TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMA THESIS

SDR-based RFID Reader with Sub-Symbol-Synchronization

supervised by

Assistant Prof. Dipl.-Ing. Dr.techn. Holger Arthaber
and
Univ.Ass. Dipl.-Ing. Dr.techn. Thomas Faseth

performed at the
Institute of Electrodymamics, Microwave and Circuit Engineering

by

Florian Galler, BSc.
Matr.Nr. 0826275
Carl-Appel-Straße 5/1108, 1100 Vienna

Vienna, November 2014

Abstract

Radio-frequency identification (RFID) is a wireless technology for identification of objects based on attached tags. RFID readers used for identification of tags are already commercially available. A recent research field is the localization of passive RFID tags.

During the course of this thesis an electronic product code ultra high frequency RFID reader based on a National Instruments USRP software defined radio (SDR) platform was designed and evaluated. This work provides a versatile platform for developing real time localization systems in upcoming research projects. Therefore, special requirements for a localization algorithm based on a superimposed Direct Sequence-Spread Spectrum signal were considered. Since some of the requirements couldn't be fulfilled with the driver framework provided by the manufacturer of the SDR platform a custom field-programmable gate array implementation was necessary.

The implementation is based on a soft core processor system with attached user generated custom logic peripherals. An important element of the design is the decoder, which not only decodes the backscattered data, but also provides real time sub-symbol synchronization for the planned localization algorithm. Different decoder strategies have been analyzed and compared within the thesis. Considerable attention was drawn on a hardware efficient design, such that future developments are possible and have enough free FPGA resources left.

Finally, tests with commercially available tags were made to verify the overall reader design.

Kurzfassung

Radio-frequency identification (RFID) ist eine funkbasierte Technologie zur Identifizierung von Objekten durch angebrachte Transponder. RFID Lesegeräte für die Identifikation von RFID Transpondern sind bereits kommerziell verfügbar. Gegenwärtige Forschungen untersuchen Lokalisierungsverfahren für passive RFID Systeme.

Ziel dieser Diplomarbeit ist die Entwicklung eines RFID Lesegeräts basierend auf einer National Instruments USRP Software Defined Radio (SDR) Plattform für EPC UHF RFID Tags. Dieses Lesegerät soll eine vielseitige Plattform für die Entwicklung von Echtzeit-Lokalisierungssystemen für kommende Forschungsprojekte bieten. Besondere Anforderungen eines auf Grundlage von überlagerten Direct Sequence-Spread Spectrum Signalen basierten Entfernungsmessverfahrens wurden berücksichtigt. Die vom Hersteller der SDR Plattform bereitgestellten Treiber können diese Anforderungen nicht erfüllen. Daher wurde im Zuge dieser Arbeit eine FPGA basierte maßgeschneiderte Implementierung erarbeitet.

Das Design basiert auf einem Softcore Mikroprozessorsystem an welchem selbst entwickelte Userlogic Peripherie angeschlossen ist. Ein wichtiger Teil dieser Peripherie ist der Dekoder, welcher neben dem Empfangen der vom Tag gesendeten Daten, eine Echtzeit sub-symbol Synchronisation für das geplante Entfernungsmessverfahren bereitstellt. Verschiedene Dekoder Strategien wurden im Zuge der Diplomarbeit analysiert und verglichen. Auf Hardwareeffizienz wurde während des gesamten Entwurfsprozess geachtet, so dass zukünftige Entwicklungen über genügend freie FPGA-Ressourcen verfügen können.

Abschließend wurde das entwickelte RFID Lesegerät mit kommerziell erhältlichen Transpondern getestet.

To my parents

Contents

Abbreviations

ADC	...	Analog to digital converter
AGC	...	Automatic gain control
ASK	...	Amplitude shift keying
AWGN	...	Additive white gaussian noise
BER	...	Bit error rate
BLF	...	Backscatter link frequency
BRAM	...	Block RAM
CW	...	Continuous wave
DAC	...	Digital to analog converter
DS-SS	...	Direct Sequence - Spread Spectrum
DSB	...	Double side band
EDK	...	Embedded development kit
ETSI	...	European telecommunications standards institute
EPC	...	Electronic product code
FPGA	...	Field-programmable gate array
ISE	...	Integrated software environment
LO	...	Local oscillator
LMB	...	Local memory bus
LNA	...	Low noise amplifier
PA	...	Power amplifier
PIE	...	Pulse interval encoding
PLB	...	Processor local bus
PR	...	Phase reversal
RAM	...	Random-access memory
RF	...	Radio frequency
RFID	...	Radio frequency identification
RX	...	Receiver
SDK	...	Software development kit
SDR	...	Software defined radio
SNR	...	Signal to noise ratio
SSB	...	Single side band
TX	...	Transmitter
UHF	...	Ultra high frequency

Chapter 1

Introduction

Radio-frequency identification (RFID) is a wireless technology for identifying and tracking of objects using attached transponders. RFID offers several advantages compared to optical identification methods like bar-codes, e.g. RFID tags can also be read when they are not in line of sight of the interrogator. There are several applications for RFID systems: Access management, contactless payment systems, tracking of goods or animals, electronic article surveillance (EAS), logistic, biometric passports, electronic toll collection etc.

RFID systems can usually be divided into two main parts - the reader (interrogator) and the transponders (tags). These systems can be classified by their operating frequency, power supply of the tags, operating range, costs per tag, and so on.

During the work on this thesis an interrogator compliant to the *EPC^{TM} Radio-Frequency Identity Protocols Generation-2 UHF RFID* [1] standard will be developed. The EPC RFID standard describes an RFID system for product identification with passive tags operating in the frequency range between 860 MHz and 960 MHz. The typical reading ranges are below 10 meters and are mainly dependent on the minimum operating power of the tags. For example, the UCODE G2XM tag from NXP needs a minimum operation power of -15 dBm which results in conjunction with a maximum ERP power of 33 dBm in a theoretical maximum reading range of 7.1 m according to the data sheet [2]. Practically, the reading distance is significantly lower due to attenuation of the electromagnetic waves by objects, destructive interference in multipath environments or other phenomena like bad antenna design. UHF EPC tags are passive or semi-passive meaning that they receive their operating power from the RF field of the interrogator by a carrier beacon, hence they need no batteries or other forms of power supplies. The modulation format and other important properties of EPC RFID

systems will be discussed in chapter 2.

An interesting active research question is the localization of RFID tags. The interrogator developed during this thesis should provide a versatile test platform for localization systems. The platform is intended to be used during the upcoming research project REFlex (RFID Real-Time Localization for Flexible Production Environments) (Project number: 845630) funded by the FFG (Österreichische Forschungsförderungsgesellschaft mbH) as a demonstrator. This project investigates localization systems for passive RFID tags utilized for intelligent process control and production systems. Besides other research topics a localization algorithm based on a superimposed Direct Sequence-Spread Spectrum (DS-SS) signal introduced in [3] should be further investigated. Therefore, special requirements for the localization have to be respected during the design and implementation of the interrogator, e.g. the decoder must not only be able to decode the received data of the tag, it has to produce also precise timing information of the backscattered signal edges to be used by the correlation algorithm used for localization.

A software defined radio (SDR) is an RF-transmission system where main parts of the signal processing are done by software instead of hardware. This allows the reuse of hardware for multiple communication scenarios because the software is easily exchangeable. The concept of SDRs is not new, but in recent years there has been a rapid development in this sector. The available bandwidth and resolution has increased and also the devices got more affordable. The benefit of using an SDR platform is that development time for the hardware can effectively be saved. Some of the commercially available SDR systems are closed source, meaning that the user has no access to the schematics or the program code. In a closed source system the user is only able to use the hardware as a blackbox system with the already prepared interfaces provided by the manufacturer, unless the user reverse-engineers the system which is normally a really demanding task.

During the course of this thesis an EPC RFID interrogator was implemented on an off-the-shelf SDR from National Instruments namely the USRP-2922. The USRP-2922 is essentially a relabeled version of the Ettus Research N210 with a SBX daughter board. These units are open hardware to some extent since their schematics are available for download and the FPGA design is open source and available via GitHub. Consequently, a detailed understanding of the hardware can be gained. It also allows the implementation of own software on the device and to exploit the full capabilities of the hardware.

Chapter 2

EPC^{TM}Radio-Frequency Identity Protocol

During the course of this thesis an interrogator for EPC UHF tags was developed. This chapter shall provide an introduction into the EPC^{TM}Radio-Frequency Identity Protocol standard [1]. The standard specifies the physical layer as well as the logical operation procedures of EPC UHF RFID interrogators and tags. Section 2.1 will provide an overview over an EPC RFID system. Section 2.2 introduces the logical operation, e.g. inventorying tag populations. Section 2.3 discusses the physical layer of EPC UHF RFID systems, e.g. the modulation.

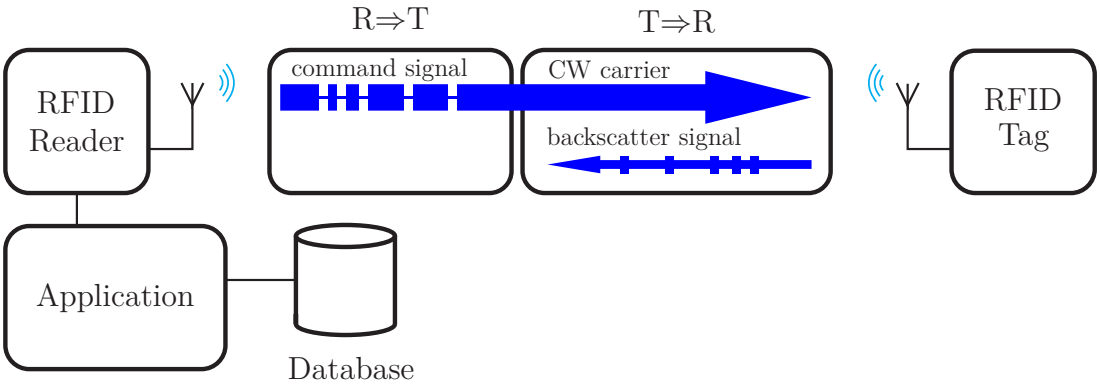

Figure 2.1: Overview of a typical RFID system

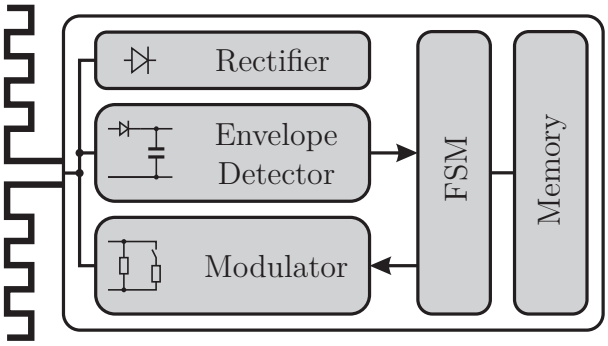

Figure 2.2: Block diagram of an RFID Tag

2.1 System Overview

In this section a typical RFID system will be discussed. In order to make it more tangible the system will be explained using an electronic article surveillance (EAS) system as example. Figure 2.1 shows an overview of an RFID system. The tag is the device which should be mounted on the object which is intended to be secured or identified. Each tag includes unique identifiers, e.g. the stored EPC and the tag ID (TID). The RFID reader is mounted in the vicinity of the warehouse exit. The antennas are mounted such that they only read tags when they leave the building. The RFID reader is continuously searching for tags and informs the stock management system (application) when a tag is leaving the warehouse, including the unique identifier of the tag. Then the stock management system looks up in the database, if the unique identifier corresponds to an article which isn't paid yet. In case of a match it triggers an alarm. In the case the good is already paid either the tag is permanently deactivated at the cash desk or it's marked in the database as already paid, therefore no alarm would be triggered.

EPC RFID systems operate at frequencies between 860 MHz and 960 MHz whereby national radio regulations must be respected. In Europe the ETSI EN 302.208 standard [4] defines usable frequencies, maximum transmit power, and an interrogator spectral mask.

2.1.1 Tags

Figure 2.2 shows a typical block diagram of an EPC RFID tag. A tag consists of an antenna with an integrated circuit attached to it. The integrated circuit chip can be further divided into an analog RF front-end and a digital data processing block. The RF front-end consists of three blocks:

- The **rectifier** block can be seen as the power supply for the remaining

integrated circuit. Since RFID EPC tags are passive their operating power must be drawn from the RF input power received via the tag antenna. Therefore, the rectifier converts the RF power into DC power.

- The **envelope detector** is used for demodulating the reader to tag (R⇒T) communication. The used modulation format will be discussed further in section 2.3.1.

- The **modulator** is used for the tag to reader (T⇒R) communication which uses so called backscatter modulation. It consists of a switch which is controlled by the finite state machine. By changing the state of the switch also the impedance of the antenna feedpoint is changed and consequently the reflected signal of the tag changes. The interrogator can detect this change in the received signal and demodulate the data. Further details of the used modulation scheme are discussed in section 2.3.2.

The digital data processing block consists of a finite state machine (FSM) used for the implementation of the protocol stack. This FSM is connected to some kind of non volatile memory for storing data, e.g. EPC, TID, kill password, access password, and user memory.

2.1.2 Interrogator

The basic task of the interrogator is to read and write memory content of tags located within its reading range. The EPC protocol specifies a half duplex data transmission. During the whole communication the RF carrier must be turned on to supply the passive tags with sufficient power.

During the R⇒T communication the reader transfers data to the tag by switching off the RF carrier for short intervals, where the information is coded into the time duration between these pulses. The short pulses can be detected by the envelope detector of the tag and are decoded by the FSM. This is called pulse interval encoding (PIE) and will be discussed in detail in section 2.3.2. During these pulses the power supply to the tag is interrupted. According to the EPC standard [1] the maximal duration of these off pulses is 13.125 µs. Since the rectifier inside the tag must be able to supply the other blocks continuously with power, it has to store energy in capacitors for the periods where no power is supplied from the interrogator.

As already explained the T⇒R communication is done by backscatter modulation. Therefore, the reader has to transmit an unmodulated CW carrier during an expected uplink communication. The reader has to synchronize to the tag response and decode it.

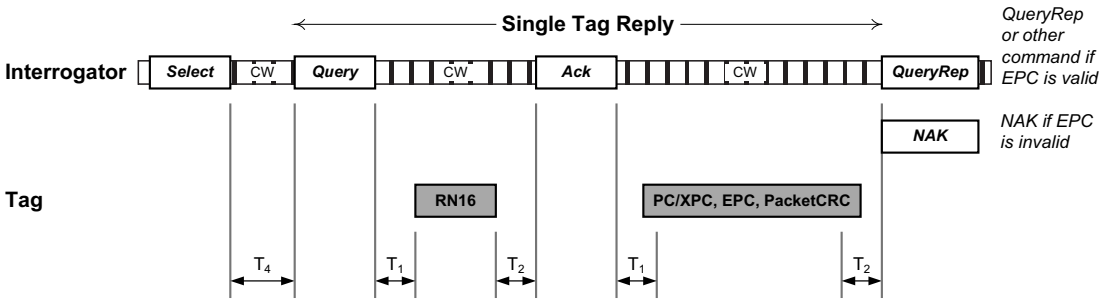

Figure 2.3: Link timing for reading out the EPC (from [1])

Furthermore, the reader also has to provide an interface which is used by the application to control and configure the reader, e.g. it gives the reader the command to read out the EPC of all tags and report the readout data. This is normally done via a standardized interface with custom commands, e.g. Ethernet, serial port or USB.

2.2 Logical Operation

Since EPC UHF RFID tags offer read distances up to a few meters, multiple tags can simultaneously be located within the reader's reading range. Therefore, the EPC standard defines an access scheme to separate multiple tags. The communication is always initiated by the interrogator. Figure 2.3 shows a typical communication flow to read out the EPC of a tag.

In the beginning the interrogator selects a part of or the whole tag population by using the *Select* command as specified in the EPC standard [1]. This allows altering flags in the tags based on masked memory content within the tags. These flags are later used by the *Query* command to inventories only a sub population of the tags located within the reading range.

The inventory round is initiated by a *Query* command. The *Query* command specifies the link parameters for the next inventory round, e.g. the used backscatter link frequency (BLF), the used backscatter modulation format, the number of slots (Q), and flags which must be set to participate. When a tag receives a correspondingly valid *Query* command it initializes its slot counter with a random number between 0 and $(2^Q - 1)$. All tags which have now a zero in the slot counter reply immediately with a 16-bit random number (RN16). The interrogator must send an *ACK* including the previously sent RN16 within a stringent

timing requirement

$$\frac{3}{f_{BLF}} < T_2 < \frac{20}{f_{BLF}}. \tag{2.1}$$

After the tag receives the *ACK* including the right *RN16* it backscatters its stored EPC. The interrogator then has two opportunities. First it can access the tag by sending a *Req_RN* command, where afterwards for example the tag's memory could be altered. Or the reader can use the *QueryRep* command, which forces all tags participating the inventory round to decrease their slot counter by one. Again, tags with slot counter equal to zero reply with an *RN16* like it would be done after the *Query* command. This can now be done until all slots (2^Q) are read.

The parameter Q has to be chosen carefully according to the expected number of tags such that probability of a collision is low. Of course, when two tags choose the same slot counter after the *Query* command, their *RN16* responses will collide and the tags will not be inventoried because the interrogator can't receive the correct *RN16* and therefore, can't send a valid *ACK* message. This results in the fact that these tags can't be read during the actual inventory round. On the other hand, choosing a too high Q value results in many free slots and therefore longer readout cycles and decreased tag read rates.

The problem regarding the implementation of an EPC RFID reader within an SDR is the following. At the maximum backscatter link frequency (BLF) the time between the end of the *RN16* response of the tag and the beginning of the *ACK* of the reader during an inventory round must be less than 31 µs. It is obvious that this is a stringent timing requirement which can't be fulfilled with using an implementation with the provided driver framework as will be discussed in the latter section 3.1.

2.3 Physical Layer

The following section discusses the modulation formats used in the up-link (R⇒T) and in the down-link (T⇒R) communication of an EPC RFID interrogator.

2.3.1 Reader to Tag Communication

According to the EPC standard, an interrogator communicates to a tag population within its reading range by modulating an RF carrier using double-sideband amplitude shift keying (DSB-ASK), single-sideband amplitude shift keying (SSB-ASK), or phase reversal amplitude shift keying (PR-ASK). The binary data is coded into different time intervals between short pulses. This is called pulse interval encoding (PIE). This modulation format can be decoded very hardware

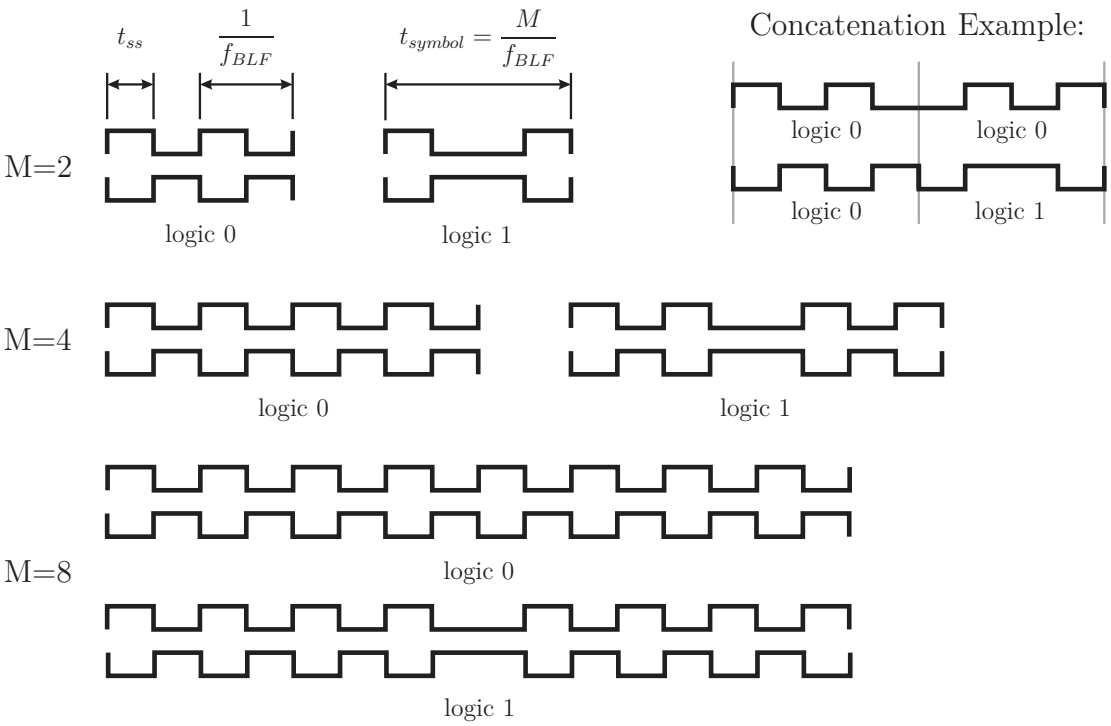

Figure 2.4: Miller symbols with both starting phases and a concatenation example

efficient by the tags using an envelope detector. This is essential since the available processing power in the tags is limited due to their passive operation with the power supplied over the RF field of the interrogator.

The (R⇒T) communication is started by a Preamble or a Frame-Sync. Both are used to synchronize the tag to the length of the PIE symbols used for the subsequent communication. A Preamble is only sent in front of a *Query* command. It additionally includes the T⇒R calibration (*TRcal*) period, which together with the divide ratio (DR) specifies the tag's BLF used during the inventory round.

An important point is the pulse shaping of the transmit signal. Since transmit signals have to fulfill stringent requirements in time domain as well as in frequency domain. The requirements of the EPC standard [1] as well as national radio regulations rules have to be respected. In Europe the ETSI EN 302.208 [4] defines the spectral mask of the interrogator signal and the usable frequencies.

2.3.2 Tag to Reader Communication

A tag communicates to an interrogator using backscatter modulation. Thereby the tag switches the antenna reflection coefficient between two states in accor-

dance to the data to be sent. The data is either encoded as FM0 baseband or Miller sub-carrier modulation. The interrogator uses the *Query* command to specify the used modulation type for the inventory round. Therefore, an interrogator does not have to be able to decode both modulation types. During the course of this thesis only Miller modulation will be investigated.

Miller modulation is a sub-carrier backscattering modulation format, where the sub-carrier frequency equals the tag's BLF (f_{BLF}). According to the standard [1] the BLF has to be chosen between 40 kHz and 640 kHz. The BLF is specified by the interrogator during the *Query* command by length of the *TRcal* period and the divide ratio (DR). Miller modulation can somehow be seen as some kind of binary phase shift keying. The M value is 2, 4, or 8 and sets the number of sub-carrier cycles by bit, therefore the Miller symbol rate is

$$f_{symbol} = \frac{1}{t_{symbol}} = \frac{f_{\mathrm{BLF}}}{M}. \tag{2.2}$$

Figure 2.4 shows the different possible miller symbols. It can be seen that a logic one is transmitted by inverting the phase at the middle of the symbol, a logic zero lacks of this inversion. The rule for concatenation of two Miller symbols is that a phase change between two symbols is only made if two consecutive logical zeros have to be transmitted. Miller symbols can be split into Miller sub-symbols, e.g. a Miller M=2 symbol can be split into four Miller sub-symbols. The duration t_{ss} of the sub-symbols is equal to half a BLF cycle.

$$t_{ss} = \frac{1}{2f_{\mathrm{BLF}}} \tag{2.3}$$

Figure 2.5 shows the different preambles which precede the T⇒R communication. By setting the *TRext* bit within the *Query* command the reader is able to choose if the normal preamble or an extended preamble shall be used by the tag for its response. The preamble is used by the reader to find the beginning of the tag response and to synchronize to the tag's BLF which has tolerances of up to ±22 %.

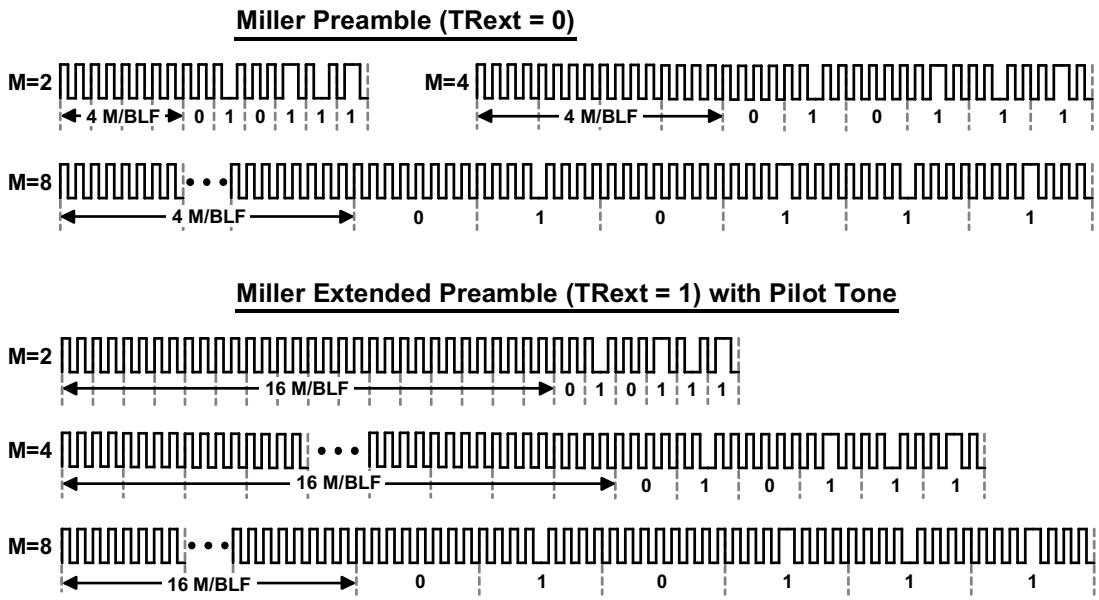

Figure 2.5: Different Miller preambles (from [1])

Chapter 3

NI USRP-2922 Software Defined Radio Platform

The focus of the following chapter is on the introduction of the SDR platform used during the course of this thesis. Section 3.1 presents the reasons for a custom FPGA implementation of the RFID reader instead of realizing the reader using the provided driver framework. Section 3.2 provides an overview of the hardware components and specifications of the used SDR platform.

3.1 Customized Implementation versus Available Framework

The USRP-2922 is intended to be used as an SDR connected to a personal computer (PC) running Matlab, Simulink, Labview, or GNU Radio. A library of drivers is available, such that the unit can be integrated very easily into these software environments. By using the pre-built drivers the base band signal processing can be carried out using the PC. In such a scenario the raw analog to digital converter (ADC) and digital to analog converter (DAC) samples are transferred to the PC via the Ethernet connection. This is a very convenient way to use the SDR. However, limitations are introduced by the Gigabit Ethernet connection.

The first limitation is the data rate of the Gigabit Ethernet interface. The maximum raw data rate of the ADC is 2.8 GBits/s and the maximum raw data rate of the DAC is 3.2 GBits/s. This is apparently too much for the Gigabit Ethernet (1 GBits/s) interface. Therefore, in order to transfer the sampled data, the SDR has to reduce the sampling rate and/or the resolution. According to the

datasheet of the USRP-2922 [5] the maximum sampling rate achievable via the Ethernet interface is 50 MSamples/s for a resolution of 8 bit or 25 MSamples/s for a resolution of 16 bit. Since the ranging algorithm which is planned to be implemented on the reader uses low spectral power density broadband signals a reduction of the sampling rate and resolution would considerably limit the performance of the localization estimation.

A second issue is the latency that is created when the software running on the PC communicates to the USPR platform. The EPC standard [1] has stringent timing requirements especially during inventorying a tag population as discussed in section 2.2. The large latency would lead to problems especially when higher backscatter link frequencies are used. The problems arising when GNU Radio is used for implementing a UHF RFID reader are summarized in [6]. Although the authors are not using exactly the same hardware, the problems can be assumed to be very similar. Furthermore, for the intended localization algorithm the partly unpredictable latency would cause reasonable problems.

A custom FPGA implementation doesn't suffer from these problems at all and generally results in a higher flexibility for testing and developing of the localization algorithms. Since the main target is to build a versatile test platform, a custom implementation for the FPGA has been realized. The drawback of this solution is that much more implementation effort is needed. Furthermore, for a custom FPGA implementation a detailed understanding of the hardware is necessary. Therefore, the available schematics [7] [8] and the datasheets of the components have been analyzed. The next section provides an overview of the system components of the USRP-2922.

3.2 National Instruments USRP-2922

Within this chapter the hardware components of the USRP-2922 are discussed. This unit essentially is a relabeled version of the Ettus Research N210 with a SBX daughter board. Ettus Research is a subsidiary of National Instruments and their products are sold directly and also via National Instruments.

The SDR used for this diploma project consists of two printed circuit boards (PCB), namely the N210 and the SBX. The N210 is the lower board depicted in figure 3.1 and is mounted directly on the case of the SDR. It includes the circuits used for baseband signal generation and processing and also the interfaces for the communication to a PC. On top of the N210 the SBX daughter board is mounted, which carries the RF relevant components like mixers, variable attenuators, synthesizers and amplifiers. It is used for up- and down-converting the analog baseband signals.

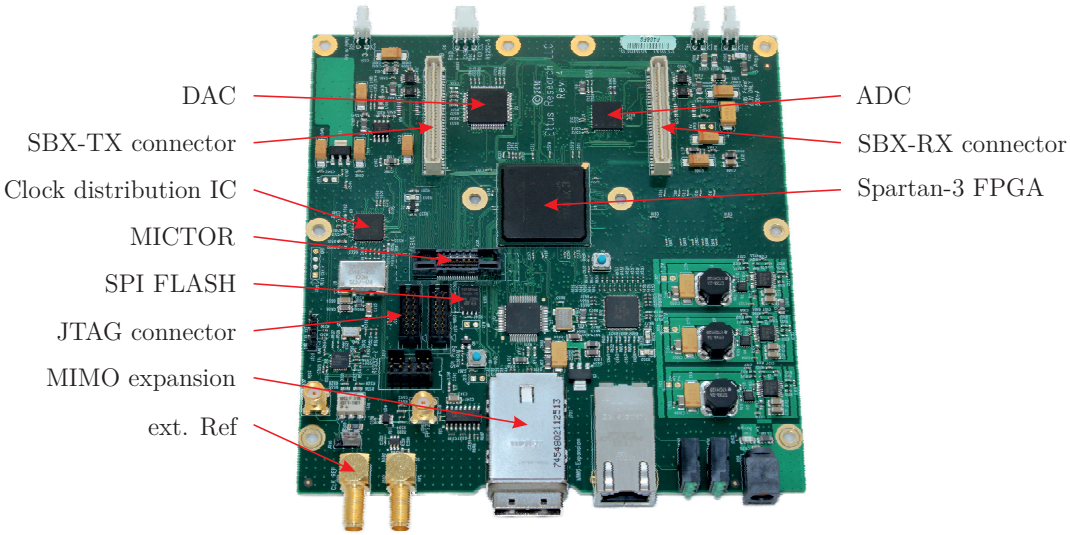

Figure 3.1: Picture of the N210 circuit board

3.2.1 N210

Figure 3.2 shows an overview of the N210 circuit board and the included components. Blue dashed lines indicate clock lines, black lines indicate data connections. In the middle of this picture the FPGA can be found which is connected to almost everything for configuration and data transfer. The FPGA is also the only user programmable logic inside the USRP-2922 platform. The utilized FPGA is a Xilinx Spartan-3A DSP XC3SD3400A in a high performance speed grade and FG676 packaging [7].

This FPGA has no non-volatile memory inside the package. Therefore, the configuration data must be stored externally. In the case of the N210 the configuration could be loaded into the FPGA either from an SPI Flash memory or directly from the PC running a design environment by using a JTAG adapter cable. The SPI Flash can be directly programmed using the SPI programming header or via JTAG over the FPGA. More information about the configuration of the FPGA can be found in the Spartan-3 Configuration User Guide [9].

The clock distribution IC Analog Devices AD9510 contains a PLL for the generation of the main clock. Either an internal or external reference clock can be chosen by an RF switch controlled by the FPGA. The external clock is either derived from the REF IN SMA connector or from the MIMO expansion connector. Furthermore, the AD9510 is used as clock distributor of the 100 MHz main VCO. Different clock dividers can be applied to each clock output. After powering up the AD9510 the PLL is powered down, therefore the 100 MHz main VCO

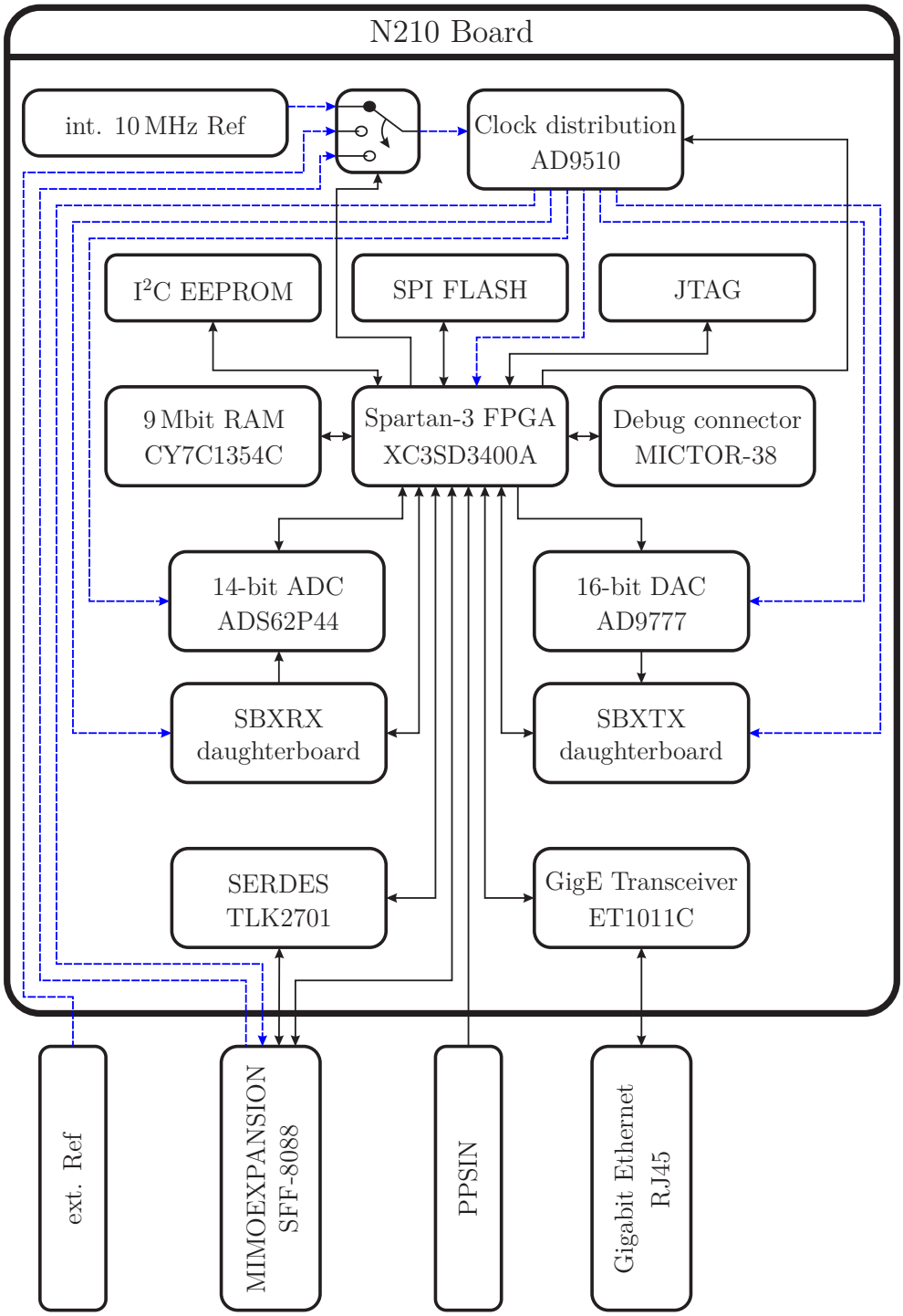

Figure 3.2: Overview of the hardware on the N210 circuit board. Blue dashed lines indicate clock lines, black lines indicate data connections.

driving the clock distribution is free running at first. A clock divider of two is set to the FPGA clock by default also. Since the whole design is planned to run in the 100 MHz clock domain, a state machine for power up configuration was implemented in the FPGA, see section 5.1.

The digital to analog converter (DAC) Analog Devices AD9777 is used to generate the analog baseband output. Its maximum input sampling rate is 160 MSamples/s and it has a resolution of 16 bits. The DAC is connected to the FPGA via a 32 bits parallel bus used as data interface for the sampling data and an SPI interface used for configuration. The DAC receives its sampling clock directly from the clock distribution IC. In the final implementation the DAC runs at a sampling rate of 100 MSamples/s.

The analog to digital converter (ADC) Texas Instruments ADS62P44 is used for sampling the analog baseband input signal. This ADC is a dual channel 14 bit 105 MSamples/s ADC. It's connected to the FPGA via a 28 bits parallel bus which is transferring the sampled data and an SPI interface used for configuration. The ADC gets its sample clock directly from the clock distribution IC. Therefore, the sampling jitter between ADC and DAC is very small. According to [3] this is a crucial requirement for the localization algorithm.

A Cypress CY7C1354C-AC 9 Mbit SRAM is placed on the printed circuit board. In the implemented FPGA design this RAM is used as storage for the custom built debug module which is intended to capture and playback ADC data at full sampling rate.

Furthermore, a Microchip 24LC024 2 Kbit I^2C EEPROM is available, which can be used to store board specific calibration data. During the work on the thesis such a storage was not needed but the functionality is designated for future use.

Another very useful component is the MICTOR 38 debug connector. This connector is directly connected to the FPGA and provides direct access to 34 FPGA pins. Since MICTOR connectors are commonly used e.g. breakout boards are available. For the implementation of the reader in this thesis the MICTOR connector is used for the serial interface to the PC.

A gigabit Ethernet transceiver is also available on the N210 board which can be used for interfacing to the PC. The Ethernet interface is not used in the FPGA implementation built during the work on this thesis, since the interface to the PC was implemented by a simple serial connection.

Another interesting interface is the MIMO expansion connector which is intended to be used to synchronize two N210 boards. On this SFF-8088 connector eight differential line pairs are available. Four pairs are directly connected to the FPGA, two pairs are connected to a Texas Instruments TLK2701 transceiver, and the remaining two pairs are connected to the AD9510 clock distribution IC. This allows flexible synchronization of two N210 baseband clocks. The Serializer/Deserializer Texas Instruments TLK2701 would allow full duplex 1.6 Gbit/s

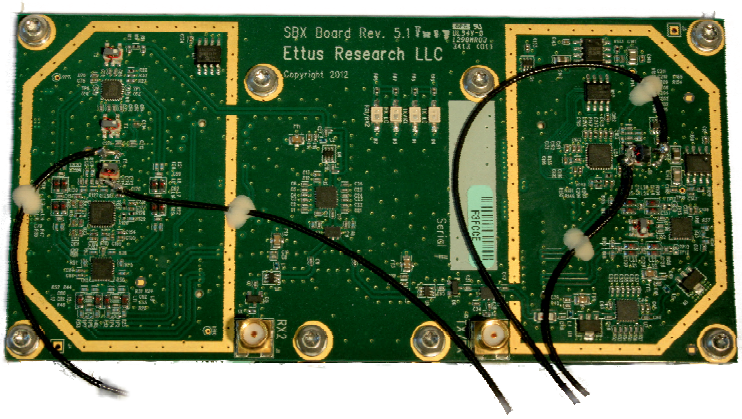

Figure 3.3: Picture of the SBX circuit board with the hardware modifcation presented in chapter 6

communication to another device. Therefore, the MIMO expansion interface could be used for future applications to provide proper synchronization between multiple SDRs, e.g. for a MIMO RFID reader consisting of multiple USRPs.

3.2.2 SBX

Figure 3.3 shows the SBX daughter board which is placed on top of the N210 board. The SBX board converts the baseband signals to the desired radio frequency (RF) and vice versa. The components of the SBX are all controlled by the FPGA located on the N210 board. Figure 3.4 shows an overview of the RF-components. It can be seen that the SBX uses direct conversion since there are no intermediate frequency stages [8].

The transmit branch and the receive branch can be seen as separate units, except for the RF front-end. The two RF-switches at the front-end allow that the TX/RX ANT port could be used as transmitter or receiver (half-duplex). The other configuration option is that for RX and TX separate ports are used. Since an RFID reader has to transmit a CW carrier during the tag response the RX and TX branches have to work simultaneously. Therefore, the switches in the RF front-end have to be configured to the latter configuration.

In order to allow dual-band operation, where RX and TX frequencies are different, the SBX board is equipped with two separate synthesizers for the generation of the local oscillator (LO) signals. These synthesizers utilize the same reference clock provided by the N210 board. Therefore, their outputs are phase synchronized if they are configured to deliver the same output frequency. However, the separate synthesizer implementation introduces additional phase noise,

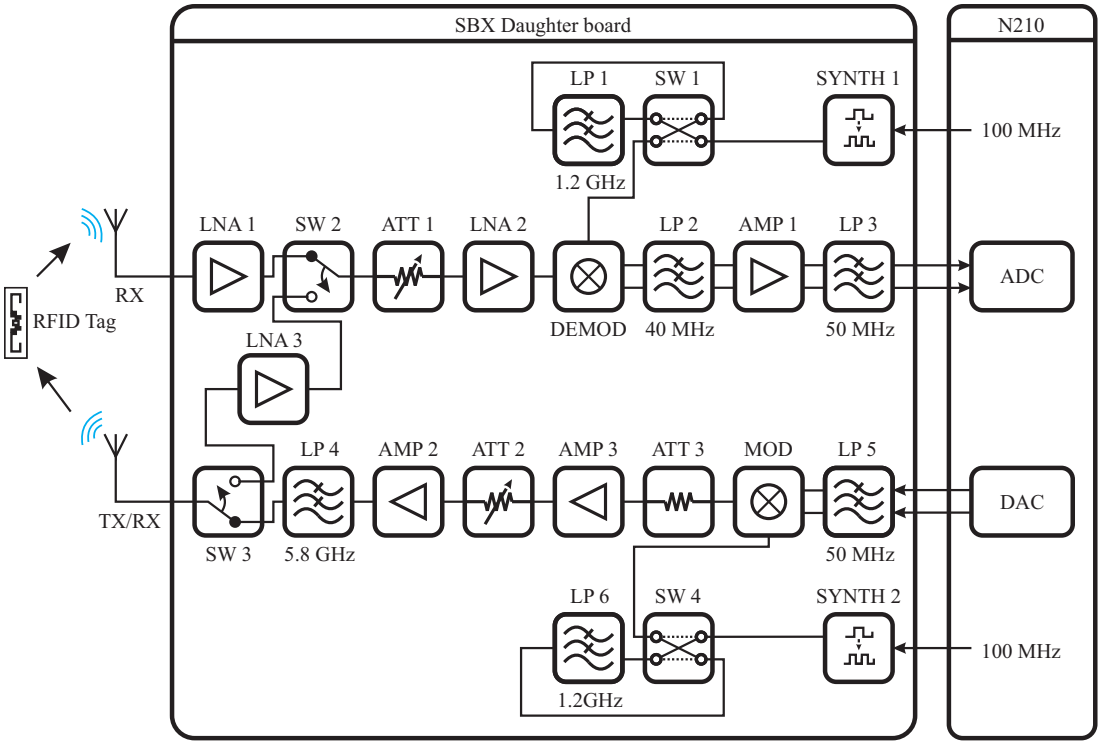

Figure 3.4: Overview of the hardware on the SBX circuit board

which degrades the performance of the RFID reader. An analysis of this effect and a solution found is presented in chapter 6. The synthesizers are connected to the FPGA via SPI and some GPIOs for configuration. The output of the synthesizers can be filtered by a 1.2 GHz low pass to suppress harmonics.

In the transmit and receive path variable attenuators are used for level adjustment. They can be controlled from the FPGA via a parallel interface and can be adjusted in 0.5 dB steps from 0 dB to 31.5 dB.

Again, two Microchip 24LC024 2 Kbit I^{2}C EEPROM are available on this board equal to the N210. As explained before, this storage can be used to store board specific data, e.g. calibration data. During the work on this thesis such a storage was not needed,nevertheless, the design is prepared for later integration.

Chapter 4

Decoder

The focus of the following chapter is on the design of the decoder for the RFID reader. In section 4.1 the requirements of the decoder will be discussed. Section 4.2 introduces the maximum likelihood sequence decoder as the optimum sequence decoder for the AWGN case. Problems regarding the implementation of such a decoder will be identified. Section 4.3 will explain the synchronous sub-symbol decoder developed and implemented within this thesis.

4.1 Requirements

An interrogator communicates to a passive tag by first transmitting a reader command (as shown in section 2.3.1) which is followed by an unmodulated RF carrier and listening for the backscattered reply. The tag backscatters data by varying the reflection coefficient of its antenna between two states in accordance with the modulation scheme selected by the reader. Therefore, the reflected signal of the tag changes, is received by the antenna of the reader again, and is used for demodulation of the exchanged data.

Figure 4.1 shows a typical constellation diagram of a backscattered signal. The large offset from the centerpoint of the IQ diagram is caused by the self interference of the unmodulated CW carrier used for supplying the tag with power. This interference is caused by the coupling between transmit and receive antenna and maybe also by scattering objects in the close vicinity of the antennas. Consequently, the overall received signal must be attenuated in a way that the receiver hardware isn't overdriven. When the received overall signal is attenuated also the tag response is attenuated. This results in a smaller distance between the constellation points s_1 and s_2 at the ADC and therefore reduces the sensitivity of the receive path. Hence, self interference caused e.g. by antenna coupling is a

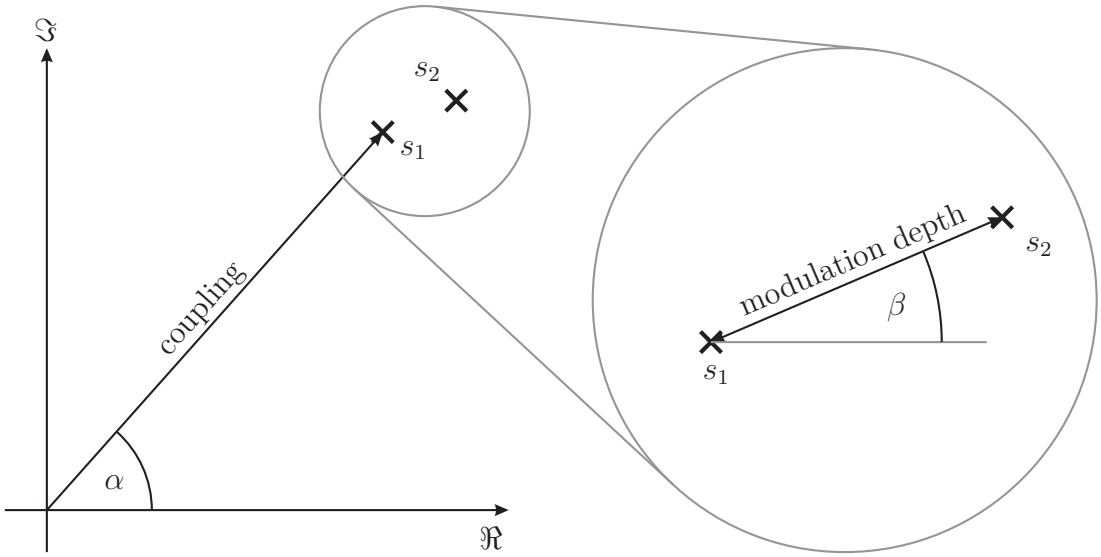

Figure 4.1: Constellation

critical system parameter [10]. The magnitude and angle α of the coupling can change significantly when scattering objects move.

The second position of the constellation s_2 is obtained when the tag is in its second backscattering state reached by switching the reflection coefficient of the antenna. The distance between s_1 and s_2 is called the modulation depth. The angle β mainly depends on the tag to reader distance. A moving tag would result in a continuously changing angle β. Therefore, if moving tags should be read also, the decoder must also be tolerant to slow changes of the positions of the constellation points.

As explained in chapter 2 the backscatter modulation of the UHF-EPC tags is either FM0 or Miller modulated. In order to reduce the complexity of the receiver only Miller modulation will be implemented because anyway the larger amount of data will be needed for the future implementation of the localization algorithm.

Another crucial issue is the large frequency tolerance of the backscatter-link frequency which is allowed by the EPC standard [1] of up to $\pm 22\,\%$. Furthermore, the decoder must also be tolerant to backscatter-link frequency variations of $\pm 2.5\,\%$ during the backscatter. These large frequency variations are allowed in order to save processing power inside the tags and to allow cheap mass production of tags.

Moreover, for the planned localization system some requirements have to be considered for the decoder implementation. The UHF RFID DS-SS range estimation algorithm introduced in [3] uses the demodulated data for suppressing static

scattering objects. Therefore, the demodulated data, especially the positions of the sub-symbols' edges, must be available before the correlation of the spreading sequence with the received data is carried out. This leads to the requirement that the demodulated data must be available as soon as possible, since all delays in the decoding process have to be compensated by delaying the ADC data for the ranging process, which is limited by the available memory of the FPGA. For example, storing the ADC data of one Miller 8 symbol with the lowest BLF would need 64 BRAMs, more than half of the available BRAMs of the used FPGA. Of course this is not feasible.

A regular RFID reader which only demodulates the data sent by the tag would use down sampling to reduce the amount of data to be processed. However, the localization algorithm needs precise timing information of the sub-symbol edge position. Consequently, down sampling of the received data for demodulation would introduce here a timing uncertainty. Therefore, it is a main goal of the design not to reduce the sample rate. On the other side this limits the possible complexity of the receiver a lot. For example, an FIR filter correlating the sampled data with a Miller symbol would not be possible in the 100 MHz clock domain, since too much filter taps would be needed and timing closure would be impossible.

4.2 Maximum Likelihood Sequence Detector

Miller modulation has four possible Miller-encoded symbols where some state transitions are not allowed. Hence Miller modulation is not memory free. Therefore, the optimum detector in the additive white Gaussian noise (AWGN) case would be the maximum likelihood sequence detector[11]. Practically this decoder would be problematic in the implementation especially when the requirements for the localization are addressed.

First of all it would require a large number of correlator banks which would be exponentially growing with the number of bits to decode. This could be avoided by using the Viterbi algorithm.

Another problem would be the frequency tolerance and the frequency variation during backscattering of the BLF. This problem could be tackled by using the preamble to synchronize to the tag's BLF initially. Afterwards the actual BLF must be tracked, e.g. by multiple correlators running on different BLFs. But the complexity would be too high to be implemented within the 100 MHz clock domain inside the given FPGA.

As explained in section 4.1 a problem would be that the demodulated bits would be available after some Miller symbol periods. Therefore the ADC samples used for the planned localization system must be delayed which would require an

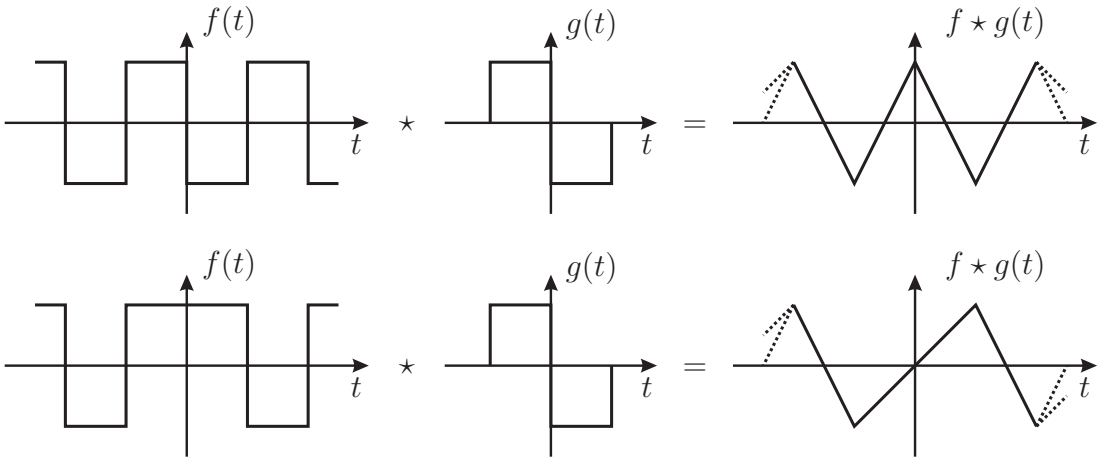

Figure 4.2: Example of correlation results

unfeasible amount of memory.

Furthermore, the complexity of this decoder would also be too large for an implementation inside the given FPGA without a reduction of the sampling rate. When the sampling rate is reduced also the time resolution reduces, which is not desirable for the implementation of the planned localization algorithm.

With regard to above mentioned problems for the implementation within this thesis a decoder design had to be chosen that is suboptimal from the point of view of signal and detection algorithms, however, fitting better the needs of the localization algorithm.

4.3 Synchronous Sub-Symbol Decoder

In this section the implemented decoder will be discussed. This decoder was designed, simulated and implemented during the work on this diploma thesis. The design can be separated into two main parts. First, the received signal is correlated with one period of a square wave which has a frequency equal to the expected BLF. Second, a decoder converts the correlation results into data bits. The whole system has been simulated using Matlab and has afterwards been implemented using Xilinx ISE (see section 5.3).

4.3.1 Correlator

Miller modulation encodes the data bits into phase changes of the backscattered sub-carrier. Therefore, the data bits can be decoded when the time differences between the phase changes are known. To estimate this differences the received

signal is correlated with $g(t)$ which is one period of a square wave which has a frequency equal to the expected BLF f_{BLF}.

$$g(t) = \begin{cases} -1, & -\frac{1}{2f_{\mathrm{BLF}}} < t \leq 0 \\ 1, & 0 < t \leq \frac{1}{2f_{\mathrm{BLF}}} \\ 0, & otherwise \end{cases} \tag{4.1}$$

Figure 4.2 shows the correlation result of the function $g(t)$ with two possible received signals $f(t)$ - one with a phase change and the other without a phase change at $t = 0$. One could see that at each edge of the received signal correlation maxima or minima occur. Furthermore, for the phase change the correlation result is zero. Using that information the decoder is now able to decode the data bits. A positive side effect of this correlation is that an offset of the BLF doesn't affect the correlation result much and that this kind of correlation can be realized time discrete in a very hardware efficient, since it can be based on an iterative algorithm:

$$(f \star g)[n] = (f \star g)[n-1] - f[n-a] + 2f[n] - f[n+a] \tag{4.2}$$

where

$$a = \left\lceil \frac{f_{sample}}{2f_{\mathrm{BLF}}} \right\rfloor. \tag{4.3}$$

Another highly desirable property of the decoder can be found when correlation is performed on the complex baseband signal of a backscattered signal. Here, self interference caused e.g. by antenna coupling will be suppressed since the mean of the base function $g(t)$ is zero. Therefore, as long as the coupling changes much slower than the BLF, which will be true for almost all usecases of the designed reader, the influence of self interference or static reflections on the correlation result of the decoder is negligible.

Furthermore, the correlation result of the complex baseband signal can be separated into magnitude and phase. The magnitude is independent of the unknown angle β between the backscatter constellation points. Nevertheless, it includes the information of the phase changes of the sub-carrier and is therefore used by the decoder for decoding the data bits. The phase of the correlation result jumps between β and $\beta + \pi$. This could be used to estimate the unknown angle β. As mentioned before, time variations of the angle β could be used in the future, e.g. to estimate the speed of the tag in relation to the RFID reader.

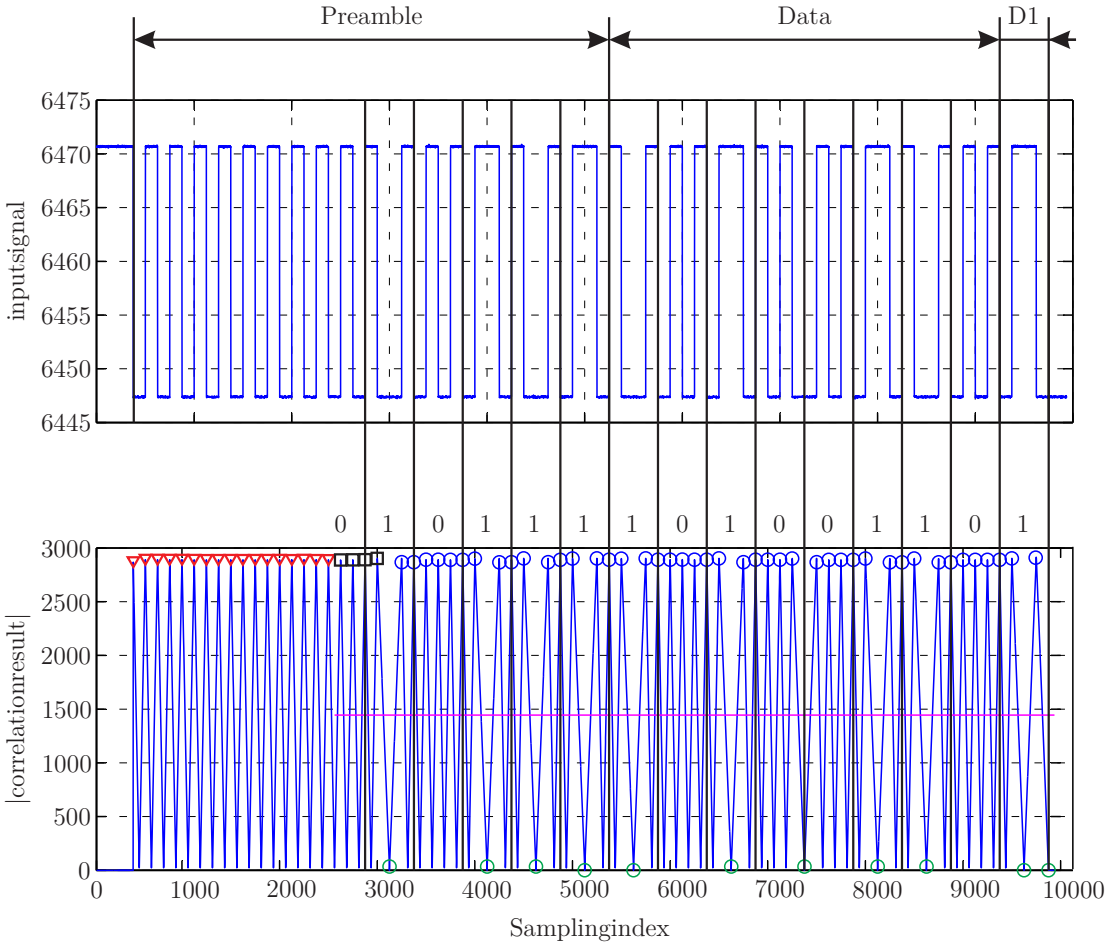

Figure 4.3: Example of a Miller M=2 modulated tag response with the corresponding magnitude of the correlation result

4.3.2 Decoder

The decoder uses the magnitude of the previously mentioned correlator for decoding the data bits transmitted by the tag. The bits are decoded by mapping the time distance between two phase inversions onto received bits. The decoder must be synchronized to the BLF for deciding between a phase inversion and no phase inversion. As already discussed the BLF could have a large frequency offset. Furthermore, a decision threshold must be found since the value of the correlation maxima is dependent on the unknown distance between the constellation points s_1 and s_2 (see figure 4.1).

Therefore, it is necessary to synchronize to the BLF in the beginning of the decoding process. Normally, a Miller modulated tag response is preceded by a preamble. Here, the tag starts by backscattering an unmodulated sub-carrier for a certain period see figure 4.3. Since no phase changes occur (because there is no data modulation) the magnitude of the correlation result shows peaks with a time difference of $1/2f_{\mathrm{BLF}}$ and a certain maximum value. The information about the time difference of the correlation peaks is used to estimate the actual BLF and in a similar manner the threshold is found by taking half of the correlation peak magnitude. To get a more accurate result the average over multiple peaks is calculated. The used peaks are indicated using red triangles in figure 4.3.

In order to keep synchronized to the BLF the sampling points used for the detection of a phase change are derived based on the location of the last peak and the BLF estimated during the preamble. As mentioned before, a phase change is detected when the magnitude of the correlation result is lower than the threshold.

The next step is to find the first bits so that the decoding can begin. Since the first bits are "0" and "1" according to the Miller preamble the decoder waits for this first phase inversion (black squares in figure 4.3). When the first phase inversion is detected, it is assumed that it is the inversion located in the middle of the first "1" bit. Afterwards the decoding operation begins. Based on the number of peaks between the last phase inversion and the actual phase inversion and the knowledge of the last decoded bit the final bitstream can be decoded.

The decoding process is stopped when an unfeasible time distance between two consecutive phase inversions is detected. This can be the end of the message or a decoding error. This is not a problem since the communication is either saved by a CRC checksum or the number of bits to be received is known beforehand because the reader requested a certain amount of data from the tag.

Finally, it is worth to mention that some performance improvements to the decoder would be possible. For example, a recovery from a decoding error would be feasible by checking the phase of the correlation result from the previous and next correlation maximum. Also the synchronization process could probably be improved by not only taking the last peak as reference. Since simulations con-

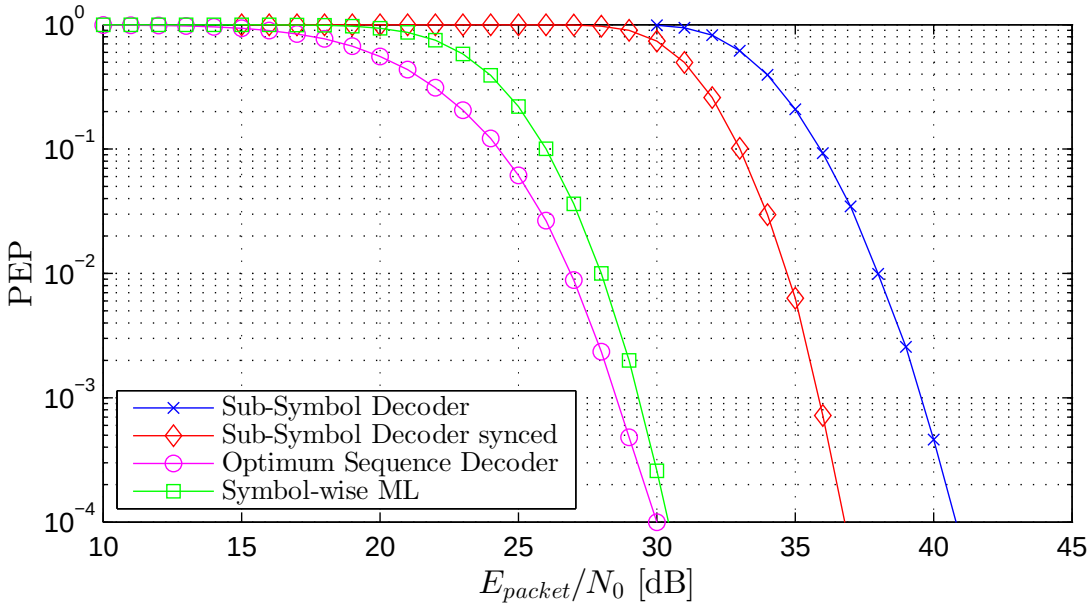

Figure 4.4: Comparison of different decoders decoding an RN16

ducted with measured tag responses, recorded using the SDR platform, showed that the SNR for decoding the tag response is usually large enough when the tag gets enough energy to reply in the downlink channel, no further investigation of the suggested improvements was done during this thesis.

4.3.3 Performance of the Sub-Symbol Decoder

It is obvious that this kind of decoder is not optimum from the point of view of signal and detection. Figure 4.4 shows a comparison of various decoders discussed above. The simulation results present the packet error probability (PEP) for a RN16 response with M=2 over the packet energy to noise ratio. This unusual representation has been chosen to provide a fair comparison between the decoders. Since also the effects of imperfect synchronization of the synchronous sub-symbol decoder should be discussed, it is necessary that the test signal also includes the preamble for synchronization. Therefore, as a standardized case the decoding performance of an RN16 tag response has been compared. All depicted results are simulation results. The synchronous sub-symbol decoder introduced in section 4.3 and implemented in the final design shows a performance difference of about 12 dB compared to the optimum sequence decoder discussed in section 4.2. Since the decoder implemented in this thesis doesn't utilize the memory of the Miller coding, the performance of a symbol-wise maximum likelihood detector was also simulated.

The performance of the implemented decoder is about 10 dB worse compared to the symbol wise maximum likelihood detector. Since this large gap could not be explained well first, a simulation with a perfectly synchronized sub-symbol decoder has been done. This still demonstrates a difference of about 7 dB to the symbol wise ML decoder. Nevertheless, this can easily be explained by three effects. First, the fact that the correlation is only taken with respect to two sub-symbols. Therefore, in the case of M=2 as indicated above the energy available for the correct decision is only half of the total amount when it is compared to the whole Miller symbol. Consequently, 3 dB of decoding performance are lost.

Furthermore, the signal space of receiving a phase change or not can be described by two orthogonal basis functions. Since the correlation is taken only with respect to one of these functions ($g(t)$) another 3 dB performance loss can theoretically be explained. Since the second basis function would not have zero mean value the coupling must be estimated and subtracted to get the correct correlation value. Of course, this would result in an additional complexity of the implemented receiver and has been omitted therefore. Furthermore, for detecting a correct Miller symbol, four correct decisions are needed in advance. This also degrades the performance and explains the last missing gap of 1 dB.

Finally, the performance difference between the perfectly synchronized sub-symbol decoder and the implemented decoder can be explained by the imperfect synchronization algorithm that is typical for any practically realizable solution.

Chapter 5

Implementation within the FPGA

The focus of the following chapter is on the implementation of the RFID interrogator using the National Instruments USRP-2922 SDR platform. In section 5.1 an overview of the implemented design within the FGPA will be presented. Section 5.2 and 5.3 will discuss the implemented user logic used for generating the transmit signal and decoding the backscattered signal in detail.

5.1 Overview

A field-programmable gate array (FPGA) is a user configurable integrated circuit, which is able to process information fully parallel since it consists of many programmable logic blocks, which can be connected via reconfigurable interconnects. Since all these logic blocks work independent of each other, an FPGA is ideally suited for parallel high data rate signal processing and interfacing. The user can configure the behavior of the FPGA by loading a binary file called bitfile, e.g. during the startup procedure. Usually an FPGA has no non-volatile memory inside the package, therefore the configuration data must be stored externally. In the case of the N210 the configuration could be loaded into the FPGA either from an SPI Flash memory or directly from the personal computer and the design environment using a JTAG adapter cable. To generate this bitfile the user describes the wanted behavior by a hardware description language (HDL) or a schematic design. There are different HDL languages available, during this diploma thesis VHSIC hardware description language (VHDL) was used. Since the board utilizes a Xilinx Spartan 3 DSP FPGA the Xilinx ISE (Integrated Software Environment) has been used as the development tool for synthesis, implementation

and simulation of the presented design. The desired parallel computing performance comes at the cost that it needs more implementation effort compared to microprocessors. Furthermore, designs realized within FPGAs are not easily to be debugged compared to microprocessors. Therefore searching for and fixing programming errors is a challenging task.

The use of FPGAs has big advantages when parallel processing is needed, but complex sequential tasks can be realized much more hardware efficient in microprocessor platforms since the used logic resources are reused for multiple tasks. In general, program codes for microprocessors are easier to debug, since the development environment usually offers more debug features in comparison to FPGA tools, e.g. halting the processor when it reaches a certain point in the program. In order to combine the benefits of both concepts a, so called softcore microprocessor which is directly placed in the FPGA is often used. Xilinx ISE features the *MicroBlaze* softcore processor which is fully integrated in the design workflow and is easily configurable.

In the implementation of the RFID reader developed within this thesis such a processor is used for the less time critical tasks like configuration of the RF hardware on the boards or the communication to the PC via the serial interface. Furthermore, it is possible to attach user defined logic written in HDL language to the processor local bus (PLB) for sharing memory resources and reading and writing to registers. Hence, the time critical signal generation and signal detection was realized in a user logic FPGA block, which can then be controlled by the software running on the *MicroBlaze*. This module is called rfiduhf and combines the receiver and transmitter logic.

In figure 5.1 a block diagram is presented which shows all modules within the FPGA design and the main signal flows. The *top_level* contains the three main modules of the design which are the *adc_dac_interface*, the *MicroBlaze XMP* project and the *clk_reset*. Additionally, it shows the connections of the FPGA's internal blocks and the external interfaces.

The *clk_reset* module is connected to the clock distribution IC (AD9510) on the N210 board. The module includes a state machine which is used for the start-up configuration of the clock distribution IC. The implemented procedure is needed since a clock divider by two is programmed on the clock output of the FPGA by default after powering up the system. This would result in an initial FPGA clock of 50 MHz but since the whole design should run on 100 MHz this clock divider must be deactivated. After the initial configuration the FPGA resets itself and from this point on the SPI interface of the AD9510 is forwarded directly to the *MicroBlaze* so that all further configuration can be done via software running on the processor. The *clk_reset* module also includes the digital clock manager which is used to generate the 50 MHz, 100 MHz and 200 MHz system clocks used by other blocks in the overall design.

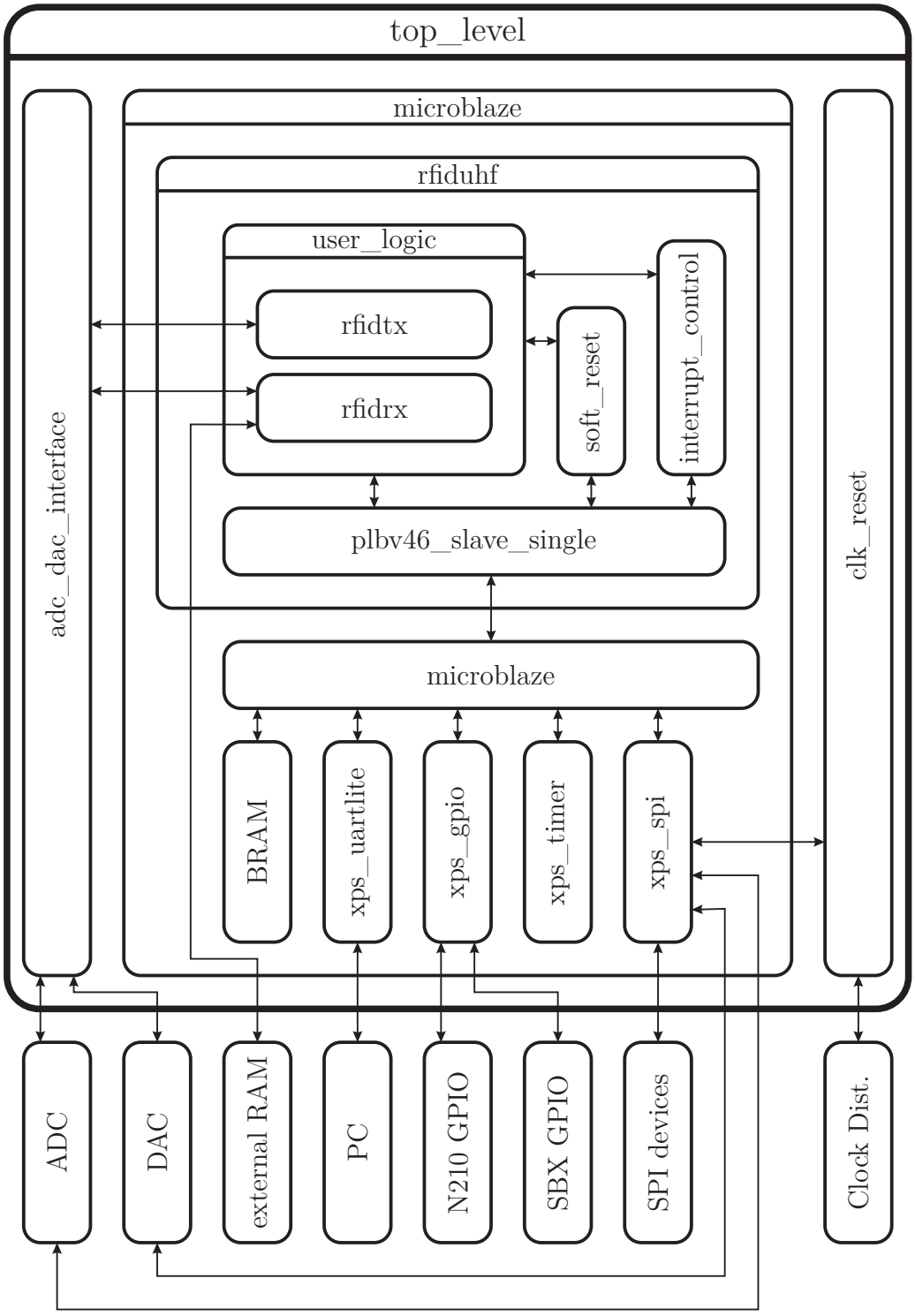

Figure 5.1: Overview of the FPGA implementation

The *adc_dac_interface* contains user logic which is used to guarantee the interface timing required by the external ADC and the DAC so that the subsequent logic can sample this data at the rising edge of the global 100 MHz clock.

The *microblaze* module contains the *MicroBlaze* softcore which is connected to some block RAM (BRAM) memory blocks via the LMB as well as the remaining peripherals connected via PLB to the processor.

The *xps_spi* and *xps_gpio* IP-cores from Xilinx are used for configuring the external hardware like the ADC, DAC, synthesizer, variable attenuators, and clock distribution. *xps_timer* provides basic timer functionality for the *MicroBlaze* and is used to get precise delays which are needed for example for the timing of the RFID protocol. The *xps_uartlite* is used for communication with a PC.

For generating custom peripherals Xilinx provides the *EDK's Create and Import Peripherals Wizard*. This is a tool for generating a framework for custom logic implementations. It was used to generate the template of the *rfiduhf* logic block which consists of four main blocks. The *soft_reset*, the *plbv46_slave_single* and the *interrupt_control* block are used to provide a framework for communication with the PLB and are automatically created by the *EDK's Create and Import Peripherals Wizard*. The *user_logic* block contains all logic needed for pulse shaping, encoding, decoding, and also a module which can be used for recording raw ADC data. The presented *user_logic* block incorporates the *rfidtx* and *rfidrx* blocks which are one of the main achievements of the development process performed within the thesis. Therefore, these will be discussed in detail in the following sections.

5.2 Transmit Module (rfidtx)

In this section the rfidtx module which is used to realize a EPC standard [1] conformal interrogator-to-tag (R⇒T) communication, is discussed. A state-machine capable of producing a PIE DSB-ASK or PIE PR-ASK modulated output signal was designed. Subsequently, this signal is fed to a 51 tap finite impulse response filter (FIR) [12], which is used for pulse shaping purposes in order to accomplish the stringent EPC transmission mask. This signal utilizes a 2 MHz sampling clock which is then resampled to the 100 MHz domain by using a cascaded integrator–comb filter (CIC) [13]. The 2 MHz sampling rate of the state machine and the attached FIR-filter was chosen in order to reduce the required number of taps of the FIR-filter (51) and to provide sufficient time resolution (0.5 µs) to be able to choose different down- and uplink-datarates according to the EPC protocol.

5.2.1 Pulse Generation

It was decided at the beginning of the work which block must be implemented inside the FPGA as dedicated logic and what functionality could be implemented within the *MicroBlaze* processor or on an attached personal computer running e.g. MATLAB. Of course, dedicated logic must be used to generate the input signals for the DAC on the N210 board since the data rate on this interface is too large to be provided directly by the *MicroBlaze* processor. However, the EPC protocol handling can be easily done by the processor. The interface between the user logic and the processor is defined in a way, that the processor writes the data bits to be transmitted into a BRAM which can also be accessed by the user logic. Some registers are used for configuring the user logic, e.g. setting the number of bits to transmit, configuring the PIE timings (Tari, Tone, RTcal, TRcal, PW) and so on. The configuration registers are all well described in appendix A.1.2. By writing a logic 1 to the *start_tx* bit in the register *rfiduhf_txconf1*, the state machine begins to generate the PIE encoded data for the pulse shaping and sets the *busy_tx* bit of register *rfiduhf_txconf1* to logic 1. The *start_bit* is cleared automatically after the *busy_tx* bit is set. When the transmission has finished the state machine clears the *busy_tx* bit indicating that the unit is ready for the next transmission.

Furthermore, in the state machine a feature has been implemented to detect if the R⇒T signaling has to begin with an EPC Preamble or a Frame-Sync. This feature is activated by default, but, it can also be manually chosen if the transmission should start with an EPC Preamble or a Frame-Sync.

Also both checksums, CRC5 and CRC16, are calculated during the transmit operation and can be appended at the end of the data transmission. Either the first bits of the message indicating the type of EPC command are used to determine which checksum should be applied in an automatic mode or otherwise the checksum can be chosen manually.

The *pwr_on_tx* in register *rfiduhf_txconf1* is used to switch on and off the RF carrier. After switching the RF carrier on the state machine ensures that for 1500 µs no R⇒T communication starts. Furthermore, the state machine ensures that the RF carrier cannot be turned on again before it has been turned off for at least a duration of 1 ms. This power-up and power-down timing requirements are implemented according the EPC standard [1].

5.2.2 Pulse Shaping

On the one hand the transmit pulse has to be filtered to generate an output signal compliant to the spectral transmission mask specified in the EPC standard [1] and ETSI norm [4]. On the other hand, timing requirements listed in the EPC

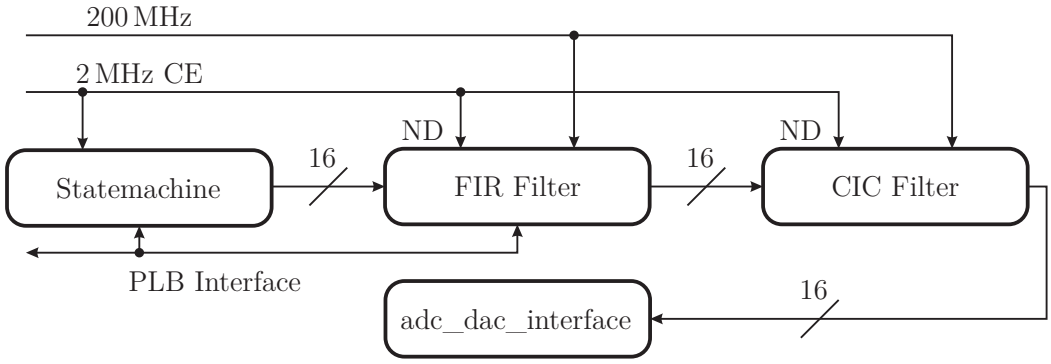

Figure 5.2: TX module overview

standard have to be accomplished. Consequently, the filter design process is a critical task. In order to gain some flexibility for pulse shaping purposes a reconfigurable FIR filter is implemented. The length of the filter is chosen based on the maximal length of a data-0 symbol (25 µs). Furthermore, this is the reason for the reduction of the sampling rate of the FIR filter and the state machine. A sampling rate of 100 MHz would lead to a filter using 2500 taps. For example the implementation of the filter in the FPGA would result in the need of 158 DSP slices while the used FPGA offers 126 DSP slices at all. In order to provide enough free resources for future extension of the FPGA RFID reader design the sampling rate of the FIR Filter is chosen to be 2 MHz. This reduces the required number of taps to 50 but also limits the time resolution of the state machine to 0.5 µs.

The FIR filter block can be clocked by a frequency larger than the sampling frequency, in order to save resources. A 200 MHz clock rate is used and only two DSP slices and two BRAMs are needed which seems to be reasonable for this task. A CIC filter is used to resample the signal to the 100 MSamples/s domain which is the sampling rate of the DAC. The CIC filter uses again three DSP slices. Figure 5.2 shows an overview of the connections of the filters. The FIR filter coefficients used for pulse shaping can be reconfigured by a configuration register. The usage of this configuration register is explained in appendix A.1.7.

5.3 Receive Module (rfidrx)

In this section the rfidrx module which is used to demodulate and decode the data backscattered by the tag will be discussed. The decoder implementation is based on the synchronous sub-symbol decoder discussed in section 4.3. This module consists of three state-machines each responsible for a different task. Figure 5.3 gives an overview of the module. The debug block provides a signal

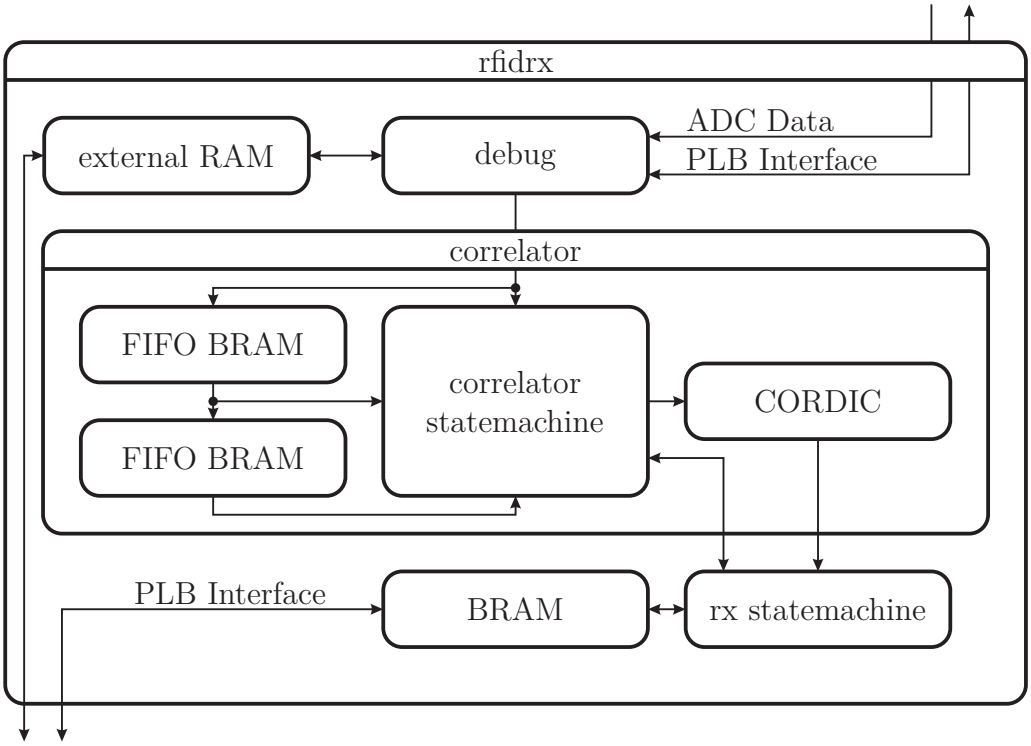

Figure 5.3: RX module overview

capturing and playback functionality used for analysis and tests of the module. The correlator block calculates the correlation of the ADC signal and one period out of a square wave with frequency equal to the expected BLF (see section 4.3.1). The magnitude of the correlation result is then used by the decoding state machine to estimate the actual backscatter-link frequency (BLF) of the tag and to decode the backscattered bits of the tag response. As already stated in chapter 4 the implemented receiver is not optimal in the sense of an optimum sequence decoder or matched filter. The advantages of the implemented receiver are that it is easy to implement, uses not much logical resources on the FPGA and it provides the data of the sub-symbol edges which is mandatory for the planned future implementation of an additional localization algorithm [3]. In the following sections all these blocks will be discussed in detail.

5.3.1 Correlator

In this block the cross correlation between the ADC data received from the debug block and one period out of a square wave with frequency equal to the expected BLF is computed. A correlator can easily be built using an FIR filter, but. this

would lead to the same problems as discussed before in section 5.2.2 where a FIR filter is used for pulse shaping purposes. It is one of the main goals of the design process of the RFID reader not to reduce the sample rate because of the planned future implementation of a localization algorithm. Consequently, another solution has to be found. An iterative algorithm has been found during the work on this thesis which bases on the fact that the signal $g[n]$ is one period out of a square wave. Therefore it is possible to calculate the correlation of this signal $g[n]$ with the ADC data $f[n]$ by

$$g[n] = \begin{cases} 1, & -a < n \leq 0 \\ -1, & 0 < n \leq a \\ 0, & otherwise \end{cases} \tag{5.1a}$$

$$(f \star g)[n] = (f \star g)[n-1] - f[n-a] + 2f[n] - f[n+a] \tag{5.1b}$$

where

$$a = \left\lceil \frac{f_{sample}}{2f_{\mathrm{BLF}}} \right\rfloor. \tag{5.2}$$

It can be seen from equation 5.1b that the cross-correlation can be calculated by adding four values and a simple multiplication by a factor of two. For the summation the previous correlation value, the actual ADC value, the ADC values delayed by a and $2a$ are needed. The delay is realized by BRAMs used as circular buffers. The multiplication by the factor of two can be realized as a simple shift-operation on the binary integer data. As demonstrated, a iterative calculation is possible in real-time using the FPGA of the SDR platform. The remaining problem is to find a suitable starting value for the correlator. This is solved by setting the correlator value to zero and turning on the used adders in a sequence according to equation 5.3 by the correlator state machine.

$$(f \star g)[n] = \begin{cases} 0, & n = 0 \\ (f \star g)[n-1] - f[n+a], & 0 < n \leq a \\ (f \star g)[n-1] + 2f[n] - f[n+a], & a < n \leq 2a \\ (f \star g)[n-1] - f[n-a] + 2f[n] - f[n+a], & 2a < n \end{cases} \tag{5.3}$$

After changing the period length a the correlator state machine has to be reset. Afterwards the output of the correlator stage is invalid for $2a$ samples. The correlation is done in the I and Q channel separately. The I and Q correlation results are then processed by a coordinate rotation digital computer (CORDIC) core to obtain the magnitude and angle of the correlation result. A CORDIC is an efficient algorithm to implement mathematical functions, e.g. trigonometric functions. Xilinx ISE provides a CORDIC IP core which was used for the implementation [14].

5.3.2 Decoder

The decoder converts the results of the correlator module to data bits, which are afterwards stored in a BRAM which is accessible also from the *MicroBlaze*. Before starting the decoder the configuration registers must be set to the desired values. The decoder is started by writing a logic one to the start bit in the configuration register (see Appendix A.1.11). After the start the decoder state machine resets the correlator and waits until the correlation values become valid as discussed above. Afterwards it waits until the magnitude of the correlation result reaches a certain threshold set in the configuration register to advance to the next state where the actual BLF and the magnitude of the correlation peaks are estimated. This is done by finding and storing the maximum values of the magnitude of the correlation result. The values of the correlation peaks are added up during the Miller Preamble. The result is then divided by the number of peaks which gives the average value of the maxima. The average is an indication of the signaling power of the tag and is made available to the processor via a register for further utilization. The second result of the processed preamble is an estimate of the BLF, which is calculated by the number of samples between the first and the last observed peak divided by the number of gaps between the peaks. Again this value is accessible to the processor via a register.

Subsequently, the state machine starts decoding the actual tag message by waiting for the first two sub-symbols without modulation state inversion in between them which indicates the first logic one of the preamble and thus the start of the tag data to be decoded. This is done by comparing the correlation magnitude between two sub-symbols with half the value of the average peak value estimated before. When the first bit is found the state machine evaluates the interval between the sub-symbols without modulation state inversion and uses this information to decode the backscattered data bits which are written to a BRAM accessible by the processor. If there is any unexpected interval detected the state machine switches its state to idle and clears its busy flag. Decoding is stopped and the end of the tag message is found. The processor can read the received data as well as the estimated BLF, the estimated signal power, and the number of received bits.

5.3.3 Debug

The design of the receiver also includes a debug module which is placed in between the incoming ADC data and the correlator module. Either, it can be used to capture live data or to test the correlator and decoder user logic by playing back some stored signals. The debug module uses the external RAM on the N210 board and provides a memory depth of 256 kSamples. Keeping in mind the sample rate

of 100 MSamples/s the maximal duration of one recording is 2.6 ms. Because longer record and playback durations might be useful a simple sampling clock divider is incorporated. It is important to notice that this feature doesn't apply any filtering technique to suppress aliasing distortion. The external memory is not dual ported. Thus, only one read or write operation can be performed at a time and the memory is only accessible via the LMB when the debug module is in the initial state.

Furthermore, the module has some triggering functionality from the processor as well as directly from the user logic. It can be operated in four modes:

- The **single capture** mode waits for a trigger event and records one shot afterwards, then it goes back to the initial state.
- The **continuous capture** mode records the ADC data using the RAM as a circular buffer. When recording is stopped by a trigger event the state machine goes back to the initial state and the current address of the circular buffer can be read out from a register.
- The **single playback** mode waits for a trigger event and plays out the stored data one time afterwards. During the waiting time for the trigger the data stored in the default register is output.
- The **continuous playback** works like the single playback with the difference that the playback is not stopped after one cycle. The continuous playback is stopped by a reset of the module.

Chapter 6

Phase Noise

Since the USRP-2922 is intended to be a versatile SDR it features dual-band operation, meaning that it is possible to transmit and receive on different frequencies. Because of that fact the SDR has two independent synthesizers for TX- and RX-LO generation. But, in an RFID reader the RX and TX frequency are the same since backscattering is used for the T⇒R communication. Therefore, the two synthesizers have to be tuned to the same frequency. Since the reference clock of both synthesizers is the same, their outputs are phase coherent. But the inherent phase noise of the two PLLs is uncorrelated. This causes distortion of the received signal by the phase noise.

A regular RFID reader would use one LO for up and down conversion so that this problems would not be encountered. Utilizing the same oscillator for TX and RX is only possible when some hardware changes are made. Section 6.1 discusses the hardware changes made and section 6.2 shows measurement results.

6.1 Hardware Changes

A short look in the documentation of the SDR shows that it is not possible to use one oscillator for RX and TX path just by a software reconfiguration. Consequently, some hardware modifications had to be done. Two RF-Baluns (T5 and T6 according to the schematic of the SBX [8]) were resoldered with SMA pigtails attached to them. These SMA connectors were mounted on the back plane of the SDR (see figure 6.1) allowing flexible external configuration of the LO signals. Figure 6.2 shows the logical points where the external wiring was attached to.

The four SMA connectors on the backplane (see figure 6.1 and table 6.1) now allow many different configurations:

Figure 6.1: Picture of the hardware modification of the USRP

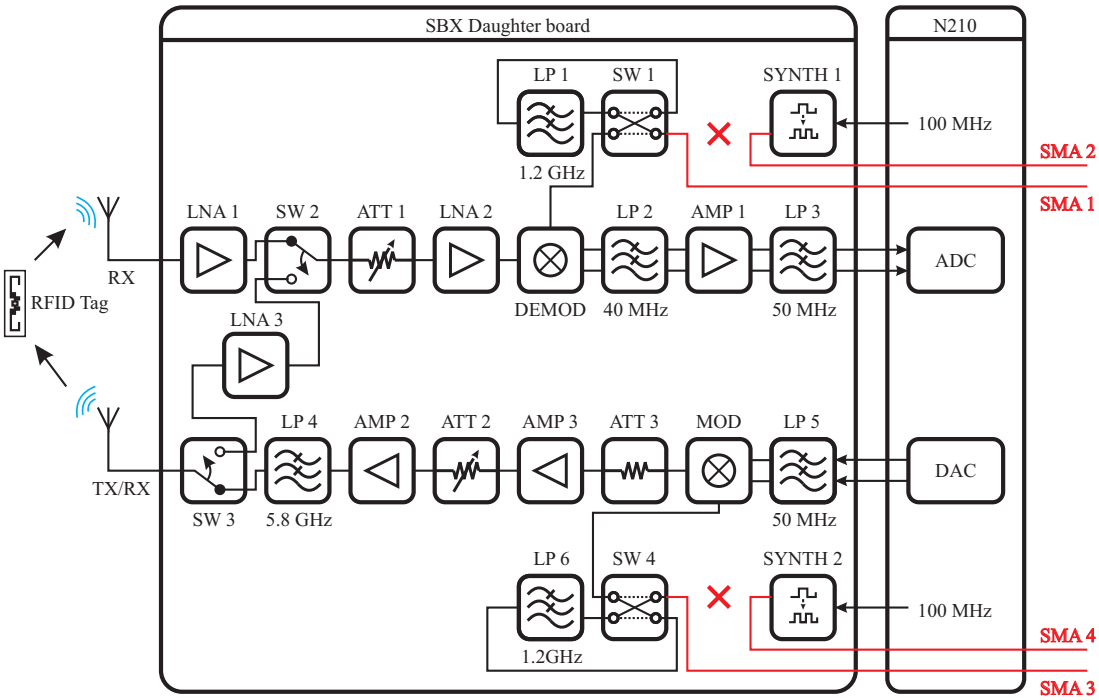

Figure 6.2: Hardware changes on the SBX daughter board to allow flexible LO configuration

SMA 1	Demodulator input LO RX
SMA 2	Synthesizer output LO RX
SMA 3	Modulator input LO TX
SMA 4	Synthesizer output LO RX

Table 6.1: List of connections on the back plane of the modified N210

- By using two SMA jumpers ($2 \rightarrow 1$) and ($4 \rightarrow 3$) the normal operation (two independent synthesizers) of the USRP can be rebuilt.
- By connecting one of the two synthesizer outputs (2 or 4) to an external power divider and connecting its output to the modulator inputs (1 and 3) one could use one synthesizer to drive both LO inputs. The other free synthesizer output should of course be terminated properly into a $50\,\Omega$ load. The result is phase coherence between the RX and TX without additional phase noise. The synthesizers on the SBX have programmable RF output power levels ($-4\,\mathrm{dBm}$ to $5\,\mathrm{dBm}$). Therefore, a passive $3\,\mathrm{dB}$ power divider can be used to resolve the problems caused by the phase noise.
- Another possibility is to drive both LO inputs (1 and 3) using an external RF source. The input drive levels of the modulator and demodulator have to be respected ($-6\,\mathrm{dBm}$ to $6\,\mathrm{dBm}$) in order not to damage the hardware. This also allows phase coherence between multiple USRPs. This also opens the opportunity to use this SDR platform in a MIMO reader scenario.

The presented list demonstrates various opportunities achieved by this hardware modification, but is not meant to be complete.

6.2 Measurement Results

In order to visualize the problem and to demonstrate that the hardware changes solved the phase noise problem, measurements have been done. The SDR was configured to transmit a CW carrier at the desired measurement frequency (e.g. $865.7\,\mathrm{MHz}$). The TX-port of the SDR was connected to the RX-port via a carefully chosen attenuator such that the ADC isn't overdriven. Samples have been recorded at full sampling rate using the *debug module* introduced in section 5.3.3.

In order to illustrate the problem the sampled data was post processed and normalized. Figure 6.3 illustrates the coordinate system transformation which was used to split the overall noise into a phase dependent part $\vec{e_\varphi}$ and an amplitude dependent part $\vec{e_r}$.

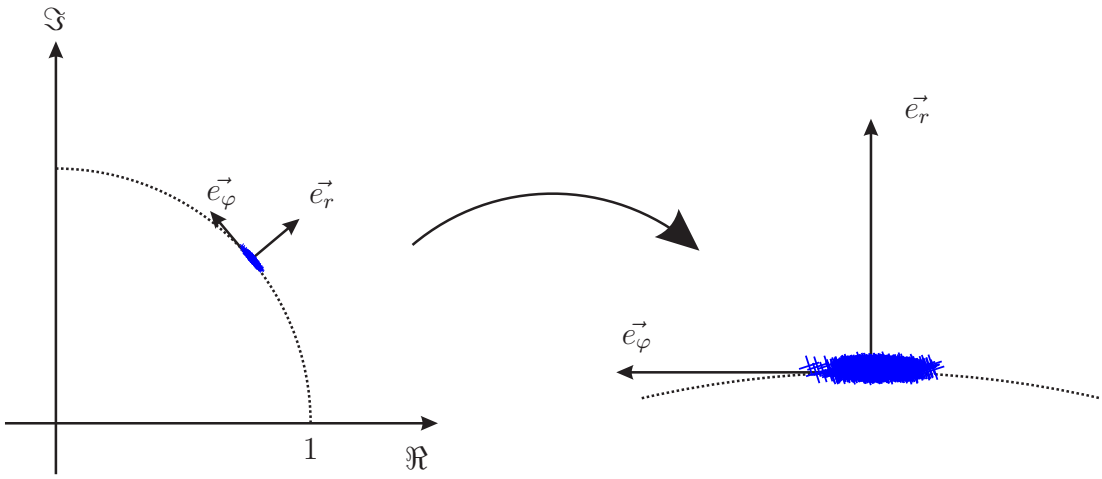

Figure 6.3: Coordinate transformation used to split the overall noise into an phase dependent part and an amplitude dependent part

Figure 6.4 shows the histograms of the distribution of the recorded data without hardware changes. The right histogram shows the distribution in $\vec{e_r}$ direction, the left in $\vec{e_\varphi}$ direction. It can be clearly seen that the distribution is much broader in the $\vec{e_\varphi}$ direction due to the inherent influence of the phase noise.

Figure 6.5 as a comparison shows the measurement results after the applied hardware modification, where the TX synthesizer output has been connected to an external power divider which splits the oscillator signal for the up- and down-converter. For the synchronized case, the measurement results show that the distributions in $\vec{e_r}$ and $\vec{e_\varphi}$ are nearly the same. Therefore, a short look into the measurement results demonstrates that the phase noise problem was solved by moderate hardware changes to the SBX daughter board.

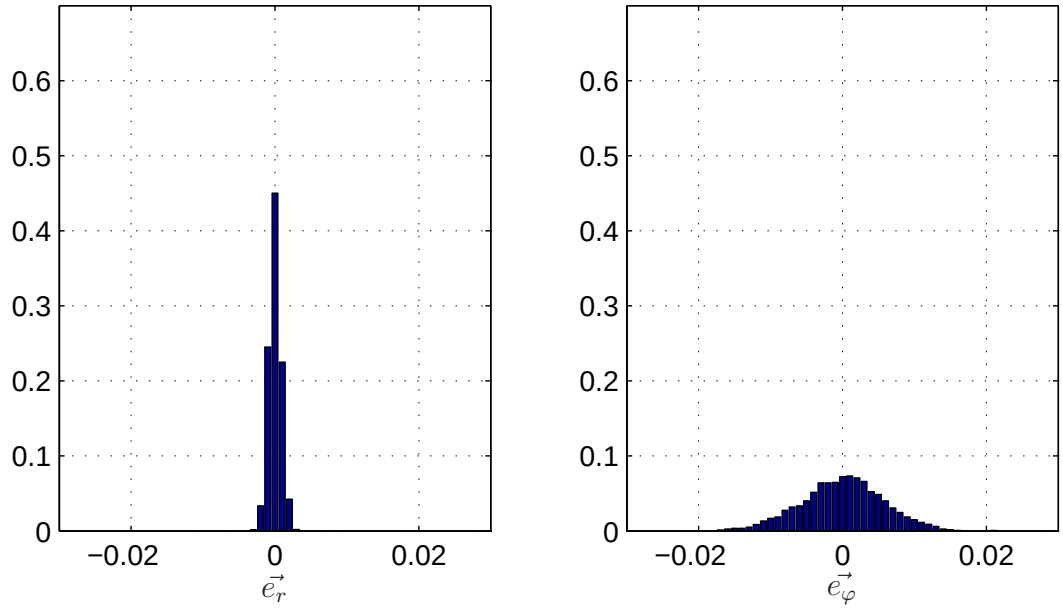

Figure 6.4: Measurement results without hardware modification

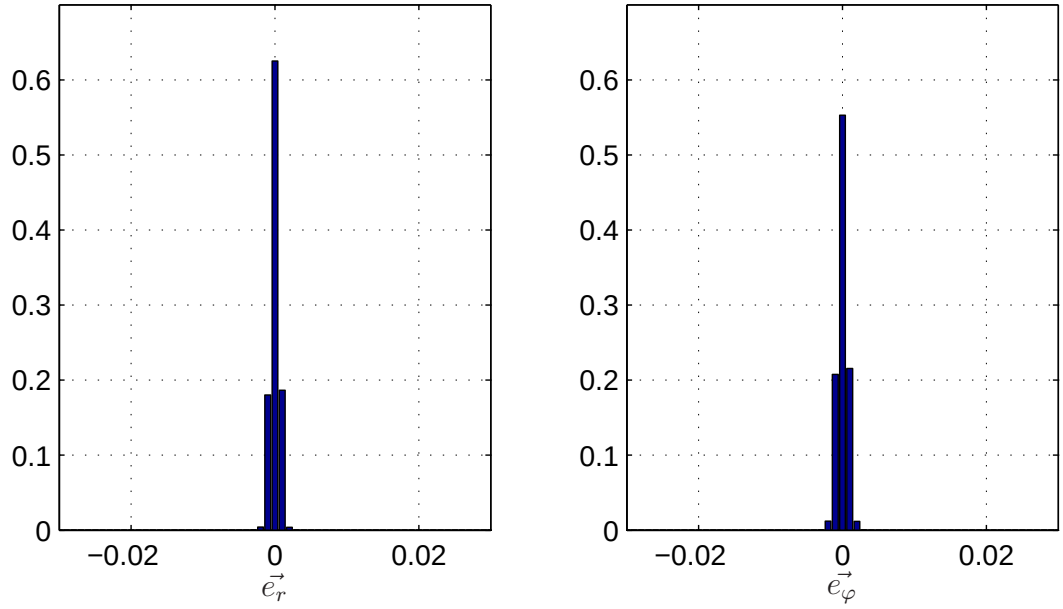

Figure 6.5: Measurement results in the synchronized case

Chapter 7

Verification of the Implementation

The implemented UHF EPC reader design was tested using commercially available Philips (NXP) UCODE EPC G2 V4 tags. Figure 7.1 shows the used single antenna setup where a directional coupler is used for decoupling the transmit and receive path. Tests demonstrate that reading out the stored EPC of an EPC tag is possible within the whole range of standardized BLF frequencies. Figure 7.2 depicts the recorded ADC Data of such a successful EPC readout at the maximum BLF (640 kHz). The reading range of this setup has been compared to a commercially available chip based RFID reader evaluation board from austriamicrosystems (AS3992)[15] connected to the same antenna.

The conducted measurements shows that the reading range of both readers is approximately the same since the SNR of the back scatted signal seems to be large enough for successful fault free decoding as long as the tag gets enough energy to be powered up.

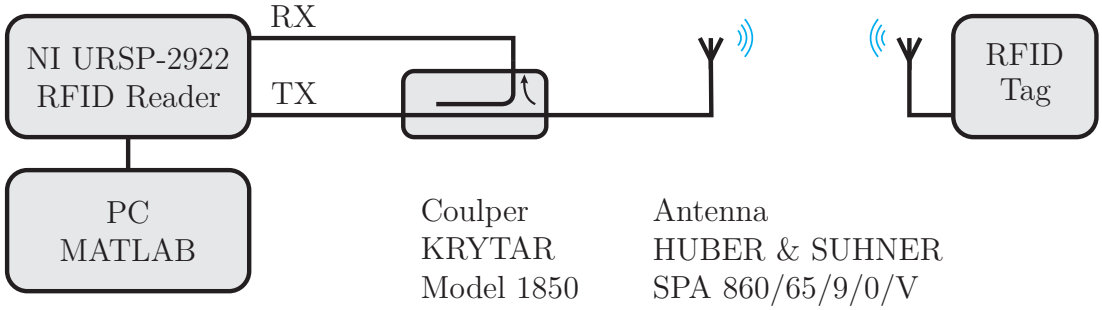

Figure 7.1: Measurment setup for verification of the system

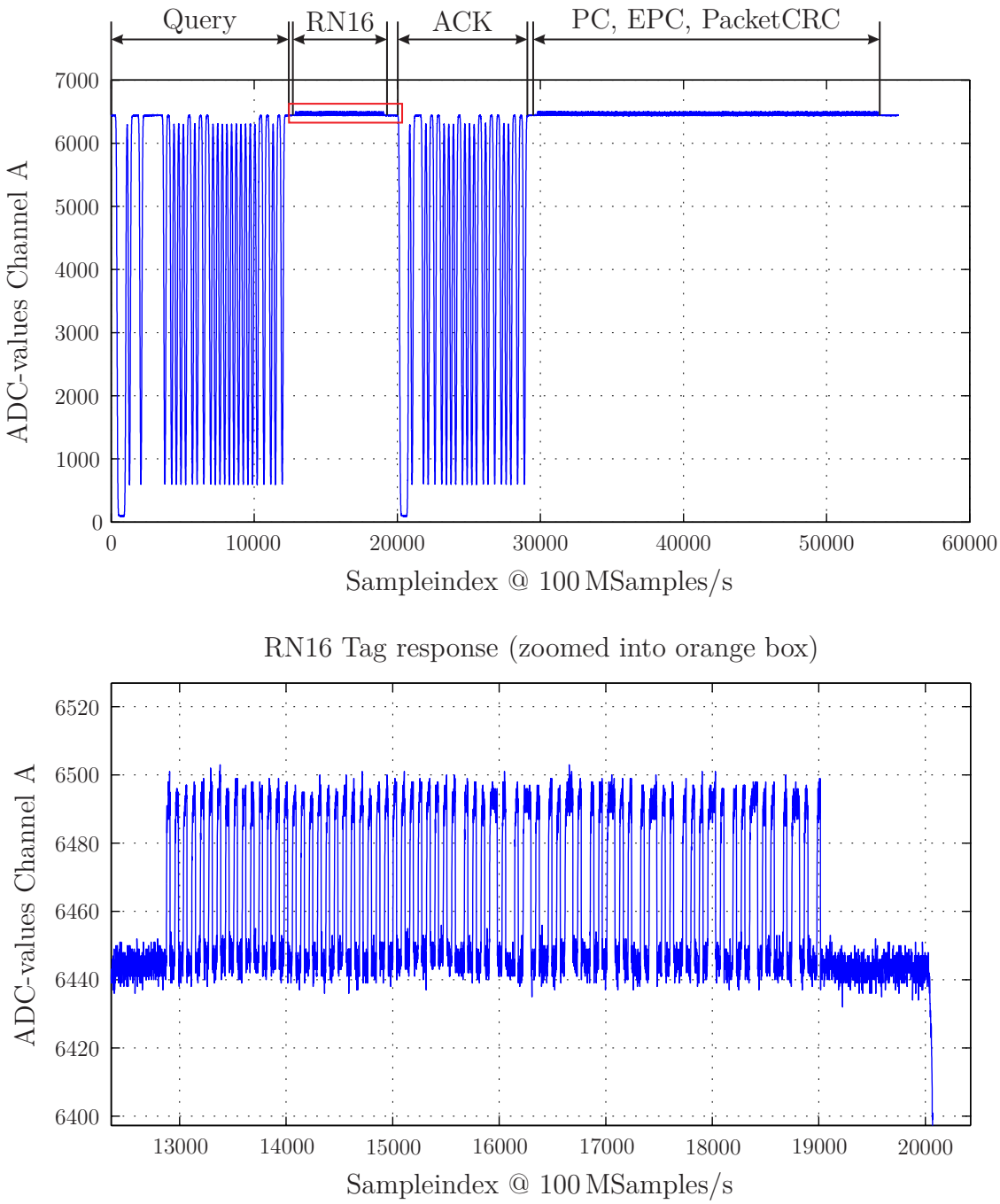

Figure 7.2: ADC recording of a successful readout of an EPC Tag

Chapter 8

Conclusion and Outlook

During the work on this thesis an RFID reader for EPC UHF tags was built based on an off-the-shelf available Software Defined Radio platform. The reader was designed to provide a flexible test platform for RFID localization for future investigations within the project REFlex (project number: 845630) funded by the Österreichische Forschungsförderungsgesellschaft mbH (FFG).

Therefore, special requirements for a localization algorithm introduced in [3] were included during the development of the reader. The ranging algorithm, which is planned to be implemented on the reader, uses low spectral power density broadband signals. Therefore a high sampling rate and resolution is desirable. The hardware of the USRP-2922 features a 14-bit ADC and a 16-bit DAC with a maximum sampling rate of 100 MSamples/s. Unfortunately the bit rate provided by the Gigabit Ethernet interface is too low to stream the sampling data with full sampling rate and resolution to a personal computer to perform the signal processing there. Therefore, if the SDR is used with the provided driver framework of the manufacturer, the sampling rate and or the resolution is reduced in the SDR and consequently the hardware capabilities cannot be fully exploited. Since a reduction of the sampling rate and resolution would considerably limit the performance for the localization algorithm, a custom FPGA implementation was developed during the work on this thesis which doesn't suffer from these problems.

The implementation of the RFID reader developed within this thesis uses a combination of a soft microprocessor and custom user logic attached to that processor. These custom user logic modules combine the receiver and transmitter logic needed for UHF RFID.

A decoder for Miller modulated tag responses was developed, simulated, implemented within custom user logic, and also tested with commercially available

tags. The decoder is suboptimal from the point of view of signal and detection algorithms, however, from the perspective of practical realizability this decoder has only very low hardware complexity, such that the FPGA has enough free hardware resources available for the localization algorithm. Furthermore, the implemented decoder offers precise timing information needed for the localization algorithm.

Because the RFID reader provides the basis for a future research project (REFlex/FFG), the next steps of a future work will be the implementation of the localization algorithm. Furthermore, with the help of the reader realized within this thesis research of MIMO RFID readers is possible since the USRP-2922 Platform offers synchronization possibility among multiple units.

References

[1] *EPC Radio-Frequency Identity Protocols Generation-2 UHF RFID*, GS1 EPCglobal Inc. Std., Rev. V 2.0.0, 2013. [Online]. Available: http://www.gs1.org/epcglobal

[2] "SL3ICS1002/1202 UCODE G2XM and G2XL product datasheet," NXP Semiconductors, 2013. [Online]. Available: http://www.nxp.com/documents/data_sheet/SL3ICS1002_1202.pdf

[3] Thomas I. Faseth, "Wireless Localization for Intelligent Transport Systems," Ph.D. dissertation, Vienna University of Technology, 2012.

[4] *Electromagnetic compatibility and Radio spectrum Matters (ERM); Radio Frequency Identification Equipment operating in the band 865 MHz to 868 MHz with power levels up to 2 W*, ETSI Std., Rev. V 1.4.1, 2011. [Online]. Available: http://www.etsi.org

[5] "USRP N200/N210 networked series datasheet," Ettus Research, 2012. [Online]. Available: https://www.ettus.com

[6] M. Buettner and D. Wetherall, "A software radio-based UHF RFID reader for PHY/MAC experimentation," in *in Proc. IEEE Int. Conf. RFID*, April 2011, pp. 134–141.

[7] "N210 schematic," Ettus Research, 2012. [Online]. Available: http://files.ettus.com/schematics/n200/n2xx.pdf

[8] "SBX, 400MHZ-4.4GHZ Transceiver schematic REV 8," Ettus Research, 2013. [Online]. Available: http://files.ettus.com/schematics/sbx/SBX.pdf

[9] "Spartan-3 Generation Configuration User Guide - UG332," Xilinx, Inc., 2009. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug332.pdf

[10] J.-P. Curty, M. Declercq, C. Dehollain, and N. Joehl, *Design and Optimization of Passive UHF RFID Systems*. New York, NY: Springer, 2007.

[11] A. F. Molisch, *Wireless communications, secound edition.* New York, NY: IEEE Wiley, 2011.

[12] "IP LogiCORE FIR Compiler v5.0 - DS534," Xilinx, Inc., 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/fir_compiler_ds534.pdf

[13] "LogiCORE IP CIC Compiler v2.0 - DS613," Xilinx, Inc., 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/cic_compiler_ds613.pdf

[14] "LogiCORE IP CORDIC v4.0 - DS249," Xilinx, Inc., 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/cordic_ds249.pdf

[15] "AS3992 UHF Reader IC product datasheet," ams AG. [Online]. Available: https://www.ams.com/eng/Products/UHF-RFID/UHF-RFID-Reader-ICs/AS3992

Appendix Chapter A

RFIDUHF Registers and Memories

The following section describes the via LMB accessible registers and memories of the *rfiduhf* userlogic implemented during the course of this thesis.

A.1 Register Descriptions

The addresses of the rfiduhf user logic registers are shown in table A.1. The registers are used for configuration and control of the *rfiduhf* user logic from the *MicroBlaze*.

Register Name	PLB Address
rfiduhf_adc_dcm	C_BASEADDR + 0x00
rfiduhf_txconf1	C_BASEADDR + 0x04
rfiduhf_txconf2	C_BASEADDR + 0x08
rfiduhf_txconf3	C_BASEADDR + 0x0C
rfiduhf_txconf4	C_BASEADDR + 0x10
rfiduhf_txconf5	C_BASEADDR + 0x14
rfiduhf_coefrld	C_BASEADDR + 0x18
rfiduhf_rxdebug_conf	C_BASEADDR + 0x1C
rfiduhf_rxdebug_counter	C_BASEADDR + 0x20
rfiduhf_rxdebug_default	C_BASEADDR + 0x24
rfiduhf_rxconf1	C_BASEADDR + 0x28
rfiduhf_rxconf2	C_BASEADDR + 0x2C
rfiduhf_rxconf3	C_BASEADDR + 0x30
rfiduhf_rxconf4	C_BASEADDR + 0x34

Table A.1: Register address map

A.1.1 rfiduhf_adc_dcm

Bits	Name	Description	R/W
0:7	version_fpga *unsigned*	Version number of the FPGA hardware	R
8:29	reserved		
30	dcm_adc_locked	Connected to the LOCKED status output of the DCM for the ADC interface. **0**: DCM is not locked **1**: DCM is locked	R
31	dcm_adc_reset	By writing a logic 1 the DCM for the ADC interface is reset. This bit is self clearing.	R/W

Table A.2: rfiduhf_adc_dcm register

A.1.2 rfiduhf_txconf1

Bits	Name	Description	R/W
0:7	reserved		
8	busy_tx	The busy flag is set when the TX statemachine is busy, e.g. it's sending out data.	R
9	start_tx	Writing a logical 1 to this bit enables the TX process with the set parameters. It's reset automatically when the busy flag is 1.	R/W
10	modulation_tx	The modulation bit selects between the two implemented modulation formats: **0**: ASK-PIE **1**: PR-PIE	R/W
11	pwr_on_tx	Used for switching the RF carrier: **0**: RF-carrier off **1**: RF-carrier on	R/W
12:13	crc_tx *unsigned*	Defines the type of CRC appended: **0**: auto CRC **1**: CRC5 **2**: CRC16 **3**: no CRC	R/W
14:15	preamble_tx *unsigned*	Defines if the message should be preceded by a preamble or a frame sync. The automatic mode inspects the first bits of the message to detect a *Query* command. If a *Query* is detected it starts with a preamble otherwise with a frame sync. **0 or 1**: automatic mode **2**: Frame-Sync **3**: Preamble	R/W
16:29	lenght_tx *unsigned*	Sets the number of bits to be sent.	R/W
30:31	reserved		

Table A.3: rfiduhf_txconf1 register

A.1.3 rfiduhf_txconf2

Bits	Name	Description	R/W
0:15	tone_tx *unsigned*	length of a PIE coded 1 in 0.5 µs steps, e.g. for 10 µs the value should be 20	R/W
16:31	tari_tx *unsigned*	length of a PIE coded 0 in 0.5 µs steps	R/W

Table A.4: rfiduhf_txconf2 register

A.1.4 rfiduhf_txconf3

Bits	Name	Description	R/W
0:15	trcal_tx *unsigned*	length of the TRcal period in 0.5 µs steps	R/W
16:31	rtcal_tx *unsigned*	length of the RTcal period in 0.5 µs steps	R/W

Table A.5: rfiduhf_txconf3 register

A.1.5 rfiduhf_txconf4

Bits	Name	Description	R/W
0:15	txpwr_tx *signed*	Used to set the input values to the FIR filter generated by the PIE encoding state machine. For full scale output set the value to $2^{31} / \sum$ filter coefficients.	R/W
16:31	pw_tx *unsigned*	length of the PW period in 0.5 µs steps	R/W

Table A.6: rfiduhf_txconf4 register

A.1.6 rfiduhf_txconf5

Bits	Name	Description	R/W
0:15	dacb_offset_tx *signed*	Offset compensation for DAC channel B	R/W
16:31	daca_offset_tx *signed*	Offset compensation for DAC channel A	R/W

Table A.7: rfiduhf_txconf5 register

A.1.7 rfiduhf_coefrld

The *rfiduhf_coefrld* register is used to reconfigure the FIR filter coefficients. Listing A.1 shows an example.

Bits	Name	Description	R/W
0:15	nextcoef *signed*	Register for reloading the filter coefficients of the FIR filter. Connected to COEF_DIN of the FIR filter[12]	W
16	coefwe	Writing a logic 1 to this bit, generates a pulse on the COEF_WE of the FIR filter [12] for a 200 MHz clock cycle. Afterwards the bit is reset to logic 0 automatically.	W
17	coefld	Writing a logic 1 to this bit, generates a pulse on the COEF_WE of the FIR filter [12] for a 200 MHz clock cycle. Afterwards the bit is reset to logic 0 automatically.	W

Table A.8: rfiduhf_coefrld register

```
void g_rfiduhf_writefiltercoef(u16 filtercoef[]){
  u8 i;

  rfiduhf_coefrld = (1<<coefld); //start

  for(i=0;i<51;i++){
    rfiduhf_coefrld = (1<<coefwe)|((u32)filtercoef[i]);
  }
}
```

Listing A.1: Example for reloading the filter coefficients

A.1.8 rfiduhf_rxdebug_conf

Bits	Name	Description	R/W
1	waittrigger	Indicates that the debug state machine is waiting for a trigger event	R
2	busy_debug	Indicates that the debug state machine is not in the initial state	R
3	trigger_imterm	Writing a logic 1 initiates a trigger event. This bit is self clearing.	W
4	reset_debug	Writing a logic 1 resets the debug state machine to the initial state. This bit is self clearing.	W
5	start_debug	Writing a logic 1 starts the debug state machine. This bit is self clearing.	W
6:7	mode_debug *unsigned*	Sets the mode of the debug module: **0**: continues record **1**: continues playback **2**: single record **3**: single playback	R/W
8:11	trigger_mask	Sets the trigger mask for the hardware triggers	R/W
12:15	reserved		
16:31	clk_div_debug *unsigned*	Sets the clock divider of the debug unit. Note: no filtering is applied to suppress aliasing distortion.	R/W

Table A.9: rfiduhf_rxdebug_conf register

A.1.9 rfiduhf_rxdebug_counter

Bits	Name	Description	R/W
0:17	countervalue *unsigned*	Can be read out in the initial state to get a pointer to the last written RAM address during recording (used after stopping continues capturing).	
18:31	reserved		

Table A.10: rfiduhf_rxdebug_counter register

A.1.10 rfiduhf_rxdebug_default

Bits	Name	Description	R/W
0:3	reserved		
4:17	adca_default *signed*	Default play out value ADC A	
18:31	adcb_default *signed*	Default play out value ADC B	

Table A.11: rfiduhf_rxdebug_default register

A.1.11 rfiduhf_rxconf1

Bits	Name	Description	R/W
0	reserved		
1	reset_rx	Writing a logic 1 resets the state machine of the receiver to the initial state. This bit is self clearing.	R/W
2	start_rx	Writing a logic 1 starts the state machine of the receiver. This bit is self clearing.	R/W
3	trext_rx	Set this bit to logic 1 when the tag response is preceded by a extended preamble	R/W
4:5	m_rx *unsigned*	Miller modulation format of the tag response: **0**: reserved **1**: M=2 **2**: M=4 **3**: M=8	R/W
6:31	threshold_rx *signed*	Sets the threshold value for detection of the start of the tag response.	R/W

Table A.12: rfiduhf_rxconf1 register

A.1.12 rfiduhf_rxconf2

Bits	Name	Description	R/W
0:15	lencorrpulse_rx *unsigned*	Set this register to length of a half correlation pulse in 10 ns steps. $\frac{\text{TRcal} \cdot 100\,\text{MHz}}{2\text{DR}}$ (A.1)	R/W
16:31	bfl_rx *unsigned*	Set this register to the length of a BLF period in 10 ns steps. $\frac{\text{TRcal} \cdot 100\,\text{MHz}}{\text{DR}}$ (A.2)	R/W

Table A.13: rfiduhf_rxconf2 register

A.1.13 rfiduhf_rxconf3

Bits	Name	Description	R/W
0:4	reserved		
5	busy_rx	A logic 1 indicates that the RX state machine is not in the initial state.	R
6:31	pwrestimate_rx *signed*	Can be read out after receiving a message indicating the average value of the correlation peaks during the first part of the preamble.	R

Table A.14: rfiduhf_rxconf3 register

A.1.14 rfiduhf_rxconf4

Bits	Name	Description	R/W
0:15	bitsread_rx *unsigned*	Indicates the number of received bit by the module after reception is finished.	R
16:31	bflestimate_rx *unsigned*	Can be read out after receiving a message indicating the estimated length of a BLF period in 10 ns steps, during the first part of the preamble.	R

Table A.15: rfiduhf_rxconf4 register

A.2 Memory Descriptions

Table A.16 lists the from the *MicroBlaze* accessible memories of the *rfiduhf* user logic. The *bram_tx* memory is used to configure the data to be sent with the *rfiduhf* block. The *rfiduhf* user logic stores in the *bram_rx* memory the received data to be read from the *MicroBlaze*. The *ram_debug* memory is used for reading out the by the debug module captured data or for storing the data to be played out by the debug module. The *ram_debug* memory can only be accessed when the debug state machine is in the initial state ($busy\_debug = 0$).

Memory Name	PLB Base Address	Size
bram_tx	C_MEM0_BASEADDR	2 kB
bram_rx	C_MEM1_BASEADDR	2 kB
ram_debug	C_MEM2_BASEADDR	1 MB

Table A.16: Memory address map

Appendix Chapter B

Example Code

The following section presents a code example to read out the EPC of a single Tag in the reading range. Section B.1 shows an example configuration of the hardware platform. Section B.2 shows an example program which shows how the *MicroBlaze* has to interact with the *rfiduhf* user logic to read out the EPC.

B.1 Initializing the Hardware

B.1.1 Initializing the N210 and SBX GPIOs

In the beginning the GPIOs connected to the N210 and SBX are configured such that all ICs needed for the subsequent procedures are enabled and supplied with power. Also the RF switch selecting the reference clock for the clock distribution IC is set to the internal 10 MHz reference. Listing B.1 shows a working configuration example.

```
gpio1_data = 0xF18CE69C;
gpio2_data = 0x00008F0E;
```

Listing B.1: Example for initializing the SBX GPIOs

B.1.2 Initializing the Clock Distribution IC

Listing B.2 shows a sample configuration for the clock distribution IC. The PLL is activated and the distribution is configured to provide 100 MHz to all other ICs, e.g. ADC, DAC, and the synthesizers.

```
// configure PLL:
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000400); //A divider not used
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000500); //B divider MSB[4:0]
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000605); //B divider LSB[7:0]
```

```
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000B00); //R divider MSB[5:0]
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000C01); //R divider LSB[7:0]

g_spi_write(SPI_DEV_MAINCLK, 24, 0x000700); // PLL 1
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000847); // PLL 2
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000940); // PLL 3
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000A04); // PLL 4
g_spi_write(SPI_DEV_MAINCLK, 24, 0x000D00); // PLL 5


// configure clock distribution:
g_spi_write(SPI_DEV_MAINCLK, 24, 0x003C08); // (testclk: LVPECL 810 mV)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x003D08); // (fpgaclk: LVPECL 810 mV)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x003E08); // (adcclk: LVPECL 810 mV)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x003F08); // (dacclk: LVPECL 810 mV)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x004003); // (SERDES: Output off)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x004102); // (CLKTXDB: LVDS 3.5 mA)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x004203); // (CLKEXPOUT: Output off)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x004302); // (CLKRXDB: LVDS 3.5 mA)

g_spi_write(SPI_DEV_MAINCLK, 24, 0x004980); // Divider0 (Bypass MODE)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x004B80); // Divider1 (Bypass MODE)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x004D80); // Divider2 (Bypass MODE)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x004F80); // Divider3 (Bypass MODE)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x005180); // Divider4 (Bypass MODE)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x005380); // Divider5 (Bypass MODE)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x005580); // Divider6 (Bypass MODE)
g_spi_write(SPI_DEV_MAINCLK, 24, 0x005780); // Divider7 (Bypass MODE)


// update registers:
g_spi_write(SPI_DEV_MAINCLK, 24, 0x005A01);
```

Listing B.2: Example for initializing the clock distribution IC (AD 9510)

B.1.3 Initializing the DAC

Listing B.3 shows an example configuration for the Analog Devices AD 9777 DAC.

```
// configure DAC:
g_spi_write(SPI_DEV_DAC, 16, 0x0004);
g_spi_write(SPI_DEV_DAC, 16, 0x0480);
g_spi_write(SPI_DEV_DAC, 16, 0x0104);
g_spi_write(SPI_DEV_DAC, 16, 0x0301);
```

Listing B.3: Example for initializing the DAC (AD 9777)

B.1.4 Initializing the ADC

Listing B.4 shows an example configuration for the Texas Instruments ADS62P44 ADC. It also shows how the DCM for the ADC interface is reset properly.

```
g_spi_write(SPI_DEV_ADC, 16, 0x0002); // Perform reset
g_timer_delay(TIMER_TICKS_US(800)); // wait 800 us

// configure ADC:
```

```
g_spi_write(SPI_DEV_ADC, 16, 0x1480);
g_spi_write(SPI_DEV_ADC, 16, 0x1600);
g_spi_write(SPI_DEV_ADC, 16, 0x1700);
g_spi_write(SPI_DEV_ADC, 16, 0x1A80);
g_spi_write(SPI_DEV_ADC, 16, 0x1B03);
g_spi_write(SPI_DEV_ADC, 16, 0x1D00);

// reset the ADC DCM:
rfiduhf_adc_dcm = rfiduhf_adc_dcm | (1 << dcm_adc_reset);

// wait till locked:
while(!(rfiduhf_adc_dcm & (1<<dcm_adc_locked)));
```

Listing B.4: Example for initializing the ADC (ADS62P44)

B.1.5 Initializing the Synthesizers

Listing B.5 and figure B.1 show a example configuration for the two Analog Devices ADF4351 synthesizers. This configuration example targets a RF output frequency of 865.7 MHz.

```
g_spi_write_rxtxsynt(32, 0x00580005); // Register 5
g_spi_write_rxtxsynt(32, 0x002C8234); // Register 4
g_spi_write_rxtxsynt(32, 0x00013E83); // Register 3
g_spi_write_rxtxsynt(32, 0x60010642); // Register 2
g_spi_write_rxtxsynt(32, 0x000087D1); // Register 1
g_spi_write_rxtxsynt(32, 0x001104E8); // Register 0
g_spi_write_rxtxsynt(32, 0x001104E8); // Register 0
```

Listing B.5: Example for initializing the synthesizers (ADF4351)

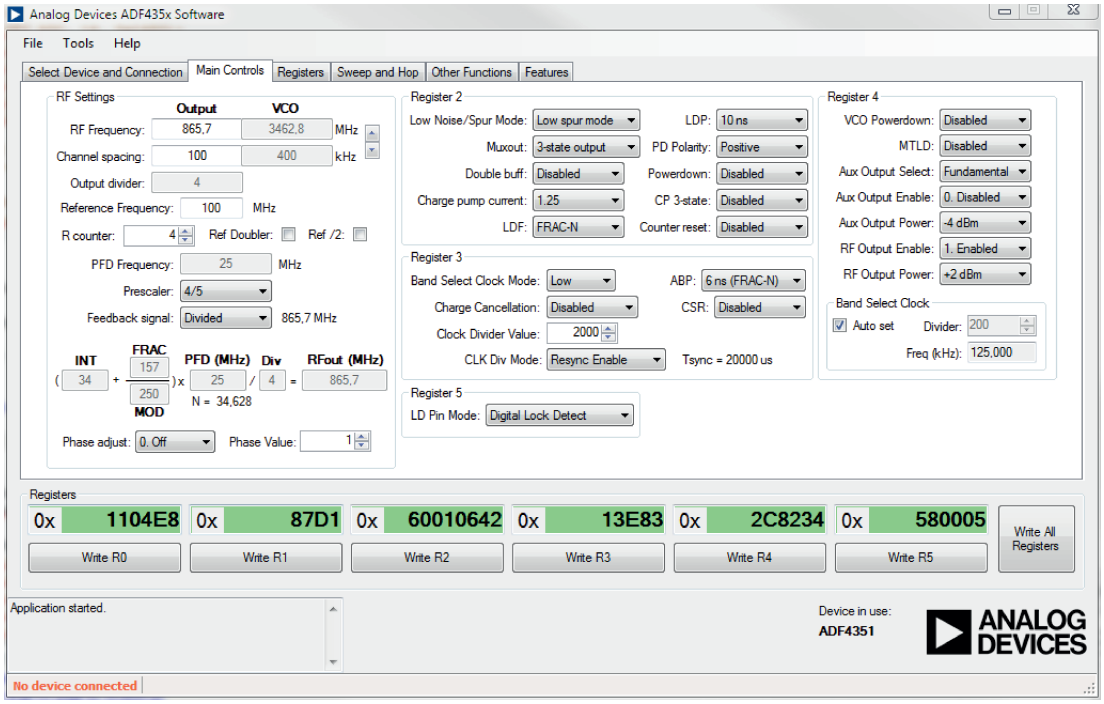

Figure B.1: Example configuration for the synthesizers (ADF4351)

B.2 Reading the EPC from a single tag in reading range

The following example listing B.6 should show how the during this thesis developed user logic can be used to readout the EPC of an RFID tag in reading range of the USRP-2292 platform.

```
u16 RN16;

//Parameters for the Inventory round:
u8 DR = 0; // DR = 8 see Query command
u8 M = 1; // M = 2 see Query command
u8 TRext = 0;
u8 sel = 0; // all tags should participate in the inventory round
u8 session = 0; // Session S0
u8 target = 0; // Target A
u8 Q = 0; // only one tag is expected in the reading range

u16 PW = 25; // PW in 0.5us steps -> 12.5us
u16 Tari = 50; // Tari in 0.5us steps -> 25us
u16 Tone = 100; // Tone in 0.5us steps -> 50us
u16 RTcal = Tari+Tone;
u16 TRcal = 320; // TRcal in 0.5us steps -> 160us
u16 modulation = 0; // ASK modulation
u16 txpwr_tx = 4800; // according to the default filtercoef
```

```
u16 blf_samples = 2000; // 50kHz BLF -> 2000 samples in 100 MSample/s
u16 lencorrpulse = 1000; // 50 kHz BLF
u32 threshold = 5000;

// switch on the RF-carrier:
rfiduhf_txconf1 = (1<<pwr_on_tx); // set the pwr_on_tx bit
while((rfiduhf_txconf1 & (1<<busy_tx)); // wait till busy_tx is 0

// configure rfidrx:
rfiduhf_rxconf1 = (threshold<<6)|(M<<4)|(TRext<<3)|(1<<reset_rx);
rfiduhf_rxconf2 = (blf_samples<<16)|(lencorrpulse);

// configure rfidtx:
Xil_Out32(C_MEM0_BASEADDR,(0b1000<<28)|(DR<<27)|(M<<25)|\
  (TRext<<24)|(sel<<22)|(session<<20)|(target<<19)|(Q<<15));
rfiduhf_txconf2 = (Tari<<16)|(Tone);
rfiduhf_txconf3 = (RTcal<<16)|(TRcal);
rfiduhf_txconf4 = (PW<<16)|(txpwr_tx);
rfiduhf_txconf5 = 0;

// start rfidtx send 17 bits:
rfiduhf_txconf1 = (17<<lenght_tx)|(1<<pwr_on_tx)|(1<<start_tx);

// wait till rfidtx is finished:
while((rfiduhf_txconf1 & (1<<busy_tx));

// start rfidrx:
rfiduhf_rxconf1 |= (1<<start_rx);

// wait till RN16 is received:
while((rfiduhf_rxconf3 & (1<<busy_rx));

// read out the RN16 from the bram_rx:
RN16 = (Xil_In32(C_MEM1_BASEADDR)>>10);

// wait T2:
g_timer_delay(TIMER_TICKS_100MHZ(3*blf_samples));

// send ACK:
Xil_Out32(C_MEM0_BASEADDR,(0b01<<30)|((RN16<<14)&0x3FFFC000));
rfiduhf_txconf1 = (18<<lenght_tx)|(1<<pwr_on_tx)|(1<<start_tx);

// wait till rfidtx is finished:
while((rfiduhf_txconf1 & (1<<busy_tx));

// start rfidrx:
rfiduhf_rxconf1 |= (1<<start_rx);

// wait till EPC is received:
while((rfiduhf_rxconf3 & (1<<busy_rx));

// the EPC can now be found in the bram_rx
```

Listing B.6: Reading the EPC of a single Tag using the rfiduhf user logic