTXRX_Simple_16QAM`.

As described in Section 2.5.13.4, it is furthermore possible to detect the end of the data reception of Castor by using the *'create .received file'* flag. This makes it possible to change the channel while the received data is still being saved to the hard-disk of Castor after reception.

2.4.1 Handshaking of Pollux and Castor

Triggered by a `.do`-file, Pollux starts its operation as previously described in Section 2.3.3 but with the difference that now data is also received. This circumstance makes the overall process a lot more complicated (see Figure 2.4.1).

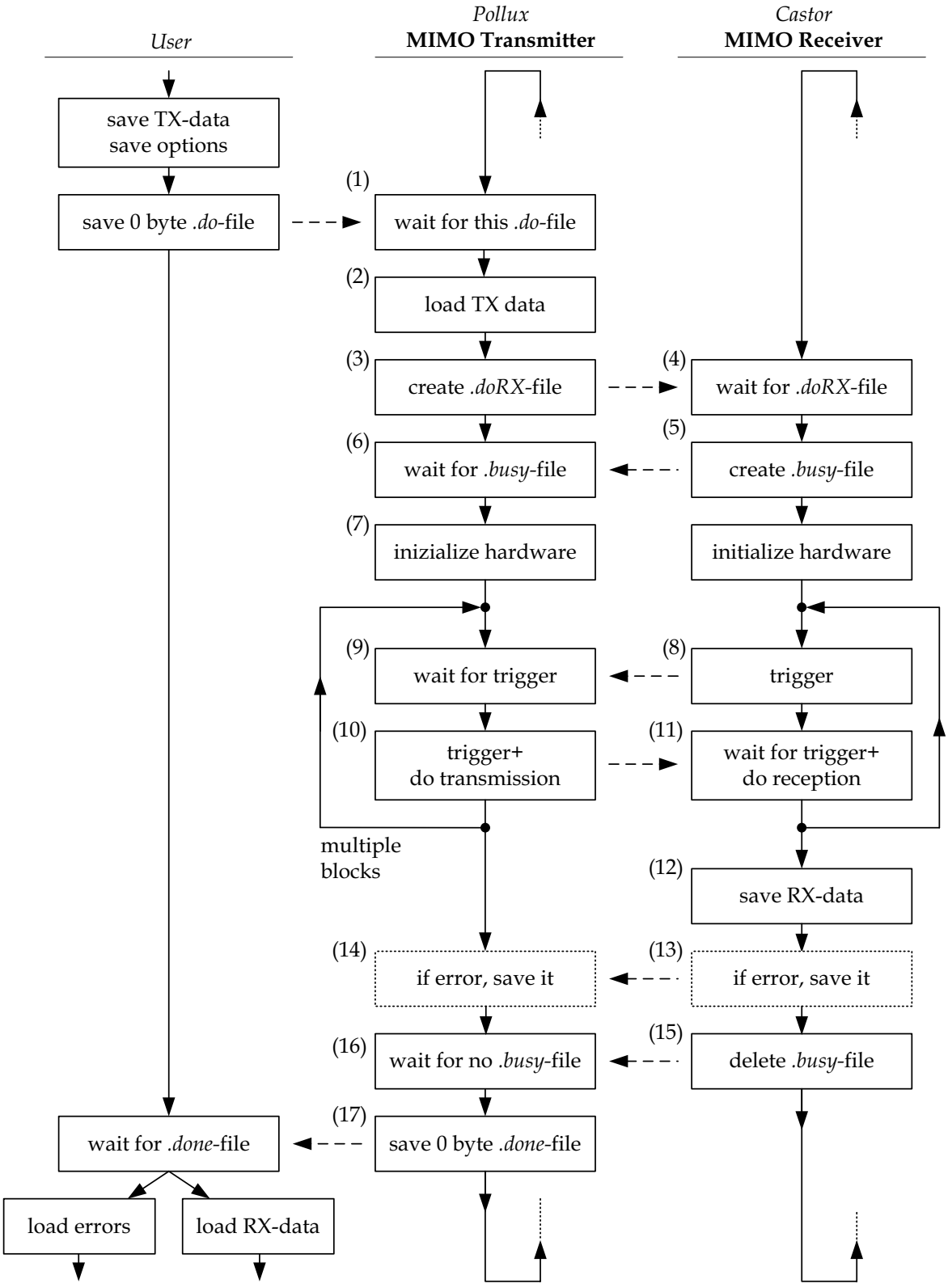

Figure 2.4.1: Handshaking of the User, Pollux, and Castor.

Although the overall process is carried out automatically, it is necessary to understand it in order to deal with errors returned by the MATLAB INTERFACE (the item numbers correspond to Figure 2.4.1):

1. If idle, Pollux continuously polls for the existence of a `.do`-file to start operation. Multiple `.do`-files are processed one after the other in alphabetical order.

2. Next, all the transmit data samples are loaded into the internal memory of Pollux. In the case of multiple transmit blocks, this can take quite a long time, thus *'Load Data'* is displayed on Pollux's screen.

3. Now Pollux has loaded all data, it displays a *'Mode: TX&RX'* message. But before a transmission can be started, Pollux gets the receiving PC, Castor, ready by creating a `.doRX`-file[11] file.

4. Castor on his part continuously polls for the existence of this `.doRX`-file to start operation.

5. Displaying *'Mode: TX&RX'*, Castor gets ready to operate and creates a `.busy`-file to signal this to Pollux.

6. Pollux, awaiting this `.busy`-file displays *'RX ready ??'* while waiting. Obviously, if there is still no `.busy`-file after some time, Castor is out of order. Displaying *'Timeout while waiting for receiver to be ready.'*, Pollux terminates operation and continuous at item 14 with reporting this error to the user.

7. At the same time Pollux and Castor initialize their up- (ICS-554 board), or rather down-converter board (ICS-554 board) and display *'TX'* or rather *'RX'*.

The data transmission itself consists of one or more blocks that are transmitted and received at exactly the same time. To do so, additional triggering is needed before each transmission:

8. To indicate that the internal down-converter board is ready to receive, Castor sets the BNC Trigger output (7c), shown in Figure 2.2.2, high.

[11] All internal handshaking files are stored in `\\Pollux\MIMO_TXData\_Internal\` or `\\Castor\MIMO_RXData\_Internal\`, so that a PC always polls for the existence of a file on the own hard-disk.

9. In Pollux, this high level on the BNC trigger input (7p) creates an interrupt and starts the up-converter hardware.

10. This hardware sets a trigger impulse on the SMA trigger connection (5p) and synchronously begins to transmit data.

11. Triggered by this triggering pulse, the receiving hardware on Castor starts its operation and receives all the data of one block. Furthermore the BNC trigger output (7c) is automatically set to low by the hardware when the first data sample is received.

The next figure shows this triggering process, which is carried out for every block of transmitted data:

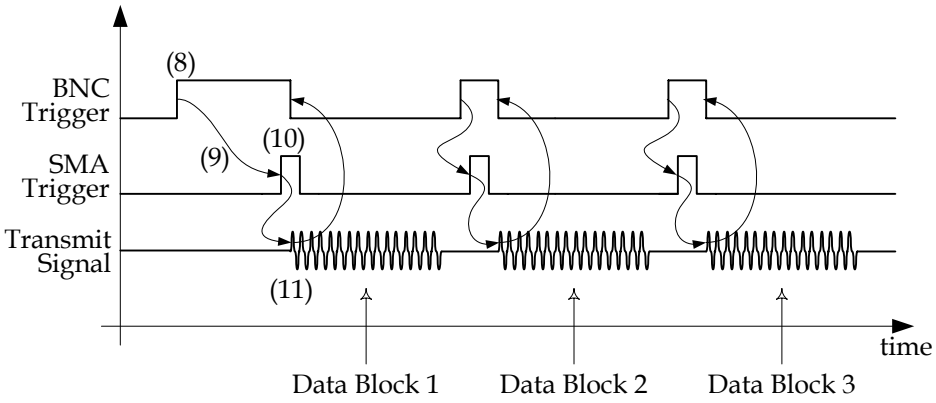

Figure 2.4.2: Triggering of the transmit and receive hardware.

Step (9) is often longer before the first transmit block because Pollux needs a lot more time to initialize its internal hardware and is, therefore, later ready to transmit.

Several tests showed that the process of receiving and transmitting data is not disturbed by any other hard-disk operation (e.g. loading or saving data via the LAN to the hard-disk). Further, a task running at low system priority on Pollux or Castor does not affect the overall performance.

12. After the transmission is complete, Castor displays *'Saving xx Blocks'* (*xx* is the actual number of blocks) and stores all the received data samples from the internal memory to the internal hard-disk at a rate of about 50 MBytes/s.

13. If the operation of Castor had to be terminated due to an error, it is continued here. The error is displayed and saved to a `_RXerr.mat`-file.

14. Pollux on his parts also continuous operation here if an error has occurred. But Pollux also checks if there has been an error on Castor and saves all errors to a `_err.mat`-file to be read by the user. Therefore, the user only has to examine this `_err.mat`-file as previously described to receive all error messages from Pollux and Castor.

15. Displaying *'Finished'* Castor deletes the `.busy`-file and returns to idle mode.

16. After the `.busy`-file has been erased, Pollux also displays *'Finished'* which indicates that the overall transmission is complete.

17. As a last step the `.done`-file is created to signalize the user, that either the received complex baseband data samples are ready, or a corresponding error file has been crated.

1. Now, the overall process starts again. If idle, Pollux continuously polls for the existence of a `.do`-file to start...

2.4.2 The Automatic Digital Down-conversion

In this subsection the data flow in Castor is considered, where the four intermediate frequency signals at first get in touch with the ICS-554 board (see Appendix B) [13, 14, 15] which performs the down-conversion operated by the MATLAB INTERFACE (see Figure 2.4.3):

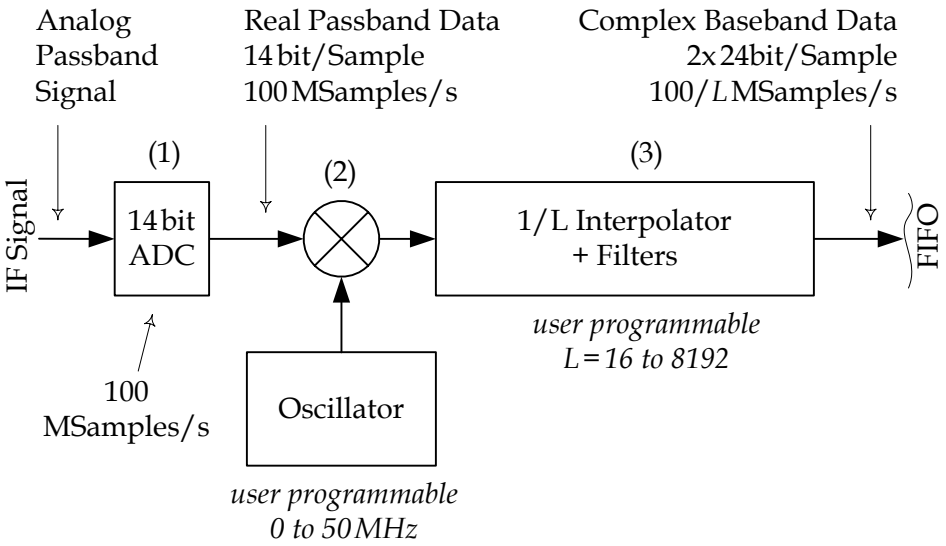

Figure 2.4.3: Simplified data flow in Castor's ICS-554 board.

1. At first, the analog intermediate frequency signals are sampled at a rate of 100 MSamples/s and converted to 14 bit digital values. The AD6645 [7] used for this conversion is capable of sampling IF signals with a center frequency of up to 200 MHz[12] with a specified SNR of 72 dB. The full scale input voltage is 1.2 V_{pp}.

Digital down conversion is then performed by the Texas Instruments GC4016 multi-standard quad DDC chip [9] (one quad chip for each channel). This chip has a multitude of capabilities but only basic operation was implemented in this diploma thesis as described in the following. The interested user should consult the documentation of the chip for further details [9]. Changing its configuration is only possible by modifying the source codes of the MATLAB server on Castor. Nevertheless, with the standard configuration implemented in the MATLAB INTERFACE, everything required should be possible because the user can always implement additional features (e.g. filtering) in his own MATLAB code.

2. A mixer and a numerically controlled oscillator are used to quadrature down-convert the signal from a user configurable center frequency (see Section 2.5.10) to the base band. Like in the up-converter, the theoretical resolution of the oscillator frequency is smaller than 0.1 Hz, while the accuracy is limited by the internal crystal oscillator to a much higher value. Although only tuning frequencies of 0 to 50 MHz are generated internally, programming higher center frequencies is possible to perform undersampling of the bandwidth-limited IF signal.

   Setting a center frequency of 70 MHz at the transmitter (the internal oscillator will be automatically set to 70 MHz) and at the receiver (the internal oscillator will be automatically set to 30 MHz to perform undersampling) while using external synchronization does not lead to exact frequency synchronization. This is due to numerical limitations in the digital generation of the oscillator frequencies.

3. After the digital down-conversion the signal is filtered. First a cascaded integrator comb filter decimates the signal. This filter is followed by a compensating finite impulse response filter (the standard filter used is the 'CFIR_80' [9]). A programmable finite impulse response filter ('PFIR_80' is the standard filter used [9]) followed by a resampler completes the filtering process. Only a switching between predefined filters

[12] The 100 MSamples/s ADC is able to perform undersampling of the IF signal which must be bandwidth-limited to satisfy the Nyquist criteria.

was implemented in the MATLAB INTERFACE but user configuration of the filter coefficients is not possible.

The MATLAB INTERFACE allows the user to set the overall decimation factor L of the down-conversion in a range of four times a value between 4 to 2048 (see Section 2.5.11). The value set in the MATLAB INTERFACE (2·L) is two times higher. This has the advantage that the same value of interpolation in the transmitter and in the receiver leads to the same baseband sample rate, although the IF sampling rates are different. E.g. setting an interpolation rate of 32 for the transmitter and the receiver leads to a baseband data rate of 6.25 MHz (=200 MHz/32) in the transmitter and the receiver.

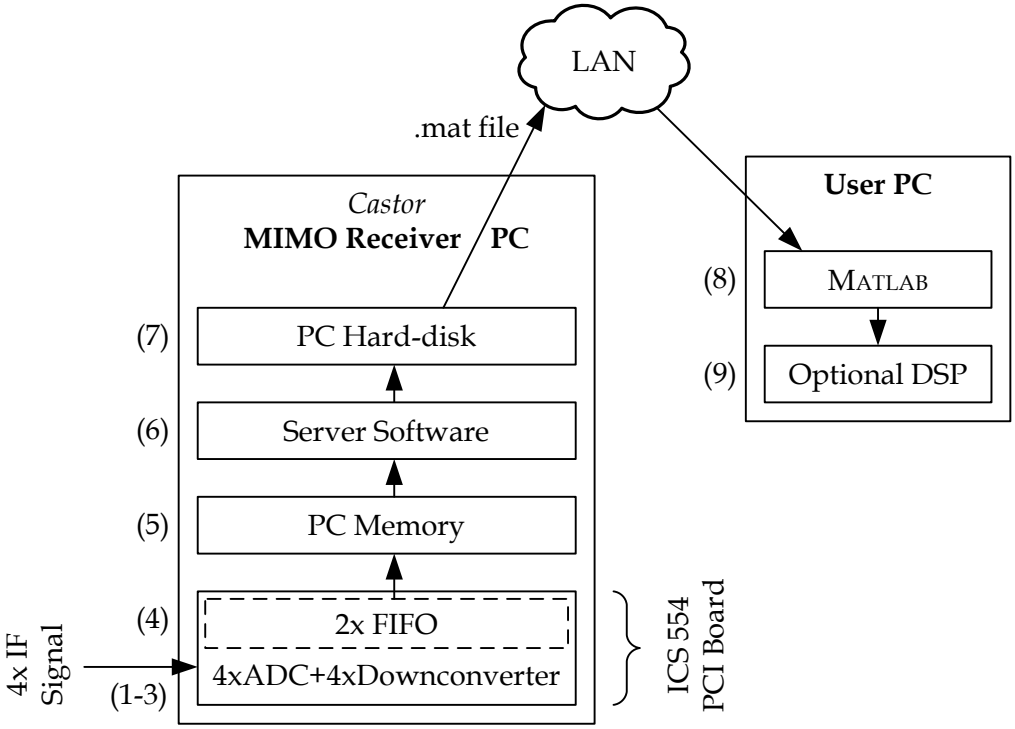

Figure 2.4.4: Simplified data flow for a reception.

4. The received data samples are stored in real-time and alternating between the two available FIFOs [8] to maximize the possible block length of a transmission (see Figure 2.4.4).

5. After moving all the received data samples to the internal memory of Castor...

6. ... they are preprocessed by the server software. This processing includes de-interleaving of the serialized data stream from four channels and two FIFOs to a MATLAB compatible data matrix for each received block.

7. The de-interleaved data samples are next stored to a `.mat`-file on the internal hard-disk. If multiple blocks are received, an `i_.mat`-file (`i` denotes the block number) is created for each block as described in Section 2.5.9.

8. Via a local area network connection, the user can now load these received 2×24 bit complex baseband data samples by using the `load(...)` command into MATLAB.

9. As a last optional step, this data can be moved to a DSP/FPGA board to perform real-time processing (see Chapter 3).

2.4.3 The Complex Baseband Data Samples

As an example, the following measurement setup is used to receive (not transmit) complex baseband data samples:

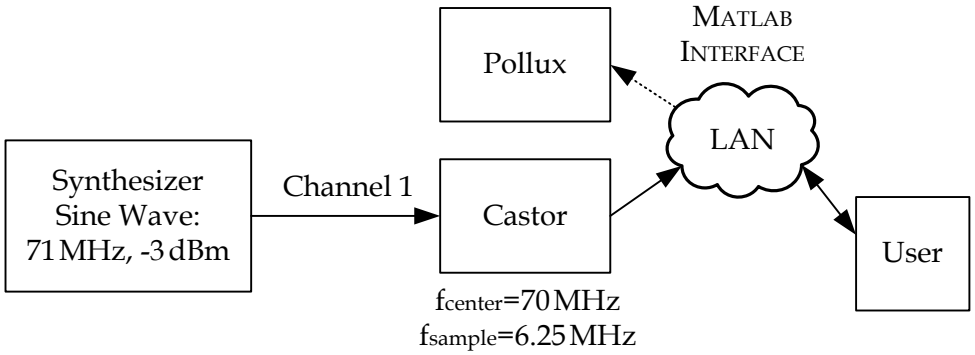

Figure 2.4.5: Measurement setup.

A synthesizer creates a sine wave at 71 MHz with a power of -3 dBm. This signal is then received with the MATLAB INTERFACE at a carrier frequency of 70 MHz using the following options:

```
xvTXDataOptions = [2,0,0,0,0,0,2^17,1,1,70.0,32,0,0]
```

A baseband sample rate of 6.25 MHz was chosen (=200 MHz divided by an interpolation factor of 32). Thus the period of the sine wave is 6.25 samples. The drop of signal power outside 80% of the spectrum, shown in Figure 2.4.6, is due to the standard filtering in the down-conversion. The

complete MATLAB code can be found in `\samples\xmRX_Simple.m`. This all leads to the following spectrum and baseband signal:

```
plot(([1:6001]-3000)/3000*3.125, 20*log10(...
    fftshift(abs(fft(hanning(6001).*xvRXData(1000:7000)))))-200)
```

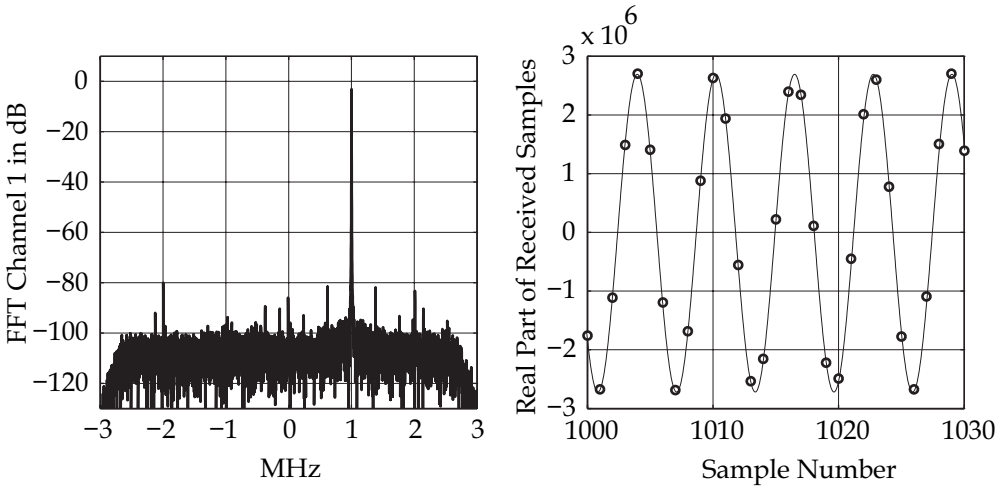

Figure 2.4.6: Receiving a sine wave at 71 MHz.

```
xvTXDataOptions = [2,0,0,0,0,0,2^17,1,1,70.0,32,0,0]
```

As previously described, a matrix of received data samples is loaded by using the MATLAB `load(...)` command. It has exactly the same format as the transmitted data samples described in Section 2.3.1: Each column represents a received channel, and the rows represent consecutive baseband data samples.

- The complex baseband data samples consist of 24 bits for the real part and 24 bits for the imaginary part. Due to a lack of a `int24` type in MATLAB they are internally stored as `int32` values and thus, 8 bytes per sample are needed.
- The **number of channels** received can only be set to '1' or '4'. In the former case only channel 1 is received, and four times less memory is needed to store the data samples.
- The **maximum block length** which can be received is 2^{17} samples for one channel and 2^{15} if data is collected at all four channels. Continuous reception of one channel is always possible. For four channels, the data rate has to be half of the maximum possible, in order to receive data continuously (see Section 2.5.13.1).

- The **maximum bandwidth** (5 MHz with the default filters) of the received intermediate frequency signal is determined by the maximum baseband sample rate of 6.25 MHz (=100 MHz/(4×4)) and some margin which is set to 80% by the internal filters. This default filter configuration can be changed to almost 100% by using the *'RX filter'* option in the MATLAB INTERFACE as described in Section 2.5.12.

  A further enhancement of this theoretically possible 6.25 MHz bandwidth is *not* possible without modifying the firmware of the FPGAs (since it has been changed in order to maximize the FIFO space) and rewriting the MATLAB server software of Castor. The modifications which are necessary to change from the so called "Split-IQ mode" to the "Wideband configuration" mode, without disturbing the current MATLAB INTERFACE would be quite complicated. Further details can be obtained from the ICS-554 board manual [13].

2.4.4 Example: 16 QAM

As another example, the 16 QAM signal shown in Figure 2.3.4 is now transmitted and received by using the MATLAB INTERFACE at a center frequency of 70 MHz. The internal clocks of Pollux and Castor are externally synchronized but the phase offset is not since this is not possible. The following setup has been used:

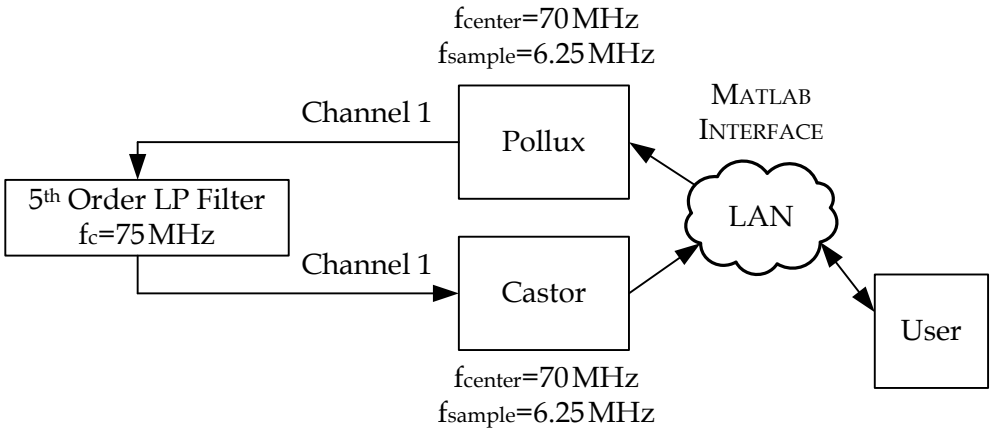

Figure 2.4.7: Measurement setup.

To suppress the higher harmonic frequencies, an analog 5th order lowpass filter with a cut off frequency of 75 MHz has been used. Its attenuation at 130 MHz is approximately -40 dB.

Transmitting, receiving, and displaying the following spectrum and baseband signal was performed using the `\samples\xmTXRX_16QAM` example. Pollux needs approximately 0.3 seconds to complete this operation (see Figure 2.4.8):

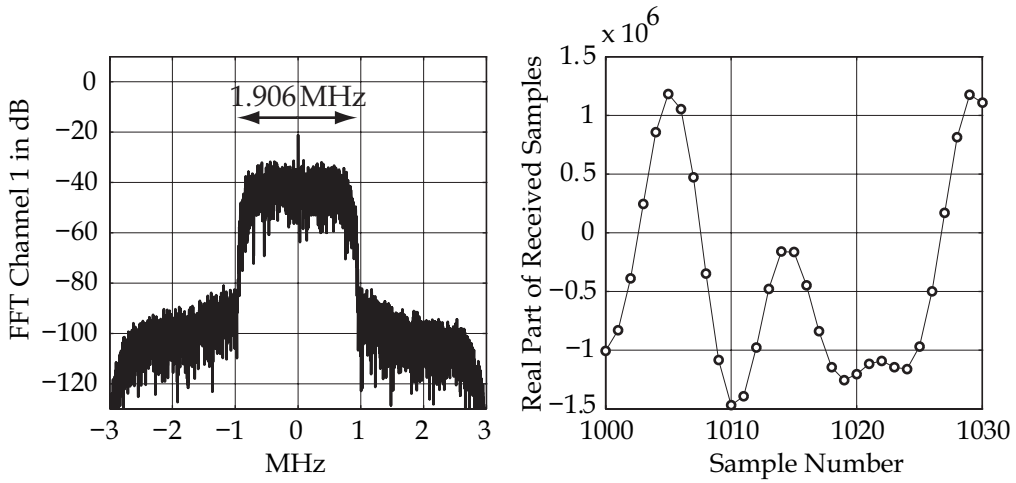

Figure 2.4.8: A received 16 QAM signal (not synchronized).
`xvTXDataOptions = [3,1,70,32,1,0,0,1,1,0,0,0,0]`

The interpolation of the root raised cosine filter was four, thus there are four data samples per symbol. During his diploma thesis, Christian Mehlführer wrote a MATLAB script to synchronize this QAM signal to receive blocks of random data without bit errors (see Figure 2.4.9). This script can be found in `\samples\ber\cmber.m`. See [3] for further information.

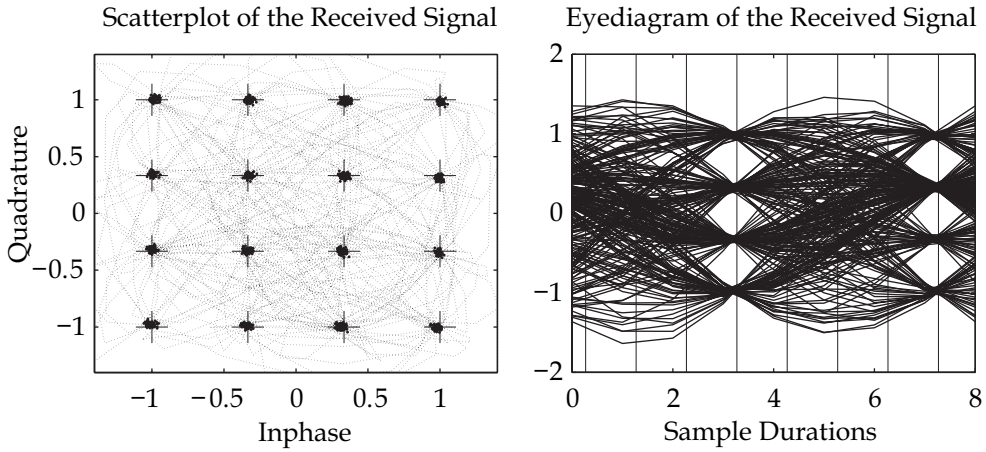

Figure 2.4.9: A received 16 QAM signal (synchronized).
`xvTXDataOptions = [3,1,70,32,1,0,0,1,1,0,0,0,0]`

2.5 "xvTXDataOptions"

The way *how* the MATLAB INTERFACE up-converts the complex baseband data samples to the intermediate frequency and back is basically determined by two different sets of input data:

1. The complex baseband data samples stored in `xvTXData` do not only designate the sample values but also the **number of transmit channels** (=number of columns in `xvTXData`) and the overall **block length in samples** (=number of rows in `xvTXData`).

   The column number equals the channel number on which the data is transmitted. At a maximum four transmit channels can be served but any value less is also possible and decreases the amount of memory needed for baseband data. The maximum number of samples possible is described in Section 2.5.14.

2. All the other **user configurable parameters** are stored in a row vector called `xvTXDataOptions`. Each of the 13 elements has a specific function the user must understand in order to use the MATLAB INTERFACE, for example:

```
xvTXDataOptions = ...
      [ 3     , ...  % ( 1) 1 to 4  operation mode
        1     , ...  % ( 2) int     TX Number of Repetitions
        70.0  , ...  % ( 3) double  TX center frequency in MHz
        32    , ...  % ( 4) int     TX interpolation factor
        1     , ...  % ( 5) boolean TX external clock
        0     , ...  % ( 6) int     TX delay betw. blocks in ms
        0     , ...  % ( 7) int     RX number of samples
        1     , ...  % ( 8) 1 or 4  RX number of channels
        1     , ...  % ( 9) int     RX Number of Repetitions
        0     , ...  % (10) double  RX center frequency
        0     , ...  % (11) int     RX interpolation ratio
        0     , ...  % (12) int     RX filter
        0*1   + ...  % flag 1       TX continuous mode
        0*2   + ...  % flag 2       RX continuous mode
        0*4   + ...  % flag 4       TX do not delete .mat file
        0*8   + ...  % flag 8       RX use .dat file
        0*16  + ...  % flag 16      create .received file
        0*32  + ...  % flag 32      TX force HDD operation
        0*64  + ...  % flag 64      RX force HDD operation
        0*128 + ...  % flag 128     sync via file
        0*256 ];     % flag 256     RX Check CW Data
```

or in short form:

```
xvTXDataOptions = [3,1,70,32,1,0,0,1,1,0,0,0,0]
```

In the following subsection, each parameter will be discussed in detail. By agreement, 0 does always mean that a parameter is *not* set. In case of a not required parameter the MATLAB INTERFACE will evaluate this parameter if needed. (For example, if 0, the receiver center frequency will be set to the transmitter center frequency.)

> In order to allow for a multitude of new, different setups,
> **the MATLAB INTERFACE does *not* check**
> **the options for correctness!**

This provides maximum flexibility to the user but may lead to strange results if erroneous options are set.

2.5.1 Operation Mode

(The subsection numbers correspond to element numbers in xvTXDataOptions.)

The first element of `xvTXDataOptions` determines the operation mode. It can be set to 1, 2, 3, or 4. The MATLAB INTERFACE is always used in the same way as described in Section 2.3 and Section 2.4.

1. **Only Transmit Data:**

   If set to 1, the MATLAB INTERFACE only performs a transmission. No data is received and, therefore, Castor is not needed at all. Parameters 1 to 4 are required in this mode, the others can be set to 0 if not needed.

2. **Only Receive Data:**

   If set to 2, the MATLAB INTERFACE only receives complex data samples through Castor. The matrix `xvTXData` is not needed for this operation and can be omitted. Parameters 1 and 7 to 11 are required for this operation, the others can be set to 0 if not needed.

3. **Receive Transmitted Data:**

   If set to 3, the MATLAB INTERFACE transmits data via Pollux and receives data via Castor at the same time. This mode is basically used to transmit data through an external channel which is set up between Castor and Pollux. Parameters 1 to 4 are required for this operation, the others can be set to 0 if not needed.

4. **Hibernate Pollux and Castor:**

   If set to 4, Pollux and Castor will hibernate instead of transmitting or receiving data. No other parameter or data is needed for this operation.

2.5.2 TX Number of Repetitions

This parameter determines how often the complex baseband data samples are repeated during one operation. It can be set to:

- **any positive integer value**

  The given baseband data blocks (there can be more than one as describes in Section 2.3.7) are repeated as often as set by this integer value. Flag *'TX Continuous Mode'* (see Section 2.5.13.0) determines if there is any space between the data blocks. Parameter *'Delay Between Blocks'* (see Section 2.5.6) can be used to set this space to a dedicated value in ms. This works for *'Operation Mode'* 1 and 3.

  The `\samples\xmTXMultiRXMulti_Sine.m` program shows how to repeat three blocks of data four times with minimum space in between. The output signal of channel one was measured by using an Agilent 54622A oscilloscope:

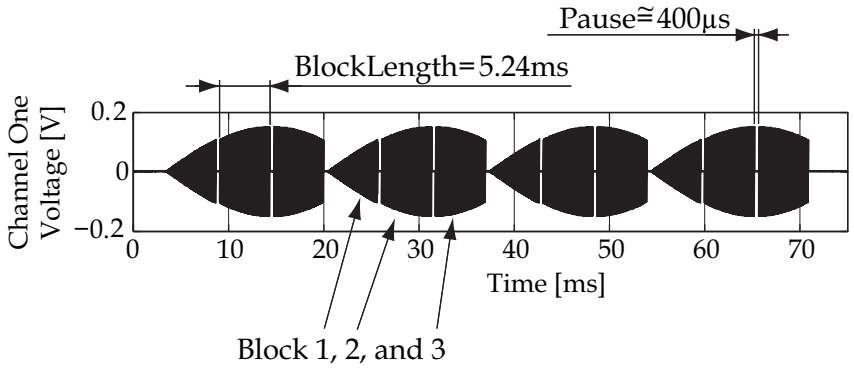

Figure 2.5.1: Measurement, repeated multiple transmit blocks.

```
xvTXDataOptions = [3,4,70,32,1,0,0,0,0,0,0,0,0]
```

The transmit length of one block (5.24 ms) can be calculated by dividing the number of samples per block (2^{15} in the above example) by the baseband sample rate (200 MHz/32 times interpolation).

- **inf**

  The samples in `xvTXData` (in this mode only one block is allowed) are repeated in an endless loop with no space in between. This loop can be only stopped by the next transmission or if Pollux is turned off.

  This can be very useful to generate a continuous wave signal for measuring the signal power with a power meter or the spectrum with a spectrum analyzer.
  e.g.: `xvTXDataOptions = [1,inf,70,32,0,0,0,0,0,0,0,0,0]`

  It can also be used to receive multiple blocks of this continuous wave signal.
  e.g.: `xvTXDataOptions = [3,inf,70,32,1,0,2^17,1,1000,0,0,0,2]`

2.5.3 TX Center Frequency

This is the center frequency of the up-converted bandpass signal. It can be set to any value between 0 and 100 MHz. The theoretical resolution is smaller than 0.1 Hz while the accuracy is limited by the internal crystal oscillator to a much higher value.

See *'TX external clock'* (Section 2.5.5) for further details on how to use an external clock reference for the center frequency to either synchronize it with the receiver or increase the accuracy.

2.5.4 TX Interpolation Factor

As described in autorefupconversion, the digital baseband data samples are interpolated by a user configurable factor before they are analog to digital converted at a sample rate of 200 MSamples/s. This factor can be determined by this parameter to an integer value between 4 and 252 in steps of four. The baseband sample rate is therefore 200 MHz divided by the interpolation factor. The interpolation rate of the transmitter is independent from the interpolation rate of the receiver.

2.5.5 TX External Clock

All internal clocks of the transmitting hardware (and therefore also the center frequency of the intermediate frequency signal) are derived from an on-board 50 MHz crystal oscillator if this parameter is set to 0.

By changing this parameter to 1, the user can connect an external 50 MHz clock source (square or sine wave at LVTTL levels) to the SMA connector (6p) shown in Figure 2.2.3.

- By connecting an external oscillator, the clock of the hardware can be set arbitrarily accurately.
- By connecting the clock output of Castor (6c) via a frequency divider to the transmit hardware, the internal clock of this hardware is exactly synchronized with the receiver hardware. This leads to exactly synchronized intermediate frequencies and baseband data sample rates.

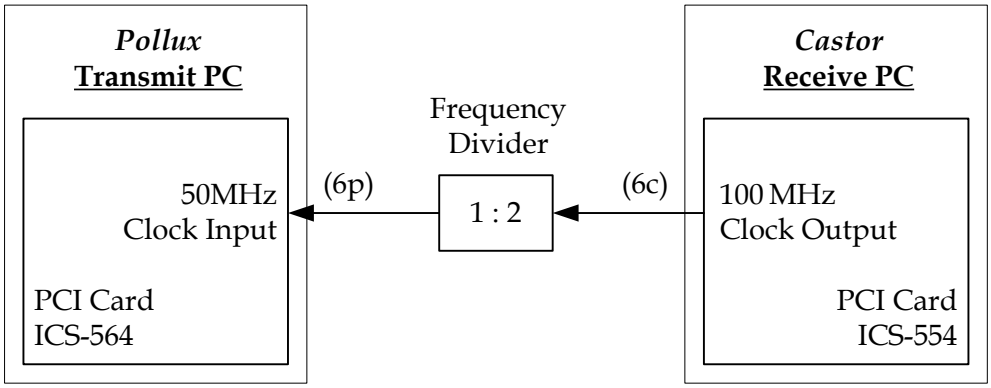

Figure 2.5.2: External clock synchronization.
(See Figure 2.2.3 on page 10 for the location of the connectors.)

It is not possible to provide an external clock to the receiver board of Castor without modifying the hardware. This option was therefore not considered in the MATLAB INTERFACE.

2.5.6 TX Delay Between Blocks

The MATLAB INTERFACE offers two ways to introduce a delay in the transmission of subsequent blocks:

2.5.6.1 Delaying by Using the C++ Sleep(...) Function:

If multiple blocks of data are transmitted, this parameter sets the delay between subsequent blocks in ms. Therefore, it can be set to any **positive integer value**.

The transmissions is delayed by using the C++ `Sleep(...)` function. Measurements showed, that the delay introduced by this function is quantized. The resulting delays last 10 ms, 25 ms, 41 ms, and so on (measured values). However, this delay is highly constant, as several tests proved. If,

for example, a delay of 5 ms is set, the resulting delay is 10 ms, and it is always 10 ms. The exact delay measured is jittering about ±0.08 ms.

Because it is not possible to detect the end of a transmission with the used hardware, the delay is always set between the end of the reception (data stored to the PC Memory, BNC trigger signal goes high) and the beginning of the next transmission (BNC trigger signal goes low). The data is stored continuously during the reception, thus the time marked "Store Data" in the following Figure 2.5.3 is the time which is needed to complete the ongoing storage process. See Section 2.4.1 for further details on the triggering process.

Using the same setup as for the example in Figure 2.5.1 but setting a delay of 5 ms, the following transmit signal was measured. The complete example can be obtained from `\samples\xmTXMultiRXMulti_Sine.m` by changing the *'TX Delay Between Blocks'* parameter to '5'.

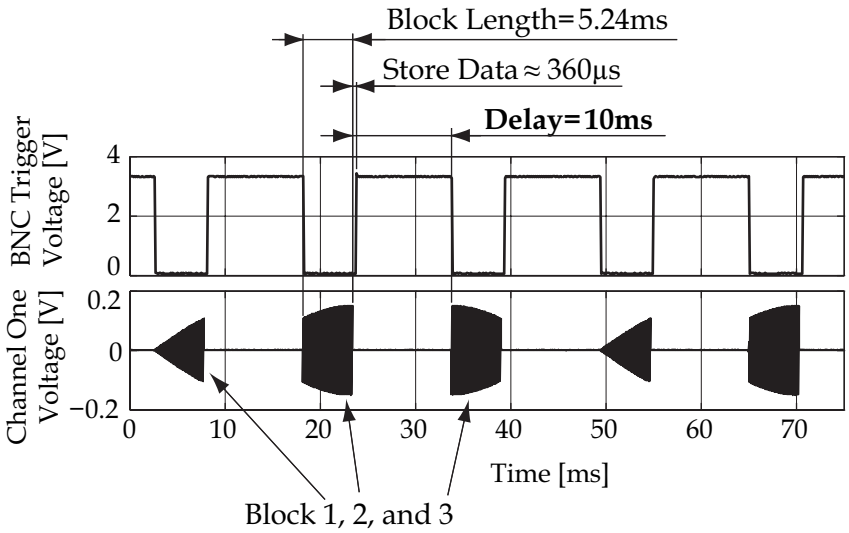

Figure 2.5.3: Measurement, delayed multiple transmit blocks.
`xvTXDataOptions = [3,4,70,32,1,5,0,0,0,0,0,0,0]`

The overall delay between subsequent transmit blocks is 10.36 ms (measured value).

2.5.6.2 Delaying by Using the CPU's High Performance Counter:

To introduce highly accurate delays, the MATLAB INTERFACE is able to delay the transmission of subsequent blocks by using the CPU's internal High Performance Counter. Therefore, the time is measured by using the `QueryPerformanceCounter(...)` Windows function.

The *'TX Delay Between Blocks'* parameter can be set to any **negative value** to introduce such a delay. If, for example, a delay of -11.25 ms is set, the resulting delay between the beginnings of subsequent blocks is 11.25 ms, and it is always 11.25 ms. The exact delay measured is jittering about ±0.06 ms.

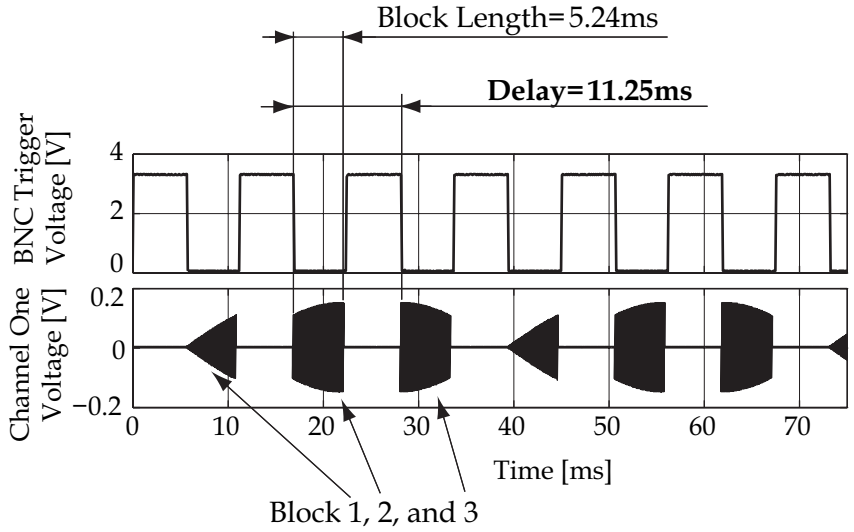

Figure 2.5.4: Measurement, delayed multiple transmit blocks.
`xvTXDataOptions = [3,4,70,32,1,-11.25,0,0,0,0,0,0,0]`

As a basic drawback, the waiting function consumes 100 % of the CPU's processing power to introduce this delay. It has not been tested yet if this has any negative side effects.

As an advantage, the delay introduced does not depend on the time needed to save data to the Castor's hard-disk. This is especially important, if the *'RX Force HDD Operation'* flag is set to store data directly to the hard-disk (see Section 2.5.13.5).

2.5.7 RX Number of Samples

This value only has to be set if the *'Operation Mode'* is 2. In this case, it determines the amount of received data samples per channel in one block. This corresponds to the number of rows in `xvRXData`. The actual number of received data samples is sometimes enlarged to fit the needs of the receiver hardware, e.g. it must be a multiple of 128.

If data is only received (*'Operation Mode'*=2), the maximum possible value is 2^{17} if one channel, and 2^{15} if four channels are received (*'RX Number of Channels'*)

If *'RX Number of Samples'* is set to '0', the value is automatically determined as the number of transmit samples (rows in `xvTXData`) multiplied with the ratio of the interpolation factors of the transmitted and received data streams (*'TX Interpolation Factor'*/*'RX Interpolation Factor'*).

2.5.8 RX Number of Channels

This value, the number of received channels (the number of columns in `xvRXData`), is completely independent from the number of transmit channels (the number of columns in `xvTXData`).

It can be set to 1 or 4. No other value is possible. In the former case data will be received only on channel one. Receiving data, e.g., only on channel two is not possible.

2.5.9 RX Number of Repetitions

This value has to be set if the *'Operation Mode'* is 2. It determines the number of blocks received.

If set to 0, the value is automatically determined as the number of transmit blocks (see Section 2.3.7) multiplied with the number of their repetitions (*'TX Repeat How Often'*).

If *'TX Repeat How Often'* is set to *'inf'*, only a continuous reception of data is possible (see *'Flag: TX Continuous Mode'*). The value must also be set because it cannot be determined automatically:
e.g.: `xvTXDataOptions =[3,inf,70,32,1,0,1,2^15,1,10,0,0,2]`

See `\samples\xmTXInfRXMulti_Sine.m` for an example on how to perform this operation using the MATLAB INTERFACE.

In contrast to Item 6 on page 23 the received blocks of data are now stored in more than one file. Therefore, each block of data must be read separately:

```
xcRXDir  = '\\Castor\MIMO_RXData\';
load([xcRXDir xvFileName '_1.mat']);
xvRXData1 = double(xvRXData);
load([xcRXDir xvFileName '_2.mat']);
xvRXData2 = double(xvRXData);
...
```

The rest of the receiving procedure remains the same. See *'Flag: RX use .dat File'* for a possibility to avoid too many `.mat`-files.

2.5.10 RX Center Frequency

This is the center frequency of the bandpass signal to be down-converted. It can be set to any value between 0 and 100 MHz. The theoretical resolution is smaller than 0.1 Hz while the accuracy is limited by the internal crystal oscillator to a much higher value.

If set to 0, the center frequency of the receiver is set to the same value as that of the transmitter. There is no need to do so.

2.5.11 RX Interpolation Factor

As described in Section 2.4.2 on, the intermediate frequency signal is interpolated by a user configurable factor after the analog to digital conversion. This factor can be determined by this parameter to an integer value between 32 and 16384 in steps of 8. Thus the baseband sample rate is 200 MHz divided by the interpolation factor. The interpolation factor of the transmitter is independent from the interpolation factor of the receiver.

2.5.12 RX Filter

This parameter sets the receive filter to be used in the down-converter. A CFIR[13] and a PFIR[14] filter can be chosen independently. If, e.g., a CFIR_68 (=3) and a PFIR_150 (=5) filter is desired, *'RX Filter'* must be set to 3+5·16.

[13] CFIR = Cascaded Integrator Comb Finite Impulse Response Filter, digital filter
[14] PFIR = Programmabe Finite Impulse Response Filter, digital filter

```
lower 4 bits:          higher 4 bits:
CFIR_80   = 0          PFIR_80       = 0
CFIR_17   = 1          PFIR_17       = 1
CFIR_34   = 2          PFIR_34       = 2
CFIR_68   = 3          PFIR_68       = 3
CFIR_100  = 4          PFIR_100      = 4
CFIR_150  = 5          PFIR_150      = 5
CFIR_GSM  = 6          PFIR_GSM      = 6
CFIR_IS95 = 7          PFIR_IS95     = 7
CFIR_UMTS = 8          PFIR_UMTS     = 8
CFIR_4014 = 9          PFIR_UMTS_24X = 9
                       PFIR_DAMPS    = 10
```

The standard value is 0. See [13] and [9] for further information on the filters.

2.5.13 Flags

This last parameter is used to set several additional features in the MATLAB INTERFACE. Each of these flags can be set by just adding the corresponding value. If, for example, *'TX Continuous Mode'*, *'RX Continuous Mode'* and *'Create .received File'* are desired, set the Flags to 1+2+16.

2.5.13.0 Flag 1: TX Continuous Mode

(Two to the power of the section number corresponds to the flag value)

Setting this flag forces the transmission to be continuous with time gaps in between the blocks. If a reception is carried out, this also has to be continuous in this case.

Due to the limited speed of the internal bus of Pollux this operation mode does not work with a too low interpolation factor in the up-converter board of the transmitter. To guarantee a continuous stream of baseband data, the *'TX Interpolation Factor'* must be an integer value bigger than four times the number of channels transmitted (columns in `xvTXData`).

If the data is directly transferred from the hard-disk by using the 'TX force HDD operation' flag, the interpolation rate has to be a lot larger. The `\samples\xmTXMultiple_FMradio.m` example uses this ability to transmit one block after the other to create an FM radio transmit signal from a music file. It can be received by using an ordinary FM radio. Further information can be obtained by examining the example.

2.5.13.1 Flag 2: RX Continuous Mode

Setting this flag, forces the reception to be continuous with no time gaps in between the blocks. If a transmission is carried out, this also has to be continuous or looped *'inf'* in this case.

To guarantee a continuous stream of baseband data, the *'RX Interpolation Factor'* must be an integer value bigger than eight[15] times the number of channels transmitted (columns in `xvRXData`).

Using the same setup as for the example in Figure 2.5.1 but transmitting continuously with no spaces between the blocks, the following transmit signal was measured:

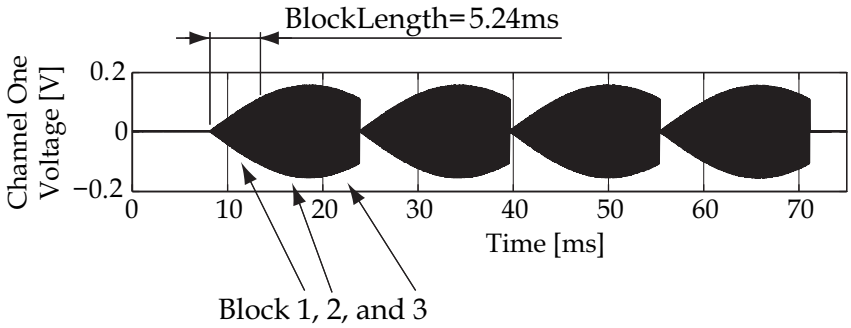

Figure 2.5.5: Measurement, continuously transmitted blocks.
`xvTXDataOptions = [3,4,70,32,1,0,0,0,0,0,0,0,1+2]`

The `\samples\xmTXMultiRXMulti_Sine.m` example can be simply modified by setting the *'RX Continuous Mode'* and *'RX Continuous Mode'* flags to show this behavior.

2.5.13.2 Flag 4: TX do not Delete .mat File

This flag forces the transmitter to not delete the `.mat` file containing all the data needed for a transmission. Setting this flag, it is possible to trigger identical transmissions by just creating a `.do`-file.

The `\samples\xmTXRX_Sine_DoNotDeleteMat.m` example shows how to use this flag.

[15] In contrast to the transmitter, a two times greater interpolation factor is needed because the received data requires eight instead of four bytes per sample.

2.5.13.3 Flag 8: RX Use .dat File

This flag forces the receiver to create a *single* `.dat`-file instead of multiple `_i.mat`-files, where `i` is the number of the received block. This `.dat`-file can be read by using the `\samples\xmICS_ReceiveData.dll` MATLAB function. The following source code therefore replaces item 6 on page 23 if multiple blocks of data are received:

```
xcRXDir  = '\\Castor\MIMO_RXData\';
xvRXDataOptions = ...
  [ 0    , ...  %      int     Position in Samples
    2^15 , ...  % ( 6) int     RX Number of Samples
    1    , ...  % ( 7) int     RX Number of Channels
    0           %      boolean RX Check CW Data
  ];
xvRXData=double(...
  xmICS_LoadData(xvRXDataOptions,[xcRXDir xvFileName '.dat']));
```

In fact, Castor always creates a `.dat`-file internally and afterwards stores the data to multiple `_i.mat`-files. This last step can be omitted by setting the *'RX Use .dat File'* flag. The transmission gets a lot faster (the data is not loaded and saved to `_i.mat`-files), and only one file is created instead of many single ones.

Used to load data from the internal hard-disk of a PC (not from a network path), the `xmICS_LoadData(...)` is a lot faster in reading data than the MATLAB `load(...)` command. For example, 60 MBytes/s instead of 10 MBytes/s were measured on a common PC. Data must be read from positions which are multiples of the sector size of the hard-disk to show this behavior. For a sector size of 512 bytes this is automatically ensured.

- **'Position in Samples'** sets the position where to start reading the data samples. It can be set to any number, even somewhere in the middle of a block.
- **'RX Number of Samples'** specifies the amount of data samples to be read. Even multiple blocks of data can be read at once, by setting this value greater than the actual block length.
- **'RX Number of Channels'** must be set to the number of channels which were received. If, for example, the actual transmission received four channels, this value must be set to '4', even if only one channel is needed.

- **'RX Check CW Data'** can be set to '0' or '1'. If set to '1', the loaded data samples are checked for being continuously received without any gaps in between. See *'RX Check CW Data'* flag for further details.

The `\samples\xmTXMultiRXMulti_Sine_Dat.m` example contains further details on how to implement this feature.

2.5.13.4 Flag 16: Create .received File

This flag forces the receiver to create a `.received` file when the actual transmission of data has completed. In the handshaking diagram on page 24 this is right before the block marked '(12)'. The user is able to, e.g., change the channel while the data is saved to the hard-disk of Castor which can need quite a long time.

To do so, the following code has to be inserted between item 3 and 4 on page 22:

3a. Wait for a `.received`-file (polling operation) on Pollux. This `.received`-file indicates that the actual transmission over the channel has been completed. The received data is *not* available yet.

```
while ((size(dir([xcTXDir xvFileName '.done'    ]),1)==0)&&...
       (size(dir([xcTXDir xvFileName '.received']),1)==0))
   pause(0.2);
   end;

[Insert your own code here, e.g. to change the channel]
```

The `\samples\xmTXMultiRXMulti_Sine_Received.m` example contains futher details. Before a transmission is initiated, the previous `received`-file has to be deleted:

```
if size(dir([xcTXDir xvFileName '.received']),1)~=0
   delete([xcTXDir xvFileName '.received']);
   end;
```

2.5.13.5 Flag 32: TX force HDD operation

This flag forces the transmitter to load the transmit data samples directly from the hard-disk via the internal memory into the transmit FIFOs. Therefore, less memory is needed to carry out a transmission. Instead of sufficient memory to store all data blocks, now only enough memory for one block is needed.

- The delay between subsequent blocks is not constant because of the not constant transfer rate from the hard-disk. Negative *'TX Delay Between Blocks'* values can be used to deal with this problem.
- *'TX/RX Continuous Mode'* will require much higher interpolation rates to work properly.

Using this mode is only recommended if the amount of memory is not big enough to store all data samples needed for a transmission.

2.5.13.6 Flag 64: RX Force HDD Operation

The same as *'TX force HDD operation'* but for the receiving PC.

2.5.13.7 Flag 128: Sync via File

This mode is used to omit the 'BNC Trigger connection' described in Item 1. on page 10. Block (8) and (9) shown on page 24 still stay the same but are realized by creating a file (8) and polling for this file (9).

As a basic drawback of this flag, synchronization via files is neither fast nor accurate. Small and/or constant delays between subsequent blocks are impossible. Using negative *'TX Delay Between Blocks'* values may solve the last problem if the delay set between the blocks is big enough.

2.5.13.8 Flag 256: RX Check CW Data

The received data samples are numbered in the receiving hardware and can therefore be checked by the MATLAB INTERFACE for being continuously received. Use this flag to ensure that the *'RX Continuous Mode'* is working properly. Using this flag requires a lot of computing power; thus, omit it if not needed.

2.5.14 The Maximum Number of Samples per Block

The following strategy can be used to determine the maximum number of transmit samples per channel and block (= the number of rows in `xvTXData`):

1. Assume that the number of samples per channel and block is 2^{17}.
2. If more than one block of data is transmitted, it is 2^{16}. (For receiving data only, this limit does not apply. It also does not apply if the transmit data is looped *'inf'*)
3. If four channels are received, it is 2^{15}. (For transmitting data only, this limit does not apply.)

Although not recommended, it is also possible to receive 2^{16} samples with four channels. Because a very time critical process is used for this operation, there is no guarantee that this really works in an operating system as Windows. Therefore, avoid to receive 2^{16} samples on four channels if possible.

2.6 Interfacing the MIMO Testbed via FTP

As previously described, the MATLAB INTERFACE not only enables the possibility to access the MIMO testbed developed from the local area network but also from anywhere in the world. Because the MATLAB INTERFACE uses ordinary files to handshake, it can be easily adapted to work over a Secure FTP connection. The `\samples\ftp\xmTXRX_ftp.m` example shows how to do so and transmit a 16 QAM signal.

Tests showed, that a `.do`-file created via a Secure FTP connection cannot be deleted immediately[16]. Therefore, the flag numbered '512' has to be set by the user to solve this problem. The example contains further details on the implementation.

[16] It is possible to delete this file immediately after it was created but for unknown reasons it will be created again.

Chapter 3

Extending the Testbed with DSP/FPGA Boards

By using DSP/FPGA boards, the user is able to extend the possibilities of the developed MIMO testbed significantly:

- These boards run **a lot faster** than ordinary PCs.
- This enables the opportunity implement and test DSP codes in an **real-time environment** not just for transmission but also for detection.
- Furthermore, they are **closer to a final product** since not all that works in MATLAB can be implemented in a final product.

Figure 3.1 shows how the MIMO testbed can be extended by Sundance SMT-365 DSP/FPGA boards (see Appendix C) [16]. There is *no* direct connection between the DSP/FPGA boards and the MIMO testbed:

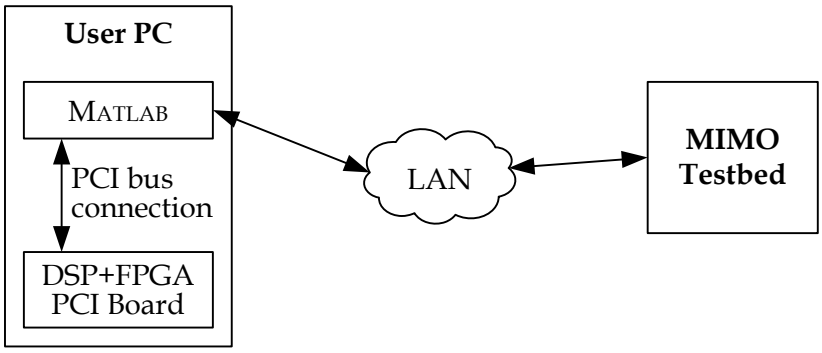

Figure 3.1: The Sundance SMT-365 DSP/FPGA board.

The DSP/FPGA boards are equipped with a 600 MHz Texas Instruments TMS320C6416 64 bit fixed point DSP (4800 MIPS peak performance), a Xilinx Virtex II XC2V1000-4-FF896 FPGA, and 8 Mbytes of high speed RAM.

Figure 3.2 shows a block diagram of the overall testbed extended by these DSP/FPGA boards (User 3) and a Secure FTP connection (User 1):

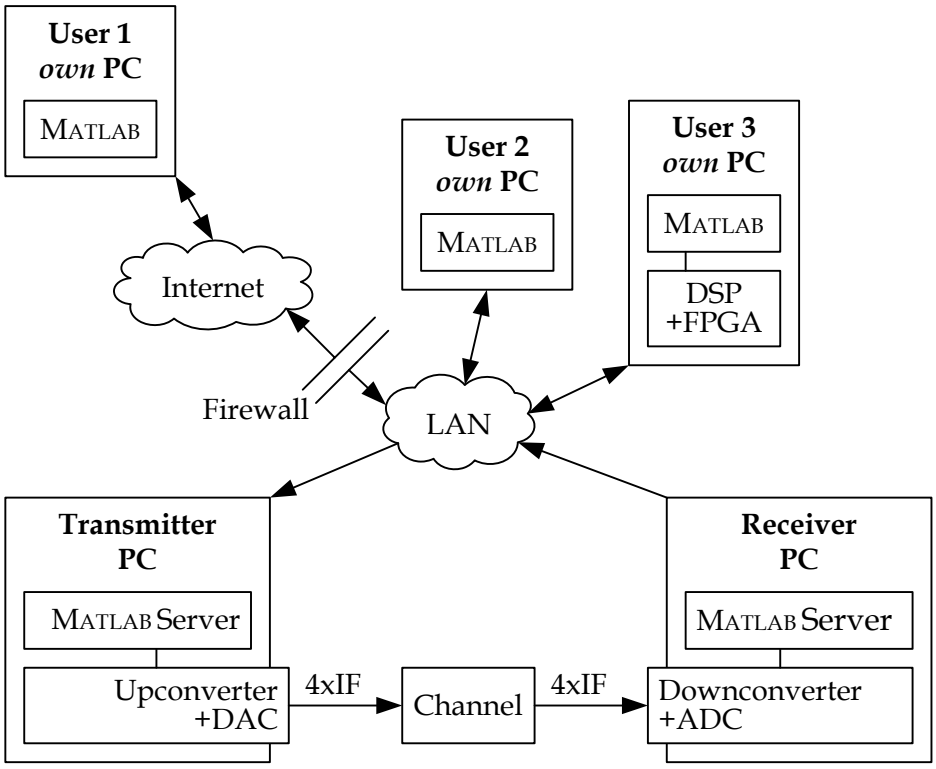

Figure 3.2: The complete MIMO testbed with several users.

Inserting the DSP/FPGA boards into the user PC and not into the transmit/receive PC to generate complex baseband data samples has the following advantages:

- Using the MATLAB INTERFACE there is the possibility to test code in an DSP/FPGA environment without knowing the hardware of the up- and down-converter boards in detail.
- Multiple users can simultaneously develop code on multiple DSP/FPGA boards on their own PCs without interrupting the operation of the MIMO testbed.

- New types of DSP/FPGA boards can be integrated into the existing structure of the testbed without any need for complicated connections in hardware.
- By using MATLAB to transfer the data to the MIMO testbed, this MATLAB code can also be adapted to preprocess the data. E.g., this enables the possibility to implement receiver functions part by part onto the DSP while still simulating all the other parts in MATLAB.
- Furthermore, MATLAB can be used to display and analyze the results of the data processed in the DSP.

Due to a lack of time this setup was not *systematically* tested. Examples on how to transfer data from the DSP/FPGA board can be obtained from `\\Pollux\MIMO_TXData\samples\sundance\*.*` and [16].

Conclusion

During this diploma thesis, a 4×4 MIMO testbed has been set up. Several firmware related issues of the hardware have been solved to put the overall system into operation. Much emphasis was put on the development of an easy to use interface.

This so called MATLAB INTERFACE is the major innovation for such a testbed. It dramatically reduces the code development time to perform a transmission. For example, not more than seven lines of MATLAB code are needed to carry out a transmission.

The usage of MATLAB, in combination with the MATLAB INTERFACE developed for accessing the testbed, allows a user to test a multitude of different setups and signal parameters with nearly no effort. It is the power of this combination that makes the MIMO testbed developed so attractive for code development and testing.

By using DSP/FPGA boards and transmitting the complex baseband data samples via the MATLAB INTERFACE, coding and encoding schemes can be implemented in a real-time environment being close to a final product. Multiple users can simultaneously develop DSP/FPGA code on multiple boards on their own PCs without interrupting the operation of each other and the MIMO testbed.

Presently, the testbed is operating continuously. More then five Tera bytes of data have already been transmitted and received without any trouble. Bit error ratio measurements showed exact matching with existing simulation results, promising great possibilities for future work with this testbed.

Appendix A

Picture of the ICS-564 board

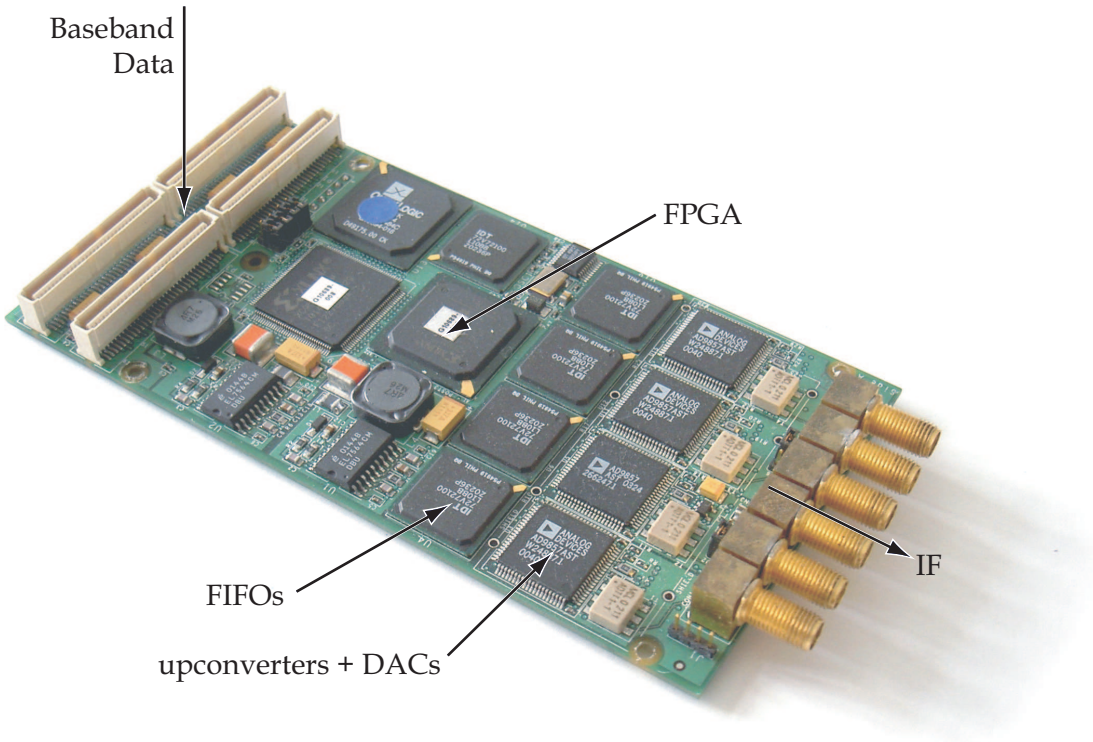

Figure A.1: The ICS-554 up-converter+ADC board.
(This board is plugged onto a PCI carrier board.)

Appendix B

Picture of the ICS-554 board

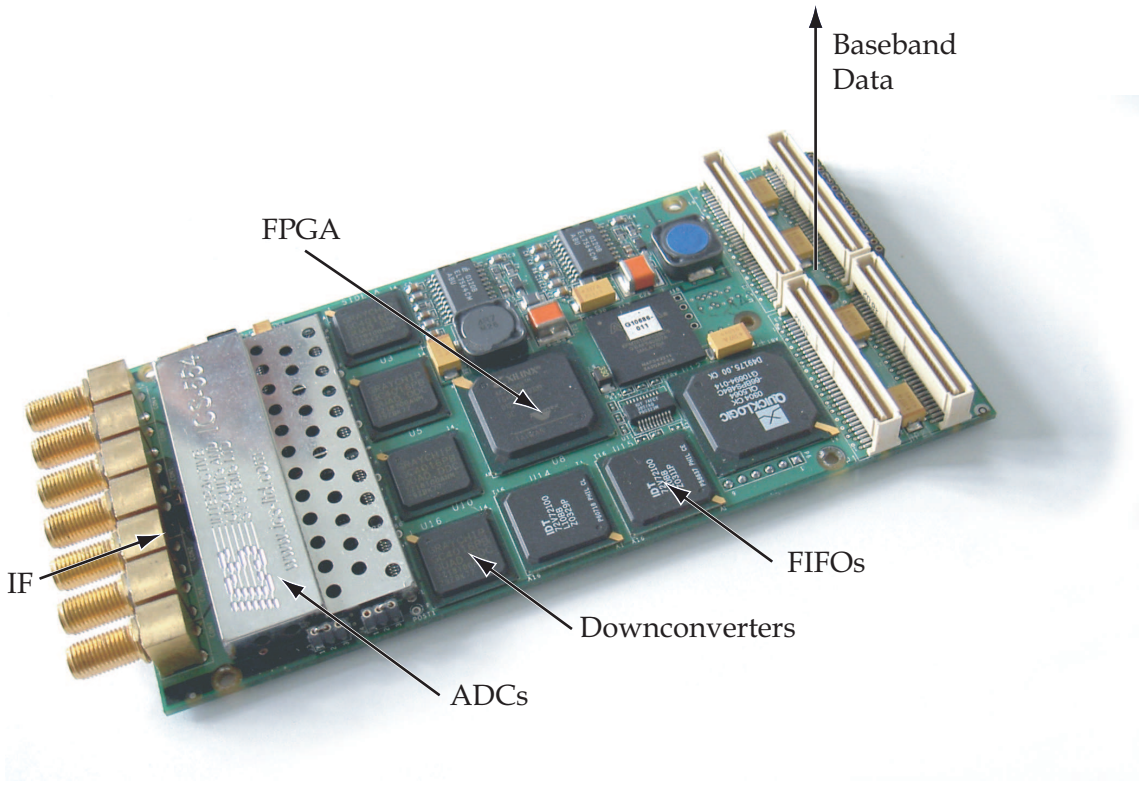

Figure B.1: The ICS-554 down-converter+ADC board.
(This board is plugged onto a PCI carrier board.)

Appendix C

Picture of the SMT-365 board

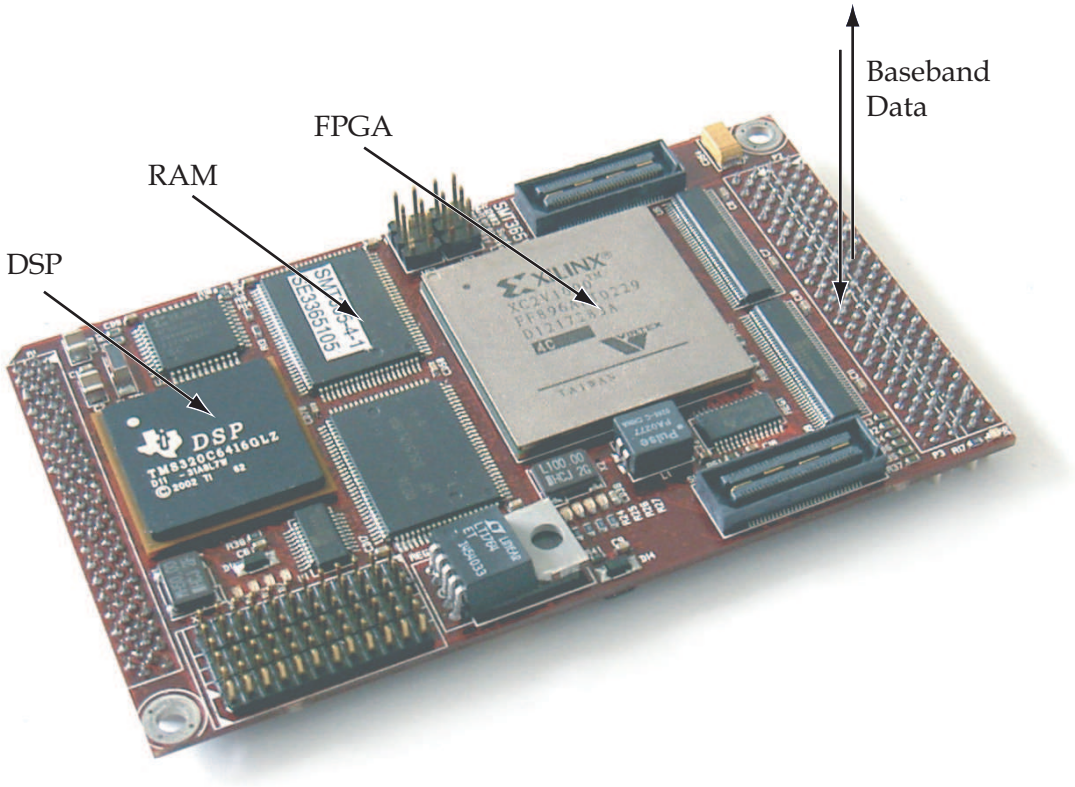

Figure C.1: The SMT-365 DSP/FPGA board.
(This board is plugged onto a PCI carrier board.)

List of Abbreviations

ADC	Analog to Digital Converter, converts digital data samples to an analog signal
CFIR	Cascaded Integrator Comb Finite Impulse Response Filter, digital filter
DAC	Digtal to Analog Converter, converts digital data samples to an analog signal
DSP	Digital Signal Processor
FIFO	First In First Out, memory to buffer a stream of data asynchronously
FPGA	Field Programmable Gate Array, a user programmable logic device
HSC	High Speed Channel, The interface between the DSP/FPGA boards and the PC via the PCI bus
ICS	Interactive Circuits and Systems Ltd, Canadian Company
IF	Intermediate Frequency, in our case between 0 and 100 MHz
LAN	Local Area Network, connection between computers
LVTTL	Low-Voltage Transistor-Transistor Logic, logic signal level standard
MIMO	Multiple Input Multiple Output, more than one antenna is used
MISO	Multiple Input Single Output, multiple antennas transmit, while just one receives
OFDM	Orthogonal Frequency Division Multiplexing, modulation scheme

PCI	Peripheral Component Interconnection, system bus in a personal computer
PFIR	Programmabe Finite Impulse Response Filter, digital filter
RF	Radio Frequency
SHB	Sundance High Speed Bus
SMT	Sundance Multiprocessor Technology Ltd., company, manufacturer of the DSP/FPGA boards
TI	Texas Instruments, company, manufacturer of the DSP on the DSP/FPGA boards
UMTS	Universal Mobile Telephone System, mobile telephone system standard
VHDL	Very High Speed Integrated Circuits Hardware Description Language, programming language used for FPGAs or ASICs
Xeon	Intel™ Xeon® Processors are processors specially designed for server platforms

Bibliography

[1] E. Aschbacher, S. Caban, C. Mehlführer, G. Maier, M. Rupp: *"Design of a Flexible and Scalable 4×4 MIMO Testbed,"* to be published in Proc. of the 11th IEEE Signal Processing Workshop, Taos Ski Valley, New Mexico, USA, August 2004.
http://www.dsp2004.nmsu.edu/
http://www.nt.tuwien.ac.at/rapid_prototyping
/publications/papers_ab/paper_dsp04.pdf

[2] S. Caban, R. Langwieser, C. Mehlführer, E. Aschbacher, W. Keim, G. Maier, B. Badic, M. Rupp, and A. L. Scholtz: *"Design and Verification of a Flexible and Scalable 4×4 MIMO Testbed,"* to be presented at RAWCON Workshop Ws2: "MIMO Implementation Aspects", Atlanta, Georgia, USA, September 2004.
http://www.rawcon.org/

[3] C. Mehlführer: *"Implementation Real-Time Testing of Space-Time Block Codes,"* Diploma Thesis, Technical University of Vienna, Institute of Communications and Radio-Frequency Engineering, June 2004.
http://www.nt.tuwien.ac.at/rapid_prototyping
/publications/diplomatheses/DA_Mehlfuehrer.pdf

[4] R. Langwieser: *"Entwicklung von HF-Baugruppen für ein MIMO Echtzeit Übertragungssystem,"* Diploma Thesis, Technical University of Vienna, Institute of Communications and Radio-Frequency Engineering, April 2004.
http://www.nt.tuwien.ac.at/rf-electronics
/rmastheses/langwieser_dipl.pdf

[5] L. Mayer: *"Development of an RF Frontend for 5.2 GHz MIMO Real-Time Transmission Experiments,"* Running Diploma Thesis, Technical University of Vienna, Institute of Communications and Radio-Frequency Engineering, Expected February 2005.
http://www.nt.tuwien.ac.at/rf-electronics

[6] *"CMOS 200 MSPS 14-Bit Quadrature Digital Upconverter AD9857,"* Analog Devices, Inc., Rev. B, 2002.
http://www.analog.com
/UploadedFiles/Data_Sheets/28341907AD9857_c.pdf

[7] Analog Devices: *"14-bit, 80/105 MSPS A/D Converter AD6645,"* Analog Devices, Inc., Rev. B 2003.
http://www.analog.com
/Data_Sheets/3945936639403804021039906151995 4AD6645_b_.pdf

[8] IDT: *"3.3V Volt High Density Supersync II 72-Bit FIFO IDT72V72100"*, IDT., September 2003.
http://www1.idt.com
/pcms/getDoc.taf?docID=8503

[9] *"GC4016 Multi-Standard Quad DDC Chip Data Sheet,"* Texas Instruments Inc., Rev. 1.0, August 2001.
http://www-s.ti.com
/sc/ds/gc4016.pdf

[10] ICS: *ICS-564 "Operating Manual,"* Interactive Circuits And Systems Ltd., Ontario, Canada, March 2003.
http://www.ics-ltd.com

[11] ICS: *"ICS-564 Hardware Delevopment Kit, User's Manual,"* Interactive Circuits And Systems Ltd., Ontario, Canada, March 2003.

[12] ICS: *"ICS-564 Windows Software Development Kit, User's Manual,"* Interactive Circuits And Systems Ltd., Ontario, Canada, 2003.

[13] ICS: *"ICS-554 Operating Manual,"* Interactive Circuits And Systems Ltd., Ontario, Canada, May 2003.

[14] ICS: *"ICS-554 Hardware Delevopment Kit, User's Manual,"* Interactive Circuits And Systems Ltd., Ontario, Canada, May 2003.

[15] ICS: *"ICS-554 Windows Software Development Kit, User's Manual,"* Interactive Circuits And Systems Ltd., Ontario, Canada, 2002.

[16] Sundance: *"SMT 365 Design Specification"*, Sundance Microprocessor Technology Limited, May 2001.
`http://www.sundance.com`