Diplomarbeit

Wireless Testbed Receiver

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Diplom-Ingenieurs

unter der Leitung von

Univ.Prof. Dipl.-Ing. Dr.techn. Markus Rupp

Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Sebastian Caban

Institute of Telecommunications

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Heinz Haderer

Fichtenweg 5
4323 Münzbach

Wien, Dezember 2011

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

Heinz Haderer
Vienna, Dezember 15, 2011

Abstract

These days, the demand for mobile transmission bandwith seems to be a never ending story. To get insights into the real world, wireless transmission measurements and not only simulations of mobile communication systems are necessary. For this task, the Vienna Wireless Testbed has been developed. To keep pace with future developments, the testbed has to be extended and updated constantly. Furthermore, before publishing measured results, one has to make certain that they are not degraded due to impurities or hardware that does not work properly.

In this thesis, the receiver, which is a main part of the Vienna Wireless Testbed, is set up for future tasks. Major improvements in quality and flexibility are achieved by replacing the analog to digital conversion plug in board. Furthermore, applying the capabilities of a novel timing responsible for triggering, a fast and secure way of transmitting huge numbers of blocks within short periods of time is achieved. Furthermore, a methodology for rapidly testing the receiver's radio frequency front end hardware is developed. Moreover, measurement results for the current testbed configuration are shown. The receiver hardware is tested in particular with an orthogonal frequency-division multiplex signal which is also used in the next generation mobile communication standard, namely Long Term Evolution (LTE). An outlook on what steps could be taken to improve the performance of the testbed receiver is given at the end of this thesis.

Kurzfassung

Aus heutiger Sicht scheint der Bedarf nach mehr Datendurchsatz in mobilen Anwendungen auch in Zukunft stets zuzunehmen. Um reale drahtlose Übertragungen besser zu verstehen sind Messungen unter realen Bedingungen und nicht nur Simulationen notwendig. Deshalb wurde in den vergangenen Jahren das Vienna Wireless Testbed aufgebaut. Um mit dem zukünftigen Entwicklungen schritt zu halten ist es notwendig, das Testbed zu erweitern, beziehungsweise zu erneuern. Außerdem ist insbesondere beim Publizieren von Messergebnissen sicherzustellen, dass diese nicht durch Störungen oder durch fehlerhafte Teile der Messausrüstung verfälscht werden.

In dieser Arbeit wird der Empfänger, welcher ein fundamentaler Bestandteil des Vienna Wireless Testbeds ist, für zukünftige Aufgaben vorbereitet. Wesentliche Verbesserungen an der Qualität von Messungen und zusätzliche Flexibilität werden durch den Austausch des Analog-Digital-Wandlermodules erzielt. Durch die Benutzung einer neuen Timing-Einheit, welche unter anderem für die Auslösung von Messungen verwendet wird, wird eine schnelle und sichere Art, um eine hohe Anzahl von Blöcken innerhalb kurzer Zeit zu übertragen implementiert. Im zweiten Teil der Arbeit wird eine geeignete Methode entwickelt, um in angemessener Zeit die Hochfrequenzbaugruppe des Testbeds zu testen. Die gemessenen Ergebnisse für die derzeitige Konfiguration werden gezeigt. Im speziellen wird die Hochfrequenzhardware mit einem „orthogonal frequency division-multiplex Signal“ für drahtlose Übertragungen, welches auch im Mobilfunkstandard der nächsten Generation verwendet wird, getestet. Am Ende wird ein Ausblick gegeben, wie man das Testbed weiter verbessern könnte.

Contents

1 Introduction

The receiver is one main part of the Vienna Wireless Testbed. The receiver's hardware and software has been developed and improved to do different measurements [1]. An overview of the testbed receiver is shown in Figure 1.1. The signal is received at a center frequency of 2.503 GHz via one to four antennas connected to the testbed's input. Afterwards the signal passes the radio frequency front end of the testbed receiver. At the end of this part, the signal is filtered and downconverted to an intermediate frequency of 70 MHz. Next, the analog signal is converted to the digital domain using an ADC[1] plug in board which is connected to the internal PCI Express[2] bus of a personal computer. After that step, the samples are temporally stored in different buffer memories and at the end, they can be used to carry out immediate calculations; for example, the feedback calculation for retransmissions and/or the samples can be stored using a hard disk RAID[3] array.

A small piece of software is needed to gain access to the functionality of the PCI Express plug in board and to establish a link between that board and the testbed environment. This software has to have small processing delays and a high data throughput to allow single block transmissions in reasonable time to get statistically meaningful results. This software is called daemon because

[1] ADC ... analog-to-digital converter

[2] PCI Express ... the Peripheral Component Interconnect Express is an input/output interconnect for computing platforms which is designed to replace the older PCI bus.

[3] RAID ... originally Redundant Array of Inexpensive Disks but nowadays it is an acronym for Redundant Array of Independent Disks

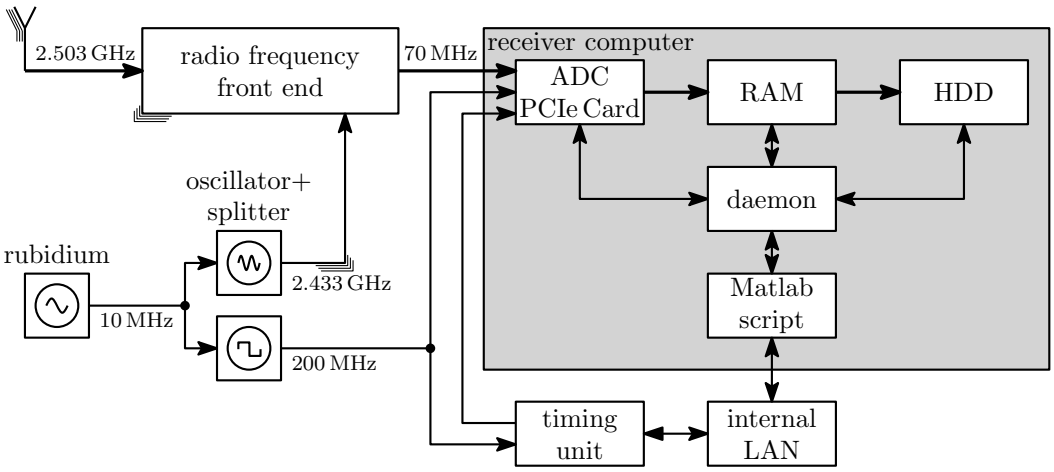

Figure 1.1: Overview of the testbed's receiver.

it is running in the background.

For controlling and to get a link to higher level programming, a Matlab script is also running on the testbed's receiver computer. Due to the fact that the testbed consists of at least one transmitter and one receiver, which can be placed at different locations, a special unit for timing has been developed (see [2]). The testbed computers and the timing units are connected through a separate local area network that is only used for signaling between the different testbed components to reach low network delays. For accurate sampling, an external oscillator at 200 MHz locked to a frequency standard is used. This clock signal is passed to the timing unit and to the ADC PCI Express plug in board, thus phase stable triggering is possible.

For future measurements, especially to get insights into the physical layer of the LTE[4] standard, upgrading the testbed receiver is unavoidable. A major improvement is reached by substituting the ADC plug in board by a new improved module.

As a replacement, the X5-RX PCI Express module which is manufactured by Innovative Integration has been chosen. The X5-RX has four channels, each of them able to sample at a rate of up to 200 MSample/s using a resolution of 16 bit. Therefore, there are enough resources to satisfy future and not only current needs. Furthermore, the X5-RX plug in board consists of a module which connects the board to the computer via the PCI Express interconnect which supports up to eight lanes to achieve a high data transfer rate. Moreover,

[4] LTE ... Long Term Evolution, a mobile network technology standard

if the requirements are more sophisticated, the application logic on an onboard FPGA[5] can be modified to fulfill them.

As a consequence of replacing the ADC plug in board, the testbed receiver has to be extended with a preamplifier to reach a signal power level which is high enough to use the full range of the ADC. Additionally, an amplifier is inserted between oscillator and X5-RX plug in board sampling clock input to satisfy the required clock level according to the datasheet.

Another adaption that was started by the thesis "Hardware-based Timing Synchronization" from Armin Dißlbacher-Fink [2] is continued. Due to the well-thought-out design of his timing unit, accurate triggering of transmitter's and receiver's sampling modules within small delays is possible.

The main functionality of the timing unit used in this work is the trigger function. Each testbed transmitter or receiver has one timing unit. For triggering, each transmitter computer and receiver computer has to send a command via an UDP datagram to the unit. This command contains the time difference from the receiving time point to the point where the trigger pulse should be fired. After firing, the unit tells either if the pulse was correctly fired or not by sending the message success or trigger error back to its corresponding computer. If a packet got lost during the trigger process, the host computer does not receive any answer.

Upgrading the computer's hardware, software, and general maintenance work in such a reengineering process is natural but will not be mentioned in this thesis.

[5] FPGA ... Field Programmable Gate Array

2 Receiver Daemon

2.1 ADC PCI Express Plug In Board

As mentioned in the introduction, the X5-RX plug in board from Innovative Integration is used as converter from the analog into the digital domain. A block diagram is given according to the X5-RX user's manual [3] in Figure 2.1. As it can be seen on the left, there are four analog inputs, each connected to a 16 bit analog to digital converter integrated circuit ADS5485 manufactured from Texas Instruments. The digital outputs are connected to a Xilinx Virtex 5 FPGA. Another input for triggering is logically directly connected to the FPGA. The X5-RX plug in board supports several different ways to obtain the desired sampling clock using the integrated circuit CDCE72010 also from Texas Instruments. To be more accurate, an external oscillator at 200 MHz coupled to a rubidium clock output at 10 MHz as reference is used. Thereby, the integrated circuit CDCE72010 is configured, using the Malibu framework in such a way that the sample clock is connected directly to an input of the FPGA. Furthermore, a computational SRAM is applied on the plug in board. This SRAM is used as local memory for applications in the FPGA if the data does not fit into internal memory structure of the FPGA. This type of memory is realized by two banks of 2 MByte QDR[1] SRAM. For maximum flexibility in memory usage, all control and data lines are directly connected to the FPGA.

[1] QDR ... Quad Data Rate is an improved SRAM technology which allows higher data transfer rates. For details see `www.qdrconsortium.org`.

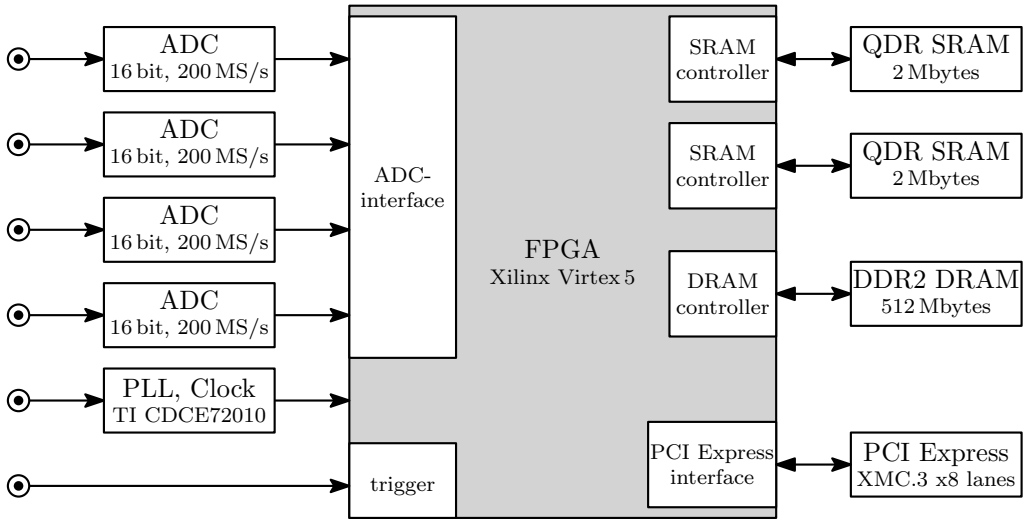

Figure 2.1: Hardware diagram of the PCI Express plug in board.

Additionally, a 512 MByte DDR2 SDRAM[2] memory is provided. The Malibu Framework Logic implements a FIFO[3] functionality to buffer the samples by using this memory. Writing to the buffer memory has the highest priority to make sure that no data is lost during the analog to digital conversion process. If the data transfer rate supported by the PCI Express interface is not high enough, data will be dropped when the buffer is full. Furthermore, an alert is used to signal that data has been dropped. The XMC.3 PCI Express interface from Innovative Integration connects the X5-RX module with the PCI Express system. This interface provides a transfer rate of up to 4 GBytes/s in full duplex. As protocol over this interface the Velocia packet system[4] is used to establish fast and flexible communication.

In Figure 2.2, the data flow between the analog connectors and the host system is shown according to [3]. First the analog signal is sampled. Then data flows into the analog to digital interface component of the FPGA which is controlled by triggering. Next, a correction of gain and offset error is done for each analog channel. After that, the enabled channel's data flows into the buffer that is implemented into the SDRAM which is organized as mentioned before as FIFO. The packetizer pulls data from the queue and creates data

[2] DDR2 SDRAM ... Double Data Rate 2 Synchronous Dynamic Random Access Memory is an interface for random access memory specified by JEDEC (see `www.jedec.org`).
[3] FIFO ... stands for first in, first out which is one way of organizing memory.
[4] The Velocia architecture is a high performance PCI Express interface from Innovative Integration.

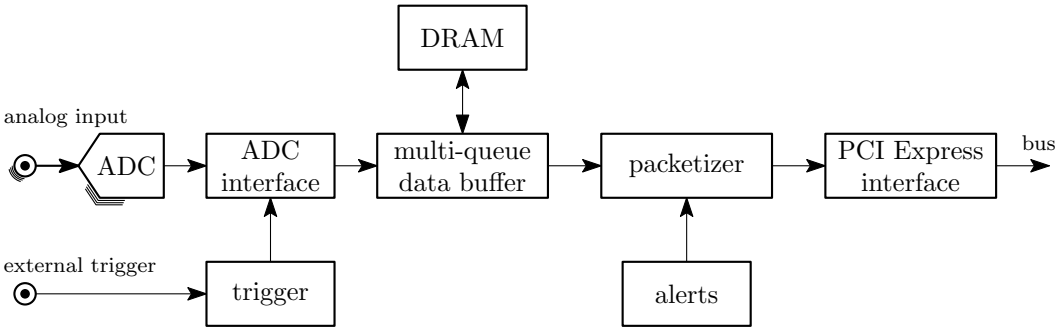

Figure 2.2: Data flow chart of X5-RX plug in board.

packets of a specified size. Then these packets are sent to the PCI Express interface logic. Now the Velocia packet system controls the data flow to the host. This system simplifies the use of direct memory access for data transfer.

2.2 Overview

The receiver daemon is a small program, written in C++, which uses the Malibu Library to get in interaction with the X5-RX plug in board. There are several threads to handle all tasks asynchronously. The main thread manages first the starting procedure, and after that the interaction between daemon and testbed environment. Another asynchronous running task is to handle the measured data. This is included in a background thread, which is managed by Malibu Library. If data is available, this thread signals this by calling an event method which is called HandleDataAvailable. An additional task is to write the received data with high speed on hard disks. This is also realized within a separate thread because complex processing in the HandleDataAvailable event method is not allowed, because of memory limitation on the X5-RX. Because text outputs slow down the processing, for monitoring reasons a separate thread is implemented. This thread only collects information from different parts of the program and prints that information in a format that is readable by humans. The interaction between the different threads is implemented thread save and by delaying the performance relevant task as little as possible.

2.3 Program Startup – Initialization

The receiver daemon has to be started with two parameters. The first parameter specifies path and name of the configuration file, the second defines the number of the testbed device. This number is in today's development state of the testbed for the receiver four, but this can be changed in future settings. For example, a start command can be

```
>X5-RX S:\01_TXRXMatlab\xmTR_Globals.m 4
```

Figure 2.3 shows a flow chart of the daemon's main thread. After starting, the parameters identification name, local listing port, internet protocol address and port number of the corresponding timing unit are read from the configuration file. The path and name of the configuration file are committed as described above. The second command line parameter specifies which part of the configuration file should be used via the testbed device number.

After switching on the computer and booting the operating system, a few tasks have to be done to get the X5-RX plug in board ready for converting signals to the digital domain. Firstly, when the daemon is starting, the Malibu module object is created. Secondly, after shared memory allocation, the baseboard activates the X5-RX module for use by calling the open() method. Now the device driver for the baseboard is loaded and internal resources are allocated. Then a reset command follows to put the module in well-defined state. More detailed information can be found in [3]. Furthermore, the UDP local listing port, which is specified in the configuration file, is bounded.

Now the receiver daemon's main thread enters a loop. In this loop, a blocking method for receiving UDP datagrams is called. If a command is received via UDP datagram, the first token specifies what section has to be executed. The individual commands are explained below.

This command's receive-execute loop will only be exited when the terminate command is received. After that, the X5-RX module is closed and the allocated memory is freed.

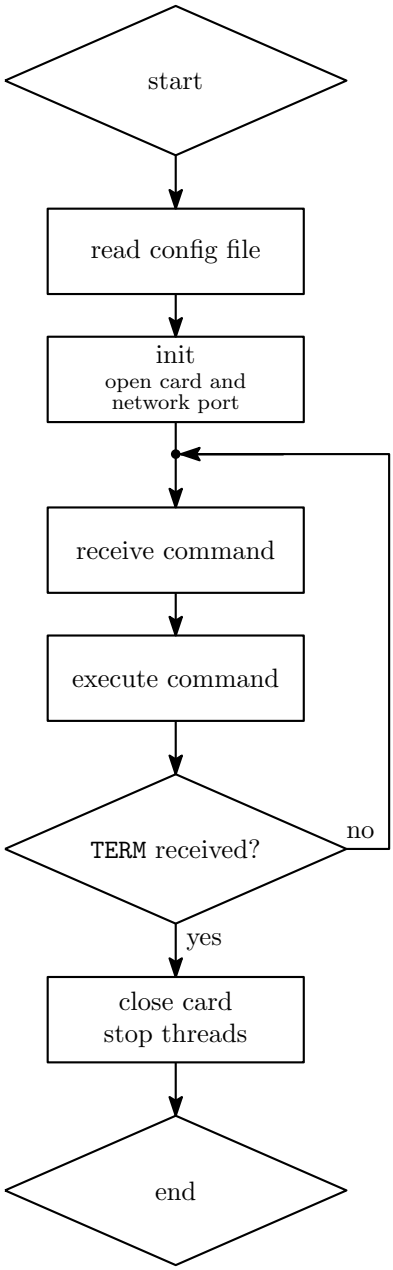

Figure 2.3: Flow chart of receivers main thread.

2.4 Commands

The commands which the receiver daemon accepts, consist of a command code at the beginning which specifies what has do be done. This code is followed by parameters if necessary. The following syntax

`command code|parameters`

is used. The amount of parameters depends on the specific command. If the command has more than one parameter, they are separated via a "|".

2.4.1 Reset

The reset command incorporates two tasks. Firstly, the daemon will be put into a well-defined state by using the parameters explained below. Secondly, the X5-RX will be set into the state where a trigger pulse has the consequence that data will be captured.

command:	**RST**
parameters:	channel count specify which channel/s is/are used. Possible arguments are 1 for using Ch1, 2 for using Ch1 and Ch2 as well as 4 for using Ch1 to Ch4 block size size of a block in bytes for all used channels packet size size of a packet in bytes for all used channels decimation factor decimation factor for the ADC channels trigger delay the trigger instant of time is calculated by the timing unit as sum of the timestamp when the UDP datagram arrives and the time specified by trigger delay. For further explanation see [2]. timout next measurement defines the period of time in ms which will be waited to receive a trigger pulse buffer size size of random access memory buffer in bytes

	done file path defines the path where the done files will be written. If an empty string is sent no done file will be created verbose mode sets the verbosity. Valid parameters are 0 for no output and maximal speed, 1 for low verbosity, 2 for high verbosity.
answer:	OK|<*name*> if reset is done successfully ER|<*name*>|<*error message*> if an error occured during the reset

2.4.2 Ping

A simple way of checking if the correct version of the daemon is running is provided by the ping command.

command:	**PING**
answer:	OK|Version␣<*daemon version as number*> if the daemon program is running

2.4.3 Measurement

The measurement command is used to tell the daemon in which file and on what position the sampled data has to be stored.

command:	**RX**
parameters:	relay output sets the relay output of the timing unit (see [2]). Valid values are defined as string in a rang from 0 to 255. external IO controls outputs on the front panel of the timing unit (see [2]). Valid values are defined as string in a rang from 0 to 63.

	internal IO controls outputs at pins which are available inside the timing unit (see [2]). Valid values are defined as string in a rang from 0 to 255. file name specifies the file name and path where the sampled data should be stored position defines the offset starting at the beginning of the file in bytes
answer:	OK|Version␣<*daemon version as number*>

2.4.4 Fill Level

The fill level command can be used to get the fill level of the daemon's buffer. Controlling the buffer usage has to be done by the testbed environment. The daemon has no algorithm included to avoid buffer overflows.

command:	**FILL**
answer:	OK|<*fill level*> returns the fill level in percent in three digits ER|<*name* >|<*error message*> if an error is occurred

2.4.5 Stop

If the daemon receives the stop command, firstly, writing the buffered data to the hard disk is forced, secondly, all files are closed, and thirdly, the X5-RX is set insensitive to trigger pulses.

command:	**STOP**
answer:	OK if exiting the stream mode of the X5-RX plug in board and writing the data to the hard disk is successfully ER|<*name* >|<*error message*> if an error has occurred

2.4.6 Sync

This command forwards the parameter part to the timing unit as a command via UDP datagram. This functionality is used in error recovery scenarios and during debugging the testbed.

command:	**SYNC**
parameter:	command which is forwarded to the timing unit
answer:	OK|<*name* >|<*received command*> if forwarding is done successfully ER|<*name* >|<*error message*> if an error has occurred

2.4.7 Terminate

As mentioned at the beginning of this section, this command is used to terminate the daemon in a proper way.

command:	**TERM**
answer:	OK|<*name* >|is terminating! if the command is received successfully

2.5 Internal Information Flow

After the main thread has received a measurement command, the information (file path, name and offset position) about a single measurement is stored in a small structure. This structure is passed to the ApplicationIO class. A trigger command to the timing unit is sent immediately. After the X5-RX plug in board gets a trigger pulse, data acquisition starts. During the acquisition, loading data to the shared memory is started. If a particular amount of data (further details in next section) is transfered, the HandleDataAvailable method is called. This method should have a small execution time, otherwise the X5-RX plug in board's buffer gets an overflow and therefore only the data is copied into the computer's RAM. After finishing one single measurement, the writing thread is called to write the buffered data asynchronously to the hard disk drives. An overview of this procedure is shown in Figure 2.4. Further explanations to classes and threads are given in the subsections below.

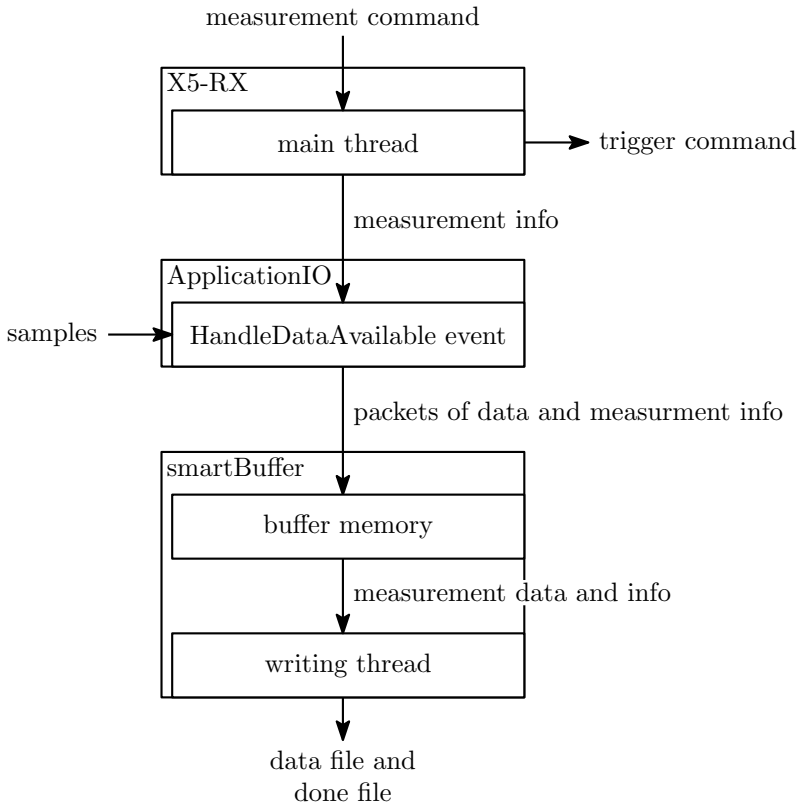

Figure 2.4: Overview of daemons internal information flow.

2.6 Internal Data Handling

If the plug in board receives a trigger pulse on its trigger input, data acquisition is started provided the board was initialized correctly. After internal buffering, the samples are loaded in packets into the shard memory. If one packet is completely loaded into the random access memory of the computer the background thread of the Malibu framework signals the availability of data. This is realized by calling the event handle method HandleDataAvailable. This method is modified for our needs. A flow chart is shown in Figure 2.5. First, a packet count is increased. This packet count is used to figure out the borders between the single measurements. After that, the packet will be extracted to get the raw byte data of the samples. These bytes and the measurement info which was previously received via an UDP datagram are added to a buffer memory by calling the add method of the buffer class. This class will be explained below in more detail. If the amount of packets reaches the number of packets

for one single measurement, the packet count is reset and the measurement information is updated. Furthermore, blocking of the main thread when receiving new measurement commands is disabled. This avoids that more than one measurement command is stored in the waiting command queue. This ensures a simple solution in case of errors, for example, if packet loss occurs.

For intelligently buffering the streamed data and writing the buffered data to hard disk, the class smartBuffer was created. For that, a sector aligned memory is allocated to increase the writing speed of the hard disks. Furthermore, each block that is written to the hard disk corresponds to one single measurement. To reach high writing speed, the size in bytes of one single measurement has to be exactly a multiple of the sector size of the used hard disk. Common sector sizes today are 512 bytes and 4096 bytes.

A newly initialized buffer is shown in Figure 2.6. The pointer that marks the current position for copying new measurement data stays at the beginning. While receiving packets for one single measurement, this pointer stays always unchanged at the first memory position of one single measurement memory frame. The offset is calculated by multiplying the packet count by the packet size in bytes. This is shown in the middle part of Figure 2.6. If the buffer's end is reached, and a whole frame would not fit without writing in non-allocated memory, the pointer which marks the current position is set to the beginning of the allocated memory. The avoidance of buffer overflows has to be done outside the buffer class and the daemon. The smartBuffer class only checks if the buffer is able to store the next measurement to avoid obscure errors while using the daemon not correctly.

As mentioned above, writing the sampled data to hard disk is implemented asynchronously in a separate thread. This thread is initialized while creating the smartBuffer class. The corresponding flow chart can be seen in Figure 2.7. This thread consists of one outer and one inner loop. The outer one will only be exited if the flag for signaling that writing should be finished is set. After entering this loop, the thread is blocked as soon as data for writing is available. The starting of the writing is signaled by the add method of the smartBuffer class, after copying the last packet of a single measurement into the buffer memory. After signaling, the inner loop will be entered. If the file from the previous measurement differs from the current one, the opened file will be closed and the current one will be opened. If no file was opened before, the step of closing will be avoided. If the file does not exist, it will be created.

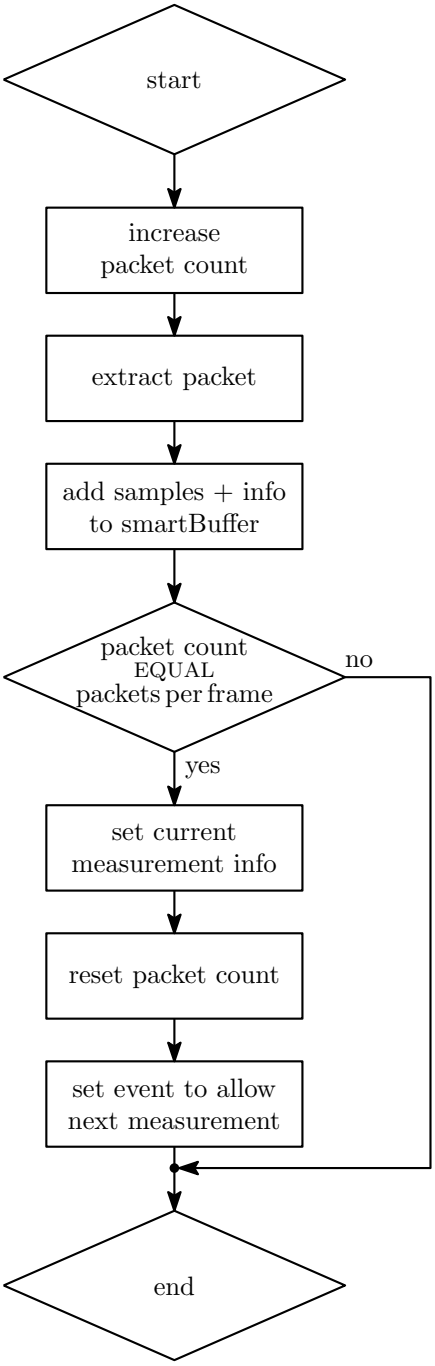

Figure 2.5: Flow chart of HandleDataAvailable method.

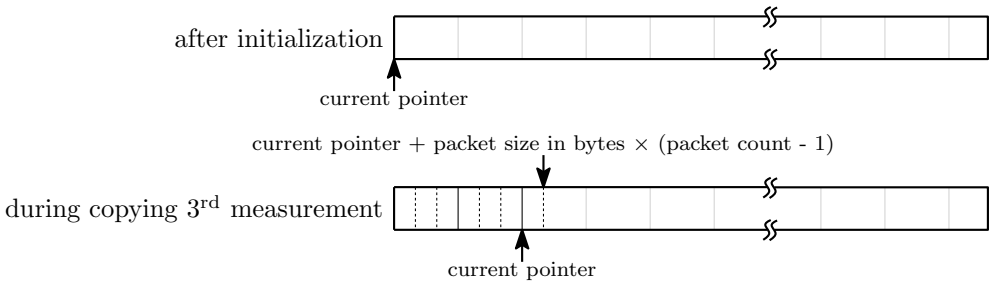

Figure 2.6: Buffer structure in RAM.

After that, if necessary, the file pointer will be set to the desired position. Finally, in the inner loop the writing command is performed. This loop runs as long as data is available in the buffer, except if a reset command is received. In this case writing is aborted which is signaled by a flag. If there is no data to write anymore, the event will be reset, the process leaves the inner loop and goes in blocking mode again due to the outer loop.

2.7 Interaction with Testbed Environment

As mentioned in the introduction, a Matlab script runs on the testbed's receiver computer which manages the whole measurement procedure. In this thesis only the part which is responsible for controlling the triggering and sampling is treated. The overall script for using the testbed is explained in [4]. A detailed description of the functionality of the transmitter daemon is given in [5].

The communication between the daemons and testbed environment is based on UDP datagrams. The main advantage of using UDP in this case is that it has small overhead. Furthermore, this way of communication is simple to handle; for example, there is not need for establishing a connection as in TCP[5]. Only binding the port at the receiving side is necessary to get datagrams from every host which has access to the network. Due to these facts, using UDP datagrams to achieve message passing between different programs locally on one host, too, is convenient. Additionally, this allows for more flexibility for future tasks, because the communication part of the programs and scripts does not need changes if they are separated on different computers or merged onto one computer.

[5] TCP ... Transmission Control Protocol

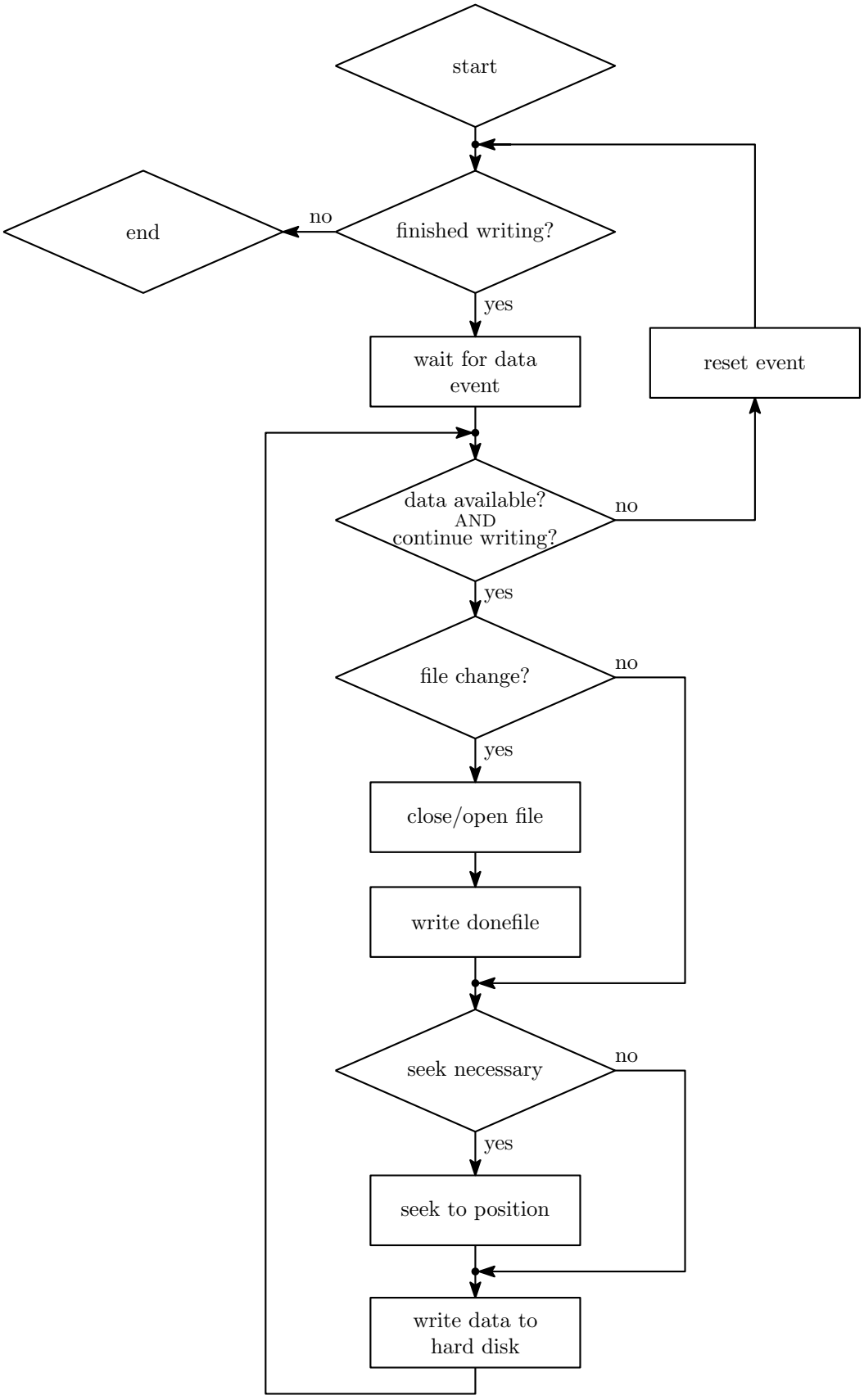

Figure 2.7: Flow chart of the writing thread.

The exchange of more static information is based on files. The advantage of this communication technique is that the usual operating system network file services can be used, and if the network connection breaks, the information is not lost; when using the network file services after reconnecting to the network share, the file is still there because the existence of the file is independent from a working network connection. In comparison, an UDP datagram is lost if a network failure occurs. Therefore, if UDP datagrams were to used for such information exchange, more complex algorithms would have to be used to compensate network failures. However, the file based way has a lot of overhead which increases the delay between the command is issued and the action is executed.

As a result, the first way of communication that has been mentioned above, is used to send commands and their responses between the testbed's components. This allows fast interaction between them. The second introduced way of information passing is used at the testbed receiver to signalize that a specific file is completely written to the hard disk (see section 2.6).

The receiver daemon is designed to run continuously at the testbed receiver computer. Normally, before doing measurements, the daemon is set to a well-known state by sending a reset command which is described in Section 2.4.1. Furthermore, the amounts of channels, frame size and other settings are configured when executing this command. If the reset process is done successfully, the daemon sends back an UDP datagram which includes an OK. Additionally, before starting a measurement, the daemon version can be queried by using the ping command to verify if the correct version is running to avoid unexpected behavior.

In Figure 2.8, a simple measurement scenario is shown. Here the initialization has already been done and so the first command which is sent via UDP is a measurement command. For simplicity in this and further sections, the measurement information, file path, name, and position is shortened to a single number which will only be increased to distinguish between the different measurements. The receiver daemon's main thread (see Figure 2.3) receives the command. After identifying the command and extracting the parameters, the measurement information is passed to the ApplicationIO class. As a next step, a trigger command is sent as UDP datagram to the timing unit. After that, all components of the receiver's daemon are checking if an error has occurred. If all steps are executed successfully and every part is working faultlessly an OK is sent back to the Matlab script. As a consequence of this checking, an

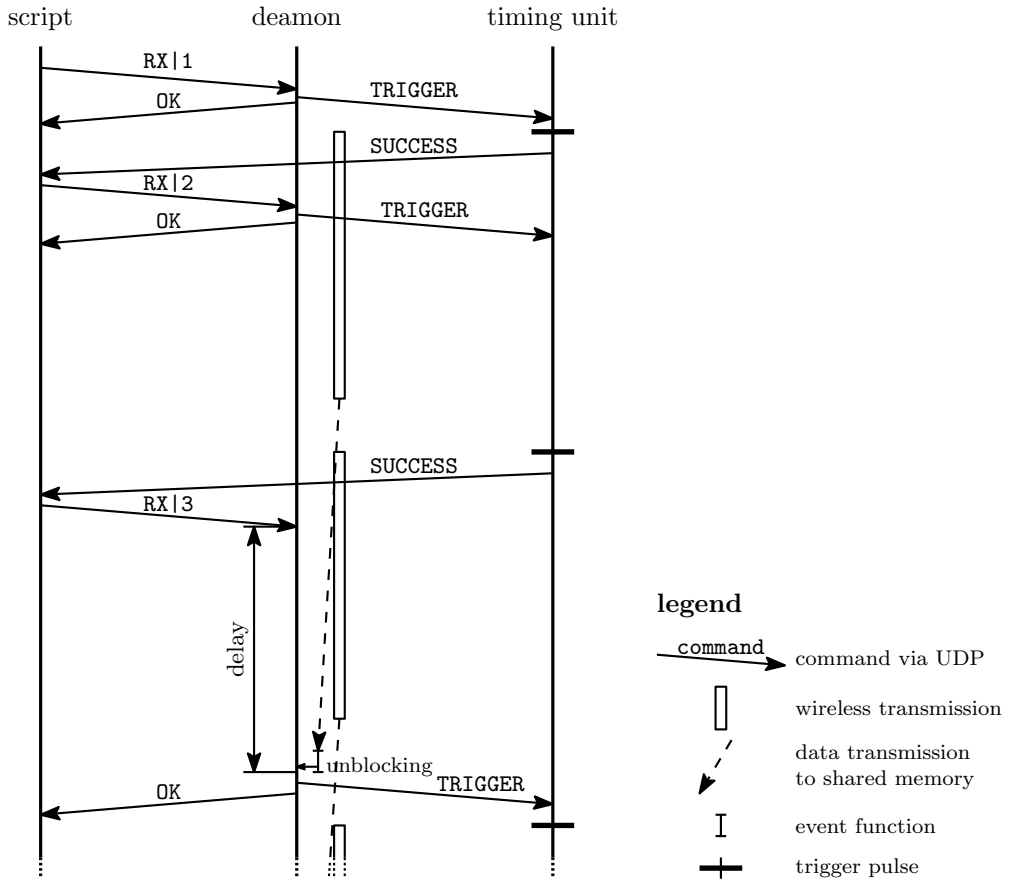

Figure 2.8: Simple measurement scenario.

OK signals additionally that the daemon worked without an error up to that time point of checking. Shortly after, the X5-RX plug in board receives a trigger pulse on its trigger input and data acquisitions is started, i.e. the samples are moved into the card's internal buffer to build packages. After finishing sampling of one package, the Malibu framework loads the samples over the PCI Express bus into the shared memory. For simplicity, in Figure 2.8, the measurement consists only of one package. In general, every measurement can be divided into an arbitrary number of packages, as long as the multiple of the package size fits exactly the measurement frame. A further aspect is that, if the package are chosen to small, the time overhead (signaling the event) is too large, so the recorded samples can not be moved over the bus fast enough and the X5-RX plug in board's buffer has an overflow. A rule of thumb is to set the packet size approximately to half of the shared memory size which is currently 32 Mbyte, so packet sizes should be set to approximately 16 Mbyte. While the packets are transfered from the X5-RX plug in board to the computer's RAM, the receiver's Matlab script sends the next measurement command. The

transmitter daemon part in such a measurement cycle is pointed out later. The trigger pulse must not be sent before sampling is finished because the X5-RX module will ignore this pulse, and one trigger pulse is lost which yields to an undefined state. Because of performance reasons, a detection of this wrong usage is not implemented in the receiver's daemon. The exact point in time for triggering in Figure 2.8 is specified through the processing within the transmitter daemon; further details are mentioned below. The second measurement command is handled similarly to the first. After receiving the success message for the trigger pulse corresponding to the second measurement, the Matlab script sends the third measurement command if further measurements are in the queue. This is done while the Malibu framework is still transferring the samples from the X5-RX plug in board to the computer's memory. Furthermore, the data acquisition corresponding to the second measurement is also still in progress. This scenario is very likely if the decimation factor is disabled and, as a consequence, the amount of data which is generated during sampling the analog signal is huge. The data transfer rate from the X5-RX plug in board over the PCI Express bus to the computers RAM is too little. In that case, the receiver daemon will delay the sending of the trigger command to the timing unit and also the confirmation via the OK command.

In contrast, Figure 2.9 shows the same scenario in a case where the transfer from the X5-RX plug in board to shared memory is fast enough. For example, the decimation factor is set to two or the samples of only one channel are of interest. In comparison, the added delay which is marked in Figure 2.8 is avoided, and the receiver daemon does not need to slow down the procedure.

Finally, in Figure 2.10 the information flow with, for example, two transmitter daemons is presented. The initialization procedure is avoided to save space. The transmitter daemon has to load the pattern which will be next sent from the computer's RAM completely into the cards memory, because only in that way the conversion of the digital samples into an analog signal is guaranteed in real time. For more information about the transmitter daemon see [5].

The first step (see Figure 2.10) after initialization is that the Matlab script sends the commands to tell the transmitter daemons, which samples have to be converted from digital to analog when receiving the next trigger pulse. After that, the daemons are loading the samples into the plug in board's memory. During this task, the script sends the measurement command immediately to

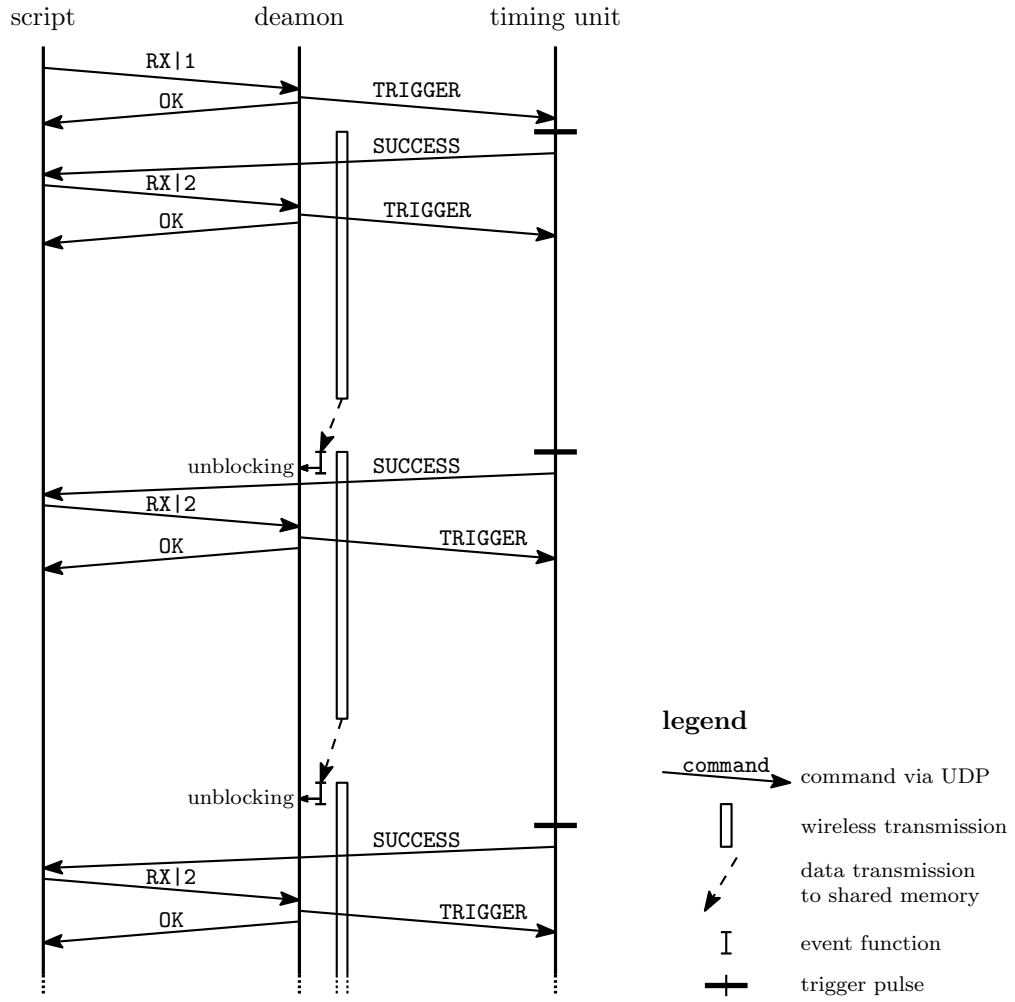

Figure 2.9: Simple measurement scenario without added delays.

the receiver daemon. This daemon sends the trigger command to a timing unit and a confirmation message back. After finishing loading, each transmitter daemon sends also a trigger command to the dedicated timing unit; and afterwards, a datagram with an OK confirmation is sent back to the receiver's Matlab script. Shortly after, a trigger pulse is provided by each timing unit on the card's trigger input. From that point on, the transmitter and receiver plug in boards convert the signal from digital to analog or vice versa. After another short period of time, if triggering was successful, each timing unit sends to each corresponding computer an UDP datagram. In this case only, the UDP datagram which is sent to the testbed receiver computer is evaluated by the Matlab script. After receiving the success message, the script sends the information for the next measurement to the daemons again but, in contrast to the first measurement, while the testbed transmitters and re-

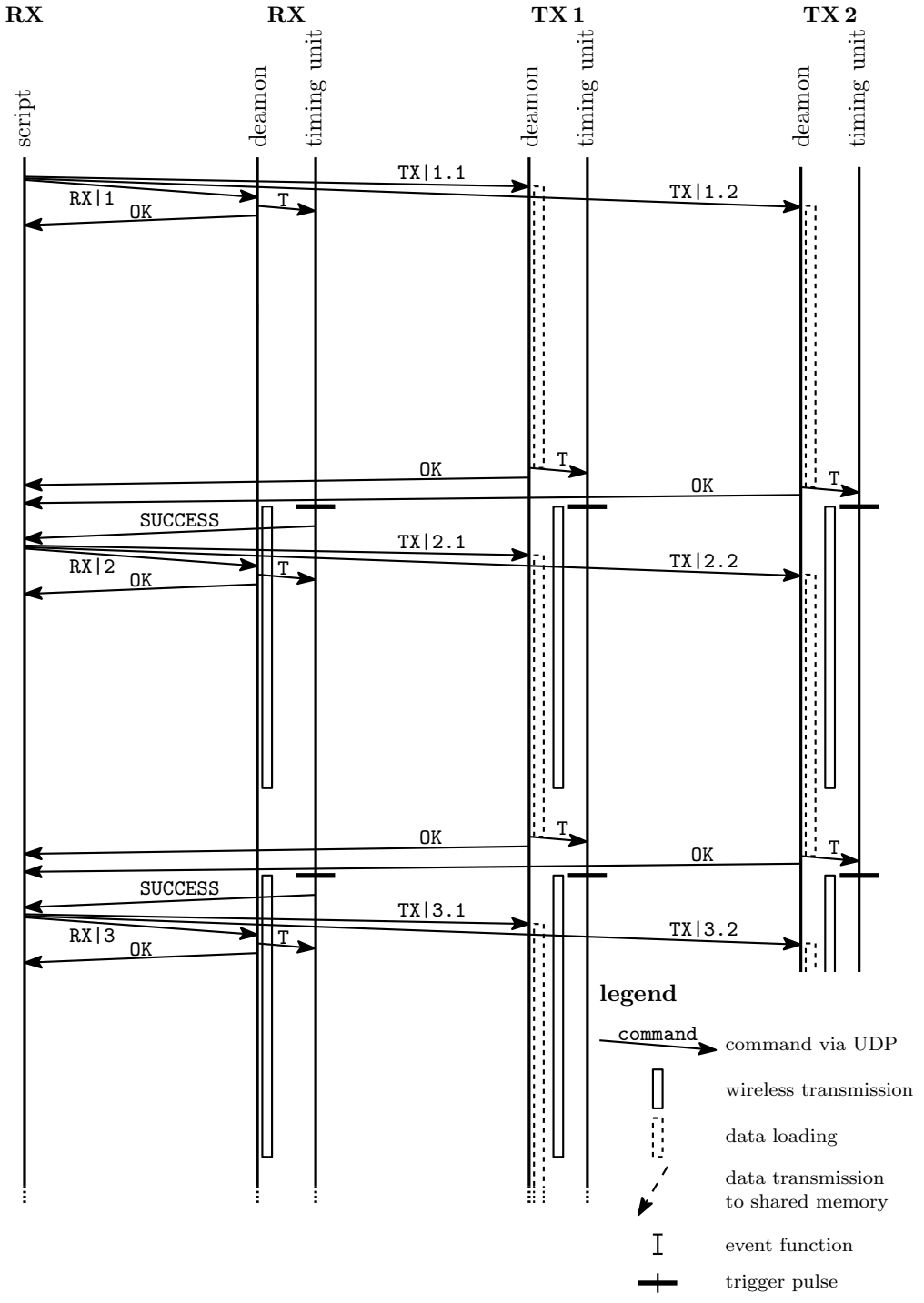

Figure 2.10: Information flow in measurement scenario with one receiver and two transmitters.

ceiver are transmitting. Further procedure is similar to the first transmission. Normally, the transmitter daemon's loading time is longer then the playing time. If this is not the case, for example, upsampling is implemented in the transmitters plug in board, the daemon waits until playing is finished. As a consequence, the transmitter daemon does not send a trigger command to the timing unit before playing is finished. Furthermore, the trigger success message will never arrive at the testbed receiver computer before the sampling is finished.

In this paragraph, the flow chart of the Matlab script which is shown in Figure 2.11 is introduced. Before the sketched starting point in Figure 2.11 is reached, the testbed is initialized correctly, which includes reseting the transmitter and receiver daemon. After this, first the `TX` and `RX` commands are sent to the daemons. After that, the OK messages which are sent back to the script from the daemons are received. However, it is important to notice that the responses can be received in arbitrary order. In this case, different packet transmission times and loading speeds at the transmitters are considered. Then the script receives the success message from the timing unit in errorless cases. As a consequence, a counter is increased to send the next measurement commands which are stored in an array. At the end of a such a cycle, the receiver daemon's fill level which is sent piggyback with the OK message is compared to the critical value. If the value is smaller, a new run can be started. Otherwise, if the fill level is greater the threshold the script pauses for while, queries the fill level again, and compares that refreshed value again. This is executed as long as the fill level is greater than the critical value. This behavior without querying the fill level corresponds exactly to the information flow diagram which is shown in Figure 2.10.

2.8 Network Transmission Errors

The usage of the UDP protocol based on the Internet Protocol layer may lead to that messages such as commands are not arrive, because there is no automatic retransmission algorithm built in. As a consequence, the receivers Matlab script, transmitter, and receiver daemon are designed to handle that commands could be get lost. A description of the different steps which will be executed in case of errors is given below.

As opposed to packet loss, packets can be duplicated in general. This is avoided

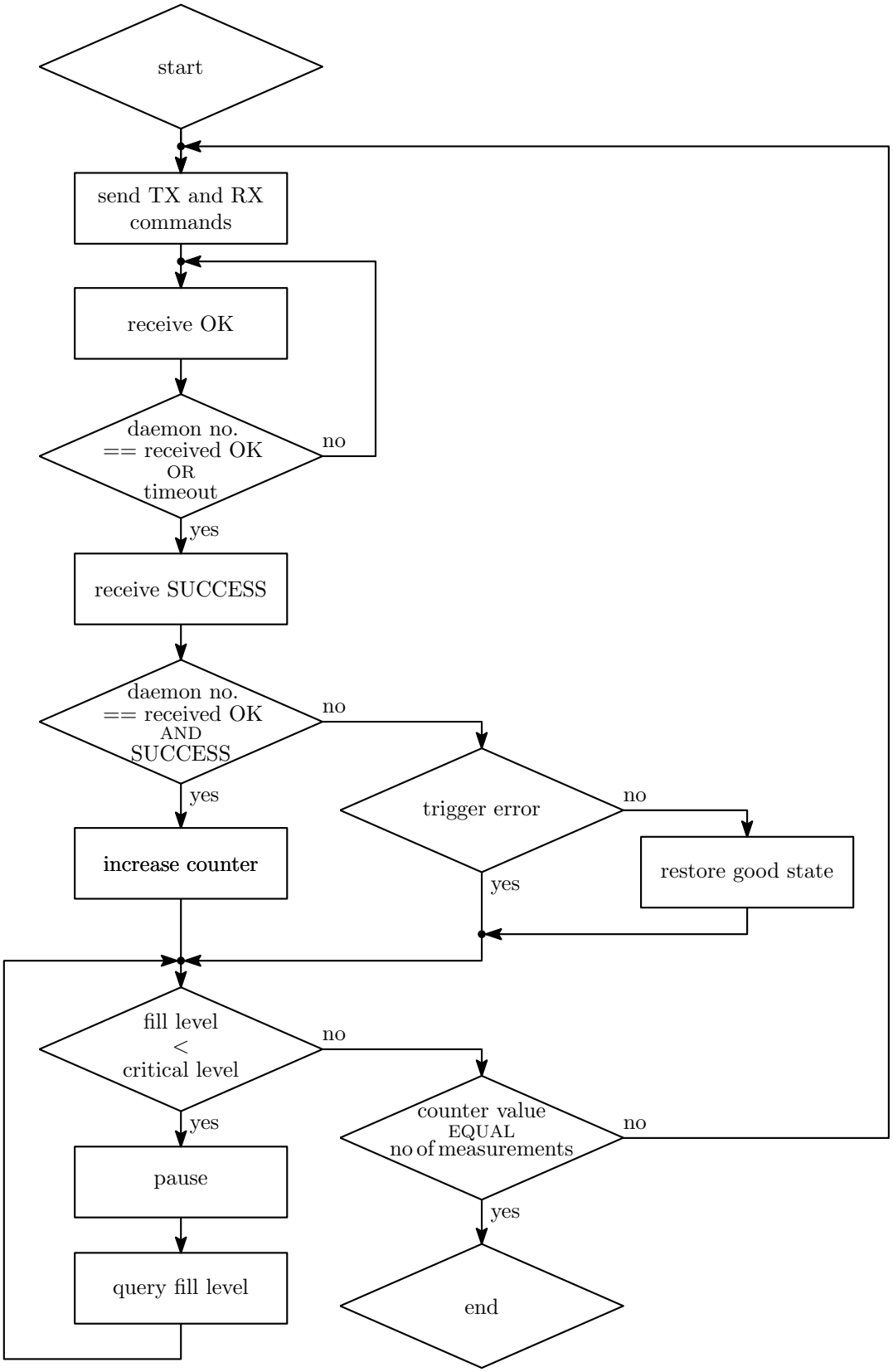

Figure 2.11: Flow chart of a Matlab script which does the coordination during the measurement process.

through the used network structure to connect the different components of the whole testbed. A detailed description of the network architecture can be found in [4]. Summarized, the network which is used to transmit the commands is dedicated only for this type of transmission and the packets are not routed between different subnetworks.

A further uncertainty which is introduced via UDP over Internet Protocol is that the transmission time can vary. Additionally, this may lead to reordered reception of packages which have been sent in a short period of time. If reordering of packages has a bad influence, the message exchange is done in a ping-pong manner. In other words, if the acknowledgment is not received, no further step will be done and, thereby, these problems are avoided; for example, see Figure 2.10. Here, the next measurement commands are sent to the transmitter and receiver daemon after receiving the success message which comes from the timing unit.

2.8.1 Trigger Error

A direct effect of packet delays is that they are sometimes delayed so much that a successful triggering is not possible. This is detected by the timing units and, instead of the success message, a trigger error message is sent back to the Matlab script. In contrast to a packet loss, when a trigger error occurs each plug in board has received a trigger pulse on its input. This is caused by the design of the timing units. Only triggering time instant is not accurate and so no restoring to the well-known state as explained in the section below hast to be done; repeating the single measurement is sufficient. In Figure 2.11 the check, if triggering was successful, fails. The check if a trigger error message was received, passes. The counter is not increased and so in the next cycle the measurement will be repeated.

2.8.2 Restore after Packet Loss

First, the different types of packet loss and how they are handled are introduced. The loss of one or more measurement commands to the transmitter and the receiver is noticed, because the confirmation will not be sent back. Otherwise, if a datagram including the confirmation is lost this has the same effect. At the Matlab script side, distinguishing between those two different cases is difficult. On the one side, when a measurement command to one

transmitter is dropped the corresponding daemon does nothing. On the other side, if the OK message gets lost the transmitter daemon loads data into the plug in board's memory. Therefore after this kind of packet loss, restoring a well-known state is necessary.

Furthermore, one or more trigger commands to the respective timing unit could be dropped. The result is that no timing unit will send the success message to the corresponding computer. Another place in the chain of datagram transmission where packet loss could happen is during the trigger process. This leads to that no success messages will be sent from each timing unit to the corresponding computer. Another possible datagram drop could effect the transmission of the trigger success message to the Matlab script. These packet losses mentioned are not distinguishable at the Matlab script side without doing additional package transmission.

The fact that losing a measurement command also causes that one of the trigger commands will not be received at a timing unit should not remain unmentioned. As a consequence, the trigger success message will not be sent.

The probability of multiple packet losses during one single measurement is negligibly small, but greater than zero, and so the algorithm also has to be aware of them. Additionally, if the maximum credible accident for network transmission, such as a network cable is unplugged, or a switch overheats and fails, occur, the probability of multiple packet loss is very high.

Because of the fact that these possible packet losses are unavoidable, the Matlab script has to handle the different kinds of their occurrence. The fact that all packet losses can be detected has been shown sufficiently. How to restore a known state and additionally, not decrease the performance in errorless working process will be shown next.

Looking at Figure 2.11, a successful measurement, a confirmation is needed from every daemon and a trigger success message has to be received. If one of the daemons confirmations does not arrive at the receivers script, but a trigger success message, the single measurement is only repeated, because an error message, which a daemon sent back after an internal error occurred, could be dropped. As one can see in the flow chart of the Matlab script in the middle of Figure 2.11, the counter will not be increased and therefore the measurement will be repeated. Before, the daemons and timing units have to be set into a well-known state. If a trigger error message is received, the timing units satisfy

by design that a trigger pulse has already been generated. Additionally, the daemons received all measurement commands, because otherwise no trigger error message could be received. In other words, if a trigger success or trigger error package is received by the Matlab script, a trigger pulse is given to each transmitter and receiver plug in board's trigger input. A missing OK message can be ignored in this case. If a trigger error occurs, a transmission is ensured and, as a consequence, the timing units and daemons are in a well-known state. If neither a trigger success nor a trigger error message arrive, it is possible that no trigger pulse occurred. In this case, the box "restore good state" will be executed.

A detailed view of this box is shown in Figure 2.12. If the occurrence of a trigger pulse is note ensured, the timing units have to be set into a well-known state by sending them an abort command. If the timing units are able to restore a well-known state they confirm the abort command with the message "abort done". Otherwise, they do not send any message back. If this is the case, the Matlab script sends the abort command again and again after waiting a short period of time between the tries. That is, they wait for the user to solve the network connection issue that has to be present in such a case. After successfully aborting, the next step is to trigger "manually" by sending the trigger command via daemon's `SYNC` command to the timing units, except for the command to the receiver's timing unit, which can be sent directly. Sending is implemented directly because the receiver daemon could be in a blocking state as described above, which would unnecessarily delay the abort process. If the Matlab script receives a trigger success message after "manually" triggering, the daemons are in a well-known state and the single measurement which was affected by the failure can be repeated. If no success message could be received, the script repeats aborting and triggering as long as a well-known state is ensured.

Practical experience proved, that it sometimes happens that network cables are unplugged. For example, when someone stumbled across them. Overheated network devices or maintenance work on routers are reason for longer network failures. A possible solution for these problem would be aborting the whole measurement. However, the consequences can be undesired as measurements with a duration of up to more than one day were conducted in the past. A better solution is that, the loops in error cases are executed until the user stops them manually. Additionally, the time spans between the re-establishments are increases up to an upper bound of one minute. For example, if someone

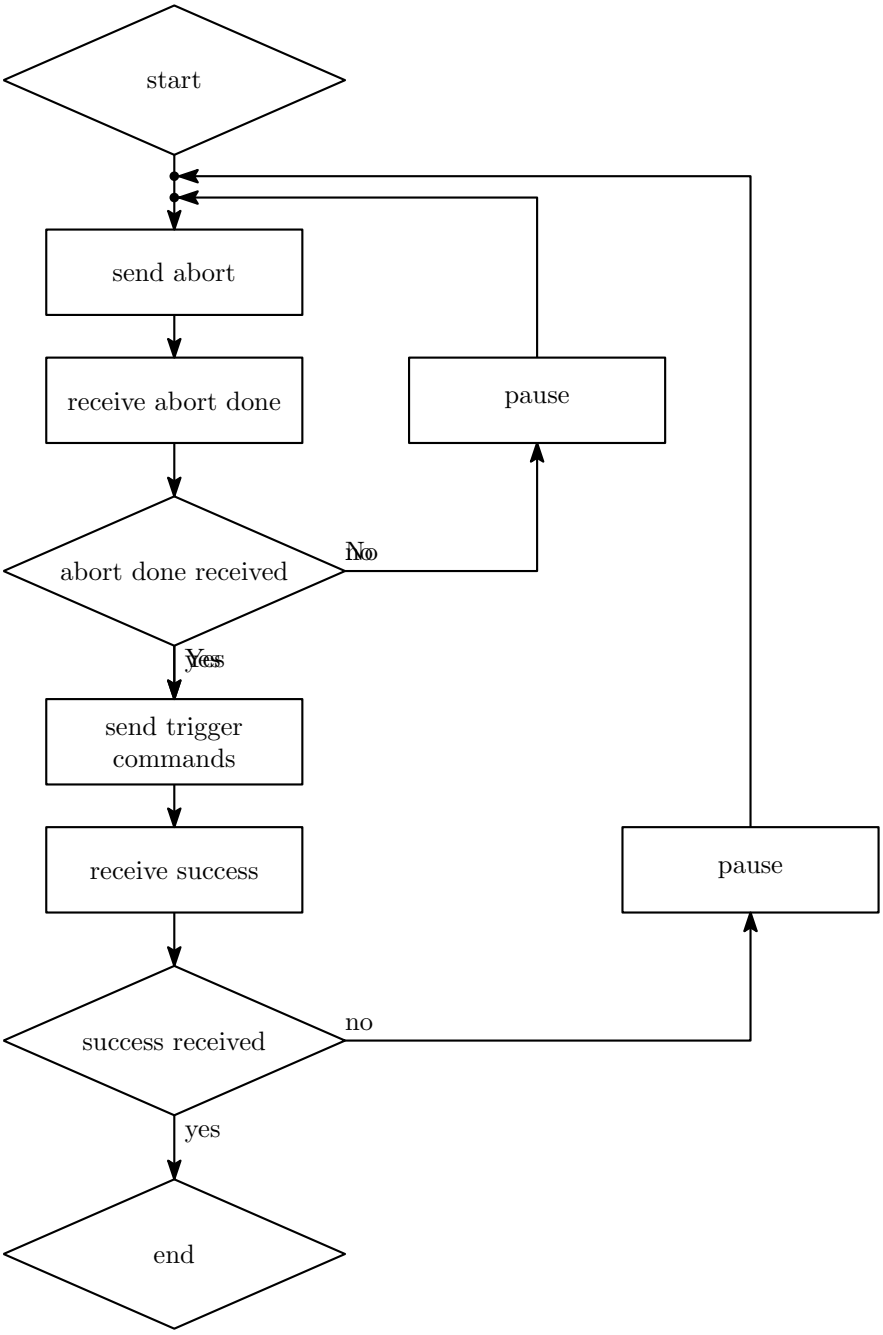

Figure 2.12: Flow chart of the Matlab script part which does restoring a well-known state.

accidentally unplugs a network cable, an error message will be shown. Then the reason can be searched and eliminated. After a successful restore process, the measurement is continued.

3 Hardware Characterization

3.1 ADC Performance

In the past, various ways to characterize analog to digital converter performance have been developed and standardized. An overview is given in [6]. A common method for dynamic ADC testing is to measure the SFDR[1]. According to [6], this performance measure is probably the most significant specification for ADCs used in communication applications. The SFDR is defined as the ratio between the RMS[2] signal amplitude and the RMS value of the peak spurious spectral content.

The measurement setup used to measure SFDR is shown in Figure 3.1. A HP 83712A Synthesized Signal Generator is used as sine-wave signal generator. According to its datasheet, the noise floor is smaller than -140 dBc/Hz and the 2nd harmonic is 65 dB below the carrier wave. Additionally, the power of the harmonics is decreased by using an external LC low-pass filter. The frequency response of this filter is shown in Figure 3.2. From this it follows that the 2nd and higher order harmonics will be suppressed with at least 78 dB. Using this setting, an input sine-wave source which has higher order harmonics 140 dB below the fundamental wave has been built. For better accuracy, the signal generator uses the same reference clock source as the oscillator which generates the sample clock. After analog to digital conversion, the signal is

[1] SFDR ... spurious-free dynamic range
[2] RMS ... root mean square

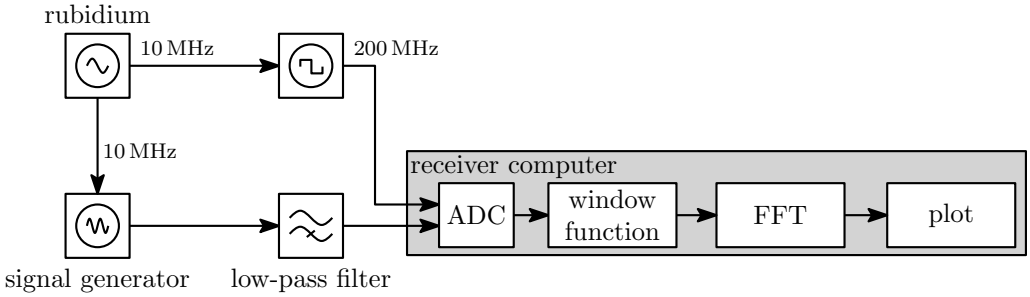

Figure 3.1: SFDR measurement setup.

Table 3.1: Harmonics of 70 MHz sine-wave sampling at 200 MHz

Order	1	2	3	4	5	6	7	8	9
Frequency in MHz	70	60	10	80	50	20	90	40	30

multiplied with a Hanning window before the discrete Fourier Transformation is performed using an FFT[3]. Because the window function reduces the signal values to zero at the boundaries, the overall signal power decreases. This factor is called coherent gain. According, to [7], for a typical $\alpha = 2$ Hanning window this factor is 0.5. Simply multiplying by the inverse compensates the influence on the signal power of the window function. Furthermore, a normalization of the signal is done in such a way that a full-scale sine-wave is defined as 0 dBFS. Finally, averaging is performed to smooth the noise floor. Such averaging does not change the level of the noise floor because only the randomness of the FFT bins is decreased (see [8]).

Figure 3.3 shows a plot generated using a sine-wave with a frequency of 70 MHz. An SFDR of approximately 75 dB is reached. The 2$^{\text{nd}}$ harmonic[4] at 60 MHz is the limiting spurious spectral content[5]. The position of further harmonics is given in Table 3.1. In general, the higher the harmonics order, the lower their power. Taking a closer look at the bins in frequency domain, some harmonics are higher as expected by theory. For example, the 8$^{\text{th}}$ harmonic at 30 MHz is approximately as high as the 2$^{\text{nd}}$ harmonic at 10 MHz.

A further examination is shown in Figure 3.4. The frequency of the sine-wave is chosen as 71.37 MHz. As a result, there are only three spurious pins above -85 dBFS. The second harmonic is found at 57.26 MHz and third har-

[3] FFT ... fast Fourier transform
[4] The 2$^{\text{nd}}$ harmonic is at 140 MHz (two times 70 MHz) and so, in the digital domain with a sampling frequency of 200 MHz, it shows at 60 MHz.
[5] The spectral components at 0 Hz are not of interest for communication applications.

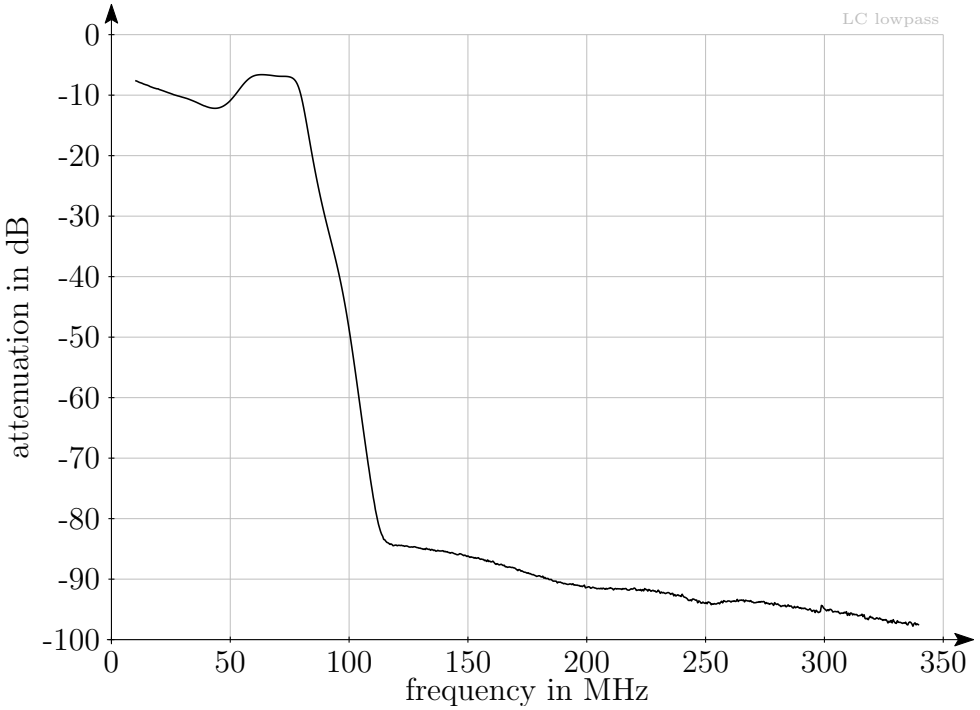

Figure 3.2: Frequency response of the used low-pass filter.

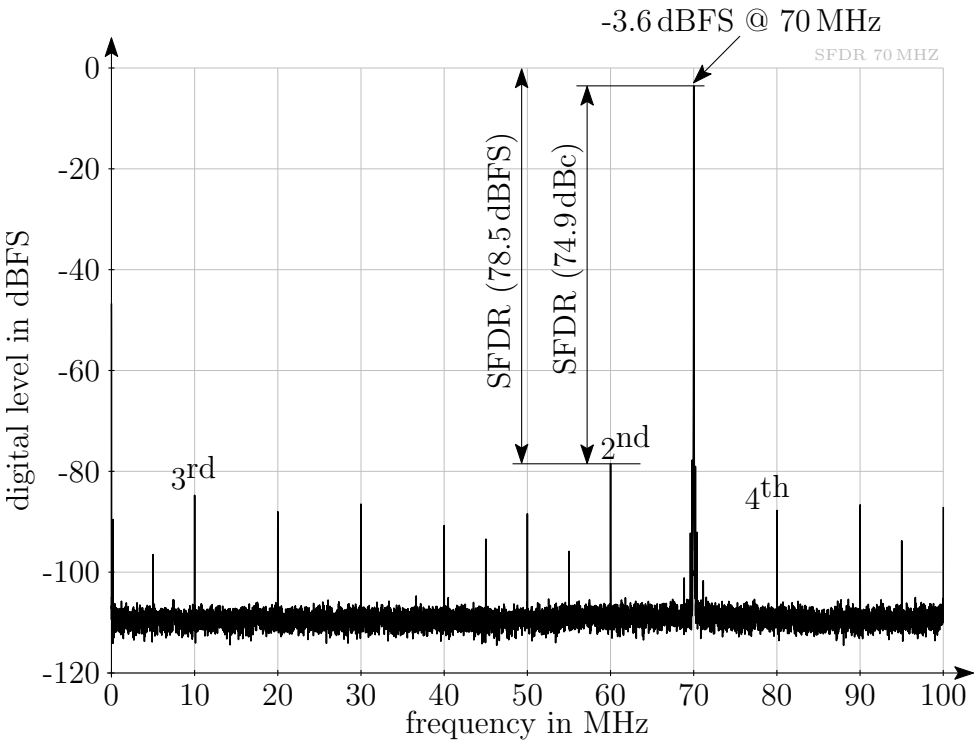

Figure 3.3: Output spectrum using a sine-wave with 70 MHz.

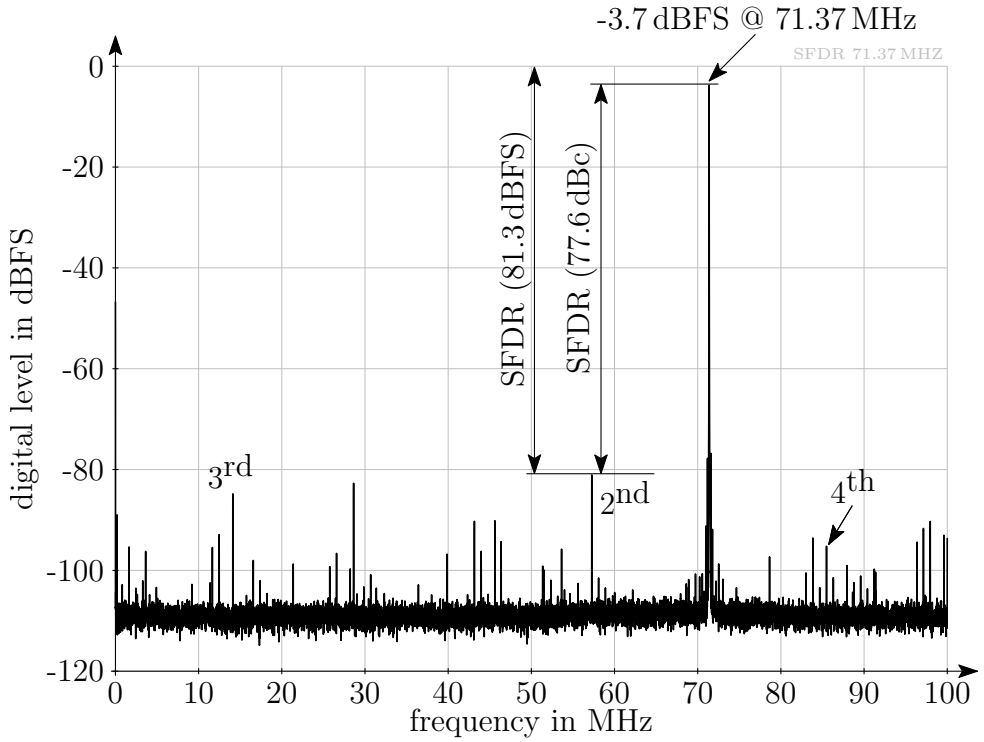

Figure 3.4: Output spectrum using a sine-wave of 71.37 MHz.

monic is found at 14.11 MHz. The bin at 28.63 MHz would be the 972^{th} harmonic which implies that height of the bin does not result from non-linearity.

Another distortion effect in radio frequency hardware design is intermodulation. For an intermodulation product, as the name implies, two signals are necessary. Coming back to Figure 3.4, it turns out that the intermodulation product between the fundamental wave and the 2^{nd} harmonic is 128.63 MHz and through sampling this frequency is transformed to 28.63 MHz. If the input sine-wave power is decreased, this bin decreases very fast which is also an indication for an intermodulation product.

Summarizing, the measured SFDR of approximately 75 dBc suggests that the X5-RX plug in board's performance is sufficient for current and future measurements.

3.2 Radio Frequency Front End

As mentioned at the beginning, the radio frequency front end is used to receive the signal at a center frequency at 2.503 GHz and correct it so that it can

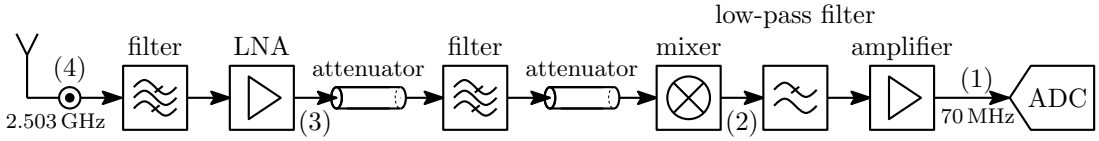

Figure 3.5: Radio frequency front end.

be sampled with high accuracy using the X5-RX plug in board. A detailed overview is given in Figure 3.5. The signal is received with an antenna which is mounted on an XYΦ-positioning table. After that, the signal is filtered with a bandpass filter, before it is amplified using an LNA[6] and bandpass filtered again. The first filter is to avoid that the LNA goes in saturation. The second one is to decrease the amplified noise at the mirror frequency so that the downmixing in the next step is not influenced. After that, the signal is downconverted. Here, the center frequency changes from 2.503 GHz to 70 MHz. Then the signal is lowpass filtered to eliminate the mirror frequencies which are introduced during downmixing. As a final step, before the signal is converted into the digital domain, an amplification is done to optimally drive the ADC which is applied on the X5-RX plug in board.

3.2.1 Receiver Noise

One point of interest is, what kind of noise the receiver hardware has. The noise level of the X5-RX plug in board can be seen in the section above in Figure 3.3 and Figure 3.4 while receiving a full scale sin-wave. This level is approximately -110 dBFS. In Figure 3.6 the noise floor of the receiver's radio frequency hardware with a match on the testbeds input is shown in the frequency domain. The above part of the graph shows the spectrum measured with a spectrum analyzer at point (2) in Figure 3.5. The lower part shows the noise sampled with the X5-RX plug in board. In order to smooth the noise floor, an average of 20 individual FFTs is taken. This does not affect the level of the noise floor. Only a reduction of the random variations in each frequency bin is achieved (see [8]). As one can see, the shape of the noise is unchanged during amplification and sampling. The observation of the increased noise floor around the center frequency of 70 MHz is caused by the radio frequency front end. To be more precise, the noise floor is increased by the LNA. Afterwards, the bandpass filtering produces the distinctive hill in the plot of the frequency

[6] LNA ... low-noise amplifier

spectrum in Figure 3.6.

The power spectral density of the region of interest is nearly flat. A small decrease of approximately 1.5 dB at the edges at 60 MHz and 80 MHz occurs. If only the range from 65 MHz to 75 MHz is considered, the variation is only 0.5 dB. However, in many applications for which the testbed receiver is used, the assumption that the power spectral density of the noise is white, is a correct choice.

In Figure 3.7 the empirical cumulative distribution function of the sampled noise is plotted after filtering. As filter, a simple bandpass is used which is designed with the Matlab filter toolbox. The parameters are $f_{\text{sample}} = 200\,\text{MHz}$, $A_{\text{stop}} = 90\,\text{dB}$, $A_{\text{pass ripple}} = 0.5\,\text{dB}$, $f_{\text{stop1}} = 60\,\text{MHz}$, $f_{\text{pass1}} = 62\,\text{MHz}$, $f_{\text{pass2}} = 78\,\text{MHz}$, $f_{\text{stop2}} = 80\,\text{MHz}$. As reference, a standard normal cumulative distribution function with the same mean and variance as the sampled noise is plotted in Figure 3.7. One can see that the two plotted cumulative distribution functions match in this figure. The practical significance, as mentioned in [9], to assume a Gaussian amplitude distribution of the noise is satisfied.

3.2.2 Characterization Using an LTE Signal

A future application area of the testbed is to investigate the different real world influences on LTE signals. For testing the performance of the testbed for this applications, the measurements described below are carried out.

Reference Signal

As reference, an LTE signal with 10 MHz bandwidth is used. As specified in the LTE standard, this is an OFDM[7] signal with FFT length of 1024. As a consequence, there are 1024 subcarriers, but only 600 of them are used. Various quadrature amplitude modulation schemes are available. The selection of the modulation scheme usually depends on the channel quality and is dynamically adapted. In our case a 64 QAM is chosen for all measurements during characterization the testbed receiver where the signal will not be transmitted over the air. In other words, in this case, the testbed receiver radio frequency front

[7] OFDM ... orthogonal frequency-division multiplexing is a multicarrier modulation scheme

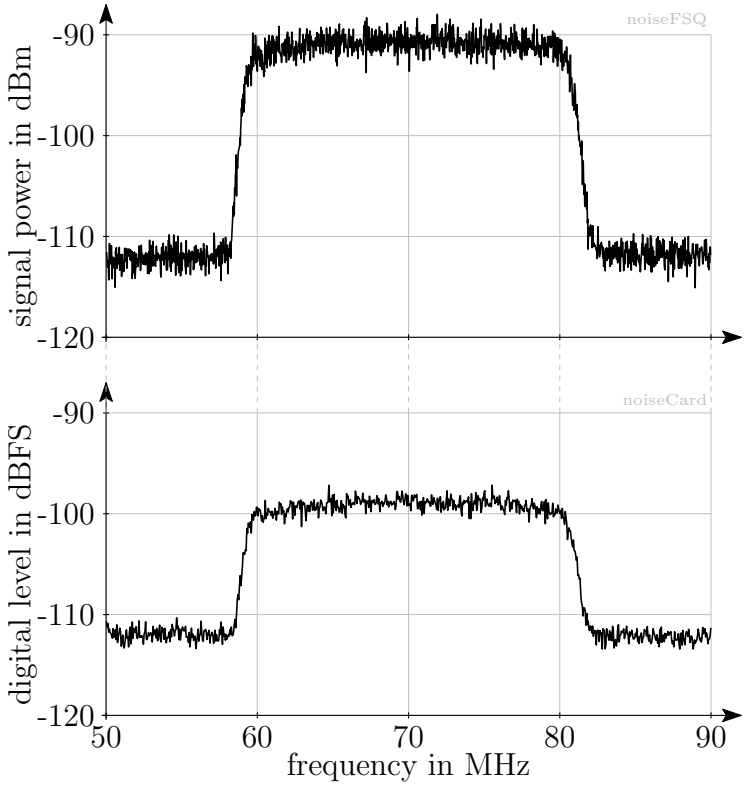

Figure 3.6: Noise floor after the downconverter measured with spectrum analyzer and with X5-RX plug in board.

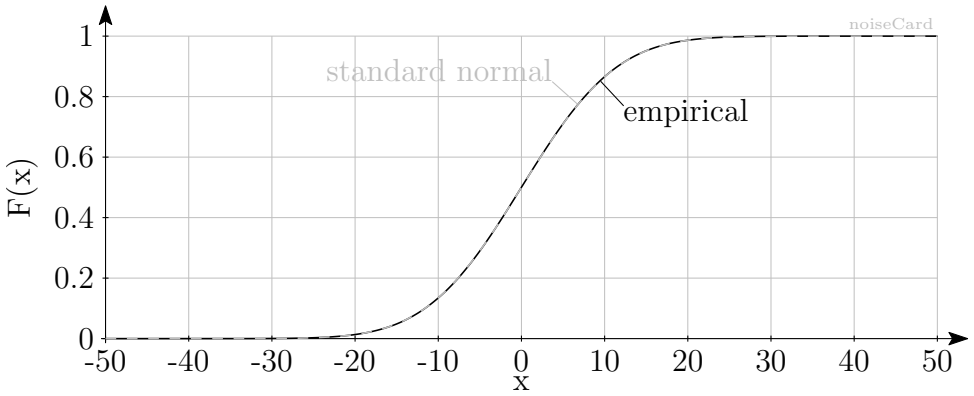

Figure 3.7: Cumulative distribution function of the sampled noise after filtering with a bandpass.

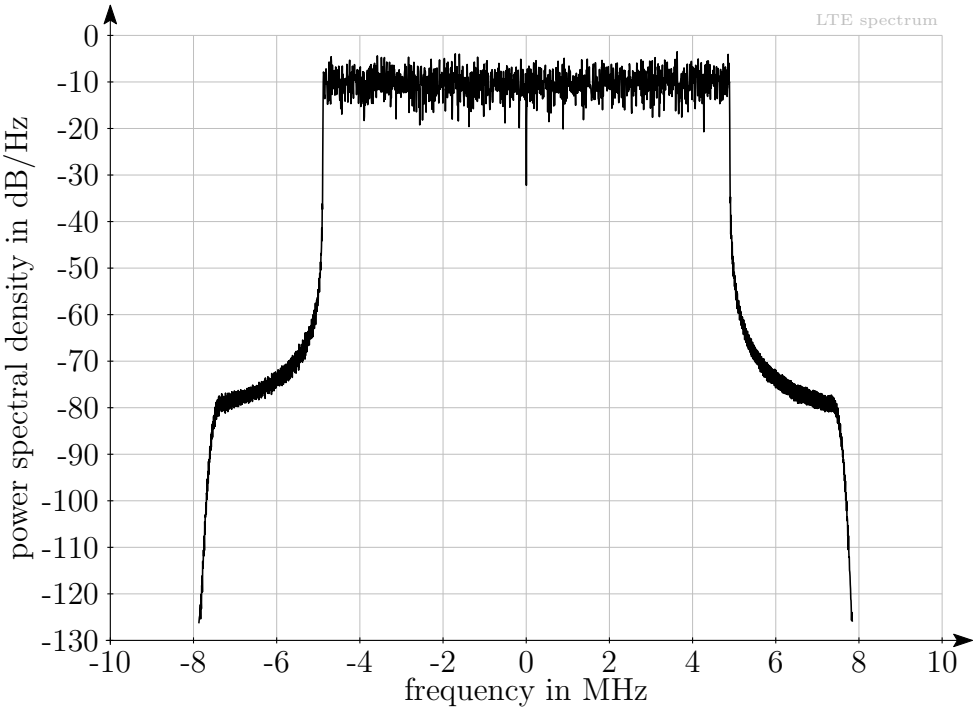

Figure 3.8: LTE signal spectrum in baseband position which is used for testing the receiver radio frequency front end.

end in addition with the ADC which is applied to the X5-RX plug in board is the channel. As a consequence, the modulation scheme for high CQI[8] values is chosen. Nevertheless, if the input signal power decreases, also the 64 QAM signal is used to get comparable results. Furthermore, the 64 QAM signal has the highest crest factor, which is useful to stress the hardware with respect to nonlinearity. The signal is generated using the LTE simulator [10]. The spectrum is shown in Figure 3.8. The signal lies in baseband position, because the used vector signal generator internally upconverts the signal to 70 MHz or respectively 2.503 MHz. The peak at zero frequency is caused because this subcarrier is not used.

The test setup used is shown in Figure 3.9. At first, the test signal is generated in baseband position at the control computer. As signal generator the Rohde & Schwarz SMU200A Vector Signal Generator is used. For compatibility reasons, the signal is upsampled to 100 MSamples/s and stored temporally on the control computer's hard disk. In the next step, the file which contains the signal samples is transmitted over the local area network to the signal

[8] CQI ... Channel Quality Indicator is used in communication system as LTE to exchange channel quality information between mobile and base station.

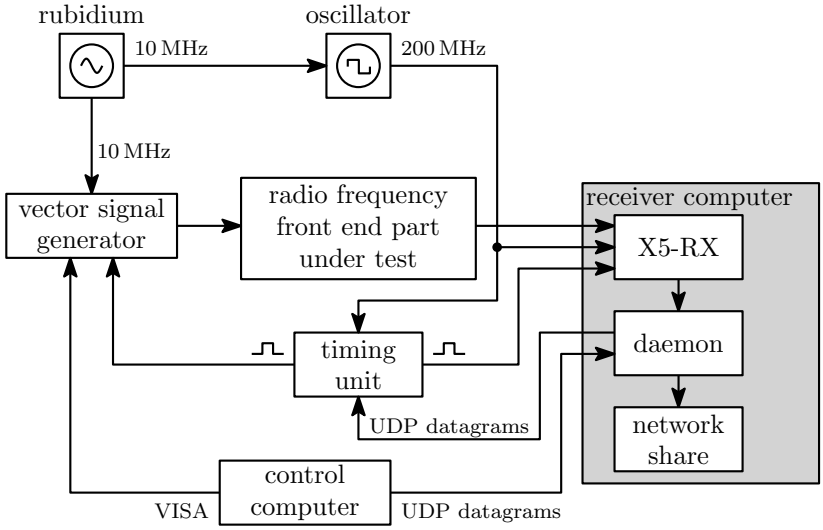

Figure 3.9: Test setup for measuring the performance of the testbed receiver's radio frequency front end.

generator. After that, the signal generator is controlled as VISA[9] object in Matlab.

A single measurement is carried out as follows: first, the output level of the signal generator is specified and its trigger input is set to sensitive. Second, a measurement command is sent to the receiver daemon. Third, the daemon sends a trigger command to the timing unit. Shortly after, the signal generator and the X5-RX plug in board receive a trigger pulse on their trigger input and the transmission of the signal starts. After that, the received signal is sampled and stored on the hard disk of the testbed receiver computer. The received signal can be evaluated on every computer which is able to run the mentioned LTE simulator and has access to the folder where the file that contains the signal is stored. In practice for evaluating the received signal several computers in the laboratory as well as the receiver's computer and control computer are used.

The points where the signal, which is generated by the vector signal generator, is supplied are marked in the Figure 3.5 with numbers in brackets. Normally, if not specifically mentioned, channel one is presented in this thesis. The other channels are built with similar components and, therefore, the results from channel one are also applicable to them. Furthermore, crosstalk between the

[9] VISA ... Virtual Instrument Software Architecture is an API (application programming interface) used to communicate with measurement devices.

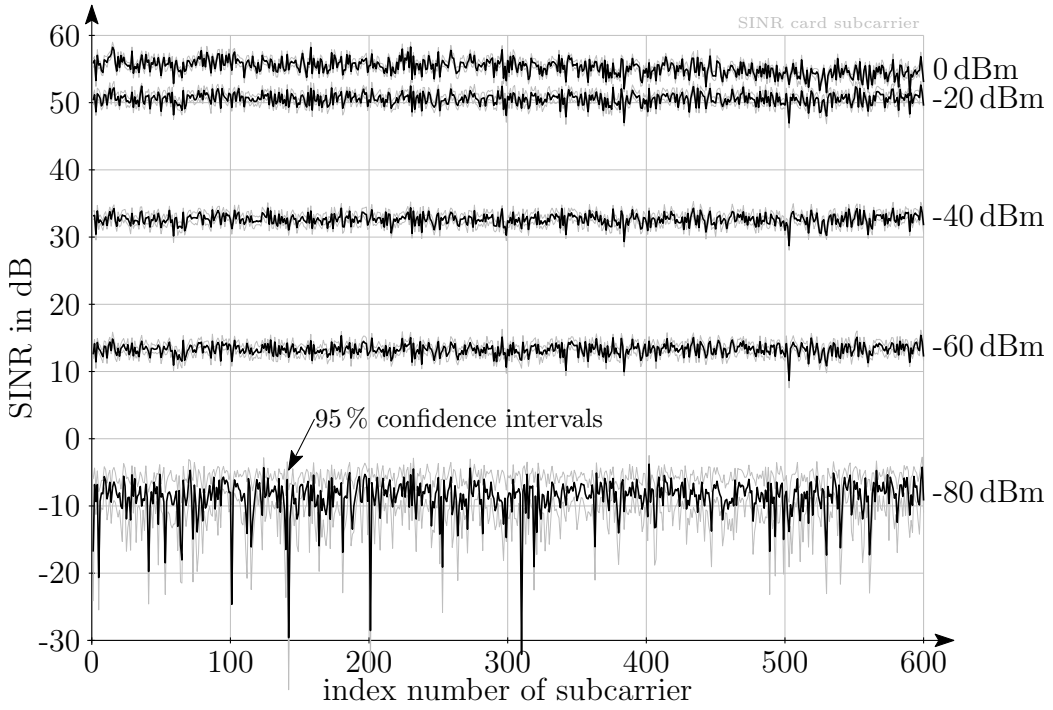

Figure 3.10: SINR over OFDM subcarrier with various input power levels. A Signal generator is directly connected to the X5-RX plug in board in board.

four channels is investigated but more than the fact that this effect is negligible could not be determined.

One way of measuring the performance is to evaluate the SINR of the sampled signal over the input signal average power.

X5-RX Plug in Board

The calculated results for each subcarrier are shown in Figure 3.10. The signal was supplied directly at point (1) into the X5-RX plug in board. One can see that for higher input power the SINR does not increase with the same rate. This will be investigated in more detail in further measurements. Another observation is that the SINR decreases slightly for higher frequencies at high power levels. Furthermore, one can see that all subcarriers have approximately the same SINR level. In other words, there are not distortions at single frequencies.

In Figure 3.11, additionally, the average over all subcarrier is taken. On high input power levels the SINR should increase because the range of the ADC would be used more efficiently. The reason for saturation is that the Rohde &

Schwarz SMU 200A Vector Signal Generators has an EVM[10] of 0.2 % according to the data sheet [11] which corresponds to an EVM of -54 dB. The EVM is defined in [12] as

$$\mathrm{EVM_{RMS}} = \sqrt{\frac{\frac{1}{N}\sum_{r=1}^{N}|S_{\mathrm{ideal},r} - S_{\mathrm{meas},r}|^2}{\frac{1}{N}\sum_{r=1}^{N}|S_{\mathrm{ideal},r}|^2}}. \tag{3.1}$$

where $S_{meas,r}$ is the normalized measured r^{th} symbol and $S_{ideal,r}$ is the normalized constellation point for the r^{th} symbol in a symbol stream of length N. The SINR is calculated in [10]

$$\mathrm{SINR} = \frac{\frac{1}{N}\sum_{r=1}^{N}|S_{\mathrm{ideal,r}}|^2}{\frac{1}{N}\sum_{r=1}^{N}|S_{\mathrm{equalized},r} - S_{\mathrm{ideal},r}|^2} \tag{3.2}$$

where $S_{\mathrm{ideal},r}$ is also the normalized r^{th} constellation point and $S_{\mathrm{equalized},r}$ is the received symbol after channel equalization. From Equation (3.1) and Equation (3.2) the relation

$$\mathrm{EVM}_{RMS} = \sqrt{\frac{1}{\mathrm{SINR}}} \tag{3.3}$$

is obtained and an EVM of -54 dB corresponds to an SINR of 54 dB.

The decrease of SINR below -2 dBm average input power is caused by an internal switch of the vector signal generator between different output circuits. The average power is linked to the PEP[11] over the PAPR[12] which is 10.4 dB in this case. The abscissa of the PEP and average power are shifted by the PAPR. Another interesting aspect is to compare this curve with a simulated one. The setting for simulation is derived from the real measurement setting: the transmit signal is generated in double precision and according to the average power sweep a quantization sweep is done. In Figure 3.10, the curve is shifted to fit the measured curve and, therefore, one can see how many bits are effectively used.

[10] EVM ... Error Vector Magnitude
[11] PEP ... peak envelope power
[12] PAPR ... peak-to-average power ratio

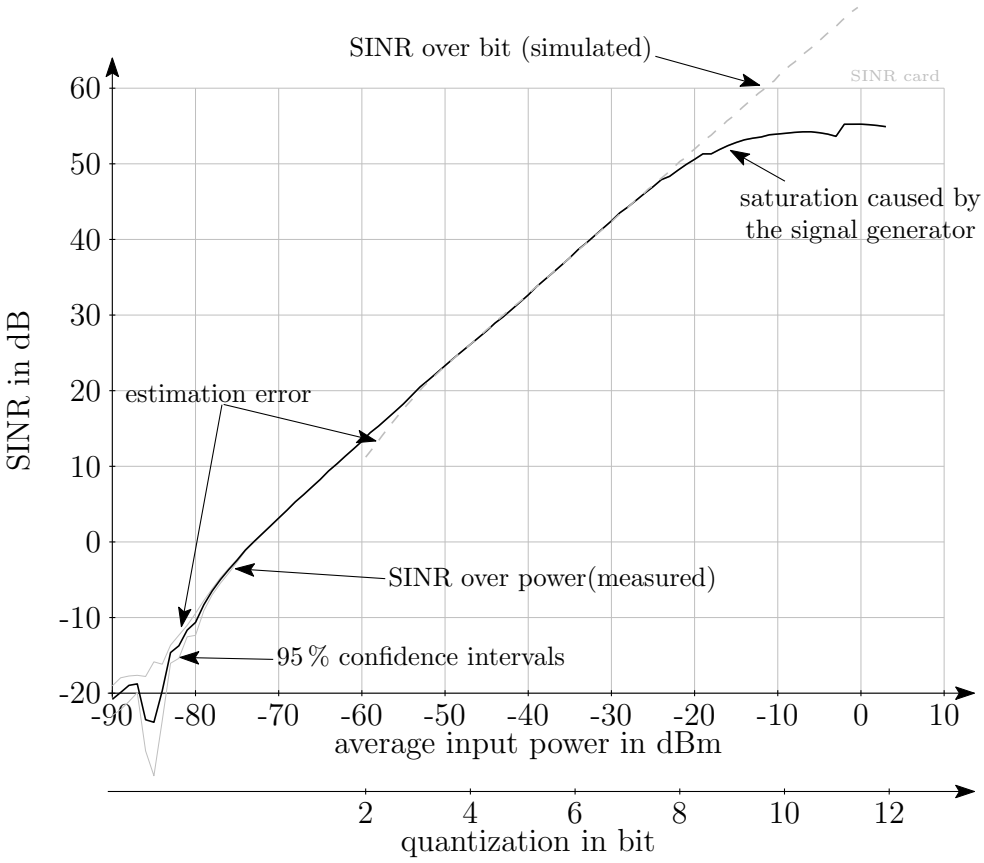

Figure 3.11: SINR over input signal power at point (1).

Preamplifier

The next device looking from the X5-RX plug in board in direction to the antennas, is the preamplifier as shown in Figure 3.5. An amplifier of type ZFL-500HLN manufactured by Mini-Circuits is applied to the testbed receiver. Its gain is specified with approximately 20 dB in the datasheet [13]. Furthermore, a typical third-order intercept point of 30 dBm is given. Figure 3.12 shows the evaluated SINR over the average input signal power. In comparison to the measurement above, the output signal power of the signal generator is adapted inverse proportional to the gain of the amplifier. The maximum SINR value does not change compared to the value which is measured with directly connected signal generator to the X5-RX plug in board. The saturation of the SINR is also caused by the Rohde & Schwarz 200A Vector Signal Generator. Indeed, this type of amplifier does not decrease the SINR at a range of up to 54 dB. Further measurements are carried out in a back-to-back manner as in [4]. In contrast to the back-to-back measurement in [4], the transmit signal is feed into the X5-RX plug in board using the ZFL-500HLN also

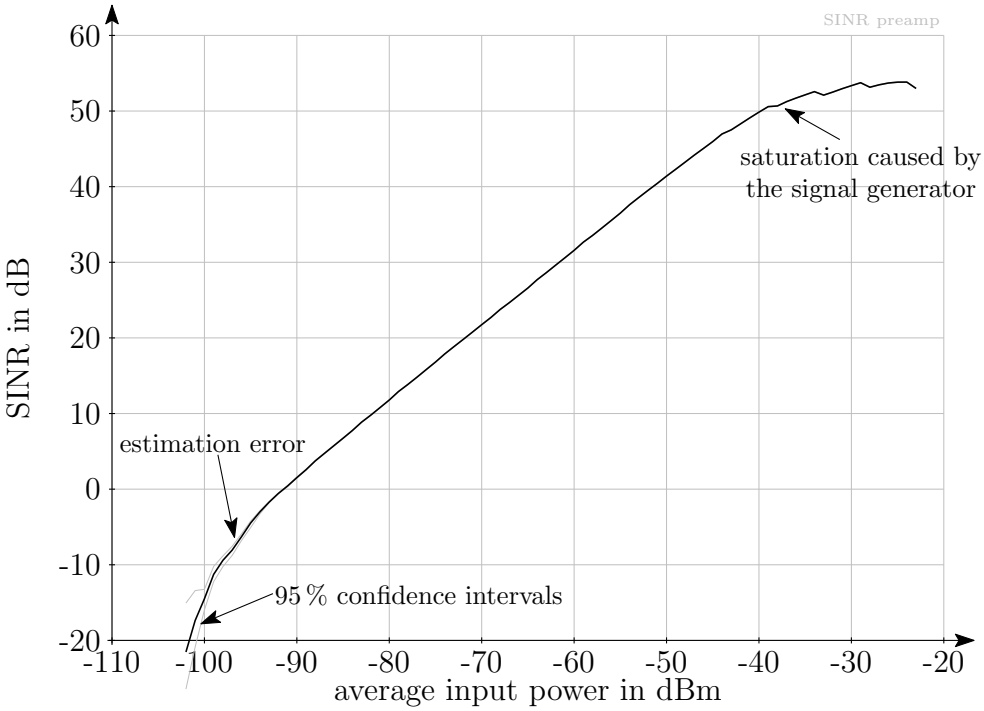

Figure 3.12: SINR over input signal power at point (2).

manufactured by Mini-Circuits as last amplifier stage. By using this configuration, the upper bound of the SINR of approximately 60 dB can be determined.

Downconverter

For converting the signal from 2.503 GHz center frequency to the intermediate center frequency of 70 MHz, a downconverter is used. As described in [14] an adapted version is currently applied on the testbed receiver (main adaption is the shifted radio frequency). The signal is feed into the testbed receiver at point (3) (see Figure 3.5). In comparison to the measurement above, a downconverter and a lowpass filter are added to the radio frequency chain. The downmixer's gain is approximately 4 dB and, therefore, the power level is decreased by this amount. Another change compared to previous measurements is that the signal generator upconverts the LTE signal to 2.503 GHz instead of 70 MHz. The evaluated result is shown in Figure 3.13. The maximum SINR which is reached at approximately -21 dBm average input power is still 53 dB. However, if the average input power is further increased, the SINR decreases. This behavior results from intermodulation by the mixer.

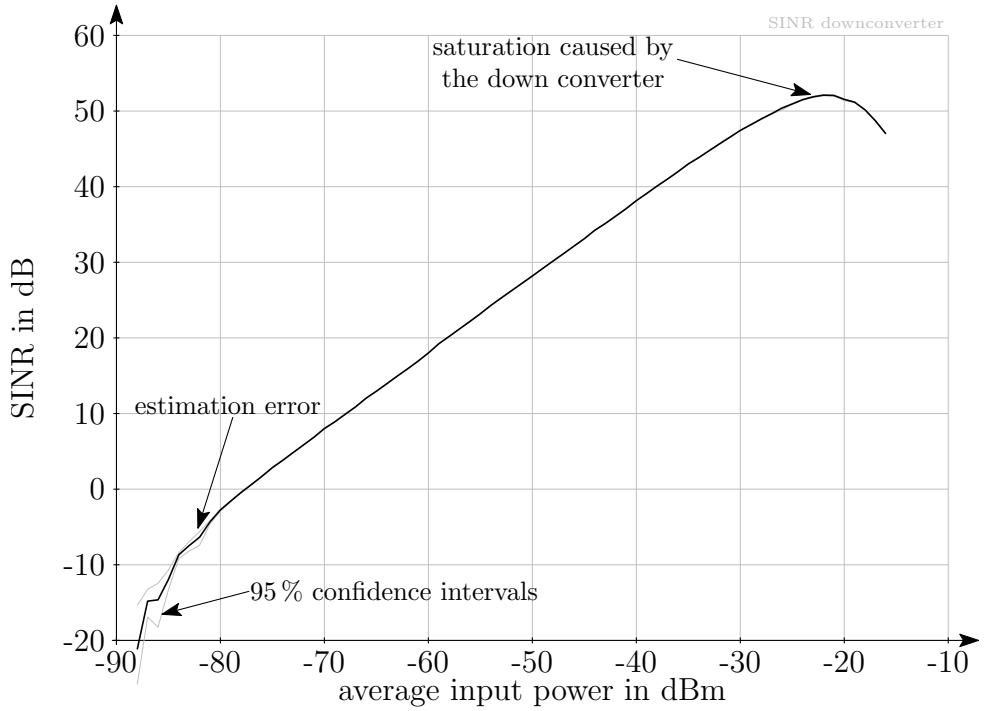

Figure 3.13: SINR over input signal power at point (3).

LNA – Testbed Receiver Input

The final measurement in this measurement series is shown in Figure 3.14. Now the signal is fed into testbed input which is point (4) in Figure 3.5. The input signal average power is decreased once again by 41 dB, which is the gain of the applied LNA of type MKU LNA 232 A, manufactured from Kuhne electronic. For comparison, Figure 3.14 shows all four channels. As mentioned above, all channels are built of the same parts and, therefore, if all components works properly, there is no significant difference expected. However, as one can see, the curve of channel two lies approximately 3 dB below the other channels. After further investigation, it turned out that the LNA of channel two does not work as well as the one of the other channels. Another interesting issue is that the maximum value of the SINR has decreased to approximately 46 dB. This is caused by intermodulation of the LNA. Furthermore, the sampled maximum value has decreased from approximately 30000 to 20000 while the signal generators output power has decreased proportionally, which also indicates that the LNA is in saturation.

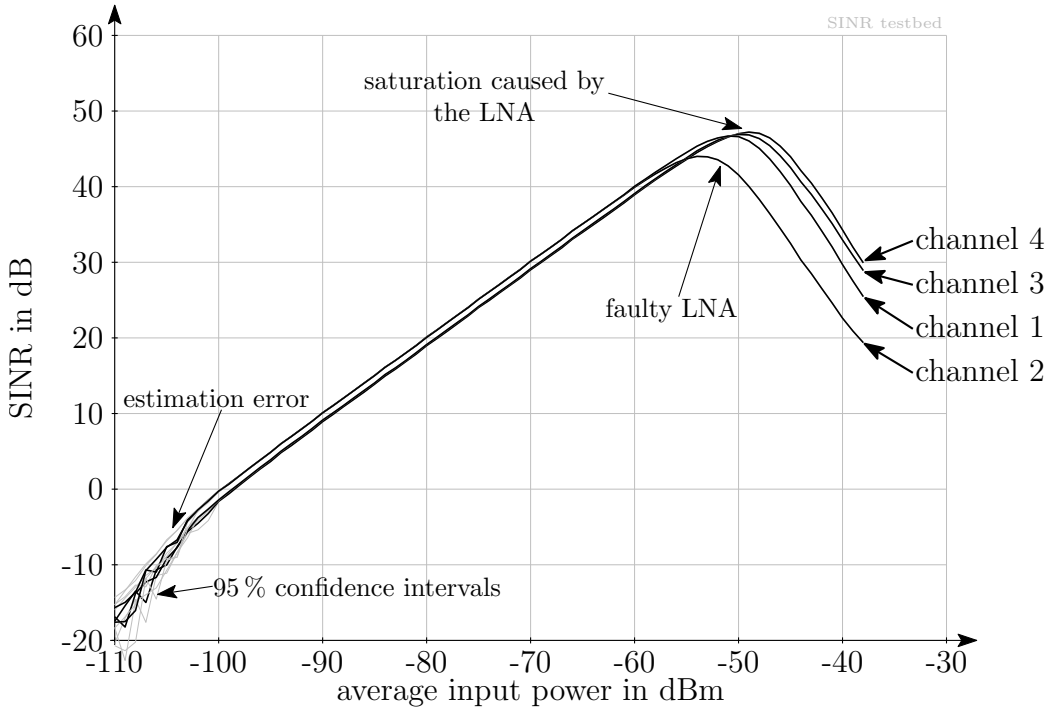

Figure 3.14: SINR over input signal power at testbed receiver input – point (4).

3.3 Frequency Spectrum

Another important task is to figure out whether the frequency range which the testbed uses to transmit, is free of spurious emissions. The testbeds center frequency is, as mentioned several times above, at 2.503 GHz. The bandwidth for the planned LTE measurements is 10 MHz, but in general, with this configuration of the testbed's transmitter and receiver, transmitting signals up to a bandwidth of 20 MHz is possible.

A simple way to check if there is disruption, is to take one testbed antenna and directly connect it to a spectrum analyzer which is, in this case, a Rohde & Schwarz FSQ 262. As receive antenna a $\lambda/4$ monopole is used.

First, the antenna is positioned outside the building. Because of the agreement between the institute and mobilkom austria AG, the used frequency range is not used for commercial mobile communication and, as result, no signal could be received. After mounting the antenna on the XYΦ-table which is approximately one meter aside the testbed receiver, the spectrum which is shown in Figure 3.15 was recorded. The peak which can be seen is caused from the testbed receiver's computer. Further investigation is done by connecting the

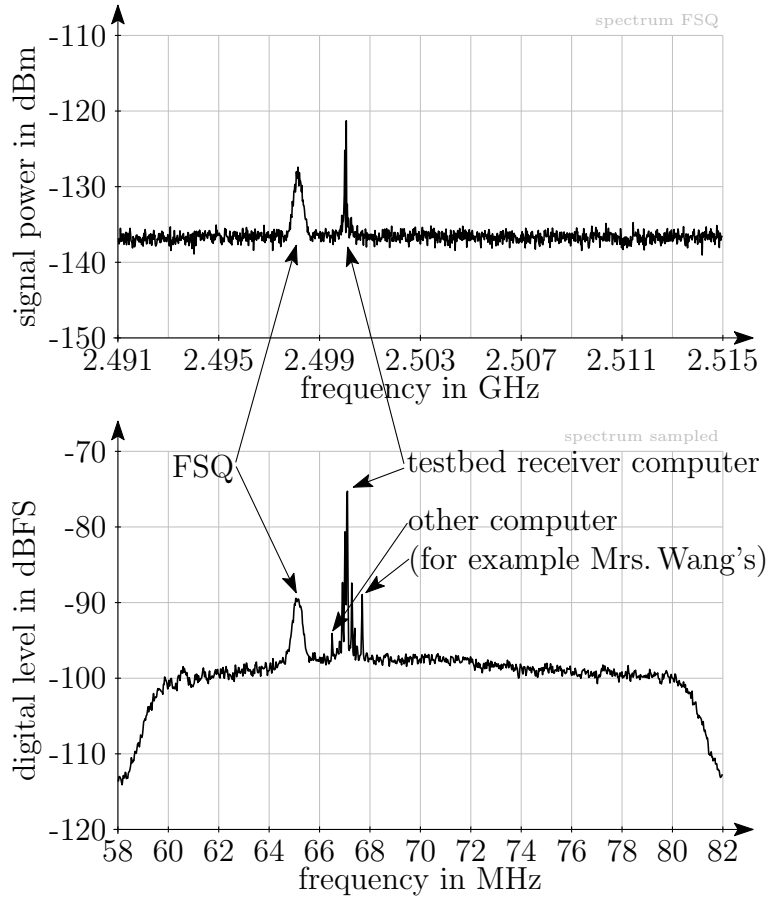

Figure 3.15: Received spectrum aside the testbed receiver. Top: measured with a spectrum analyzer. Bottom: sampled with the X5-RX plug in board, transformed and average afterwards.

spectrum analyzer at the output of the low noise amplifier. In this configuration, the received signal is filtered with a bandpass filter and amplified. The result is shown in Figure 3.15. As expected, because of low noise figure of the LNA, the peak is still there.

4 Conclusion and Outlook

Summarized in this thesis, a testbed receiver was set up to receive signals of up to 20 MHz bandwidth at a center frequency of 2.503 GHz. As part of that, a daemon has been written in C++ so that the functionality of the X5-RX plug in board is useable in terms of the testbed environment. Furthermore, a Matlab script has been written which has the main task to control the single block transmission. Using this script, which runs on the receiver's computer, together with the transmitter's timing units and receiver's timing unit, the single block transmission rate could be increased in comparison to previous setups of the Vienna Wireless Testbed. The receiver's radio frequency front end has been characterized using an LTE signal in terms of SINR. Furthermore, the spurious-free dynamic range of the ADC, which is applied on the X5-RX plug in board, has been determined. Additionally, the testbed receiver's noise and the possible added impurities have been investigated.

A faulty LNA which is applied in channel two (see Figure 3.14) should be replaced to achieve the same performance at all four channels. Moreover, it turned out that in the current configuration the achievable peak SINR value is limited by the used LNAs of type Kuhne electronic S-Band HEMT-Amplifier MKU LNA 232 A (see [15]). For test purposes, an LNA from Kuhne electronic of type Broadband Super Low Noise Amplifier KU LNA B 2227 A (see [16]) has been applied. Because the new LNA has less gain, the attenuators in the radio frequency front end chain has been removed (see Figure 3.5). The resulting graph (SINR over the average input power) using this LNA is shown in Figure 4.1. As can be seen, the maximum achieved SINR value is increased

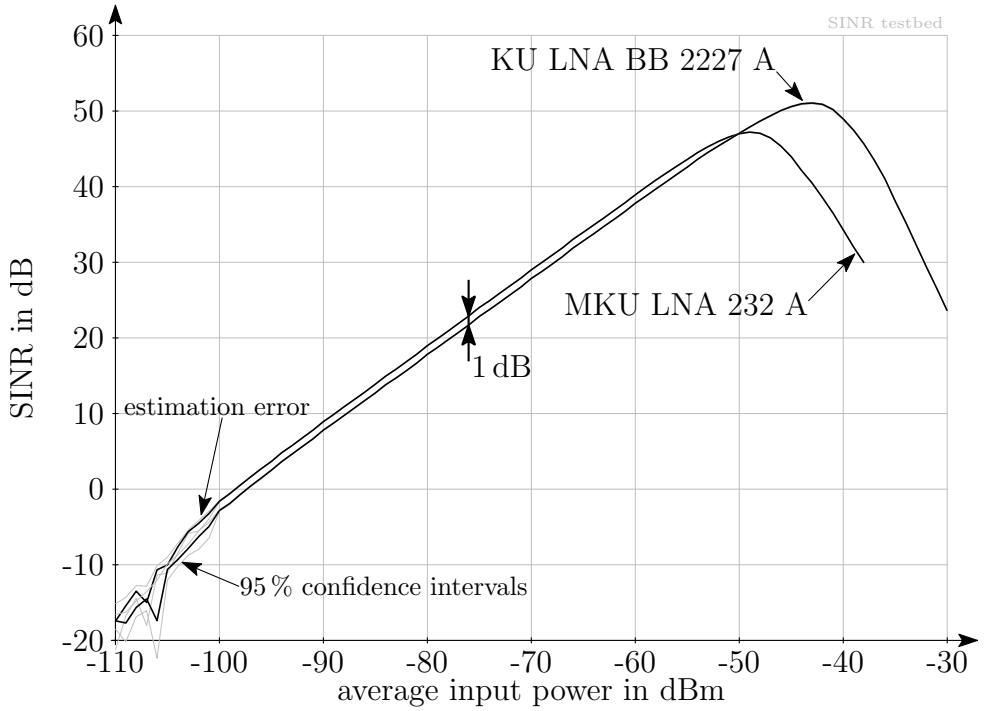

Figure 4.1: SINR over average input power at testbed input point (4) using an improved LNA.

by approximately 4 dB, because of the higher third-order intercept point in comparison to the other LNA. The decrease of approximately 1 dB is caused by the higher noise figure and the lower gain of the KU LNA BB 2227 A compared to the currently mounted LNA MKU LNA 232 A.

To avoid the spurious continuous-wave radiation of the testbed receiver's computer (see Figure 3.15) in case of transmitting OFDM signals, where the zero subcarrier is unused, the spectrum could be shifted. As a consequence, the frequency of this subcarrier matches with the frequency of the radiated spectral impurities from the computer. Another possible solution would be to shield the computer. If measurements were carried out in an office scenario the influence of the computer could be seen as a real world influence.

Bibliography

[1] S. Caban, "**Testbed-based Evaluation of Mobile Communication Systems**," Ph.D. dissertation, Institut für Nachrichtentechnik und Hochfrequenztechnik, 2009.
http://publik.tuwien.ac.at/files/PubDat_181156.pdf

[2] A. Disslbacher-Fink, "**Hardware-based Timing Synchronization**," Master's thesis, E389, 2011.
http://publik.tuwien.ac.at/files/PubDat_195758.pdf

[3] **X5-RX User's Manual**, Innovative Integration, 2390-A Ward Ave, Simi Valley, California 93065, 2010.
http://www.innovative-dsp.com

[4] L. Edlinger, "**Wireless Testbed**," Master's thesis, E389, 2011.

[5] E. Huremovic, "**Wireless Testbed Transmitter**," Master's thesis, E389, 2011.

[6] W. A. Kester, **Data Conversion Handbook**. Analog Devices, 2004.
http://www.analog.com/library/analogDialogue/archives/39-06/data_conversion_handbook.html

[7] F. J. Harris, "**On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform**," vol. 66, no. 1, pp. 51–83, 1978, doi: 10.1109/PROC.1978.10837.

[8] W. A. Kester, "**Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor**," 2008.
http://www.analog.com/static/imported-files/tutorials/MT-003.pdf

[9] **e-Handbook of Statistical Methods**. NIST/SEMATECH, 2011.
http://www.itl.nist.gov/div898/handbook/

[10] C. Mehlführer, M. Wrulich, J. C. Ikuno, D. Bosanska, and M. Rupp, "**Simulating the Long Term Evolution Physical Layer**," in *Proc. of the 17th European Signal Processing Conference (EUSIPCO 2009)*, Glasgow, Scotland, Aug. 2009.
http://publik.tuwien.ac.at/files/PubDat_175708.pdf

[11] **SMU200A Vector Signal Generator Datasheet**, Rohde & Schwarz, 2007.
http://www2.rohde-schwarz.com/file_8104/SMU_dat-sw-en.pdf

[12] M. D. McKinley, K. A. Remley, M. Myslinski, J. S. Kenney, D. Schreurs, and B. Nauwelaers, "**EVM Calculation for Broadband Modulated Signals**," *64th ARFTG Conf. Dig.*, pp. 45–52, 2004.
http://www.eeel.nist.gov/kate_papers/R1_ARFTG64_McKinley_com.pdf

[13] **ZFL-500HLN+ Low Noise Amplifier Datasheet**, Mini-Circuits.
http://www.minicircuits.com/pdfs/ZFL-500HLN.pdf

[14] L. W. Mayer, "**Wideband 4x4 MIMO Over the Air Transmission at 2.45 GHz**,"

Master's thesis, E389, 2005.
http://publik.tuwien.ac.at/files/pub-et_10521.pdf

[15] **MKU LNA 232 A2 Super Low Noise Preamplifier**.
http://www.kuhne-electronic.de/en/products/special-offer/mku-lna-232-a2.html

[16] **KU LNA BB 2227 A Low Noise Broadband Amplifier**.
http://www.kuhne-electronic.de/en/products/low-noise-amplifiers/ku-lna-bb-2227-a.html

[17] S. Caban, J. A. García Naya, and M. Rupp, "**Measuring the physical layer performance of wireless communication systems: Part 33 in a series of tutorials on instrumentation and measurement**," *IEEE Instrumentation & Measurement Magazine*, vol. 14, no. 5, pp. 8–17, 2011, doi: 10.1109/MIM.2011.6041377.

[18] S. Caban, C. Mehlführer, M. Rupp, and M. Wrulich, **Evaluation of HSDPA and LTE: From Testbed Measurements to System Level Performance**. Wiley-Blackwell, 2011.