UNIVERSITY OF CAMBRIDGE
Department of Engineering

Distributed Software-Defined Radio for RFID

Author Name: Siddharth Gupta
Supervisor: Dr Michael Crisp

Date: May 30, 2018

Distributed Software-Defined Radio for RFID

Siddharth Gupta, Pembroke College
Supervisor: Dr Michael Crisp

Technical abstract

Radio-frequency identification (RFID) is a growing market, with much recent commercial interest focussing on applications involving passive, ultra-high frequency tags. This has motivated further research into this area, creating a demand for flexible tools to perform experiments. These tools can be provided by software-defined radio (SDR) devices, which implement baseband signal processing in software instead of hardware; this allows the same hardware to be used for many different protocols and modulation techniques.

There are many potential benefits to using SDRs as RFID readers, as they enable rapid development and highly customised applications. These applications include multi-input multi-output (MIMO) RFID schemes involving multiple radios, which would be more difficult without a host computer or network to handle data transfer between devices. The LimeSDR-USB 1v4 is a recently released SDR, with excellent performance compared to other similarly priced SDRs, and includes full-duplex 2x2 MIMO in a single board.

In this project, the LimeSDR is configured and evaluated for use in passive UHF RFID applications. To achieve this, a software configuration is developed using a modified set of drivers, coupled with an existing RFID library implemented in the open-source software development tool-kit GNU Radio. A MATLAB script for analysing RFID communications is also developed.

The unmodified LimeSDR drivers and recommended software configuration cause a number of difficulties in an RFID application; these include a baseband tuning error, a DC corrector issue, desynchronisation of reader-tag communications, and excessive latency. The tuning, DC corrector, and desynchronisation problems are fully resolved. The latency is reduced such that 75% of tag-reading attempts succeed in best-case conditions, with the remaining 25% failing due to latency. In particular, the mean latency between the RN16 and the ACK is reduced from 7 ms to 475 µs.

The performance of the LimeSDR is evaluated using two metrics: the receiver sensitivity, and the transmitter spectrum characteristics. The sensitivity has been defined in this context as the minimum tag power at the receiver for which at least 25% of attempted communications with a tag are successful. The sensitivity is found to be linearly related to the transmit-receive (TX-RX) isolation for isolation values between 50.4 dB and 84.8 dB. The total RX gain is found to have little impact on sensitivity, except for causing saturation of the receiver analogue to digital converters for certain combinations of low TX-RX isolation and high RX gain. At a TX-RX isolation of 84.8 dB, the receiver sensitivity is found to be −90.3 dBm. This is comparable to common commercial RFID chips.

The transmitter spectrum is found to violate European Telecommunications Standards Institute restrictions on transmit spectral mask, for both a modulated waveform and a continuous wave. Thus, in a commercial application, analogue filtering would be required at the transmitter output.

The key implication is that the LimeSDR is now successfully configured for UHF RFID applications, with sensitivity comparable to common commercial RFID chips, providing a low-cost alternative for SDR RFID research. This has already attracted interest from other researchers.

Contents

1 Introduction

1.1 Objective

The objective of this project was to evaluate a commercially available software-defined radio (SDR), the LimeSDR-USB 1v4 [1], for use in passive ultra-high frequency (UHF) radio-frequency identification (RFID) applications.

1.2 Motivation

The RFID market is growing rapidly, with significant recent interest focussing on passive UHF tags [2]. This has motivated research into improving the range and reliability of RFID readers in more complex environments, where problems such as fading and interference occur, creating a demand for more flexible research tools.

The overall vision of this series of 4$^{\text{th}}$ year projects is to create an expandable, distributed network of synchronised RFID readers, in order to allow a large number of antennas to be used to read a tag simultaneously. Such a multi-input multi-output (MIMO) scheme offers many benefits, including improved coverage, improved ability to detect multiple tags, the ability to localise tags, and a reduction in the effects of fading [3]. If this can be accomplished by connecting the readers into a network using standard data cables, as opposed to connecting the readers directly to each other via co-axial cables, then a significant reduction in cabling complexity and cost can be achieved, greatly increasing the viability of the technique.

This project specifically focussed on evaluating the performance of a commercially available SDR, the LimeSDR-USB 1v4 (henceforth referred to as 'the LimeSDR'), as an RFID reader, so that it can be considered as a candidate platform to develop the above scheme. However, as detailed in the following sections, enabling the use of the LimeSDR as an RFID reader could have wider implications.

1.2.1 Benefits of SDR for RFID

Currently, the value of using SDR in RFID lies primarily in research, as the flexibility of SDR lends itself to rapid prototyping and customised experiments. SDR also greatly facilitates analysis, as the raw data is already on the computer, without needing to be separately extracted from a dedicated reader in a proprietary format.

However, this flexibility increases the device cost relative to dedicated RFID readers of comparable performance, which generally reduces the viability of an SDR RFID reader in a traditional commercial context. This may change in the future as RFID networks increase in complexity, where more sophisticated techniques may be used, requiring the additional flexibility and computational power of an SDR. For example, it has been suggested that wideband infrastructure might be used to extend data networks within buildings, and by sharing the same infrastructure across multiple data protocols (including RFID), overall power consumption could be reduced [4]. SDR technology would be highly suited to this application due to its flexibility and wideband capability.

In order to change the functionality of an SDR RFID reader, only the software needs to be changed. This is particularly useful for RFID research; examples of some of the RFID-relevant aspects which can typically be varied include:

- The transmit (TX) and receive (RX) gains, and the specific distribution of these gains between the various hardware amplification stages
- The carrier frequency
- The backscatter link frequency
- The particular RFID protocol used, in terms of modulation format and bit timings
- The extent of digital filtering, and in some cases onboard analogue filtering
- The sampling rate

Using an SDR, all of the above can often be varied more or less arbitrarily within the hardware limits, even outside of RFID protocol specifications, which would not be possible with a more traditional radio platform or a dedicated RFID reader.

Furthermore, as the SDR is generally connected to a host computer with much more processing power available than a typical dedicated RFID reader, more demanding or sophisticated techniques can be used, which can lead to improved performance. Some recent examples of more complex experiments using SDRs in RFID research include:

- Synchronisation of multiple SDRs to allow time-of-flight localisation of passive tags [5]
- Use of a single SDR in a SIMO (single input, multiple output) configuration to allow for phase-difference-on-arrival (PDoA) localisation of passive tags [6]
- Eavesdropping on RFID communications [7]
- Low-cost RFID tag performance characterisation [8]

These examples highlight some of the key benefits which SDR can bring to RFID: synchronisation of multiple SDRs, MIMO over multiple devices, and very flexible, customisable research tools.

1.2.2 Benefits of the LimeSDR

There are currently many commercially available SDRs; see table 1 for a comparison of some of the more popular devices. However, the LimeSDR in particular possesses many features which make it a potentially interesting RFID reader for SDR RFID research. Table 1 shows that the LimeSDR performs well on most general metrics when compared to the other SDRs of a similar price. For RFID research in particular, however, the key advantages of the LimeSDR include full duplex (simultaneous transmit and receive) 2x2 MIMO on a single board, and fully open-source hardware and software.

The other key benefit of the LimeSDR is its price; currently retailing at USD $299 [1], it is significantly cheaper than current hardware often used for SDR RFID research, which often retails for several thousand pounds [9]. Thus, successfully implementing an RFID reader with the LimeSDR could lower the barrier to entry for SDR RFID research. This is particularly true for research involving MIMO operation over multiple devices, as in these cases the cost of multiple SDR devices may be the limiting factor.

Table 1: A comparison table for a selection of commonly used commercially available SDRs. Specifications for the LimeSDR are given in the first column. Table data taken from [1]. Note that two models of the BladeRF are available: the x40, and the x115; values which differ for the x115 are given in parantheses.

	LimeSDR	LimeSDR Mini	Ettus B200	Ettus B210	BladeRF x40	RTL-SDR	HackRF One
Frequency Range	100 kHz - 3.8 GHz	10 MHz - 3.5 GHz	70 MHz - 6 GHz	70 MHz - 6 GHz	300 MHz - 3.8 GHz	22 MHz - 2.2 GHz	1 MHz - 6 GHz
RF Bandwidth	61.44 MHz	30.72 MHz	61.44 MHz	61.44 MHz	40 MHz	3.2 MHz	20 MHz
Sample Depth	12 bit	12 bit	12 bit	12 bit	12 bit	8 bit	8 bit
Sample Rate	61.44 MS/s	30.72 MS/s	61.44 MS/s	61.44 MS/s	40 MS/s	3.2 MS/s	20 MS/s
TX Channels	2	1	1	2	1	0	1
RX Channels	2	1	1	2	1	1	1
Duplex	Full	Full	Full	Full	Full	N/A	Half
Interface	USB 3.0	USB 3.0	USB 3.0	USB 3.0	USB 3.0	USB 2.0	USB 2.0
Programmable Logic Gates	40k	16K	75k	100k	40k (115k avail)	N/A	64 macrocell CPLD
Chipset	LMS7002M	LMS7002M	AD9364	AD9361	LMS6002M	RTL2832U	MAX5864, MAX2837, RFFC5072
Open Source	Full	Full	Schematic, Firmware	Schematic, Firmware	Schematic, Firmware	No	Full
Oscillator Precision	±1 ppm initial, ±4 ppm stable	±1 ppm initial, ±4 ppm stable	±2 ppm	±2 ppm	±1 ppm	unknown	±20 ppm
Transmit Power	max 10 dBm (depending on freq.)	max 10 dBm (depending on freq.)	10 dBm+	10 dBm+	6 dBm	N/A	-10 dBm+ (15 dBm @ 2.4 GHz)
Price (USD)	$299	$139	$686	$1,119	$420 ($650)	~$10	$299

1.3 Current state of the art

The novelty of this project lies primarily in the use of the LimeSDR as an RFID reader. Currently, many researchers [5, 6, 8, 10] use Universal Software Radio Peripheral (USRP) devices, produced by Ettus Research. However, to the best of this author's knowledge, no published work has implemented an RFID reader using the LimeSDR.

From a software standpoint, many SDR RFID reader implementations in the literature are based on an implementation by Buettner [11]. This was then modified in work by Kargas [10], which contained a basic implementation of an RFID reader, allowing retrieval of the EPC from a tag using only a single modulation format (FM0 line coding) and a fixed backscatter link frequency of 40 kHz. Others have modified this library to add further functionality. However, for this project, the basic library was deemed to be a sufficient starting point, as the aim was to evaluate the LimeSDR specifically, as opposed to a particular software implementation of an RFID reader.

2 Background

In this section, some background knowledge is presented to facilitate better understanding of the project.

2.1 Software-defined radio (SDR)

Conventional radios have fixed hardware dedicated to transmitting and/or receiving signals, with all of the signal processing being carried out in hardware, or on a microprocessor running dedicated firmware (so that it cannot be altered to perform a different function). In contrast, the principle behind SDR is that the majority of the baseband signal processing is carried out in software, with the hardware facilitating the conversion between analogue radio waves and digital samples in software on a computer. SDRs can also often tune over a very wide range of frequencies, whereas conventional radios typically tune over a small, protocol-specific range. Figure 1 gives a general block diagram for SDR receivers; transmitters are similar, but in reverse. The hardware of the LimeSDR specifically is described in section 2.3.

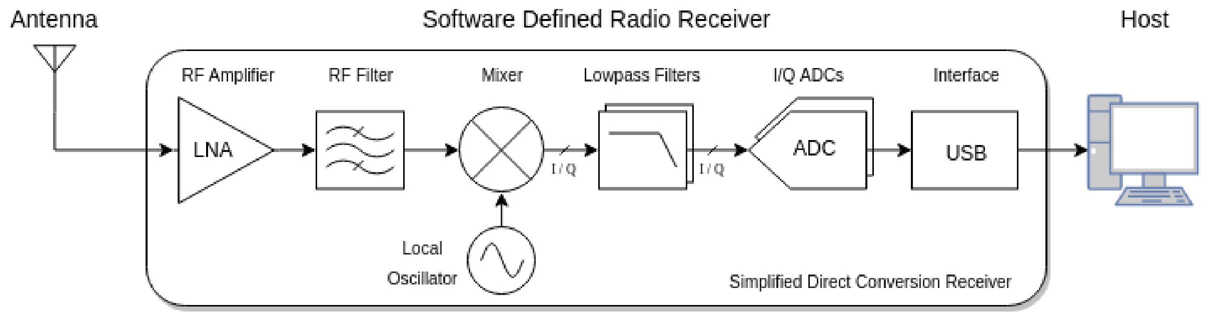


Figure 1: A block diagram showing a typical SDR receiver architecture [12]. The transmit architecture is similar, but in reverse. Note that the interface need not be USB; ethernet is also often used.

2.2 RFID

2.2.1 RFID tags

RFID uses electromagnetic waves to collect information from ‘tags’ attached to objects, generally for the purpose of tracking or identification. Three broad categories of tags are available, distinguished by how they are powered:

- Active tags have their own power source, and have an active transmitter to respond to a reader.
- Semi-passive tags have a power source for the tag circuitry, but do not include an active transmitter.

- Passive tags have no power source; they harvest energy from the waveform transmitted by the reader.

Semi-passive and passive tags lack an active transmitter. Therefore, to respond to the reader, they use a technique called 'backscattering': the tag changes the load attached to its antenna, which changes how the antenna reflects radio waves, allowing a basic form of modulation to be used. In this way, a reader can obtain a message from a tag by transmitting a constant amplitude wave at the correct frequency; the tag can then backscatter its response.

RFID tags can also be divided into three different frequency ranges [13]:

- Low-frequency (LF, 125/134 kHz) tags operate through inductive coupling between the tag and reader. This gives a short read range, but the low frequency allows significant penetration into metallic objects and water, making these tags suitable for human and animal identification.
- High-frequency (HF, 13.56 MHz) tags also use inductive coupling. The higher frequency means that the attenuation in water and metal is more significant than for LF tags. However, the higher frequency allows for a higher data rate. HF tags are therefore widely used for non-contact smart cards, where relatively complex exchanges must occur.
- Ultra-high frequency (UHF, 860-960 MHz and 2.4-2.45 GHz) tags operate through radiative coupling instead. This gives a much larger range than the other tag types, and simple antennas may be used, allowing tags to be low-cost. This is suitable for applications in automobile and rail-car tracking, as well as supply-chain management. However, the increased read-range also introduces complexities such as reflections and fading.

Passive UHF tags operating at a frequency of 910 MHz were used exclusively in this project, as their large range facilitates many potentially interesting applications.

2.2.2 Summary of RFID protocol

RFID communications can make use of a number of different protocols. The specific protocol investigated in this project is defined in ISO 18000-6 [14] (EPCglobal Class 1 Generation 2), commonly used for UHF tags. While the protocol offers many features, often the particular data of interest is the electronic product code (EPC) stored on a tag, which can be used as an identifier; thus, in this project, only retrieving the EPC was attempted. In order to retrieve the EPC from an ISO 18000-6 compliant tag, the following sequence must be followed:

1. The reader transmits a 'QUERY' command, which contains information about this particular session of communication, such as the backscatter frequency. The reader then transmits a constant-amplitude continuous wave at the carrier frequency, the CW, in order to allow the tag to backscatter a response.

2. The tag responds with an 'RN16', a random 16 bit number which is used to identify that particular tag in this communication session. The RN16 changes after each QUERY, even for the same tag.

3. The reader responds with an 'ACK', acknowledging receipt of the RN16 by effectively repeating it back to the tag. This is followed by another CW.

4. If a valid ACK was received, matching the transmitted RN16, the tag transmits its EPC. If an invalid ACK was received, no response is given.

Figure 2 shows a complete example of the above protocol as used in this project.

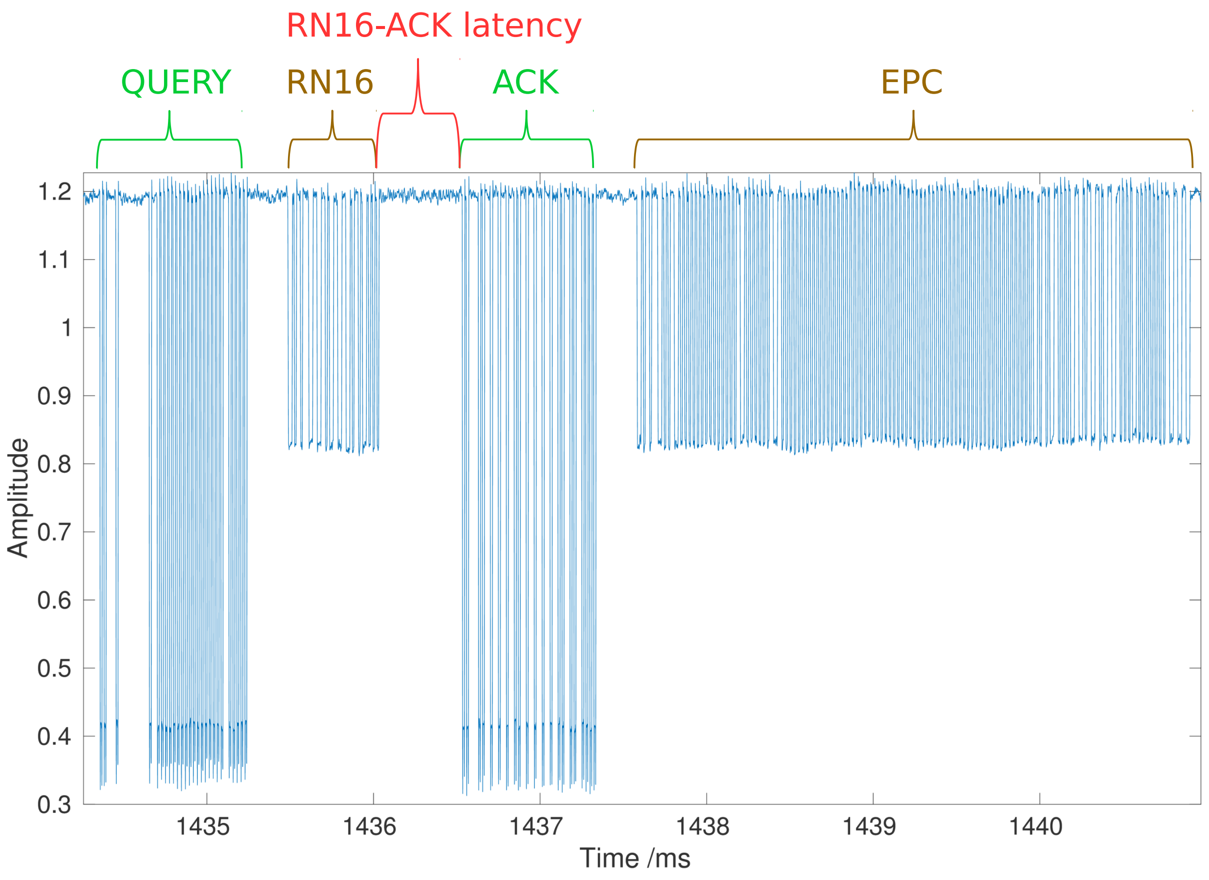


Figure 2: The raw baseband waveforms at the LimeSDR receiver for a typical example of a successful exchange between the reader and tag, such that the tag transmits its EPC. In this example, the setup was chosen such that the tag response (the RN16 and EPC, brown) would have an unusually large amplitude relative to the reader waveforms (green), in order to make it more visible; in most scenarios the tag response amplitude would be much smaller.

Protocol timing constraints The ISO 18000-6 standard defines timing constraints for all aspects of the protocol, which places constraints on both the reader and the tag. An important point to note is that the ACK must occur within a defined time interval after the end of the RN16. This interval depends on the frequency of tag modulation selected, also known as the backscatter link frequency (BLF). In this project, a BLF of 40 kHz was used, constraining the ACK to arrive between 75 µs and 500 µs after the RN16. The actual time taken between the end of the RN16 and the beginning of the ACK will henceforth be referred to as the *RN16-ACK latency*. Achieving a sufficiently low RN16-ACK latency proved to be a significant difficulty, as discussed in section 3.5.

2.3 LimeSDR hardware

This section presents a brief overview of the LimeSDR hardware, which is responsible for the key benefits of the LimeSDR. The programmable radio-frequency integrated circuit (RFIC), the LMS7002M [15], is also discussed in further detail. An annotated photograph of the upper surface of the LimeSDR is given in figure 3.

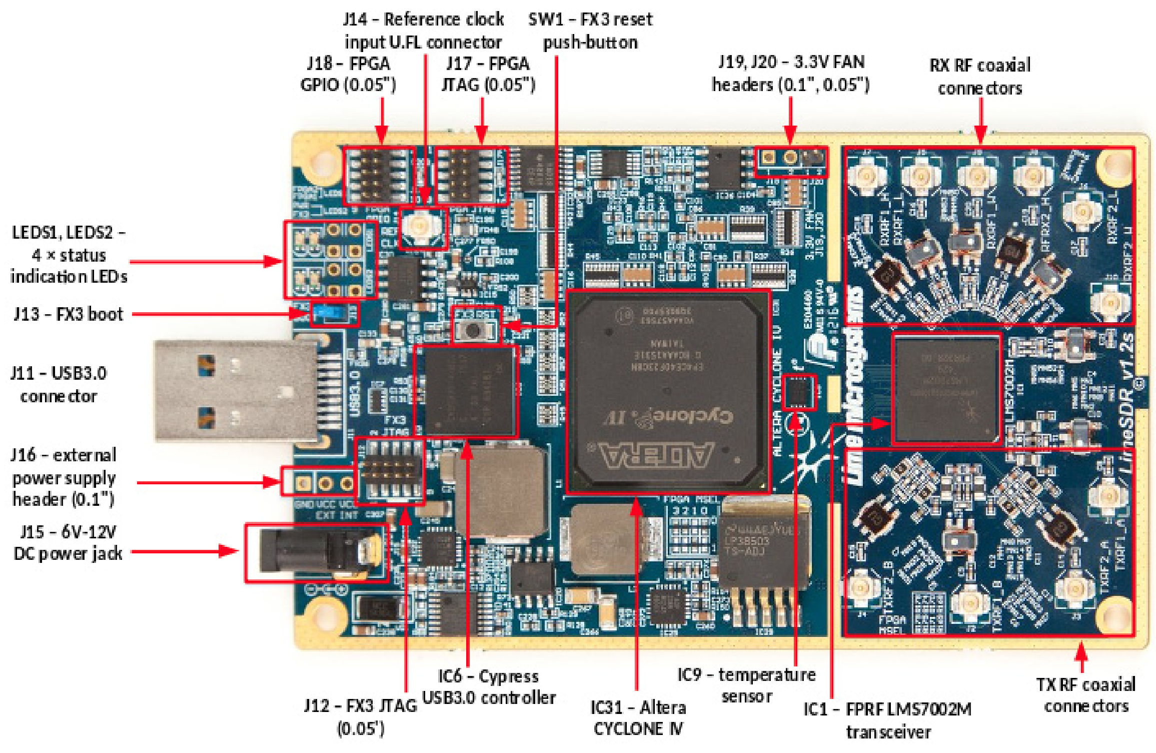

J14 - Reference clock input U.FL connector
SW1 - FX3 reset push-button
J18 - FPGA GPIO (0.05")
J17 - FPGA JTAG (0.05")
J19, J20 - 3.3V FAN headers (0.1", 0.05")
RX RF coaxial connectors
LEDS1, LEDS2 - 4 × status indication LEDs
J13 - FX3 boot
J11 - USB3.0 connector
J16 - external power supply header (0.1")
J15 - 6V-12V DC power jack
J12 - FX3 JTAG (0.05')
IC6 - Cypress USB3.0 controller
IC31 - Altera CYCLONE IV
IC9 - temperature sensor
IC1 - FPRF LMS7002M transceiver
TX RF coaxial connectors

Figure 3: An annotated photograph of the upper surface of the LimeSDR [16]

High-level overview Figure 4 gives an overall block diagram for the hardware. In this project, the sections of particular interest were those which handled samples; these included the USB3.0 sections, the Altera Cyclone IV E FPGA [17], and the LMS7002M.

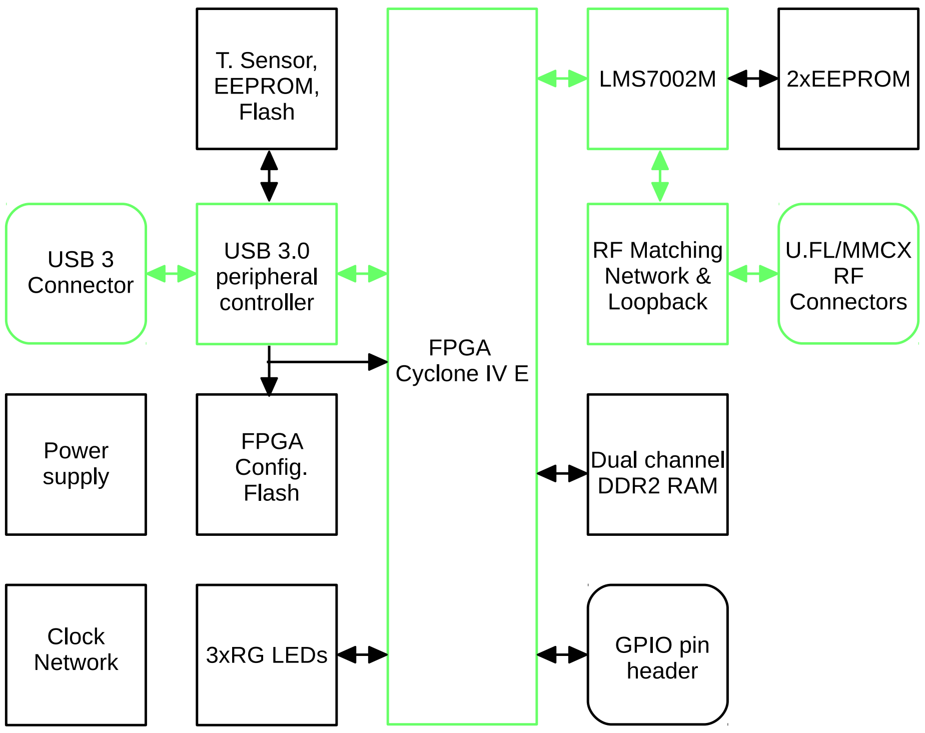


Figure 4: A high-level block diagram of the LimeSDR hardware, adapted from [18]. The sections which handle samples are coloured green.

FPGA The FPGA controls the operation of the LimeSDR, and acts as the logic interface between USB3.0 and the LMS7002M. The FPGA can be modified to perform custom functions, and for other SDRs with FPGAs, some researchers have implemented the reader logic directly on the FPGA itself [5]. This avoids transferring data from the FPGA to the host computer and back, which can be a significant source of latency. However, this approach reduces the flexibility of the solution when compared with an implementation on the host computer, as the FPGA gateware will in general be very hardware-specific. In contrast, with sufficient abstraction, software on a host PC can be almost hardware-agnostic (the software ecosystem is discussed further in section 2.4). Another problem with this approach is that if the data does not reach the computer in real-time, then it becomes much more difficult to move data between multiple devices, as this must be done directly from device to device, whereas a computer or network device could act as a central node to distribute information.

LMS7002M The RF performance and functionality of the LimeSDR is primarily determined by the LMS7002M. A block diagram of the LMS7002M is given in figure 5. From this figure, the 2x2 MIMO nature of the LMS7002M can be observed, as there are two transmit and two receive channels. For each channel, the different analogue paths are optimised for different frequency ranges, so that there are a total of 10 physical antenna connectors on the board.

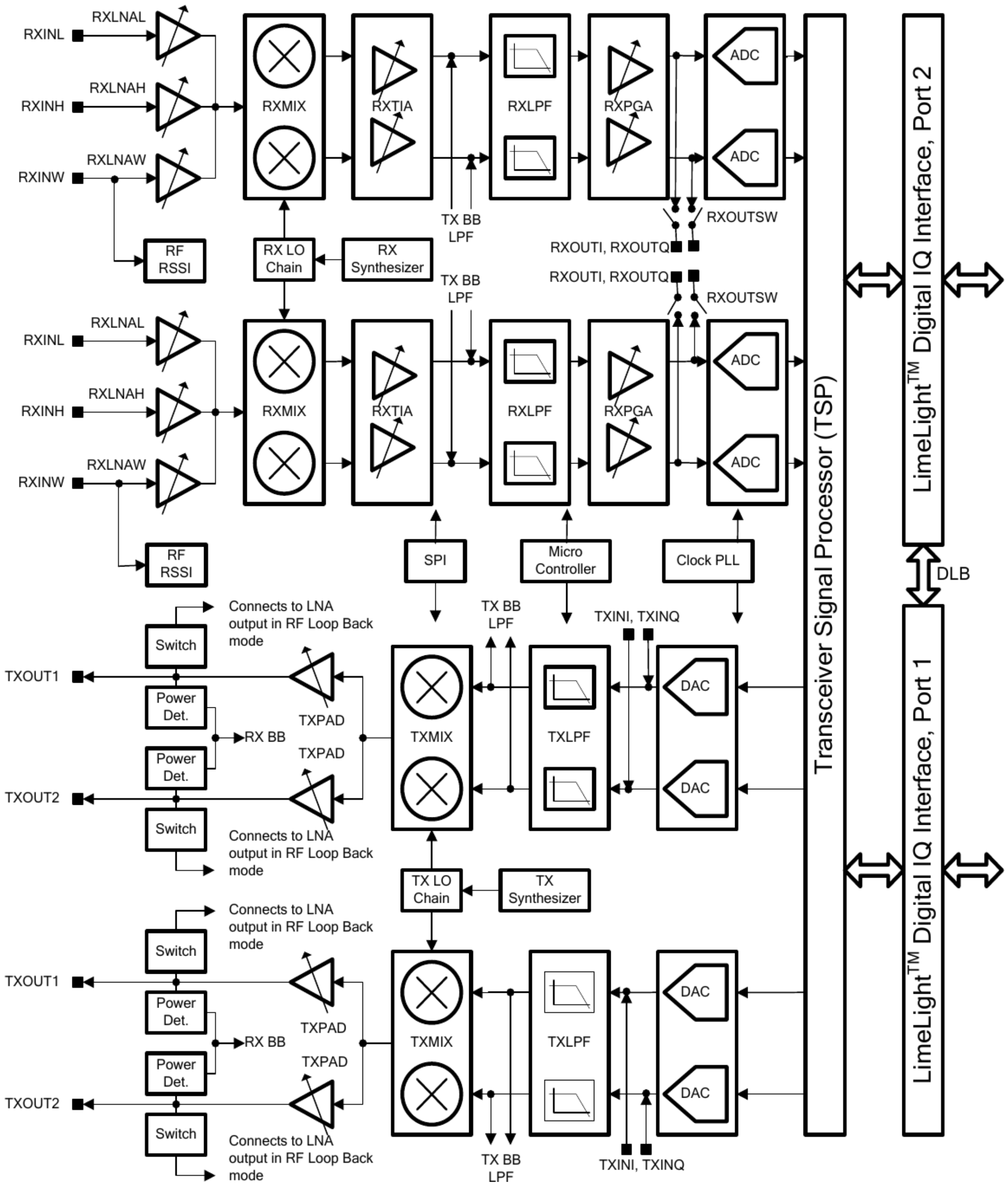


Figure 5: A high-level block diagram of the LMS7002M [19] RFIC used on the LimeSDR.

From figure 5 it can also be seen that the RX gain is divided into four stages: the RXLNA (low noise amplifier), the RXTIA (trans-impedance amplifier), the RXPGA (programmable gain amplifier), and the RXTSP (digital gain implemented in the transceiver signal processor (TSP)). The TX gain is shared between two stages; the TXTSP (digital gain implemented in the TSP), and the TXPAD (power amplifier driver). The drivers provided for the LimeSDR board automatically decide a gain distribution between the various elements for given total RX and TX gains; this algorithm was not altered.

The TSP contains (amongst other functions) programmable finite impulse response (FIR) filters for both TX and RX. These could be used to replace the matched filter implemented in the host computer software; this would be advantageous in situations where the host computer processing power is limited. However, it was found that the filtering on the host computer used here did not add significant processing time, so this functionality was not explored in detail.

Another attractive feature of the LMS7002M is the fact that the TX phase-locked loop (PLL) can be shared by both the TX and the RX chains. This causes the TX and RX mixers to use the same local oscillator, which can reduce the impact of TX-RX phase noise leakage; this is discussed further in section 4.3.1.

2.4 Software ecosystem

Software is required to carry out the signal processing and logic to implement an RFID reader on the host computer. On a high level, GNU Radio [20] is a widely-used open-source toolkit used to create SDR applications, enabling the construction of signal processing block diagrams, or ‘flowgraphs’, by connecting together standard signal processing blocks such as filters and mixers. Custom blocks can also be created, and the RFID reader program used in this project was implemented as a sequence of custom GNU Radio blocks.

In order to bridge between the high-level GNU Radio platform and low-level control of the LimeSDR, a few layers of abstraction are necessary. The software setup finally used in this project was (working from lower to higher levels of abstraction):

1. Lime Suite [21]: the core low-level drivers provided by MyriadRF which interface with the host computer USB drivers, and carry out LimeSDR-specific functions such as calibrations.

2. SoapyLMS7 [22]: a wrapper for Lime Suite to allows it to interface with SoapySDR.

3. SoapySDR [23]: an open-source generalized C/C++ application programming interface (API) and runtime library for interfacing with SDR devices.

4. SoapyUHD [24]: allows standard UHD (USRP Hardware Driver) GNU Radio blocks to be used to control devices compatible with SoapySDR.

5. GNU Radio as described above.

Alterations from the default source code were made in the SoapyLMS7 and GNU Radio layers; however, the presence of many layers added to the complexity of debugging, as it was initially difficult to determine which layer was causing problems. In GNU Radio, the particular flowgraph used was an adapted version of a Gen2 Reader program [10], which was initially developed using a USRP device.

Note that while the above programs were the final combination chosen, they are not the only means of writing software for use with the LimeSDR. Figure 6 shows diagrammatically some of the applications in the wider software ecosystem.

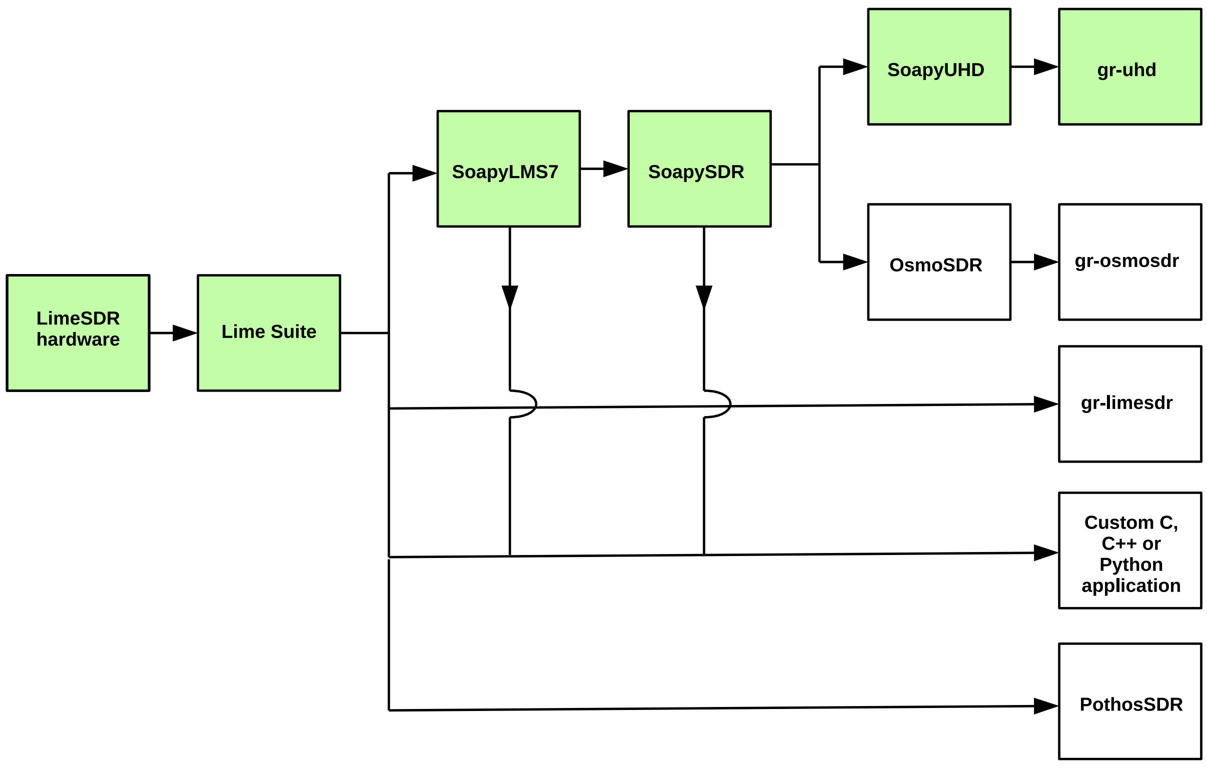


Figure 6: A block diagram summarising some of the alternative methods of producing SDR applications for the LimeSDR. Note that the blocks beginning with 'gr-' represent GNU Radio source/sink blocks, which would be used in a GNU Radio application. This is not an exhaustive diagram, but covers some of the more popular methods at the time of writing. The final path chosen in this project is highlighted in green.

2.5 GNU Radio flowgraph

The Gen2 Reader GNU Radio flowgraph used was developed by Kargas [10] in 2015. This section presents an overview of the operation of this program, as understanding this was key to the project.

An overall block diagram of the flowgraph is given in figure 7. The following sections describe the function and operation of each block in further detail. Note that the gate, decoder and reader blocks are custom blocks, and share a number of variables, so that each block can affect the other blocks directly in ways beyond passing samples.

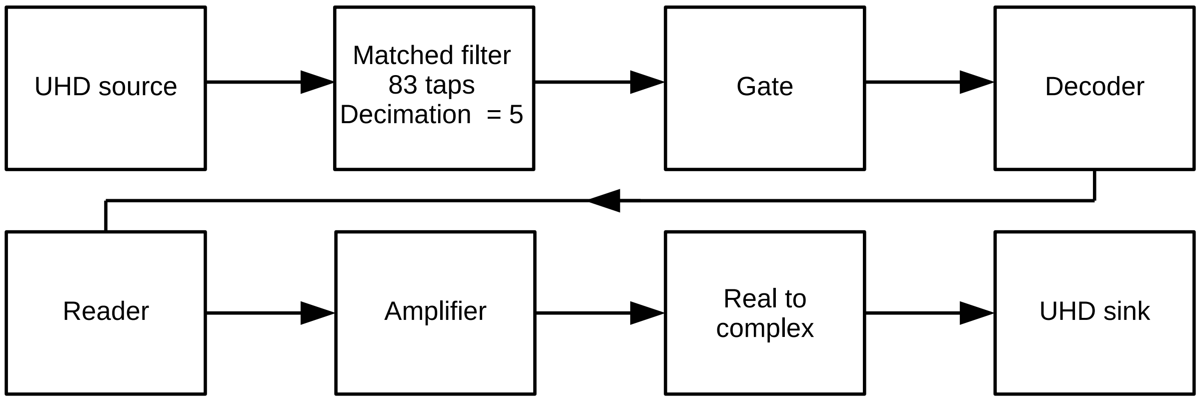


Figure 7: A block diagram for the Gen2 Reader flowgraph for GNU Radio used in this project. Note that all debugging-related blocks have been excluded from this diagram.

UHD source and sink The UHD source and sink blocks represent the connection to the hardware receiver and transmitter respectively. These blocks can be used to connect to a variety of different hardware devices; the particular device to be used is passed as a parameter to the block. Other parameters are also passed to the block, which control the hardware RF properties. These include:

- The analogue-digital converter (ADC) or digital-analogue converter (DAC) sampling rate
- The center frequency
- The total RX or TX gain, or individual gains of specific stages
- The antenna port
- The particular RX or TX channel

Depending on the hardware, other parameters may also be specifiable, but these were the parameters set for the experiments carried out here.

Matched filter The matched filter serves multiple purposes, including maximising the signal-to-noise ratio (SNR), and filtering out high frequency noise added in the channel [25]. Here, the matched filter was rectangular, with cut-off frequency of twice the BLF (40 kHz). This resulted in a required filter length of $\frac{1}{2\times40000} = 12.5\,\mu\text{s}$. Converting this into samples, using a sampling rate of 6.7 MS/s (mega samples per second), this required $12.5 \times 6.7 = 83.75$ filter taps. Rounding this down to the nearest integer gave 83 taps; rounding up would have begun to include the opposite modulation state in the averaging, which may have had an adverse effect on the efficacy of the filtering.

The matched filter also applied a decimation factor of five; this resulted in the output sample rate from the block being a factor of five lower than the input sample rate. This was done to reduce the processing requirements of later stages.

Gate The primary function of the gate is to detect reader commands and pass only the samples containing tag communications to the the decoder. It detects reader commands by calculating a rolling average DC offset, and then detecting when samples differ significantly from this DC offset. If samples have an amplitude below a certain threshold factor of the DC offset, this counts as the negative edge of a pulse; if then a positive edge of a pulse occurs after at least half a pulse width, then this pair of edges is counted as a pulse.

If at least five pulses have been detected, this suggests that a reader command has been detected. If it has also been at least a certain number of samples since the last pulse (approximately equal to the ISO standard-defined interval between reader commands and tag responses), then this suggests that the appropriate amount of time has been waited after the end of the reader command, such that it is likely that the tag response is about to begin. The gate then 'opens' and allows a certain number of the following samples through to the decoder, with the DC offset removed; the number of samples depends on whether an RN16 or an EPC is expected. After these samples have been passed on, the gate 'shuts' again and does not open until another reader command has been detected.

Decoder The decoder is responsible for recovering the RN16 or EPC in the samples received from the gate. If an RN16 is expected, an RN16 is always decoded from the provided samples, even if the samples do not actually contain a genuine RN16; clearly in these instances the decoded RN16 is meaningless, but it is not possible for the reader to know this *a priori*. However, if an EPC is expected, the decoded EPC must pass a cyclic redundancy check (CRC) in order to be interpreted as a genuine EPC.

The theory behind the decoding algorithm is described in [10]; since this algorithm was not altered in this project, it will not be discussed here.

After decoding an RN16, the decoded RN16 bits are passed to the reader block to be used to generate the corresponding ACK. If an EPC was decoded instead, nothing is passed to the reader block.

Reader The reader block is responsible for generating samples to be transmitted, including the QUERY and ACK commands, as well as the intervening CW. The timing of the transmission of a QUERY or ACK is limited by the time taken for the preceding EPC or RN16 to be processed.

Amplifier The amplifier block simply applies a fixed multiplier to the samples produced by the reader block.

Real to complex The real to complex block converts the real samples from the amplifier block into the complex samples required by the hardware sink.

3 Configuring the LimeSDR for RFID

The project initially focussed on configuring the LimeSDR to be able to retrieve EPCs from a standard commercial tag. This section provides an overview of this process, including a description of some of the hardware and software issues encountered, and an analysis script developed in MATLAB [26] to aid in troubleshooting.

3.1 Tuning error

The Lime Suite drivers implement a default tuning algorithm which first tunes the local oscillator (LO), and then corrects any remaining tuning error using the baseband (BB) numerically-controlled oscillator (NCO). However, in this application, using the default algorithm, a very low frequency tuning correction was being erroneously applied in the NCO, even though the LO had been tuned correctly. This resulted in the CW being modulated by a low frequency sine wave, which was undesirable, and prevented the RFID library from correctly interpreting the RN16s. This was corrected by manually editing the Lime Suite drivers to force the NCO tuning value to always be zero. Figure 8 shows the effect of this problem, with the DC corrector issue (see next section) resolved.

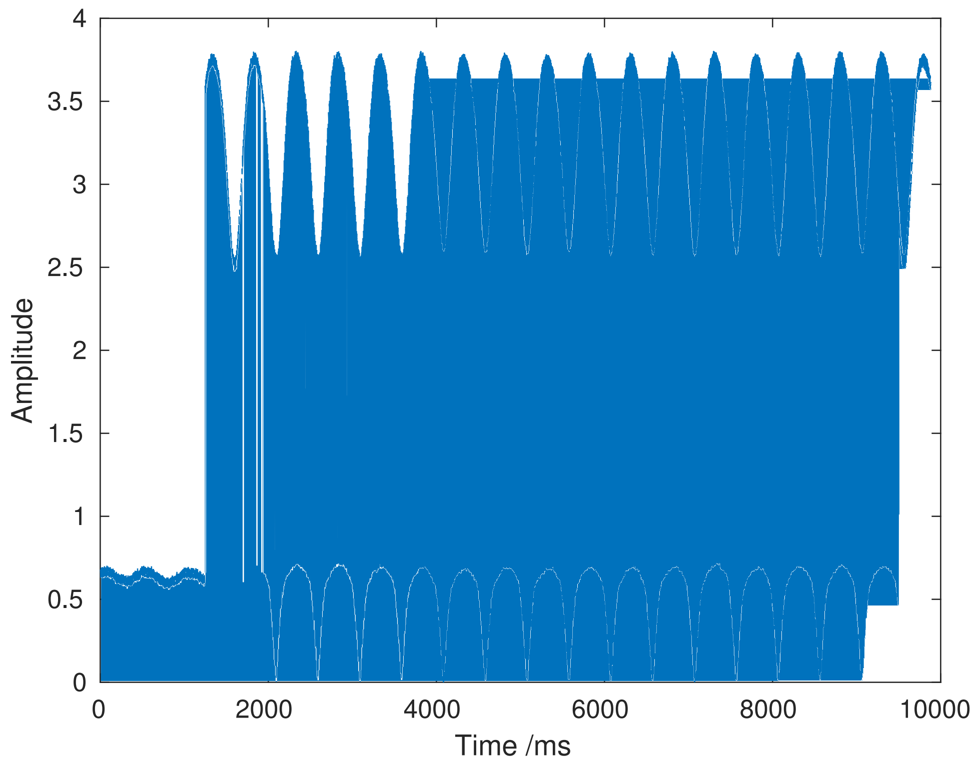


Figure 8: A plot of the received signal with the tuning error. Note the large amplitude, low frequency sinusoidal modulation.

3.2 DC corrector

By default, the Lime Suite drivers enable a DC corrector, which attempts to correct for unwanted DC offsets in signals. However, for very low BB frequency signals, the DC corrector may prematurely cut in, causing a signal with a large number of discontinuities. The tuning error discussed above was sufficiently low frequency to cause this issue, which therefore also prevented the RFID library from correctly interpreting the RN16s. Even without the tuning error, the DC corrector still heavily distorted the reader waveform. This was resolved by editing the SoapyLMS7 drivers to bypass the DC corrector. Figure 9 gives a representative example of a test run with the problems given by both the DC corrector and tuning error.

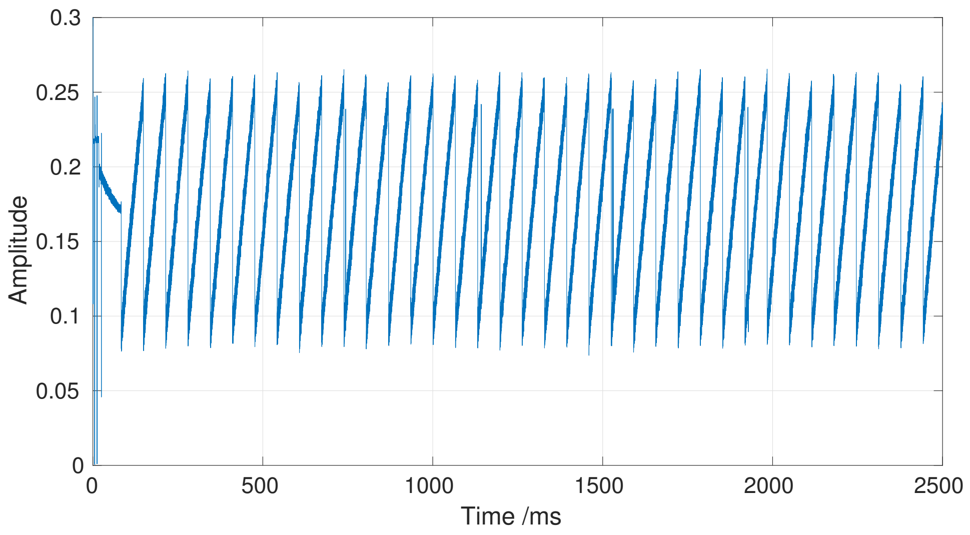


Figure 9: The received signal without DC corrector or tuning error fixes applied. Note the sawtooth-like oscillation due to the combination of the DC corrector and tuning errors; a correct signal would have constant amplitude, and so would appear as a horizontal straight line, aside from the RFID signal modulations.

3.3 MATLAB analysis script

In order to debug many of the issues described in the following sections, it was necessary to be able to analyse the signals received by the LimeSDR in detail. This included information such as the exact duration of the RN16-ACK latency, which could not easily be provided by modifying the GNU Radio library. This is because GNU Radio only deals with samples, rather than time (this is discussed further in section 3.5.1). The raw samples received in GNU Radio could be recorded as a file which could then be imported into MATLAB, so a MATLAB script was written to determine the desired information by processing the data offline.

The script located communication blocks (QUERYs, RN16s, ACKs, or EPCs) by considering the change in standard deviation of amplitude for a section of samples immediately before or after a block. A large change indicated the beginning or end of a block. The blocks were then classified using both the block duration and knowledge of the possible sequences of blocks. The blocks could be located to within a few microseconds, allowing accurate block durations and inter-block timings to be calculated.

The script also decoded the RN16s, ACKs, and EPCs, using a mid-point crossing detection algorithm, and checked the validity of the EPCs. This enabled communication errors to be identified, including, for example, checking whether an ACK contained the same data bits as its corresponding RN16. The accuracy of the script was confirmed by comparing the number of EPCs and QUERY commands detected with that reported by the GNU Radio program, and verifying that they matched in the majority of cases.

3.4 Desynchronised RN16s and ACKs

The LimeSDR carries out a calibration procedure at the start of each test. This produces waveforms which can be interpreted by the Gen2 Reader library as a reader command, which on some occasions can cause RN16s and ACKs to become 'desynchronised'. Specifically, when this desynchronisation occurs, all of the ACKs sent by the reader in response to a tag RN16 are actually the ACK corresponding to the RN16 sent by the tag in the *previous* exchange, rather than the current exchange. For example, if corresponding communication blocks are given matching subscripts, then a correct pair of exchanges would be:

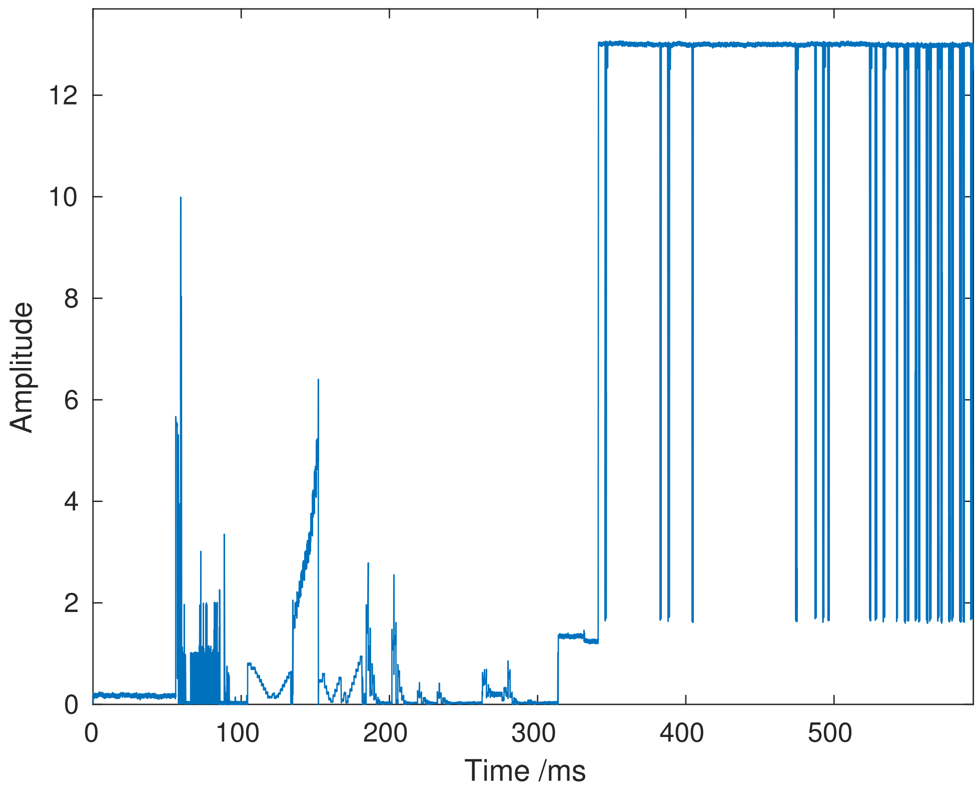
$QUERY_1$ -> $RN16_1$ -> ACK_1 -> EPC_1
$QUERY_2$ -> $RN16_2$ -> ACK_2 -> EPC_2

while a desynchronised pair of exchanges would be:

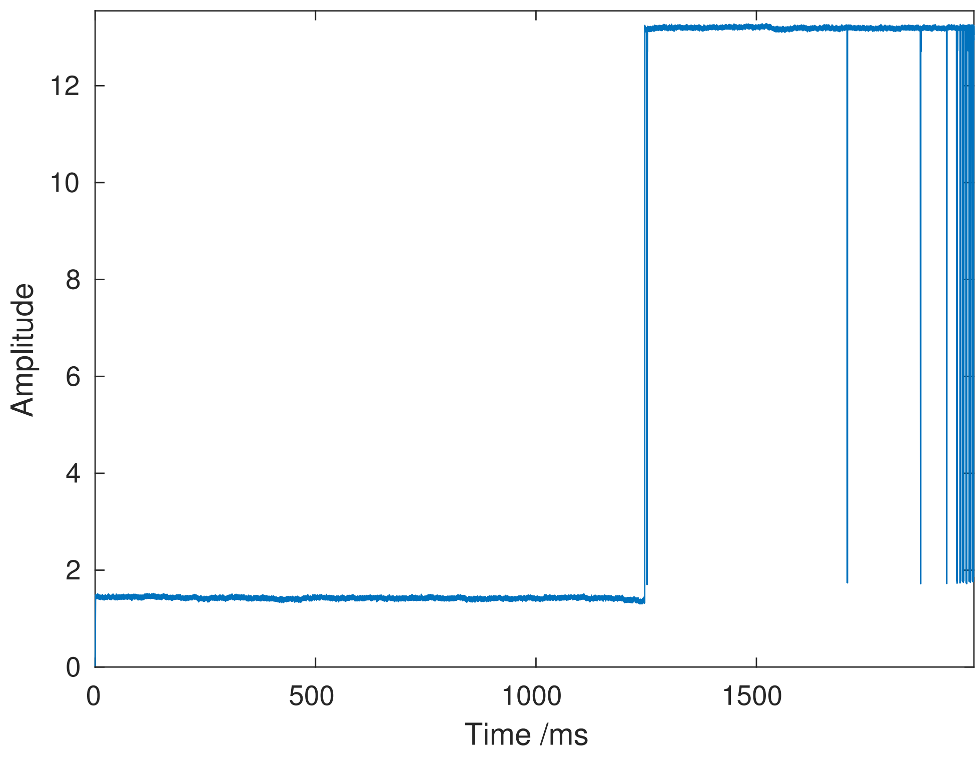
$QUERY_1$ -> $RN16_1$ -> ACK_0
$QUERY_2$ -> $RN16_2$ -> ACK_1

This can cause all attempted communications with the tag to fail in a particular test, even in conditions where a reasonable read rate has previously been achieved. Figure 10a below shows an example of the waveforms generated by the calibration procedure.

To resolve the issue, a delay was added in the SoapyLMS7 drivers after the calibration procedure had begun, but before the Gen2 Reader library had begun to request samples. This delay allows the calibration procedure to complete before the samples begin to be sent to GNU Radio, thus preventing the library from seeing the calibration waveform, and so resolving the issue. Figure 10 shows examples of the start-up waveform seen by GNU Radio before and after the solution was implemented.



(a)



(b)

Figure 10: The start-up waveform (a) before and (b) after adding the post-calibration delay in the SoapyLMS7 drivers. In (a), the normal modulation begins after approximately 340 ms, whereas in (b), this occurs after 1300 ms.

3.5 Latency

For a 40 kHz backscatter link frequency, the protocol requires the ACK to begin within 500 µs of the end of the RN16. With the default Lime Suite drivers and unaltered Gen2 Reader program, the RN16-ACK latency is approximately 7 ms, over an order of magnitude too large. In this project, this has been reduced to a mean latency of 475 µs. The following sections explore this issue in detail, describing the measures which successfully decreased the latency, along with some which were investigated but did not have a measurable impact. The final latency achieved is also summarised, and suggestions are given of methods to reduce latency further.

Note that latency is one of the primary disadvantages of SDR for RFID; in general, SDRs are not used with real-time operating systems, so the operating system cannot guarantee that the latency is kept under a certain limit. By contrast, in a dedicated RFID reader, samples do not need to be transferred from the device to a computer and back, so the majority of the sources of latency are avoided.

3.5.1 Real-time scheduling for GNU Radio

GNU Radio was run in a 'real-time scheduling' mode; this is a setting which minimises the latency caused by the GNU Radio layer. GNU Radio itself does not have a concept of global system time; it cannot give a guarantee that any sample will be processed in a specified time, as this may not be possible given the processing power of the CPU used, and given that the operating system it is running on is unlikely to be real-time. However, hardware devices do have a concept of time, in that they produce or consume samples at a rate specified by the hardware sampling rate. Therefore, as long as the GNU Radio processing is sufficiently fast, it can be guaranteed that samples cannot be taken from or sent to the hardware at a faster rate than the specified sampling rates. However, no guarantees can be given for the time taken for a sample received from a hardware block to be processed and then sent to a different hardware block. Real-time scheduling acts to minimise this time by maximising the priority of the GNU Radio application for the host computer operating system, so that if there is any processing to do in GNU Radio, this is performed before many other operating system tasks.

3.5.2 Reducing buffer sizes

A large buffer anywhere in the transmit or receive chain results in a long time taken for the buffer to fill before it is passed on to the next stage, hence increasing the latency. A large buffer, however, reduces the average overhead required in moving samples, as the same 'header' bits can be used for a greater number of data samples, and so gives a higher overall throughput. The choice of the size of a buffer in an application is therefore generally a compromise between the two competing factors of latency and throughput. In the SoapyLMS7 drivers, the `config.performanceLatency` parameter allows control over some of the internal buffer sizes in the drivers, by setting it as a value between 0 and 1. By default, it is set to 0.5, as a balance between latency and throughput. In this case, minimum latency was desired, so this parameter was set to 0. This value minimised the buffer sizes in this section of the drivers. This was not found to restrict the throughput for the sampling rates used, but it greatly improved the latency.

3.5.3 Increasing sampling rate

As discussed above, the time taken to fill and empty buffers is a significant contribution to the system latency. Increasing the sampling rate increases the number of samples being generated per second; with a fixed buffer size, this reduces the time taken to fill and empty buffers. Thus the ADC and DAC sampling rates were maximised at a rate of 6.7 MS/s and 3.35 MS/s respectively.

The maximum theoretically possible sampling rate for the LimeSDR is 61.44 MS/s (see table 1), but attempting higher sampling rates than the values stated above caused a segmentation fault on the host computer. The cause of this was determined to be a memory leak in the Gen2 Reader program, related to several `memcpy` calls in the implementation of the 'Reader' block. However, a solution could not be determined. The same error occurred when testing the program at high sampling rates with a USRP N210, which suggests that the problem is not specific to the LimeSDR.

3.5.4 Choice of GNU Radio source and sink blocks

The source and sink blocks in a GNU Radio flowgraph represent the connection to the physical hardware; all other blocks within the flowgraph are hardware agnostic. The software setup used initially in the project was slightly different to the final setup chosen; instead of SoapyUHD, OsmoSDR [27] was used, and therefore gr-OsmoSDR source and sink blocks were used in GNU Radio instead of UHD blocks (see section 2.4 for further details regarding the wider software ecosystem). It was found that switching from OsmosSDR to SoapyUHD caused a significant reduction in RN16-ACK latency, on the order of 450 µs. It is not clear why this change caused such a large latency reduction, as the source code for OsmoSDR was inspected and no buffers could be found. One possible reason is that the SoapyUHD code may have been implemented more efficiently than gr-OsmoSDR, resulting in greatly reduced processing time.

3.5.5 Final RN16-ACK latency achieved

The combination of these steps resulted in a mean RN16-ACK latency of 475 µs. The RN16-ACK latency therefore remained in specification the majority of the time, but still occasionally failed to comply, as there was a distribution of latency values about this mean. Using the MATLAB analysis script described above, the RN16-ACK latencies for a typical run of 1000 queries were recorded. In this run, 772 EPCs were successfully decoded. A histogram of these results is plotted in figure 11.

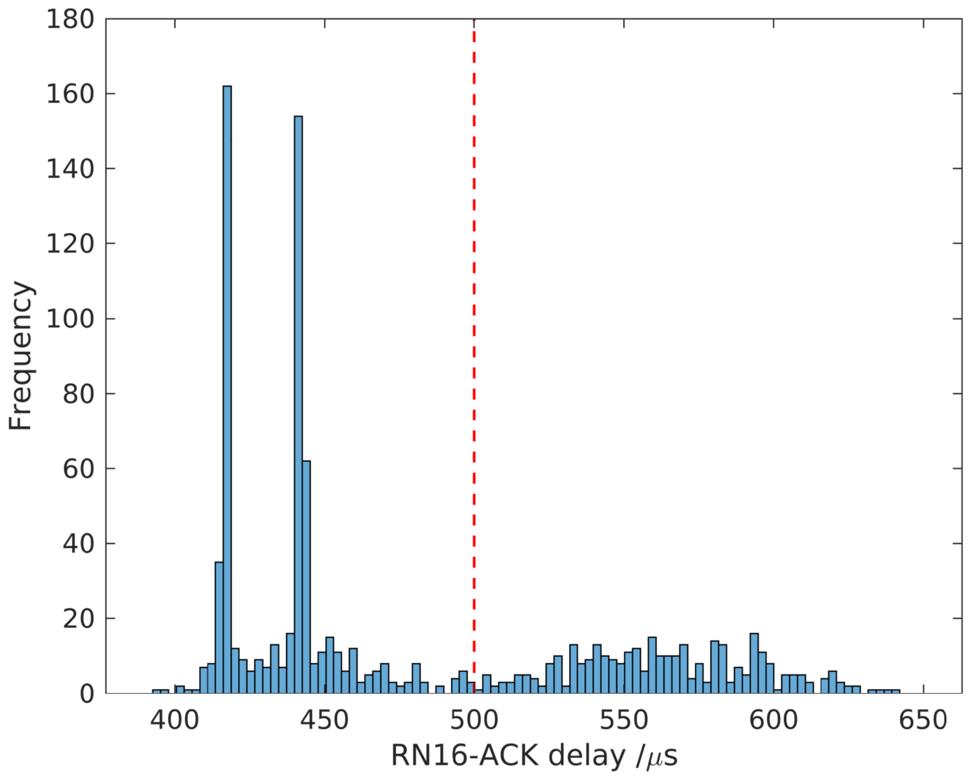


Figure 11: A histogram of the RN16-ACK latencies for a typical run of 1000 queries. The red line marks the 500 µs limit specified by the standard. Severe outliers have been excluded.

There are a few interesting aspects to note. The most striking feature is the two peaks at 417 µs and 442 µs. Note that the peaks are 25 µs apart, which corresponds to the duration of a single tag symbol. This is a result of a subtle feature of the protocol: the length of the RN16 transmitted by the tag is nominally 16 symbols; however, if the RN16 does not naturally end with a '1', a dummy '1' is appended, so that the transmission always ends on a '1'. If a dummy '1' has been appended in this way, this increases the duration of the RN16 by a single symbol, with duration 25 µs. However, the Gen2 Reader library does not wait to check for the existence of the dummy '1' before commencing processing of the RN16, so the existence of a dummy '1' reduced the measured RN16-ACK latency whenever it was appended.

Note that only 772 queries returned valid EPCs out of 1000; it was then investigated further why the remaining 228 of the queries did not return valid EPCs. Note that it was possible that these queries were returning EPCs, but these EPCs were not counted as valid due to errors introduced by noise in the channel. Another potential reason for an invalid EPC could be drift in the clock frequency of the tag; due to the low-cost and simple nature of the tag, the clock was not very accurate, so the reader may have struggled to interpret an EPC with a tag symbol rate excessively different to the nominal value.

However, the script determined that in this instance, all EPCs returned were valid (i.e. passed the CRC check), so the 228 failed queries did not result in any EPC being returned. An EPC is not returned if either the ACK sent to the tag does not match the RN16 sent by the tag, or if the RN16-ACK latency is excessive.

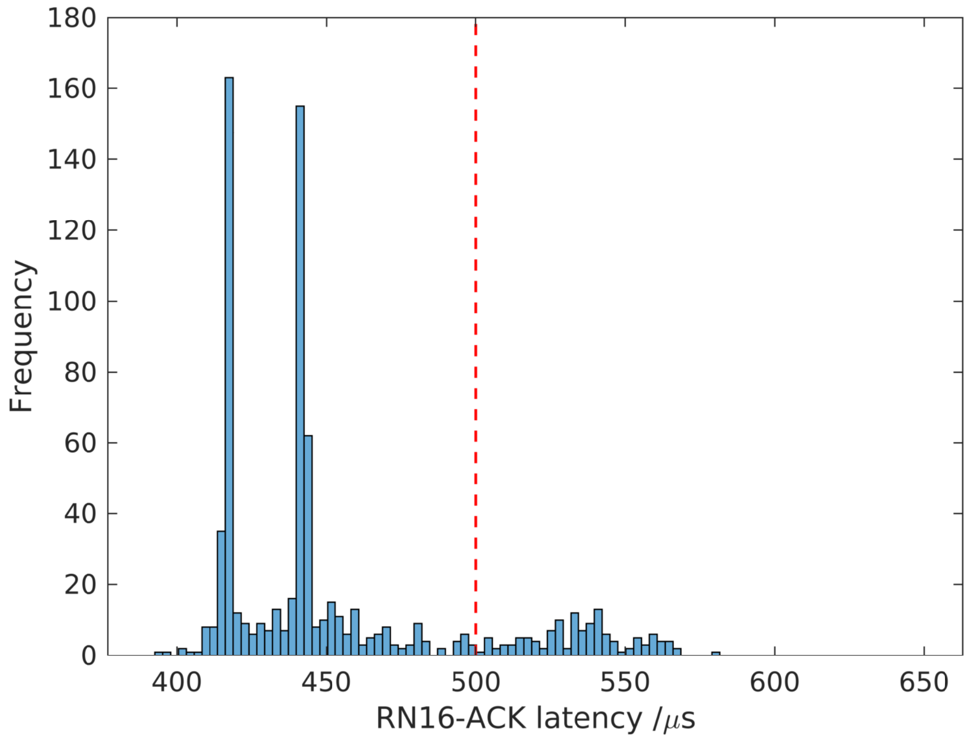
The script determined that only 17 ACKs transmitted by the reader did not match the preceding RN16, which meant that the rest of the failed queries were attributable to excessive RN16-ACK latency. Histograms of the RN16-ACK latencies which either successfully or unsuccessfully returned EPCs are given in figures 12a and 12b respectively. From figure 12b it can be seen that the vast majority of failed exchanges did indeed have an RN16-ACK latency greater than the 500 µs maximum defined in the standard, which is consistent with the idea that the majority of the failed queries were due to excessive latency. Looking at figures 12a and 12b together, the latency cut-off for this particular tag seemed to lie at approximately 550 µs, with a small degree of overlap between the two graphs. This tag had tag identifier (TID) E200-6806-CAC4-59AE.

The overall conclusion from the above analysis is that, if the conditions are otherwise favourable, with the SNR being sufficiently high at both the tag and the reader, then the read rate of this reader is limited by latency, with approximately 25% of queries exceeding the RN16-ACK latency limit. This figure of 25% was measured over multiple tests.

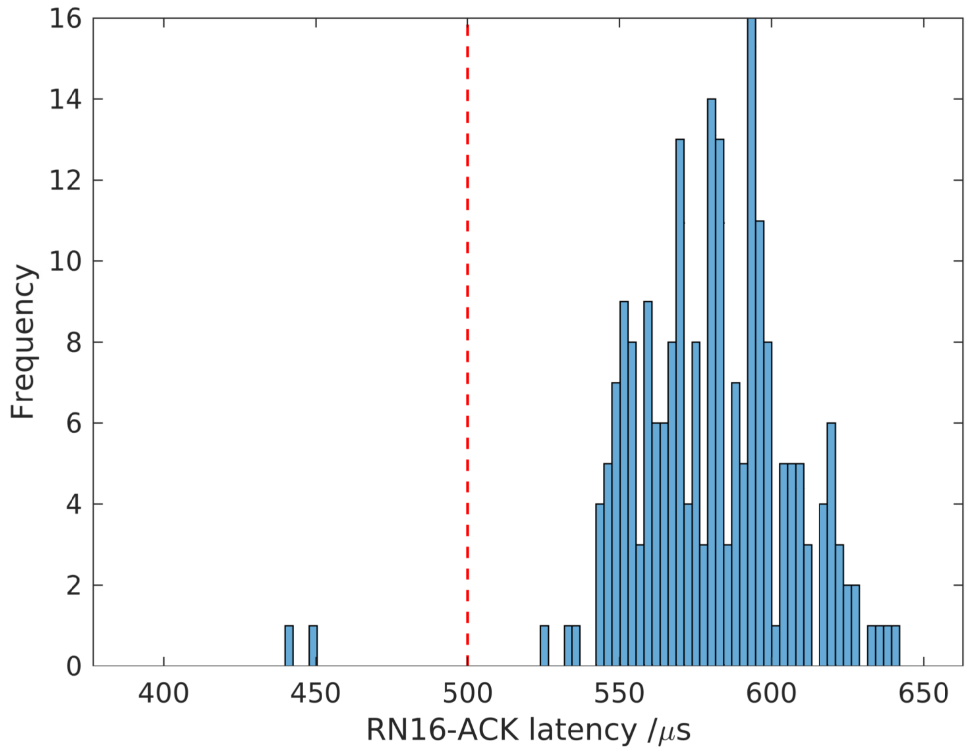
3.5.6 Effect of latency on EPC read rate

The above discussion established that increasing the RN16-ACK latency had an adverse effect on the EPC read rate. To understand this more quantitatively, artificial delays were added in the reader library to simulate the effect of increased RN16-ACK latency. For a range of values of added latency, the mean read rate and RN16-ACK latency were measured over a number of attempts, using the MATLAB script described above. The results are plotted in figure 13. Note that the graph shows the *mean* latency for the tests; there was still a distribution of latency values about each mean, which is why there were still successful EPC reads for mean latencies well above 550 µs.

From figure 13, it can be seen that the read rate drops approximately linearly with latency over much of the range, before the magnitude of the gradient decreases. One could extrapolate the curve linearly to attempt to predict the mean RN16-ACK latency which would result in a 100% read-rate; however, it is probable that this would be inaccurate, as the behaviour is unlikely to remain linear at the higher read-rates.



(a)



(b)

Figure 12: Histograms of the RN16-ACK latencies, split into exchanges were a valid EPC *was* (a) or *was not* (b) returned successfully. Note the different y-axis scales to allow the detail to be seen in both figures. A total of 1000 queries were attempted. The red lines mark the 500 µs limit specified by the standard. Severe outliers have been excluded.

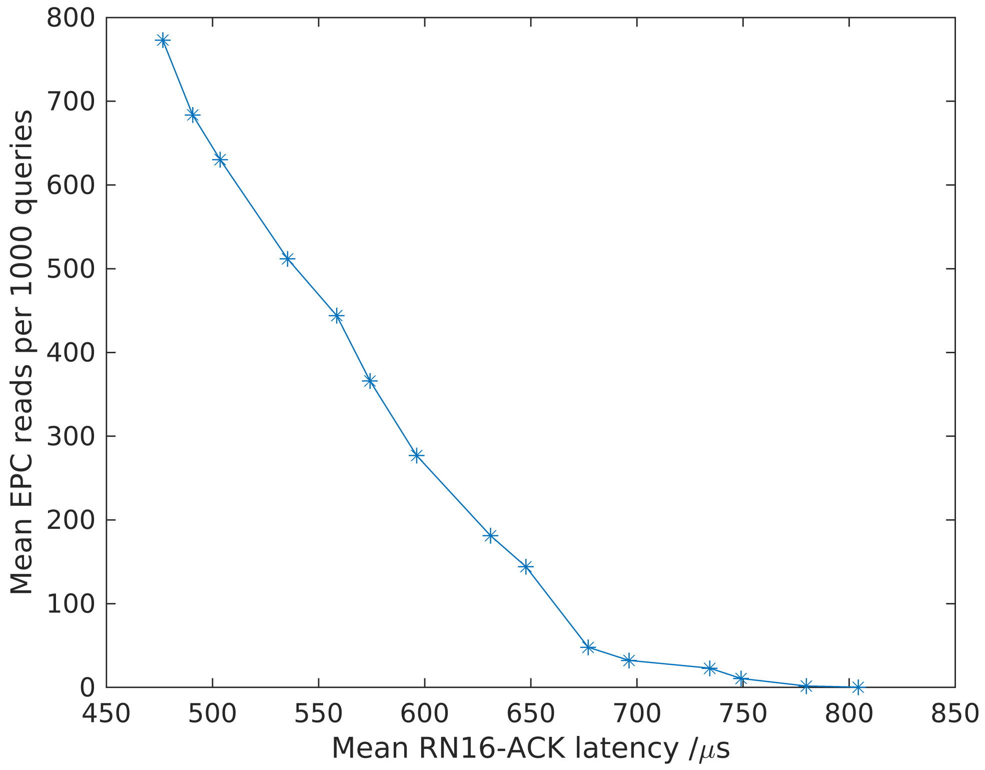


Figure 13: A plot of the mean EPC read-rate against the mean RN16-ACK latency. All measurements were carried out at an operating point where it was known that the proportion of failed EPCs due to reasons other than latency was negligible, due to the high SNR.

3.5.7 Summary of latency sources

The following is a summary of the sources of latency encountered between an RN16 arriving at the RX port and an ACK leaving the TX port of the LimeSDR, in the order encountered:

1. RF and FPGA processing
2. Filling FPGA packets to send via USB3.0
3. Transfer over USB3.0 cable
4. Lime Suite processing and RX buffering
5. SoapyLMS7, SoapySDR, and SoapyUHD processing
6. Gen2 Reader library processing in GNU Radio
7. SoapyLMS7, SoapySDR, and SoapyUHD processing
8. Lime Suite processing and formation of FPGA packets to send over USB3.0
9. Transfer over USB3.0 cable
10. RF and FPGA processing

3.5.8 Other investigated potential latency sources

In addition to the measures described above, many attempts were made to reduce latency which had no measurable impact:

- A number of different USB3.0 cables were trialled, of different lengths and from different manufacturers.
- The computer used for the project is relatively old, so the USB3.0 controller in the computer was suspected to be a source of latency. A new USB3.1 to PCI-e card was installed in the computer.
- On a forum related to the LimeSDR, some suggested that an external power supply might alleviate certain problems, rather than using power supplied directly via the USB3.0 connector. An external power supply was therefore tried.
- A low-latency version of the Ubuntu 16.04 kernel was trialled.

3.5.9 Further potential sources of latency

The following methods would potentially reduce the latency further, giving better performance, but could not be investigated due to time or resource constraints:

- The size of packets transferred via USB3.0 is defined by the FPGA gateware, as it is the size of packets which the FPGA gateware has been written to handle. If this packet size were reduced, this would potentially reduce the latency, in a manner analogous to reducing buffer sizes as discussed above. However, this packet size is hard-coded in the gateware and so might require significant changes to the gateware. Reducing USB packet sizes for a USRP device was successfully accomplished by Buettner [11], but for USB2.0 rather than USB3.0. More modern USRP devices connect via Ethernet.
- Using a more powerful computer would reduce the GNU Radio processing time, thus reducing latency. This was briefly investigated by running the project on a more powerful computer. The EPC read rate did indeed improve by a small fraction, suggesting that this is a valid method for reducing latency; however, the computer was not available for extended use for this project.
- There could be further investigation into the segmentation fault problem limiting the sampling rate, as described in section 3.5.3. For example, a different GNU Radio RFID reader program might not exhibit the same issue. Increasing sampling rates could however potentially *increase* latency if the host computer were incapable of handling the increased rate of samples; this could be resolved by using a more powerful computer, in which case the benefits of the increased sampling rate could be fully realised.

- Towards the end of the project, a new GNU Radio source/sink block called gr-LimeSDR [28] was released, which did not require the intermediate layers of SoapySDR and SoapyUHD. This could potentially further reduce the latency as the drivers would be more streamlined; however, as this was only introduced after a significant proportion of testing had been carried out, it was not investigated.
- A significant improvement in latency could be achieved by implementing all or part of the reader logic on the FPGA itself. For example, the FPGA could automatically send the CW without requiring the CW samples to be sent to the LimeSDR by the host computer, and could filter the received samples to only send RN16s and EPCs to the host computer to be decoded. This would greatly reduce the number of samples being sent over the USB3.0 connection, and would reduce the processing required on the host computer. This would be especially beneficial in a multi-device MIMO application, where the host computer would be required to handle data streams from multiple SDRs.
- The base version of the Lime Suite drivers used in this project was released in November 2017; however, there have been numerous updates to the drivers since then. These updates were not incorporated into the project as they changed the behaviour of the system, and in order to carry out consistent testing and debugging a stable configuration was required. It is possible that newer versions of the drivers could operate more efficiently, and so updating the base drivers to a newer version could potentially lower the latency.

4 Evaluation of LimeSDR sensitivity

An RFID reader contains both transmitting and receiving elements, both of which affect the system performance. Therefore, to evaluate both aspects, two primary metrics were used: the sensitivity of the LimeSDR receiver, and the spectral properties of the transmitted waveform. This section describes the evaluation of the sensitivity.

The sensitivity of the reader in this context was defined as the minimum average tag symbol power at the reader for which the successful EPC read rate was at least 25%. A better sensitivity value indicated that the reader could cope with a smaller tag signal, indicating that the reader would have a larger range in a real application.

There were many configurable parameters in the setup which could affect the sensitivity; due to time constraints, the effects of only two of these were investigated: the isolation between TX and RX, and the total RX gain in the SDR.

TX-RX isolation By necessity, in a typical RFID system, the desired tag signal at the receiver is much weaker than the transmitted signal. It is therefore critical that good isolation is achieved between TX and RX, in order to avoid the TX signal dominating the signal at the RX port. The required magnitude of this isolation for acceptable signal reception is dependent on the radio, as the effect of high TX powers at the receiver is partially dependent on the noise properties of the transmitter. Thus, the effective isolation between TX and RX was varied, and the sensitivity of the reader evaluated at each isolation value.

RX gain The total gain in the RX path can be important for sensitivity: insufficient gain, and the received signal will not be large enough to be adequately sampled by the ADC, resulting in excessive quantisation noise; too much gain, and the signal will saturate the ADC, or the amplifiers may behave non-linearly and contribute noise in the form of distortion and third order intermodulation products. The sensitivity was therefore measured for a number of different RX gain values.

4.1 Experimental setup

The key requirement of the experimental setup was that the TX-RX isolation could be varied independently of the attenuation in the RX path. This resulted in the design of the setup shown in figure 14. The variable attenuator ATTN_TX was used to vary the TX-RX isolation, while ATTN_RX was used to vary the attenuation in the RX path.

It was necessary to verify that the TX-RX isolation and RX attenuation were truly being varied independently, to ensure that there was no unexpected leakage of signal through

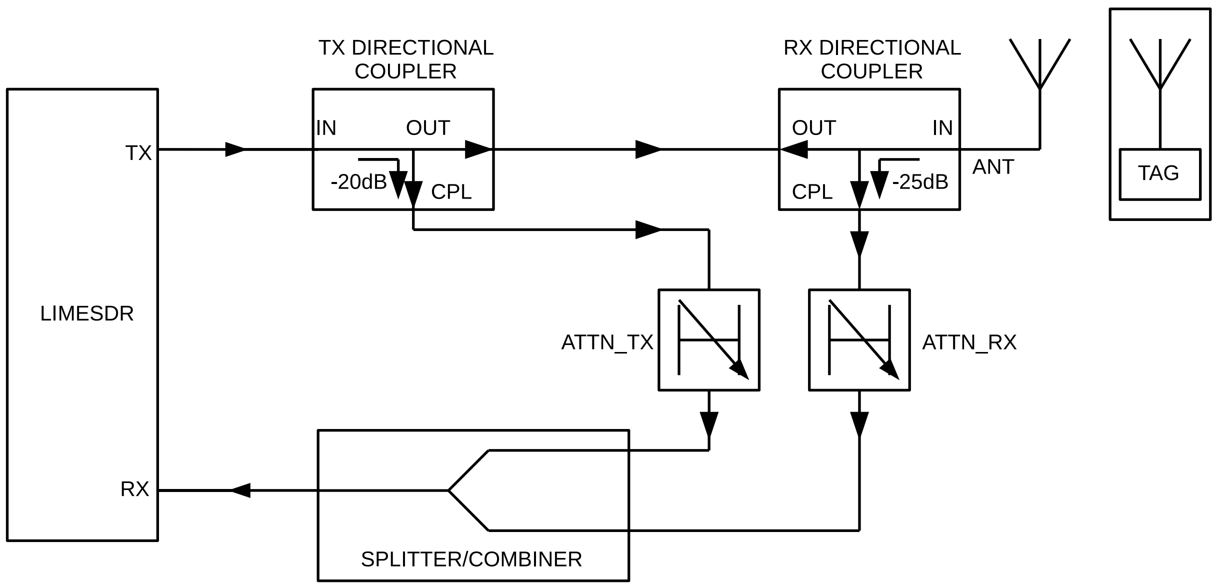
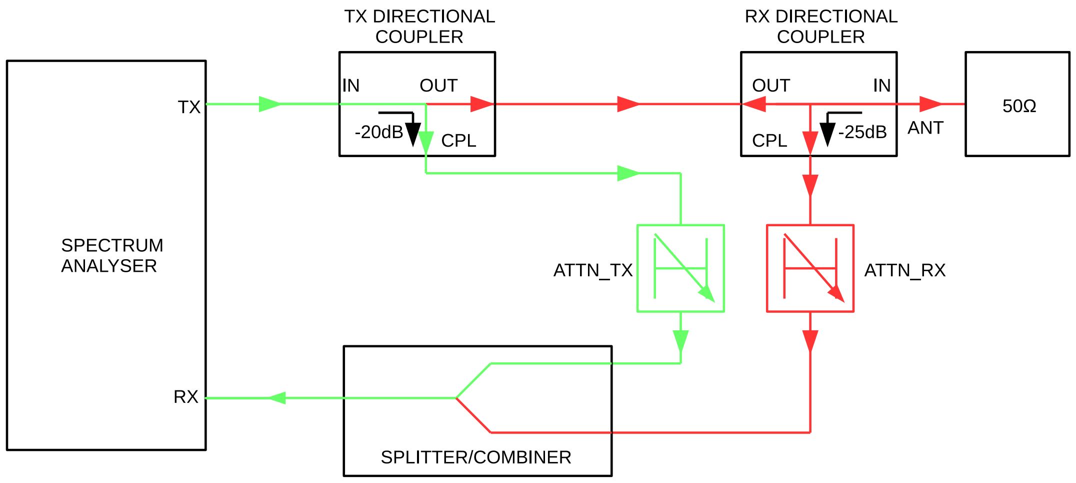


Figure 14: A block diagram of the setup used for the RX sensitivity measurements, with the desired signal paths indicated by arrows.

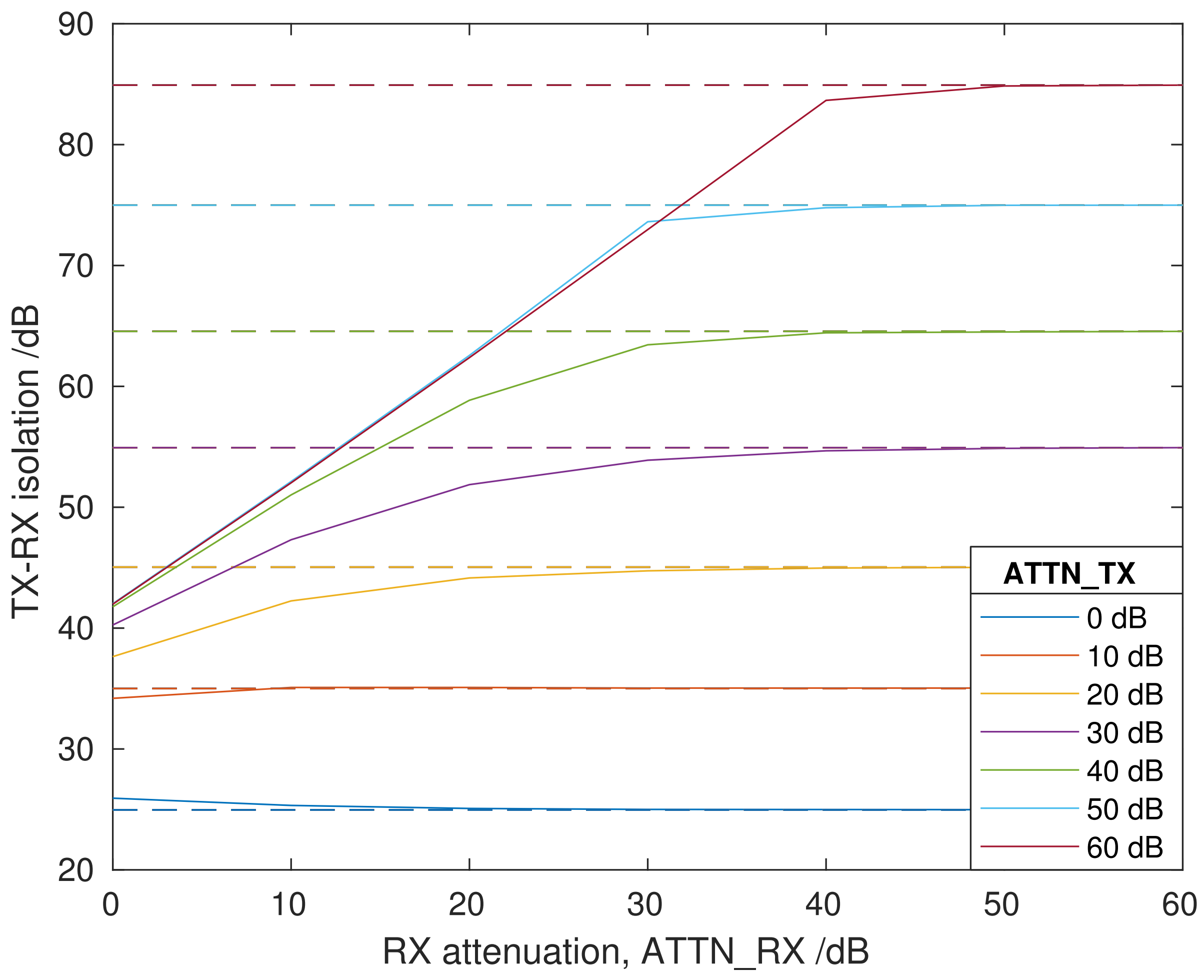
alternative paths. To check this, two experiments were carried out, as described below. Block diagrams and results of these experiments are shown in figures 15 and 16.

4.1.1 Checking isolation between TX and RX

With an ideal setup, the TX-RX isolation would be independent of the attenuation in ATTN_RX. For the setup shown in figure 15a, a sine wave at 910.04 MHz at a power of −10 dBm was applied at the TX port using a signal generator. This frequency was chosen as it was the frequency of the modulated CW during tag or reader communication. The antenna was replaced with a 50 Ω terminator, and the signal power at the RX port was measured using a spectrum analyser. The isolation was then calculated by subtracting the measured RX power from the input TX power (in dBm). For different values of ATTN_TX, the value of ATTN_RX was varied. The results are plotted in figure 15b.
From figure 15b, it can be seen that for values of ATTN_TX below 20 dB, the leakage through ATTN_TX dominated, as desired. However, the isolation between the TX port and the input to ATTN_TX was itself approximately 20 dB, whereas the unwanted leakage through the RX directional coupler was −35 dB. This meant that for higher values of ATTN_TX, and lower values of the added RX attenuation ATTN_RX, this unwanted leakage dominated, with ideal behaviour only being restored when the value of ATTN_RX was sufficiently high. While this shows that the experiment was far from ideal, it was not invalidated; it meant that when interpreting the main sensitivity experiment results, the values of ATTN_RX and ATTN_TX were both required in order to calculate the actual TX-RX isolation.



(a)



(b)

Figure 15: Figures showing the setup and results for the TX-RX isolation experiment. Figure (a) shows the setup, while figure (b) shows the results. In figure (a), the intended signal paths are shown in green, while the unwanted leakage paths are shown in red. In figure (b), each curve represents a different value of ATTN_TX. The dotted lines represent the ideal case, where the RX power is independent of the value of ATTN_RX.

This experiment made the approximation that the antenna acts as a perfect $50\,\Omega$ load, giving no reflection; in reality, the S_{11} of the antenna was measured to be -13.5 dB. The isolation between the CPL and OUT ports of the RX directional coupler was measured to be 35.0 dB, while the isolation between the IN and CPL ports was measured to be 25 dB as quoted. The isolation between the IN and OUT ports was confirmed to be negligible. This meant that the extra leakage TX signal leaving the CPL port due to the non-ideal antenna was $13.5 + 25 = 38.5$ dB lower than the incident power at the OUT port. Therefore, the actual total isolation between the CPL and OUT ports, including the effect of the antenna reflection, was 33.4 dB, giving an error in the approximation of 1.6 dB. This error is small, as it is comparable to the losses in the lengths of cable used, and does not affect the value of sensitivity obtained; it only affects the corresponding TX-RX isolation value.

4.1.2 Checking RX attenuation

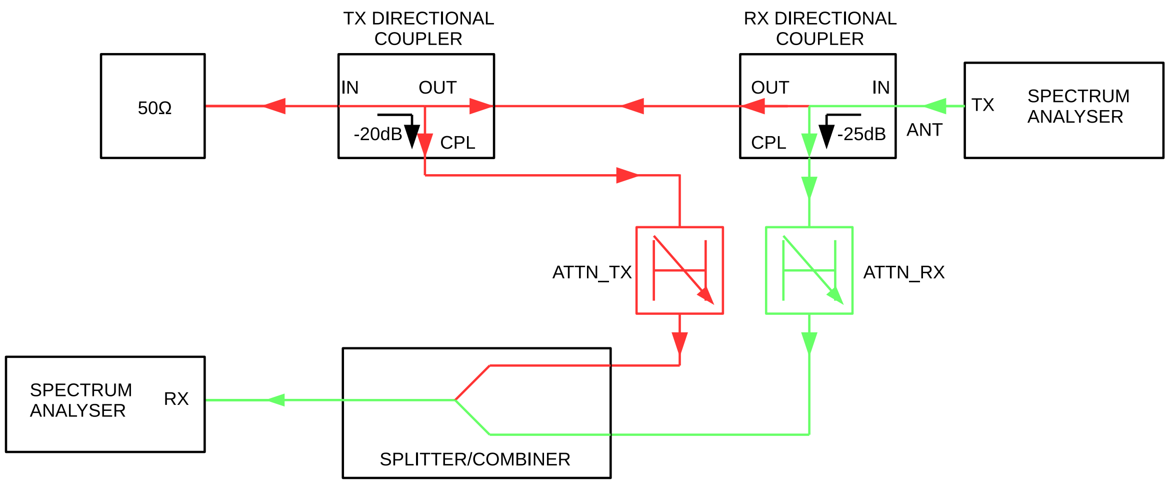
With an ideal setup, the relationship between the powers at the antenna and the RX ports would be completely independent of the value of ATTN_TX. To test this, the setup shown in figure 16a was used. A sine wave at 910.04 MHz was applied at the ANT port, at a power of -10 dBm. The TX port was terminated with a $50\,\Omega$ load, and the signal power at the RX port was measured using a spectrum analyser; the ANT-RX isolation was then calculated by subtracting the measured RX power from the input ANT power (in dBm). For different values of ATTN_TX, the value of ATTN_RX was varied. The results are plotted in figure 16b. From the figure, it can be seen that for ATTN_TX values of 30 dB or higher, the results were mostly ideal; even the curve for ATTN_TX = 20 dB remains close to the ideal case up to an ATTN_RX of 40 dB. For this reason, only values of ATTN_TX of 20 dB and above were investigated.

Note that, initially, the TX directional coupler was replaced by a splitter/combiner; however, the sensitivity experiment was carried out and it was determined that the unwanted leakage of the tag signal through the TX splitter was in many cases dominating the tag signal at the RX port. The use of a directional coupler instead added an effective 50 dB attenuation to this unwanted leakage, greatly improving the experiment.

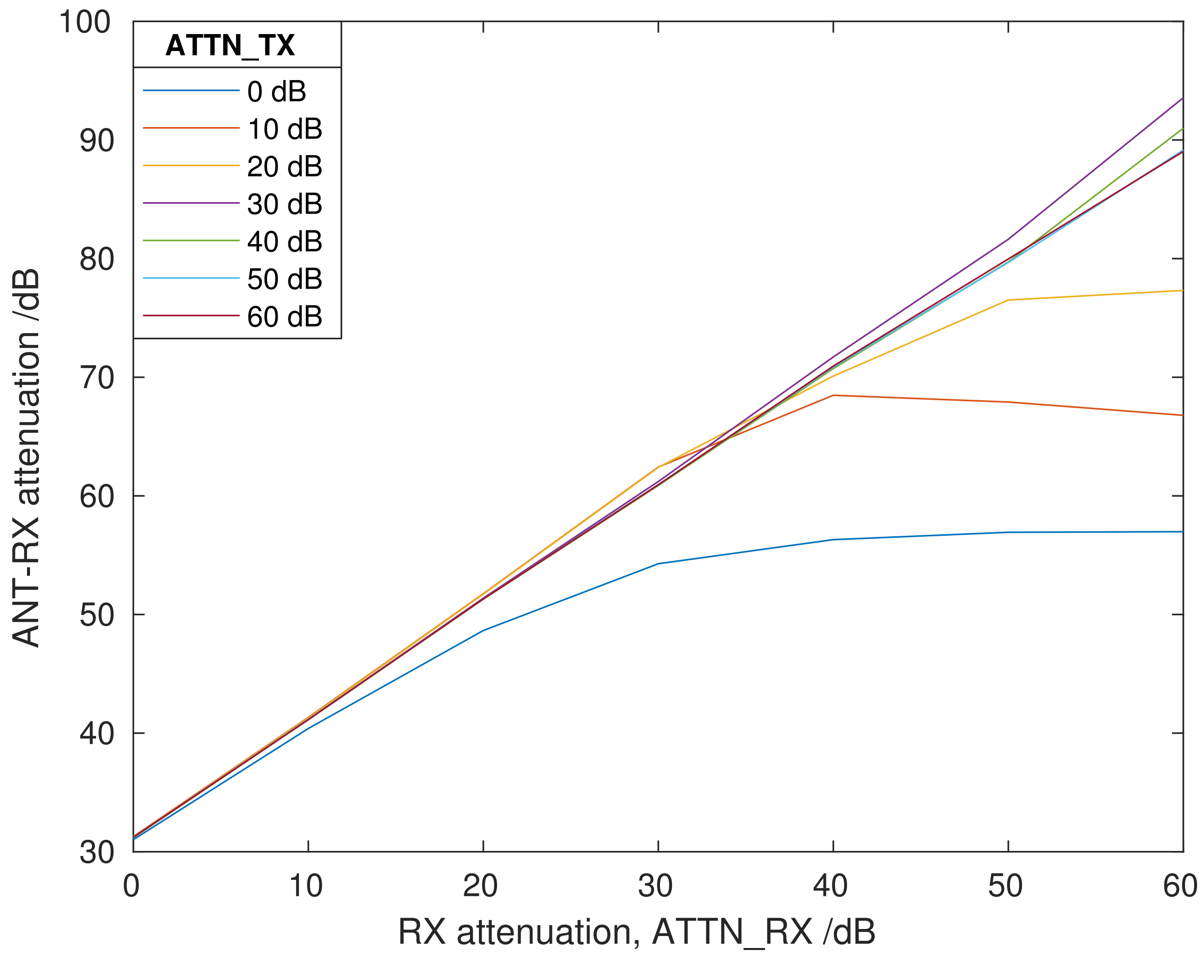
4.2 Method

Having established the validity of the experimental setup, the TX-RX isolation value (ATTN_TX) was varied between 20 dB and 60 dB in 5 or 10 dB intervals. For each of these values, the RX gain was varied in approximately 10 dB intervals from 0 dB to 60 dB.

The default algorithm in the Lime Suite drivers was used to allocate the total RX gain between the various gain stages in the RX chain. The TX gain was set to 60 dB for



(a)



(b)

Figure 16: Figures showing the setup and results for the RX attenuation experiment. Figure (a) shows the setup, while figure (b) shows the results. In figure (a), the intended signal paths are shown in green, while the unwanted leakage paths are shown in red. In figure (b), each curve represents a different value of ATTN_TX. In the ideal case the ANT-RX attenuation should rise at the same rate as the RX attenuation, giving the same straight line with gradient 1 for all curves.

all measurements, again using the default algorithm given in the Lime Suite drivers to distribute the gain between the two gain stages. This resulted in 52 dB of gain in the TXPAD, and 8 dB of gain in the TXTSP. Table 2 shows the gain distribution used for each value of RX gain.

Table 2: The gain distribution over the RX gain stages for each value of total RX gain.

Total RX gain /dB	RXLNA gain /dB	RXTIA gain /dB	RXPGA gain /dB
0	0	0	0
9	0	9	0
18	6	12	0
30	18	12	0
40	28	12	0
50	30	12	8
60	30	12	18

For each combination of TX-RX isolation and RX gain, the added RX attenuation (ATTN_RX) was increased until any further attenuation would cause the EPC read rate to drop below 25%. This limiting RX attenuation was then recorded. A separate experiment was carried out to determine the conversion between RX attenuation and sensitivity, described in the following section.

4.2.1 Determining the relationship between RX attenuation and sensitivity

It was desired to obtain a sensitivity value in terms of the tag backscatter power at the receiver. To achieve this, the RX signal that would normally go to the RX port of the LimeSDR was instead fed into another splitter/combiner; one output of this was connected to the LimeSDR RX port, and the other was connected to a spectrum analyser. This setup is shown in figure 17. The spectrum analyser was then set to track power against time, and the tag modulation was located on the trace. To calculate the tag modulation power, the high and low points of the tag modulation were converted to linear mW from dBm, the difference was found, and then this difference was converted back to dBm. This power was found to be -52.8 dBm. The extra losses introduced by the additional splitter and cables were measured, using a vector network analyser, to be 3.5 dB in total. This gave an estimated tag power at the LimeSDR RX port in normal test conditions, with ATTN_RX = 0 dB, to be -49.3 dBm. From figure 16b, it was then known that in most cases the RX attenuation would cause this power to drop by a known amount, equal to the attenuation. In this way, the limiting RX attenuation for a particular test could therefore be converted into a tag power, and hence a sensitivity.

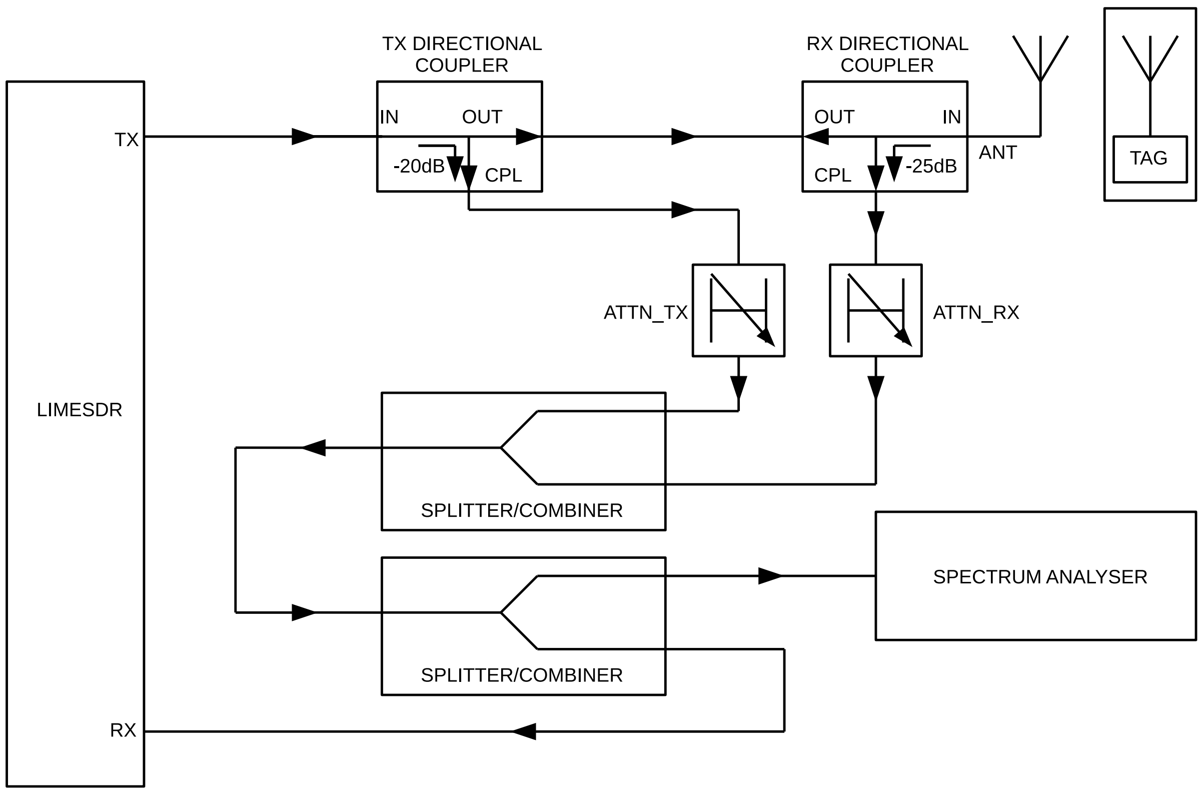


Figure 17: A block diagram for the setup used to determine the relationship between the RX attenuation value and the tag modulation power at the RX port.

4.2.2 Limitations of method

There were several factors which either complicated the method, or limited the range of valid results, which are described below. In the complete results table (table 5, appendix A), any results which were invalid due to one of the reasons below are indicated.

Variability in read rate The EPC read rate dropped over a range of values, but in the range where the read rate was changing, the read rate often varied significantly between successive repeated tests; see figure 18 for an example of this. The tests were therefore repeated several times for each attenuation value in order to increase the reliability of the results. This also gave rise to an uncertainty in the final sensitivity values of approximately ± 1 dB.

Inability to detect reader commands For higher values of TX-RX isolation, the reader was unable to detect its own commands in the received waveform; this made it impossible for the reader to continue to send queries, as it relies on detecting its own commands to time when it expects tag replies. It also does not send further reader commands until it has detected the previous command. The values of TX-RX isolation where this occurred depended, in many cases, on the value of RX gain.

Furthermore, in these instances, the occurrence of this problem was highly variable, over a wide range of RX attenuation values. This meant that for higher values of TX-RX isolation, the sensitivity was often limited by the ability of the reader to detect its own commands, as opposed to an inability to correctly decode EPCs. This variability also meant that, in these instances, the sensitivity was poorly defined.

It is likely that this is a problem with the particular GNU Radio program used, rather than an inherent limitation of the LimeSDR, as the program is able to successfully detect tag modulation of much smaller amplitude than the reader signals it fails to detect. In addition, the incidence of the error was often resolvable by increasing the RX gain, which suggests again that it was not a property of the SNR at the receiver, but a property of the reader command detection algorithm.

Receiver saturation For low values of TX-RX isolation, and with higher RX gains, the TX signal saturated the ADCs, preventing the reader from being able to interpret the tag signals.

Undesired leakage The leakage issues as described in section 4.1 were also a problem, but the effects of these could be mostly mitigated through careful interpretation of the results. Note that a Tescom TC-2600A RFID tester was also available for use in the lab, which could potentially perform a sensitivity measurement without the leakage problems. However, whilst this was initially attempted, the tester was found to work very unreliably with the LimeSDR, as the reader often failed to read any EPCs even under ideal conditions. As a result, the setup described above was designed instead.

4.3 Results

The results of the sensitivity testing are summarised in the relevant figures in the following sections, and are also given in table 5 in appendix A.

To give a better indication of a typical relationship between EPC read rate and the value of ATTN_RX, a detailed set of readings was taken for ATTN_TX = 40 dB, RX gain = 30 dB. The resulting mean EPC read rate (averaged over three attempts) against ATTN_RX is plotted in figure 18. Error bars were also plotted, giving the maximum and minimum EPC reads per 1000 queries found for each value of ATTN_RX over the three measurements. From the figure it can be seen that the read rate changed negligibly until approximately 23 dB, after which it fell until approximately 35 dB. Note that, as discussed in section 4.2.2, over the range of values where the EPC read rate was changing, the variation was very large, as can be seen by the large error bars.

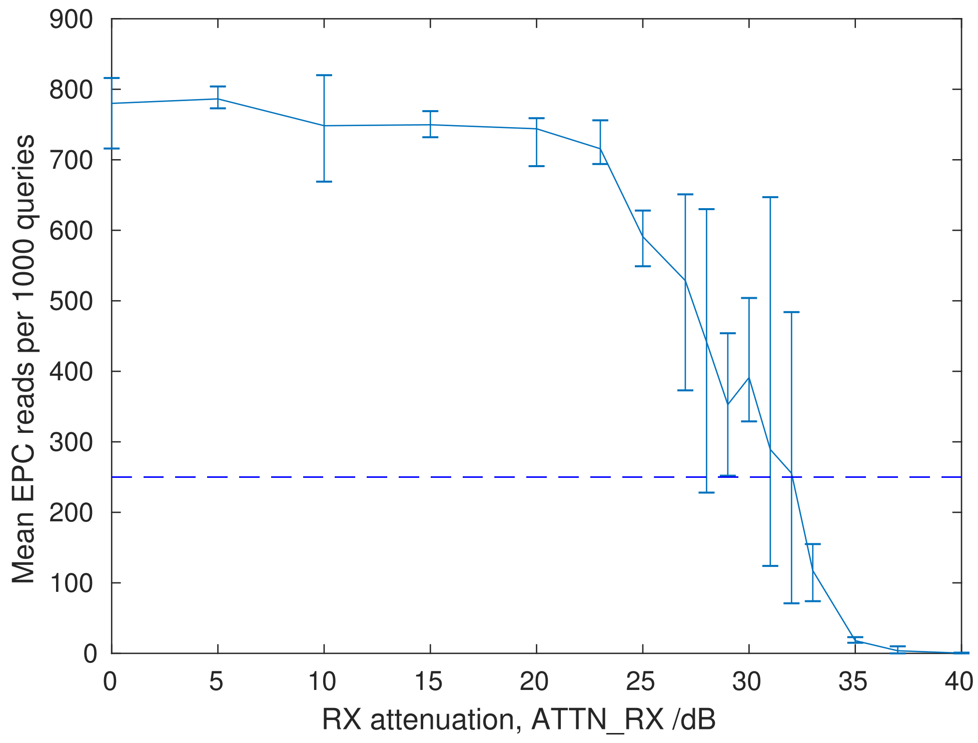


Figure 18: A plot of the mean EPC reads per 1000 queries against the value of ATTN_RX, for a typical set of parameters: ATTN_TX = 40 dB, RX gain = 30 dB. The vertical error bars represent the maximum and minimum EPC reads per 1000 queries measured at each RX attenuation. A horizontal dashed blue line has been placed at the threshold of 250 EPC reads as used in this project.

4.3.1 Effect of TX-RX isolation on sensitivity

For each value of ATTN_TX and ATTN_RX, the approximate true value of TX-RX isolation was found by linearly interpolating between the results from the TX-RX isolation experiment (figure 15b). The sensitivity for each combination of parameters was found by subtracting the corresponding limiting RX attenuation from the reference tag power of -49.3 dBm, as discussed in section 4.2.1. These values are included in table 5 in appendix A. For each value of ATTN_TX, the optimal sensitivity was then recorded. These results are summarised in table 3, and plotted in figure 19.

Table 3: A table showing the optimal sensitivity found for each value of TX-RX isolation.

TX-RX isolation /dB	Sensitivity /dBm
50.9	-56.3
60.1	-65.3
73.4	-79.3
84.8	-90.3

From figure 19 it can be seen that the optimal sensitivity varies linearly with TX-RX

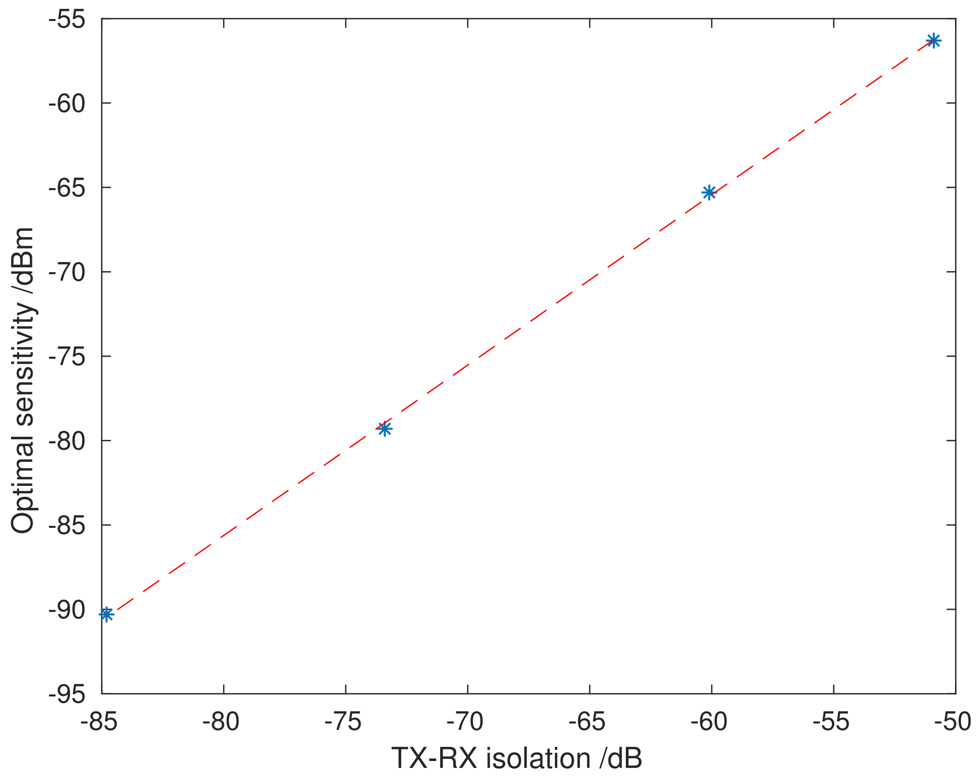


Figure 19: A plot of optimal sensitivity against TX-RX isolation. A line of best fit has also been plotted through the points, represented by the dashed-red line.

isolation, which suggests that over this range of TX-RX isolation, the factor limiting the sensitivity at the receiver was the leakage of the TX signal to the receiver. Leakage of TX signal to the receiver can cause a number of issues:

1. If the TX signal is sufficiently large, then it can saturate one or more of the stages in the receiver chain. This is relatively easy to detect by visual inspection of the waveforms.

2. A TX signal may be large enough such that it causes one or more of the receiver stages to operate in a non-linear regime, even if this does not go as far as saturation; for example, compression could occur. This would add noise in the form of non-linear distortion of the signal.

3. The in-band noise generated in the TX chain is also leaked to the receiver and cannot be filtered out.

It is worth considering item 3 further, to see where the noise is generated in the TX chain. The following list discusses some of the likely sources:

- Due to the large analogue gain in the TX chain, 52 dB here in the TXPAD, any amplitude noise (as opposed to phase noise, see below) generated in the transmitter prior to the TXPAD will be amplified in the output TX signal. Amplitude noise prior to the TXPAD is likely to come from in-band quantisation noise from the DAC, and thermal noise.

- The TXPAD amplifier itself will also contribute noise due to its noise figure.
- The TX local oscillator generates phase noise, random fluctuations in the phase of the oscillator waveform, which is then added to the TX signal in the TX mixer. This phase noise is in close proximity to the carrier frequency, as is the tag signal, which therefore means it cannot be filtered out. This is often the dominant form of transmitter noise [29]. The same local oscillator was used for both the TX and the RX mixers, which was accomplished by forcing 'TDD' mode to be enabled in the SoapyLMS7 drivers. However, radio waves take non-zero time to travel between the TX and RX mixers. This means that by the time the signal reaches the RX mixer, the phase noise has changed relative to the phase noise when the signal was generated at the TX mixer, resulting in non-zero phase noise in the baseband at the receiver. Using the same local oscillator does, however, reduce the effect of the phase noise, as for short delays, the phase noise at the TX and RX mixers is well-correlated; this effect is called range correlation [30].

For the measurements plotted in figure 19, visual inspection of the waveforms confirmed that saturation was not occurring; this was however an issue for many other parameter combinations. The linearity of the sensitivity with TX-RX isolation suggests that over the range of TX-RX isolations in the figure, the dominant noise source was not due to non-linear distortion in the RX chain either; if this were the case, then it would be expected that the sensitivity would improve more quickly at lower TX-RX isolations. Thus, it is likely that the factor limiting sensitivity was the leakage of TX noise to the receiver.

At a sufficiently high TX-RX isolation, it would be expected that eventually a different factor would limit the sensitivity, such as thermal noise in the receiver. Higher values of TX-RX isolation were tested, but unfortunately in those cases a meaningful sensitivity value could not be determined, as the reader struggled to detect its own commands; see section 4.2.2. From this data, therefore, it can be said that the inherent sensitivity of the LimeSDR configured as an RFID reader, using the Gen2 Reader library developed in [10], is at least $-90.3\,\text{dBm}$. This is comparable to the commonly used Impinj Indy R2000 RFID reader chip [31], where for a high TX-RX isolation bistatic configuration, the sensitivity at which 99% of EPC reads are successful is quoted to be $88\,\text{dB}$ for a realistic tag clock drift.

Typically achievable TX-RX isolation It is worth considering what level of TX-RX isolation is achievable in practice. In this experiment, it was possible to achieve high TX-RX isolations only when the tag signal was also attenuated, by setting ATTN_RX to a value above 0 dB. In a real scenario, the objective is to maximise this isolation whilst simultaneously minimising the attenuation of the tag signal. In this experiment, the best TX-RX isolation achieved with ATTN_RX set to 0 dB was approximately 52 dB (figure 15b). Extrapolating slightly from figure 19 suggests that the optimal sensitivity at this isolation would be approximately −55 dBm. Work in [29] suggests that even 52 dB of isolation is optimistic; it is suggested that a good isolation for a typical monostatic configuration, where the same antenna is used for transmit and receive, is 30 dB, whereas in a bistatic configuration (separate antennas), this might be improved to 40 dB. However, active cancellation techniques could be used to improve the isolation [29]. For example, in the LimeSDR, this could be implemented in analogue using the second TX channel to transmit a signal 180° out of phase with the carrier. This would then be combined with the received signal to cancel out the TX signal, after an automatically adjusted delay to account for travel time, before the signal reached the RX port. This could be controlled with the FPGA.

4.3.2 Effect of RX gain on sensitivity

The primary impact of the RX gain was to determine which factor limited the sensitivity of the reader, as opposed to directly altering the sensitivity itself. An example of this which shows the full range of behaviour is reproduced here in table 4. Table 4 shows the range of results where ATTN_TX was set to 40 dB. It can be seen that at low RX gains, the sensitivity was limited by the reader struggling to detect its own commands, whereas at high RX gains, the ADC was saturated.

At moderate RX gains, neither of these effects occurred, but the RX gain made almost no impact on the sensitivity. This suggests that over these values, the SNR at the receiver was dominated by a factor other than noise generated within the receiver, which is consistent with the idea that the noise at the receiver was dominated by noise in the TX leakage signal, as suggested in section 4.3.1.

Table 4: The sensitivity testing results for ATTN_TX = 40 dB.

RX gain /dB	Sensitivity /dBm	Notes
0	N/A	Limited by inability to detect reader signals
9	N/A	Limited by inability to detect reader signals
18	N/A	Limited by inability to detect reader signals
30	-79.3	
40	-79.3	
50	-78.3	
60	N/A	Limited by ADC saturation

5 Evaluation of LimeSDR transmitter spectrum

RFID readers are only permitted to transmit over a certain bandwidth in order to prevent excessive interference between transmitters operating in nearby channels. Bandwidth restrictions are expressed in terms of *spectral masks*, which indicate the maximum permissible transmit power in each frequency range relative to the carrier frequency. The particular spectral mask which must be adhered to depends on local regulations; for Europe, it is defined in the ETSI standard EN 302 208-1 [32]. Figure 20 shows this spectral mask, which cannot be exceeded by the transmitted signal spectrum.

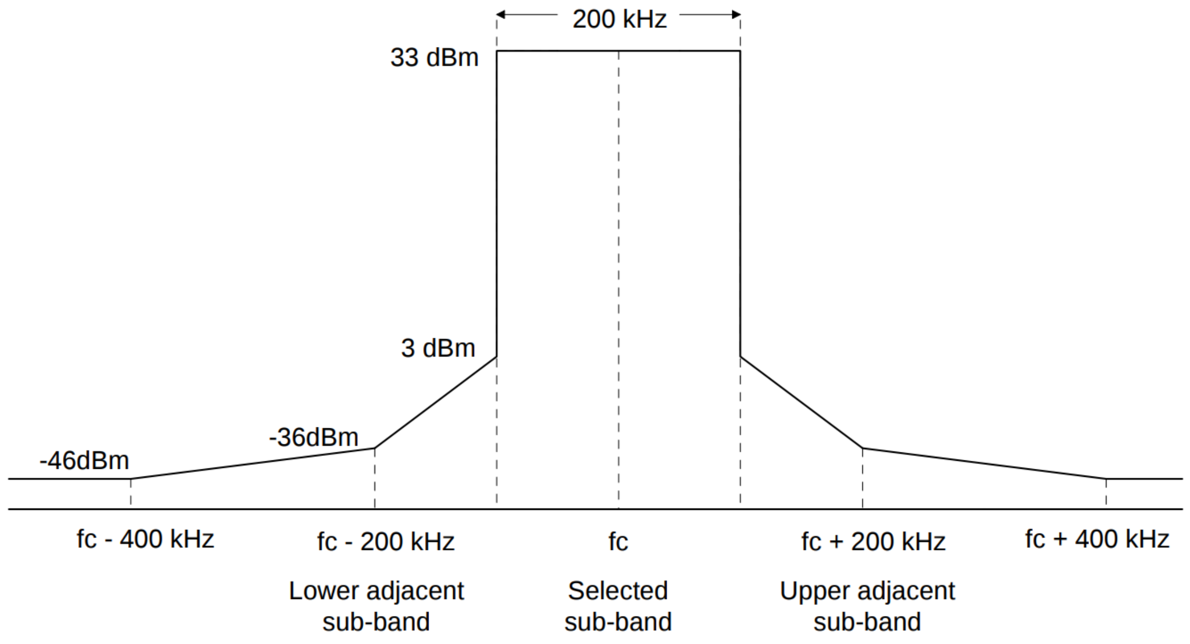


Figure 20: The European spectral mask requirements as defined in EN 302 208-1 [32]. The carrier frequency is given by f_c. Figure sourced from [32].

While the output spectrum of a transmitter is determined largely by the transmitter symbols used, the transmitter itself can also have a significant impact. A typical transmitter chain will have at least one power amplifier stage, and in the case of the LimeSDR a clock is also present. The power amplifier is likely to contribute noise in band, whereas the clock is a likely source of spurious emissions (emissions at frequencies far away from the carrier frequency), which may cause an otherwise-compliant transmitter to violate regulations.

5.1 Trade-off between reader symbol spectral width and range

No filtering was implemented in the transmitter path, either digitally or in analogue (except any filtering implemented by default in the LimeSDR). The signal sent to the LimeSDR in GNU Radio consisted of simple pulse-interval encoding (PIE), where the transmitter was simply turned on and off in order to digitally modulate the carrier wave.

This gives very deep, sharp edge transitions, which is ideal from the perspective of the tag attempting to demodulate the reader symbols; the tag can sample anywhere within the width of the pulse and detect a low power level. This maximises the range at which the tag can successfully decode the reader symbols.

However, sharp transitions in the reader result in a very large bandwidth signal, unsuitable for a commercial reader. Thus, a commercial reader would have to use transmitter symbols with a smaller spectral width, and would likely have analogue filtering at the output.

5.2 Method

To investigate the performance of the LimeSDR transmitter, a similar setup was used as for the sensitivity measurement (see figure 14). However, instead of connecting the TX coupler to the RX splitter, the coupled output from the TX splitter was plugged into a spectrum analyser. In addition, the attenuation of ATTN_RX was set to 0 dB, to ensure that the TX signal leaking through the ATTN_RX coupler was sufficient to allow the reader to detect its own commands and therefore continue to communicate with the tag. For the in-band spectrum measurement, the spectrum analyser was set to the settings defined in the ETSI standard for spectral mask measurements.

The internal TX gain of the LimeSDR was set to a total of 60 dB, close to the maximum available analogue TX gain, in order to maximise the output power; this was the same gain as used in the sensitivity testing. It is therefore possible that a better spectral performance could have been achieved with a lower TX gain, due to lower distortion, but this was not investigated due to time constraints. In addition, this gain would then have to be supplied by an external amplifier to achieve the same output power level, which may contribute its own distortion.

5.3 In-band spectrum

Figure 21 shows the unfiltered transmitted spectra from the LimeSDR, within 500 kHz of the carrier frequency, both when reading a tag (i.e. the 910 MHz carrier being modulated by a 40 kHz waveform) and when transmitting just the CW at 910 MHz. The ETSI spectral mask has also been overlaid. For the two measured curves, the power has been normalised to a peak at 33 dBm, in order to facilitate comparison. This assumes that an ideal power amplifier, with the same gain over all frequencies, would be added to the output of the LimeSDR to allow it to reach the maximum power admissible by the standard.

As can be seen in the figure, the unfiltered, modulated spectrum clearly violates the ETSI spectral mask requirements, due to the peaks at the harmonics of the modulation frequency. Digital filtering would reduce the impact of modulation, by reducing the

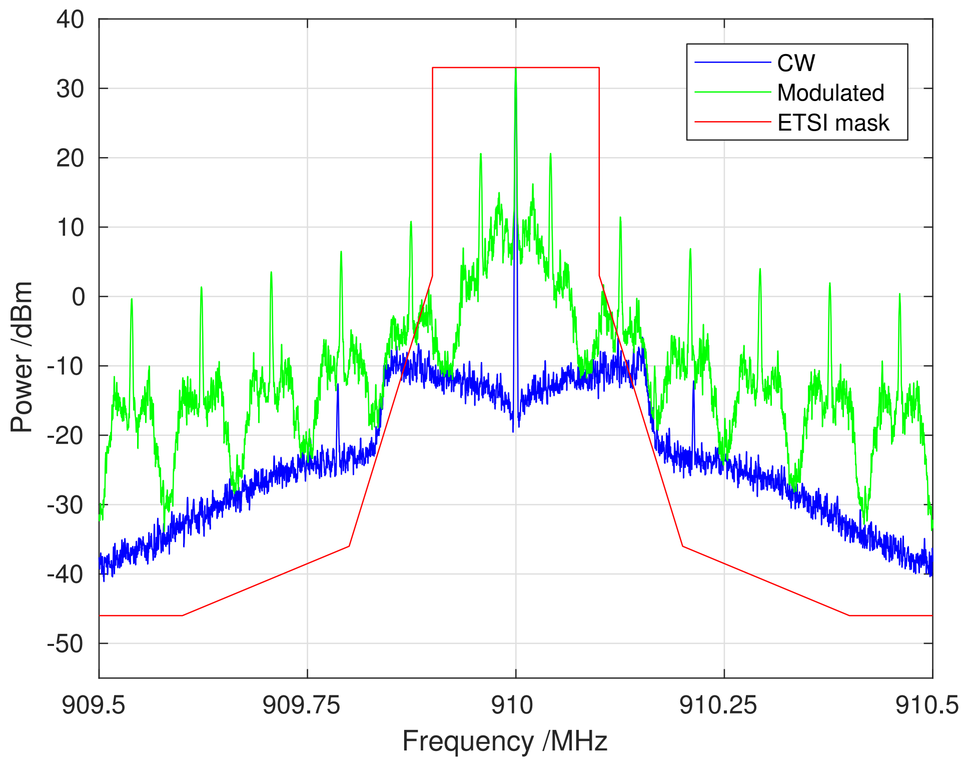


Figure 21: The output spectrum of LimeSDR transmitter, for both just the CW at 910 MHz, and the modulated waveform. The ETSI spectral mask has been overlaid. All waveforms have been normalised to a peak power of 33 dBm for comparison.

steepness of the modulation transitions and so reducing the high frequency content. In the limit, the best which digital filtering could achieve would be to remove the modulation entirely, just leaving the CW behind. Of course, this would result in no information being transmitted, but it is worth considering as a theoretical limit. Figure 21 shows that the LimeSDR transmitter also violates the ETSI spectral mask requirements even when transmitting just a CW. This indicates that in order to fulfil the ETSI requirements, at least one of the two following solutions would be needed:

1. The transmitter power would have to be reduced below the maximum allowed by ETSI. This would reduce the range at which tags could be read.

2. Analogue filtering would be required at the transmitter output to attenuate the out-of-band emissions. While the LMS7002M contains some tunable analogue low-pass filters, the lowest tunable cut-off frequency is 2 MHz [18], which is too high to have an appreciable effect on the in-band spectrum.

It is worth considering the TX spectrum of the CW further, as, due to the lack of modulation, it is defined only by LimeSDR-specific properties. Clearly, in the ideal case there would simply be a spike at 910 MHz, dropping immediately to the noise floor on either side. In the case of the LimeSDR, while the amplitude does drop by approximately 50dB immediately around the center frequency, it then *rises* with frequency by almost 10dB before it drops further. Another interesting point to note that is that there is a spike approximately 215 kHz either side of the center frequency. A possible source of this spike could be the USB power supply to the LimeSDR; it is likely that the power supply used to generate the USB power rail inside the host PC was a switch-mode supply, which would contribute noise at the switching frequency. To remove this spike the LimeSDR could be powered from a linear power supply or a battery instead, or analogue filtering could be used.

The TX spectrum was compared to that produced by a signal generator. The signal generator was set to the same output power as the LimeSDR, measured to be 9.5 dBm, and connected to the same experimental setup. The two spectra were then compared in figure 22. The plot confirms that the slow drop off outside the central 500 kHz in the LimeSDR spectrum is due to the LimeSDR transmitter, as opposed to any other properties of the setup or the spectrum analyser. In addition, the spectrum of the LimeSDR was measured for DAC sampling rates of 1, 2, and 3.35 MS/s, and no differences were found.

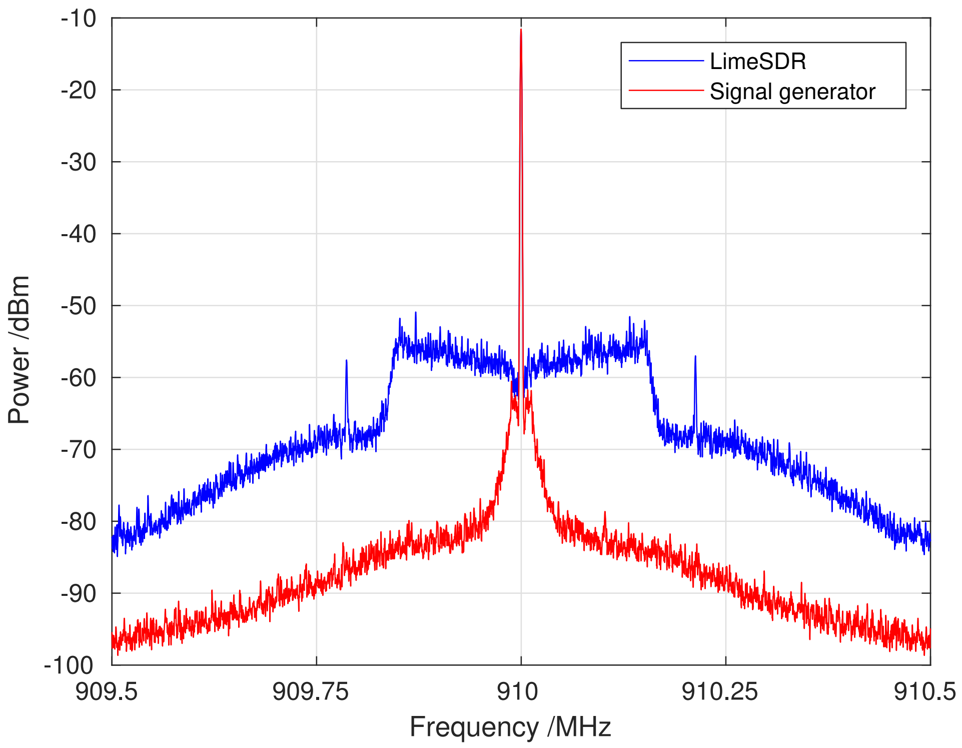


Figure 22: Comparison between the output spectrum of the LimeSDR a signal generator for a CW at 910 MHz. Note that in this case the power levels were not rescaled.

5.4 Spurious emissions

In order to locate spurious emissions further away from the center frequency, the transmitted spectrum was measured over a wide frequency range. The spectrum of the transmitter output when transmitting a CW at 910 MHz was measured over a 65 MHz span centered on 910 MHz. This spectrum is plotted in figure 23. Note the peaks at 30.72 MHz above and below the center frequency; 30.72 MHz is the frequency of the onboard oscillator on the LimeSDR board, so these peaks represent the noise contributed by the oscillator. As they are not caused by digital modulation, these peaks would need to be removed with analogue filtering. This would not be difficult given the large frequency offset of the noise peaks from the center frequency. No other significant spurious emissions were found in this frequency range.

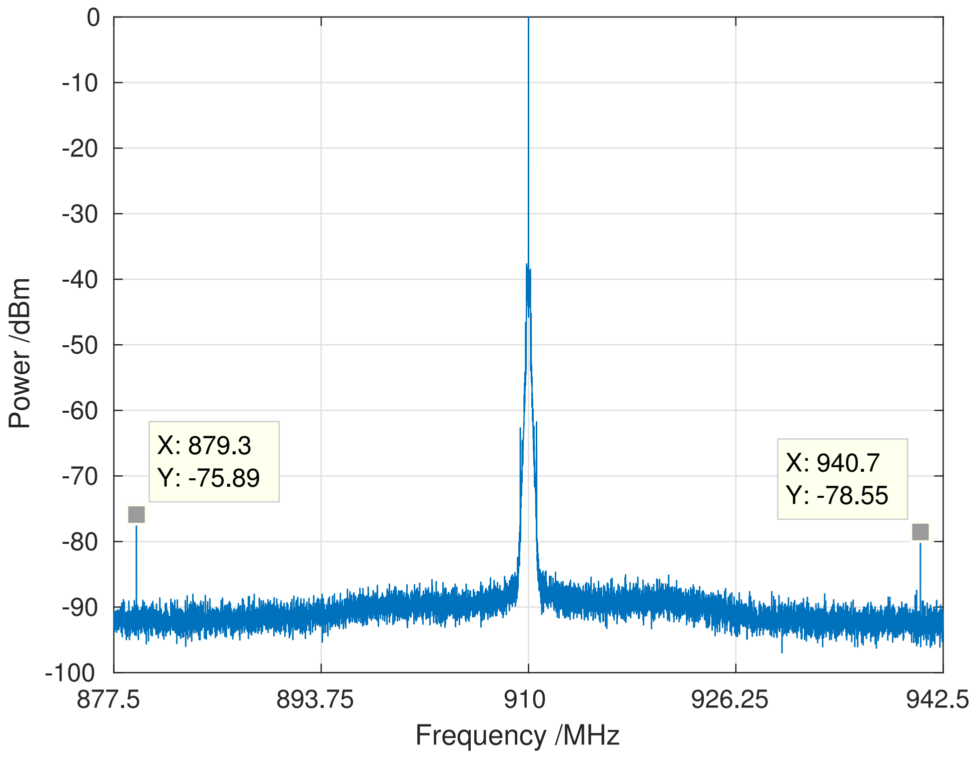


Figure 23: Wideband output spectrum of the LimeSDR transmitting a CW at 910 MHz. The frequency range is ±32.5 MHz from the center frequency. The peaks due to the onboard oscillator are marked.

6 Conclusion

6.1 Main findings

The key findings of the project include the following:

- The default LimeSDR drivers implement an erroneous tuning correction, which can be removed by modifying the SoapyLMS7 drivers.
- The default drivers implement a DC corrector which, in this context, causes an undesirable large-amplitude sawtooth-like oscillation in the received waveform. This can again be corrected by modifying the SoapyLMS7 drivers.
- The LimeSDR calibration procedure waveforms cause ACKs to become desynchronised from their corresponding RN16s; this is rectified by modifying the SoapyLMS7 drivers to allow calibration to complete before sample streaming begins.
- The mean RN16-ACK latency has been reduced from 7 ms to 475 µs. This allows the reader to successfully request and decode an EPC on approximately 75% of attempts in optimal conditions. The 25% of attempts which fail are due to excessive RN16-ACK latency.
- An offline MATLAB script has been developed to analyse RFID communications, with the ability to classify and locate RFID communication blocks from raw samples.
- With sufficient isolation between the TX and RX ports, the sensitivity of the system is at least -90.3 dBm. This is achieved at a TX-RX isolation of 84.8 dB, and is comparable to commercial RFID reader chips.
- At higher TX-RX isolations, limitations within the reader software prevent the reader from being able to detect its own modulations. This limited the range of TX-RX isolations which could be tested to a maximum of 84.8 dB.
- Over the range of TX-RX isolation values tested (50.4 dB to 84.8 dB), the sensitivity of the receiver varies linearly with TX-RX isolation. This suggests that, over this range, the receiver sensitivity is limited by transmitter noise leaking to the receiver.
- The total RX gain determines whether the receiver is saturated, limited by TX-RX isolation, or unable to detect reader commands, but does not otherwise impact the sensitivity.
- To meet ETSI spectral mask regulations in a commercial application, the reader modulation waveform would need to be adjusted, and analogue filtering would be required at the output.

- Significant spurious peaks occur in the transmitter spectrum due to the onboard reference oscillator, offset from the carrier frequency by the oscillator frequency.

6.2 Ideas for future work

Potential ideas for future work include the following:

- Attempt to reduce the RN16-ACK latency further by investigating some of the suggestions in section 3.5.9.
- Implement a TX-leakage cancellation algorithm in the FPGA in order to improve the effective TX-RX isolation.
- Investigate further the problem of the Gen2 Reader library being unable to detect the reader commands when at low amplitude, and potentially resolve it by modifying the reader command detection algorithm.
- Extend the Gen2 Reader library to support further features of the protocol, such as additional modulation schemes. Alternatively, a different Gen2 Reader library with the desired functionality could be investigated.

6.3 Wider implications

The key implication of the above is that the LimeSDR has now been successfully configured as an RFID reader, which gives an alternative, low-cost option for SDR RFID research. This has already attracted interest from other researchers, of which some have been able to successfully configure their LimeSDRs for RFID as a direct result of this project. This may be especially useful in MIMO research with multiple devices, where the cost per device may otherwise exceed budgetary constraints.

In addition, the sensitivity of the LimeSDR has been characterised over a range of RX gains and TX-RX isolations, which will enable better evaluation of whether the LimeSDR is suitable for a particular application.

Finally, the TX output spectrum has been characterised in relation to ETSI spectral mask limitations, which provides an indication of the extent of filtering which would be required in a commercial application.

A Tabulated results

Table 5: A table showing the results of the sensitivity tests. Many of the results are marked ‘N/A’; for these, a reason is given in the ‘Notes’ column. Refer to section 4.2.2 for further detail.

RX gain /dB	ATTN_TX /dB	Limiting ATTN_RX /dB	Estimated TX-RX isolation	Sensitivity /dBm	Notes
0	20	6	50.4	-55.3	
9	20	6	50.4	-55.3	
18	20	7	50.9	-56.3	
30	20	7	50.9	-56.3	
40	20	N/A	N/A	N/A	ADC saturated
50	20	N/A	N/A	N/A	ADC saturated
60	20	N/A	N/A	N/A	ADC saturated
0	30	N/A	N/A	N/A	Unable to detect reader signals
9	30	N/A	N/A	N/A	Unable to detect reader signals
18	30	N/A	N/A	N/A	Unable to detect reader signals
30	30	16	60.1	-65.3	
40	30	N/A	N/A	N/A	ADC saturated
50	30	N/A	N/A	N/A	ADC saturated
60	30	N/A	N/A	N/A	ADC saturated
0	40	N/A	N/A	N/A	Unable to detect reader signals
9	40	N/A	N/A	N/A	Unable to detect reader signals
18	40	N/A	N/A	N/A	Unable to detect reader signals
30	40	30	73.4	-79.3	
40	40	30	73.4	-79.3	
50	40	29	73.0	-78.3	
60	40	N/A	N/A	N/A	ADC saturated
0	50	N/A	N/A	N/A	Unable to detect reader signals
9	50	N/A	N/A	N/A	Unable to detect reader signals
18	50	N/A	N/A	N/A	Unable to detect reader signals
30	50	N/A	N/A	N/A	Unable to detect reader signals
40	50	41	84.8	-90.3	
50	50	39	84.7	-88.3	
60	50	40	84.8	-89.3	
0	55	N/A	N/A	N/A	Unable to detect reader signals
9	55	N/A	N/A	N/A	Unable to detect reader signals
18	55	N/A	N/A	N/A	Unable to detect reader signals
30	55	N/A	N/A	N/A	Unable to detect reader signals
40	55	N/A	N/A	N/A	Unable to detect reader signals
50	55	N/A	N/A	N/A	Unable to detect reader signals
60	55	N/A	N/A	N/A	Unable to detect reader signals
0	60	N/A	N/A	N/A	Unable to detect reader signals
9	60	N/A	N/A	N/A	Unable to detect reader signals
18	60	N/A	N/A	N/A	Unable to detect reader signals
30	60	N/A	N/A	N/A	Unable to detect reader signals
40	60	N/A	N/A	N/A	Unable to detect reader signals
50	60	N/A	N/A	N/A	Unable to detect reader signals
60	60	N/A	N/A	N/A	Unable to detect reader signals

B Risk Assessment Retrospective

The primary hazards identified in the risk assessment submitted at the beginning of the project were electrical, hazardous substances, and computer use. Additional hazards were identified during the project.

Electrical All voltages used throughout the project were initially assessed to be lower than 50 V; this was accurate in hindsight.

Hazardous substances The hazardous substances risk was identified due to the potential requirement for soldering during the project. Soldering was only carried out on one occasion during the project, and departmental guidelines were followed in this instance. Thus the risk assessment was accurate in this regard.

Computer use The project was predicted to require extensive computer use, as this was required to control the radio. This was accurate in hindsight.

Cycling to the Electrical Engineering building A risk not considered in the initial risk assessment was the risk involved in cycling to the Electrical Engineering building on the West Cambridge site. All of the experimental work was carried out in this building due to the requirement for laboratory equipment such as spectrum analysers and vector network analysers, as well as the computer used for the project; these were not available in the Trumpington site building, which would have been within walking distance. To minimise the risks involved, the relevant traffic laws were followed, and cycle paths were used wherever possible.

Electromagnetic radiation risk Another risk not considered in the initial risk assessment was the risk due to electromagnetic radiation produced by the SDR transmitter. The maximum power output of the transmitter was 9.5 dBm, or approximately 10 mW, of which the majority reached the antenna. To assess whether this was a safe level, this was compared to safe mobile phone power limits. For example, the Federal Communications Commission of the USA states that a safe specific absorption rate is $1.6\,\mathrm{Wkg}^{-1}$ for radiation from a mobile phone [33]. Thus, to exceed this safe limit, the entire radiation from the antenna would have to be absorbed within 0.006 25 kg of tissue, which could not occur under any reasonable circumstances. The radiation risk contributed by the experiment was therefore deemed to be negligible.

Summary In summary, all risks identified at the outset of the project were accurately assessed and appropriately managed. The only significant risk not initially considered was the risk involved in cycling to and from the Electrical Engineering building on a regular basis; the electromagnetic radiation risk was found to be negligible.

References

[1] MyriadRF, “LimeSDR.” [Online]. Available: https://myriadrf.org/projects/limesdr/ (Accessed 2018-05-26).

[2] C. Swedburg, “RFID Market for Retailers Forecast to Grow 39% Annually,” 2015. [Online]. Available: http://www.rfidjournal.com/articles/view?13104 (Accessed 2018-05-26).

[3] F. Zheng and T. Kaiser, “MIMO for RFID,” in *Digital Signal Processing for RFID*. Chichester, UK: John Wiley & Sons, Ltd, apr 2016, pp. 95–137. [Online]. Available: http://doi.wiley.com/10.1002/9781118824269.ch5

[4] M. Crisp, R. V. Penty, I. H. White, and A. Bell, “Wideband Radio over Fiber Distributed Antenna Systems for Energy Efficient In-Building Wireless Communications,” in *2010 IEEE 71st Vehicular Technology Conference*. IEEE, 2010, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/5493764/

[5] F. Galler, S. Grebien, T. Faseth, K. Witrisal, G. Magerl, and H. Arthaber, “Extension of an SDR UHF RFID Testbed for MIMO and Monostatic Time of Flight Based Ranging,” *IEEE Journal of Radio Frequency Identification*, vol. 1, no. 1, pp. 32–38, mar 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8038235/

[6] L. Görtschacher, J. Grosinger, H. N. Khan, B. Auinger, D. Amschl, P. Priller, U. Muehlmann, and W. Bösch, “SIMO UHF RFID-Lesegerät zur Tag-Lokalisierung mittels Sensorfusion in einer ausgewählten Umgebung,” *Elektrotechnik und Informationstechnik*, vol. 133, no. 3, pp. 183–190, jun 2016. [Online]. Available: http://link.springer.com/10.1007/s00502-016-0407-9

[7] F. Le Roy, T. Quiniou, A. Mansour, R. Lababidi, and D. Le Jeune, “RFID Eavesdropping Using SDR Platforms,” pp. 208–214, 2018.

[8] L. Catarinucci, D. De Donno, R. Colella, F. Ricciato, and L. Tarricone, “A cost-effective SDR platform for performance characterization of RFID tags,” *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 4, pp. 903–911, apr 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6094207/

[9] Ettus Research, “USRP N210 Software Defined Radio (SDR).” [Online]. Available: https://www.ettus.com/product/details/UN210-KIT (Accessed 2018-01-05).

[10] N. Kargas, F. Mavromatis, and A. Bletsas, “Fully-Coherent reader with commodity SDR for Gen2 FM0 and computational RFID,” *IEEE Wireless Communications Letters*, vol. 4, no. 6, pp. 617–620, dec 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7236869/

[11] M. Buettner and D. Wetherall, “A "Gen 2" RFID monitor based on the USRP,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 3, p. 41, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1823844.1823850

[12] LuaRadio, “SDR general architecture.” [Online]. Available: http://luaradio.io/new-to-sdr.html (Accessed 2018-05-05).

[13] D. Dobkin, *The RF in RFID.* Newnes, 2008.

[14] ISO, “ISO/IEC 18000-6 - Information technology. Radio frequency identification for item management. Parameters for air interface communications at 860 MHz to 960 MHz General,” *ISO Standards*, 2013. [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=59643

[15] Lime Microsystems, “Field programmable RF ICs: LMS7002M.” [Online]. Available: http://www.limemicro.com/products/field-programmable-rf-ics-lms7002m/ (Accessed 2018-05-05).

[16] MyriadRF, “LimeSDR Made Simple Part 1: Introduction.” [Online]. Available: https://myriadrf.org/blog/limesdr-made-simple-part-1/ (Accessed 2018-05-06).

[17] Altera, “Cyclone IV datasheet,” 2009. [Online]. Available: https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html (Accessed 2018-05-06).

[18] MyriadRF, “LimeSDR-USB hardware description.” [Online]. Available: https://wiki.myriadrf.org/LimeSDR-USB_hardware_description (Accessed 2018-05-06).

[19] MyriadRF, “LimeMicro:LMS7002M Datasheet.” [Online]. Available: https://wiki.myriadrf.org/LimeMicro:LMS7002M_Datasheet#Functional_block_diagram (Accessed 2018-05-06).

[20] GNURadio, “GNU Radio,” 2016. [Online]. Available: http://gnuradio.org/ (Accessed 2018-01-05).

[21] MyriadRF, “Lime Suite.” [Online]. Available: https://myriadrf.org/projects/lime-suite/ (Accessed 2018-01-05).

[22] MyriadRF, “SoapyLMS7 - SoapySDR LimeSuite bindings.” [Online]. Available: https://github.com/myriadrf/LimeSuite/tree/master/SoapyLMS7 (Accessed 2018-01-05).

[23] Pothosware, “SoapySDR.” [Online]. Available: https://github.com/pothosware/SoapySDR/wiki (Accessed 2018-01-05).

[24] Pothosware, "SoapyUHD." [Online]. Available: https://github.com/pothosware/SoapyUHD/wiki (Accessed 2018-05-06).

[25] G. L. Turin, "An Introduction to Matched Filters," *IRE Transactions on Information Theory*, vol. 6, no. 3, pp. 311 – 328, 1960.

[26] MathWorks, "MathWorks," 2018. [Online]. Available: https://uk.mathworks.com/ (Accessed 2018-05-13).

[27] Osmocom, "GrOsmoSDR - SDR (Software Defined Radio)." [Online]. Available: https://osmocom.org/projects/sdr/wiki/GrOsmoSDR (Accessed 2018-01-05).

[28] MyriadRF, "gr-LimeSDR: LimeSDR plugin for GNU Radio." [Online]. Available: https://github.com/myriadrf/gr-limesdr (Accessed 2018-05-08).

[29] A. Boaventura, J. Santos, A. Oliveira, and N. B. Carvalho, "Perfect Isolation: Dealing with Self-Jamming in Passive RFID Systems," *IEEE Microwave Magazine*, vol. 17, no. 11, pp. 20–39, nov 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7590203/

[30] B.-J. Jang and H. Yoon, "Range Correlation Effect on the Phase Noise of an UHF RFID Reader," *IEEE Microwave and Wireless Components Letters*, vol. 18, no. 12, pp. 827–829, dec 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4686748/

[31] Impinj, "Impinj Indy R2000 RAIN RFID High Performance Reader Chip." [Online]. Available: https://www.impinj.com/platform/connectivity/indy-r2000/ (Accessed 2018-05-29).

[32] ETSI, "EN 300 422-1 V1.3.2 - Electromagnetic compatibility and Radio spectrum Matters (ERM); Wireless microphones in the 25 MHz to 3 GHz frequency range; Part 1: Technical characteristics and methods of measurement," *European Standard (Telecommunications series)*, vol. 2, pp. 1–41, 2008. [Online]. Available: http://www.etsi.org/deliver/etsi_en/302200_302299/30220801/01.03.01_60/en_30220801v010301p.pdf

[33] Federal Communications Commission, "Specific Absorption Rate (SAR) for Cellular Telephones." [Online]. Available: https://www.fcc.gov/general/specific-absorption-rate-sar-cellular-telephones (Accessed 2018-05-25).