



OPEN HARDWARE 2016 DESIGN CONTEST

TEAM NUMBER XIL-29448

FPGA-Based All-Digital Multi-protocol RFID Reader

Author:

João Borges dos Santos
(joaoricardo@ua.pt)

Supervisor:

Dr. Arnaldo S. R. Oliveira
(arnaldo.oliveira@ua.pt)

TELECOMMUNICATIONS INSTITUTE - UNIVERSITY OF AVEIRO



Saturday 18th June, 2016

Contents

1	Introduction	6
1.1	Document's organization	7
1.2	Xilinx Software	7
1.3	Project's folder list organization	8
2	System Design	11
2.1	Overview	11
2.2	External Hardware requirements - Analog Physical Layer	12
2.3	HDL Hardware Developed - Digital Physical Layer	13
2.3.1	Overview Developed Custom IP Cores	14
2.4	Software Developed - Media Access Control (MAC) Layer	24
2.4.1	Description: Team Developed Functions	31
2.4.2	Description: Third Party Developed Functions	37
2.5	How to assemble the RFID Reader	37
2.6	Design Reuse	41
3	Results	42
3.1	Reader Multi-Standard Agility	43
3.2	Transmitted power mask at UHF Band	44
3.3	RFID reader maximum range	45
3.4	FPGA resource occupation	46
4	Conclusion	47
	Appendix A RF Front-End Schematic	48

List of Figures

2	Main directory.	8
3	VIVADO project hardware directory.	8
4	Custom IP repository directory.	9
5	SDK source directory.	10
6	System block diagram.	11
7	Custom_IP_ADRxTx.	14
8	Custom_IP_MGT_TX_AMPLITUDE.	18
9	Custom_IP_BP_CONFIGURATION.	19
10	Custom_IP_RFID_DECOCER.	21
11	Custom_IP_IIC.	23
12	Ethernet communication state machine.	25
13	EPC Gen2 RFID protocol state machine.	28
14	MIFARE RFID protocol state machine.	30
15	Change IP Address.	38
16	Platform was successfully initialized.	39
17	Status LED after Xilinx board initialization.	39
18	RFID Ethernet commands.	40
19	Status LED after Xilinx board initialization.	40
20	Project repository.	41
21	Experimental Setup.	42
22	RFID signal transmitter power spectrum.	43
23	Implemented RFID reader UHF Power Mask.	44
24	Implemented RFID reader maximum range.	45
25	Percentage of occupation.	46
26	Implemented RF Front-End electric schematic.	49

List of Tables

1	Custom_IP_ADRxTx register table.	16
2	Custom_IP_ADRxTx ports interface description.	17
3	Custom_IP_MGT_TX_AMPLITUDE register table.	18
4	Custom_IP_MGT_TX_AMPLITUDE port description.	19
5	Custom_IP_BP_CONFIGURATION.	20
6	Custom_IP_BP_CONFIGURATION port description.	20
7	Custom_IP_RFID_DECOCER port description.	21
8	Custom_IP_RFID_DECOCER.	22
9	Custom_IP_IIC port description.	23
10	Ethernet commands.	26
11	Transmission mask measured at UHF.	44
12	Implemented RF Front-End commercial components.	48

List of Acronyms

A/D	Analog-to-Digital
ADR_xT_x	All-Digital Receiver/Transmitter
ASK	Amplitude Shift Keying
ATQA	Answer to Request, Type A
BRAM	Block Random Access Memory
CRC	Cyclic Redundancy Check
DC	Direct Current
DDC	Digital Down Conversion
DDS	Digital Direct-Synthesis
EPC	Electronic Product Code
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
FIR	Finite Impulse Response
FM0	Bi-phase space encoding
FPGA	Field-Programmable Gate Array
F_s	Sampling Frequency
HDL	Hardware Description Language
HF	High Frequency
IF	Intermediate Frequency
IO	Input/Output
IoT	Internet of Things
ISO	International Standards Organization
lwIP	Lightweight IP
MAC	Media Access Control
MGT	Multi-Gigabit Transceiver
PWM	Pulse-Width Modulation
RBW	Resolution Bandwidth
REQA	Request Command, Type A
RF	Radio Frequency

RFID	Radio Frequency IDentification
RN16	16-bit Random or pseudo-Random Number
Rx	Reception
SDR	Software-Defined Radio
Tx	Transmission
UART	Universal Asynchronous Receiver/Transmitter
UHF	Ultra High Frequency
UID	Unique Identifier
VBW	Video Bandwidth

1 Introduction

The World is increasingly becoming more connected in a way that everyone and everything are interconnected. This concept is also known by Internet of Things (IoT) [1]. However, currently the disconnected devices and the way to connect them is still an important issue, that can be addressed by Radio Frequency Identification (RFID) technology.

An RFID reader capable to handle multiple Radio Frequency (RF) bands in an IoT environment, needs to have flexibility and adaptability requirements. So, to overcome these requirements, the RFID technology should present a relationship with Software-Defined Radio (SDR) architectures.

To ensure a high level of inter-operability between the physical layers and the protocol standards, the physical circuitry must be carefully designed in an efficient way. An Field-Programmable Gate Array (FPGA) seems to be a good candidate for such application, due to its reconfigurable capability allied with the high logic capacity and its high-speed IO pins. In this sense, it is understandable that several FPGA-based architectures to build fully-digital SDR transceivers have been proposed in the literature [2–4].

An RFID reader starts its reading task by exciting a tag that implements a given RFID protocol. These protocols are handled by the microprocessor (μP) implemented at the FPGA. After the baseband protocol processing, an upconversion stage is required to translate the baseband signal to an RF representation. Traditionally, this is performed in the analog domain with a mixer that multiplies the desired signal by a sinusoidal carrier (e.g. the homodyne transmitter). In an all-digital transmitter the data-path is entirely digital up to the RF stage, which provides high level flexibility. Besides that, some FPGAs have embedded serializers, called Multi-Gigabit Transceivers (MGTs) that can be used to generate an RF signal directly at the FPGA output. Therefore, FPGAs have a really important role in SDR, allowing the implementation of all-digital transmitters, providing them a greater flexibility.

Hereupon, this document features the report for the developed multi-protocol RFID reader, based on an all-digital transceiver implemented on FPGA and minimal RF front-end. The proposed radio system is a flexible RFID reader, which is able to operate at High Frequency (HF) 13.56 MHz and Ultra High Frequency (UHF) 860-960 MHz and implemented by taking advantage of the high-speed serializers/deserializers available on the FPGA.

1.1 Document's organization

This document is organized in six major sections:

- Introduction - Brief description of the implemented RFID reader and its importance for IoT environments;
- Project's folder list organization - Folder directory organization for both SDK and VIVADO projects;
- System Design - Detailed description of the implemented system. It features the external hardware requirements, the HDL hardware developed, the developed software, how to assemble the RFID reader and a link to a repository with all the parts of the implemented system;
- Results - Presents the obtained results such as the multi-standard agility, the transmitted power mask and the reading range for the implemented RFID reader. An YouTube link for a video that demonstrates the implemented RFID system is provided.
- Conclusion - Final conclusions about this work;
- Appendix A - The implemented RF Front-End schematic.

1.2 Xilinx Software

This project was developed using the Xilinx software:

- VIVADO 2014.4;
- SDK 2014.4;

1.3 Project's folder list organization

This section presents the organization of the project files..

The main directory is \src.

Fig. 2 presents the files and folders presented in the main directory.

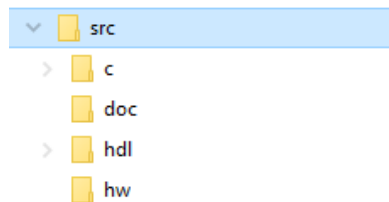


Figure 2: Main directory.

- **c**: Directory of the SDK project;
- **doc**: Directory of the documentation file;
- **hdl**: Directory of the VIVADO project hardware;
- **hw**: Directory of the binary (.bin) file.

The VIVADO project hardware directory is ...\\hdl.

In this directory there are the files and folders present in Fig. 3.

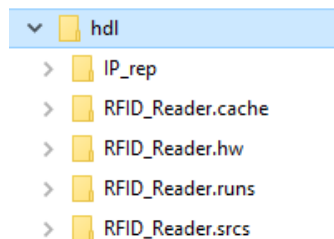


Figure 3: VIVADO project hardware directory.

- **IP_rep**: Directory of all the developed HDL IP cores;
- **RFID_Reader.cache**: Directory of the VIVADO project cache;
- **RFID_Reader.hw**: Directory of the VIVADO project hardware;
- **RFID_Reader.runs**: Directory of the VIVADO project runs;

- **RFID_Reader.srcs**: Directory of the VIVADO project sources;
- **RFID_Reader.xpr**: VIVADO project file.

The developed custom IPs for this project are under the `\src\hdl\IP_rep` directory. As shown in Fig. 4.

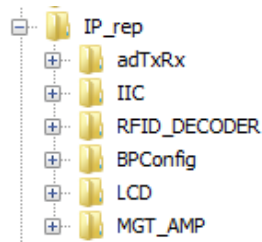


Figure 4: Custom IP repository directory.

- **adTxRx**: Directory of the developed Custom_IP_ADRxTx;
- **IIC**: Directory of the developed Custom_IP_IIC;
- **RFID_DECODER**: Directory of the developed Custom_IP_RFID_DECODER;
- **BPConfig**: Directory of the developed Custom_IP_BP_CONFIGURATION;
- **LCD**: Directory of the developed Custom_IP_LCD;
- **MGT_AMP**: Directory of the developed Custom_IP_MGTP_TXP_AMPLITUDE.

The SDK project is present in the directory `\src\c` and the source files are in the directory `\src\c\RFID_Reader\src`. See Fig. 5.

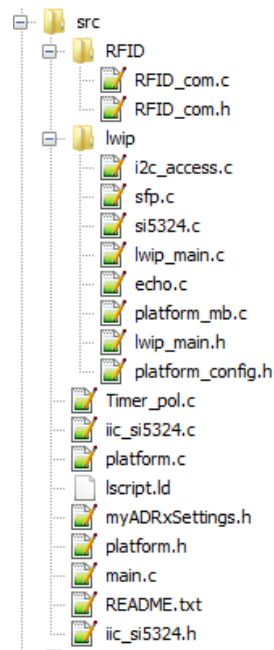


Figure 5: SDK source directory.

- **RFID**: Directory of all the RFID implemented functions.;
- **RFID_com.c**: *C* file that contains all the RFID functions;
- **RFID_com.h**: Header file that contains the declarations of all the RFID functions as well as the definitions required for the project;
- **lwip**: Directory of the Xilinx lwIP functions;
- **Timer_pol.c**: *C* file that contains the timer configuration;
- **iic_si5324.c**: *C* file that contains the *Si5324* drivers;
- **iic_si5324.h**: Header file with the declaration of the *Si5324* functions;
- **Platform.c**: *C* file where the Microblaze cache, interrupts and timers are initialized. It also initializes RFID commands, the RFID decoder and the ADRxTx;
- **Platform.h**: Header file for the Platform.c;
- **myADRxSettings.h**: Header file that contains all the ADRxTx definitions;
- **main.c**: *C* file with the project main function.

2 System Design

2.1 Overview

This section presents an overview of the developed RFID system. The system's block diagram is described in the Fig.6 and can be divided in two sections:

- FPGA section;
- RF Front-End section;

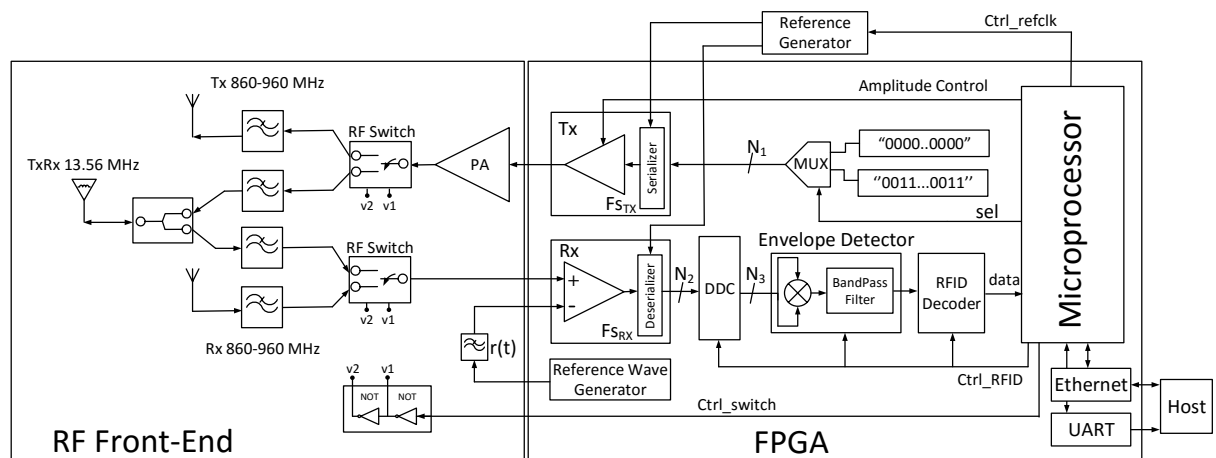


Figure 6: System block diagram.

The baseband RFID protocol as well as its coding, modulation and upconversion at the Transmission (Tx) path and its downconversion, demodulation and decoding at the Reception (Rx) path were implemented in the FPGA. It also features the host communication via Ethernet and UART.

The RF Front-end consists of signal amplification chain as well as a filtering path and antenna switching, according to the desired RFID protocol.

2.2 External Hardware requirements - Analog Physical Layer

This subsection feature all the required external hardware to implement the All-Digital RFID reader.

The main requirements are:

- 1 Xilinx Kintex-7 FPGA KC705 Evaluation Board;
- 1 Low-pass Filter with Cutoff Frequency $F_c = 44$ MHz;
- 2 Low-pass Filter with Cutoff Frequency $F_c = 1$ GHz;
- 2 Low-pass Filter with Cutoff Frequency $F_c = 20$ MHz;
- 3 DC-Blocks;
- 1 Power Amplifier wide-band with minimum range of frequencies 10 MHz to 1 GHz and 20 dB Gain;
- 1 Wilkinson power combiner/divider with minimum range of frequencies 0 to 20 MHz;
- 2 UHF Antennas (860-960 MHz);
- 1 HF coil Antenna (13.56 MHz);
- 2 RF switches with minimum range of frequencies 0 to 1 GHz;
- 2 Logical inverter ports;
- 1 Power source 15 V;
- 1 Power source 3 V.

The electrical schematic of the implemented RF Front-End is illustrated in Appendix A.

2.3 HDL Hardware Developed - Digital Physical Layer

This section presents and describes the developed IP cores of this project. For each presented IP a functional description is included together with a register table and a port interface table.

The following custom IP cores were developed:

- Custom_IP_ADRxTx;
- Custom_IP_MGT_TX_AMPLITUDE;
- Custom_IP_BP_CONFIGURATION;
- Custom_IP_RFID_DECODER;
- Custom_IP_IIC.

2.3.1 Overview Developed Custom IP Cores

• Custom_IP_ADRxTx

The *Custom_IP_ADRxTx*, presented in Fig. 7, is the IP core responsible for up-conversion/downconversion the baseband/RF RFID signals, therefore it is divided in two paths: Transmission path and Receiver path.

This IP uses two BRAMs. The *Bram_ADRx* stores data from DDC or MGT and the *Bram_Tx* stores a PWM signal to configure the Tx frequency in runtime.

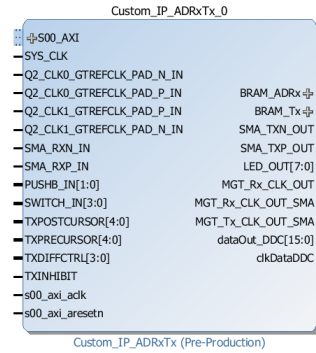


Figure 7: Custom_IP_ADRxTx.

Transmission path:

The vast majority of RFID protocols are based on ASK modulation schemes, that can be easily implemented by controlling the amplitude of the carrier wave or even turning it off depending on the symbol to be transmitted. By observing Fig. 6, depending on the symbol to transmit the μP will select a parallel word of N_1 bits to serialize that can be one of two options: “0101...01” or “0000...00”, that respectively correspond to the presence and the absence of the carrier wave. The serializer is working at frequency of F_{sTX} , which is related with a given reference clock frequency (f_{clkTx}) by a factor F_1 due to its internal PLL (1).

$$F_{sTX} = f_{clkTx} \times F_1 \quad (1)$$

Consequently, the carrier wave frequency (fc) is defined by (2).

$$fc = \frac{F_{sTX}}{2} \quad (2)$$

Therefore, just by changing the serializer clock frequency, it is possible to change its sampling rate and therefore the carrier placement. Since there is an interest in having a fine carrier tuning it can be added one more degree of freedom regarding the periodicity of the digital word “0101..01”. Then, adding a second factor F_2

that will define the periodicity of the sequence “01” in the previous word, i.e., if $F_2 = 2$ the sequence will be “0101...01” and if $F_2 = 4$ the sequence will be “00110011...0011”. Then the carrier frequency assumes a new expression defined by (3).

$$f_c = \frac{F_{sTX}}{F_2} \quad (3)$$

On the other hand, to control the carrier amplitude, the μP , dynamically changes the peak-to-peak voltage of the serialized waveform depending on the symbol to transmit.

Receiver path: The receiver path downconvert the RF RFID signal to baseband and demodulate it. To do so, the implemented system use directly the FPGA high speed differential input pins from the Multi-Gigabit Transceiver (MGT)s to build the comparator quantizer. This conversion is based on Pulse-Width Modulation (PWM) [4,5]. As presented in Fig. 7, to perform the Analog-to-Digital (A/D) conversion a single comparator is required, using one of its inputs for the RF filtered signal and the other for a reference signal ($r(t)$). The reference signal should present an amplitude greater or equal than the RF signal. Additionally, the reference signal should ideally be a triangular wave, since it presents an uniform amplitude distribution allowing the RF signal to be equally quantized [6]. Nevertheless, a sinusoidal waveform can also be used (with an acceptable decrease on the overall receiver resolution) which greatly simplifies the hardware from an implementation point of view. This is the approach used in this work by generating a square wave in the digital domain, and then with a low pass filter, a sinusoidal waveform is obtained to feed the comparator. On the other hand, the frequency of the reference signal imposes the maximum bandwidth of the signal to be acquired, which in the RFID case, is about few hundreds of kHz.

Consequently, following once again the diagram of Fig. 6, at the comparator output there is a two-level PWM representation of the analog signal, that is sampled at a rate of F_{sRX} . In this case the FPGA’s high speed differential input buffers are used to build the comparator and the sampler, which allow to obtain a single-chip integrated RFID reader system. The sampling process works at a few Gbps and then the digital PWM signal is converted to a parallel word of N_2 bits by the deserializer, allowing to process the data at a lower sampling rate of F_{sRX}/N_2 . Therefore, after sampling the RF digital signal must be down-converted to the baseband, filtered to remove the PWM distortion and decimated up to a sampling rate suitable for the baseband processing and decoding. This task is achieved at the Digital Down Conversion (DDC) block, which is performed by a polyphase Digital Direct-Synthesis (DDS) and polyphase filter with N_2 parallel paths.

After the DDC the carrier signal is centered at an Intermediate Frequency (IF)

of 2 MHz (although it can be modified to be centered between 0 to $F_{sDDC}/2$).

This IP core has four registers to configure. Table 1 presents those registers.

Table 1: Custom_IP_ADRxTx register table.

Address	Bit Organization	Description
0x00	[31 downto 0] → Define Address	Set the Rx BRAM limit address
0x04	[0 downto 0] → Enable Rx BRAM	Value '1' enables <i>Bram_ADRX</i>
0x04	[1 downto 1] → Enable Tx BRAM	Value '1' enables <i>Bram_Tx</i>
0x04	[2 downto 2] → Reset Rx MGT	Value '1' resets the Rx MGT
0x04	[3 downto 3] → Reset Tx MGT	Value '1' resets the Tx MGT
0x04	[4 downto 4] → MGT clock data recovery hold	Value '1' sets the MGT clock data recovery circuit register
0x04	[5 downto 5] → Enable DDS operation	Value '1' enables DDS
0x04	[7 downto 7] → Select the data to write on the Rx	Value '1' selects the data directly from the MGT. Value '0' selects data from the DDC (for debug purposes)
0x08	[31 downto 0] → Set the DDS phase increment	Select the carrier frequency of the received signal
0x0C	[31 downto 0] → Define Address	Set the Tx BRAM limit address

Table 2 presents the IP core ports interface.

Table 2: Custom_IP_ADRxTx ports interface description.

Port Name	Type	Description
S00_AXI	BUS	AXI Bus Interface
SYS_CLK	IN	Board System Clock
Q2_CLK0_GTREFCLK_PAD_N_IN	IN	Differential N Fs Tx Clock
Q2_CLK0_GTREFCLK_PAD_P_IN	IN	Differential P Fs Tx Clock
Q2_CLK1_GTREFCLK_PAD_N_IN	IN	Differential N Fs Rx Clock
Q2_CLK1_GTREFCLK_PAD_P_IN	IN	Differential P Fs Rx Clock
SMA_RXN_IN	IN	Differential N RF receiver SMA
SMA_RXP_IN	IN	Differential P RF receiver SMA
PUSHB_IN	IN	MGT hard reset
TXPOSTCURSOR	IN	Transmitter post-cursor Tx pre-emphasis control
TXPRECURSOR	IN	Transmitter pre-cursor Tx pre-emphasis control
TXDIFFCTRL	IN	Driver Swing Control
TXINHIBIT	IN	Block transmission
s00_axi_aclk	IN	AXI Bus Interface asynchronous clock
s00_axi_aresetn	IN	AXI Bus Interface asynchronous reset
BRAM_ADRx	BUS	Rx BRAM Bus Interface
BRAM_Tx	BUS	Tx BRAM Bus Interface
SMA_TXN_OUT	OUT	Differential N RF transmitter SMA
SMA_TXP_OUT	OUT	Differential P RF transmitter SMA
LED_OUT	OUT	Debug LEDs
MGT_Rx_CLK_OUT	OUT	Fs Rx Clock
MGT_Rx_CLK_OUT_SMA	OUT	SMA Debug Fs Rx Clock
MGT_Tx_CLK_OUT_SMA	OUT	SMA Debug Fs Tx Clock
dataOut_DDC	OUT	Baseband Data from DDC
clkDataDDC	OUT	DDC clock

• Custom_IP_MGT_TX_AMPLITUDE

The *Custom_IP_MGT_TX_AMPLITUDE*, presented in Fig. 8 is responsible to control the Tx MGT buffer amplitude.

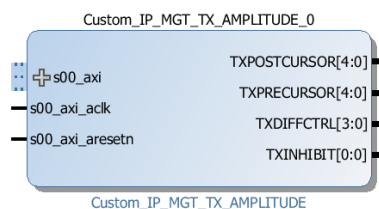


Figure 8: Custom_IP_MGT_TX_AMPLITUDE.

Table 3 features the port map registers and Table 4 presents the ports interface description.

Table 3: Custom_IP_MGT_TX_AMPLITUDE register table.

Address	Bit Organization	Description
0x00	[10 downto 10] → Define TXINHIBIT [9 downto 5] → Define TXPRECURSOR [4 downto 0] → Define TXPOSTCURSOR	This register is always defined as 0x00000000
0x04	[3 downto 0] → Define TXDIFFCTRL	This register sets the MGT buffer amplitude and uses values from 0 to 2^4

Table 4: Custom_IP_MGT_TX_AMPLITUDE port description.

Port Name	Type	Description
s00_axi	BUS	AXI Bus Interface
s00_axi_aclk	IN	AXI Bus Interface asynchronous clock
s00_axi_aresetn	IN	AXI Bus Interface asynchronous reset
TXPOSTCURSOR	OUT	Transmitter post-cursor Tx pre-emphasis control
TXPRECURSOR	OUT	Transmitter pre-cursor Tx pre-emphasis control
TXDIFFCTRL	OUT	Driver Swing Control
TXINHIBIT	OUT	Blocks transmission

• Custom_IP_BP_CONFIGURATION

The *Custom_IP_BP_CONFIGURATION*, presented in Fig. 9 is responsible for configuring the digital bandpass FIR filter.

This IP uses an BRAM to store the filter's coefficients in order to configure it on the fly.

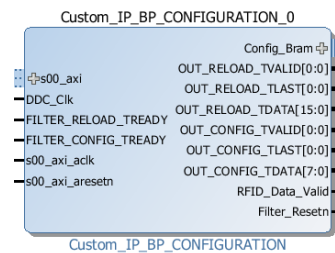


Figure 9: Custom_IP_BP_CONFIGURATION.

Table 5 features the port map registers and Table 6 presents the ports interface description.

Table 5: Custom_IP_BP_CONFIGURATION register table.

Address	Bit Organization	Description
0x00	[0 downto 0] → Define bandpass FIR filter configuration Start/Stop	Start or stop the bandpass filter configuration. Uses values 1 or 0.
0x04	[0 downto 0] → Define Filter_Resetn	Sets a soft reset to the bandpass FIR filter. Uses values 1 or 0.

Table 6: Custom_IP_BP_CONFIGURATION port description.

Port Name	Type	Description
s00_axi	BUS	AXI Bus Interface
DDC_Clk	IN	Clock from DDC
FILTER_RELOAD_TREADY	IN	Coefficient ready
FILTER_CONFIG_TREADY	IN	Configuration ready
s00_axi_aclk	IN	AXI Bus Interface asynchronous clock
s00_axi_aresetn	IN	AXI Bus Interface asynchronous reset
Config_Bram	BUS	Configuration BRAM Bus
OUT_RELOAD_TVALID	OUT	Coefficient valid
OUT_RELOAD_TLAST	OUT	Coefficient last
OUT_RELOAD_TDATA	OUT	Coefficient data
OUT_CONFIG_TVALID	OUT	Configuration valid
OUT_CONFIG_TLAST	OUT	Configuration last
OUT_CONFIG_TDATA	OUT	Configuration data
RFID_Data_Valid	OUT	DDC data valid
Filter_Resetn	OUT	Bandpass FIR filter asynchronous reset

• Custom_IP_RFID_DECOCER

The *Custom_IP_RFID_DECOCER*, presented in Fig. 10 is responsible to decode baseband RFID signals. Two different RFID protocols decoders are implemented: Manchester-decoder for MIFARE and FM0-decoder for EPC Gen2.

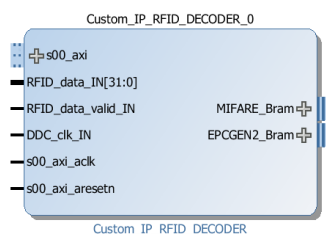


Figure 10: Custom_IP_RFID_DECOCER.

This IP uses two different BRAMs. MIFARE_Bram stores the MIFARE RFID IDs and EPCGEN2_Bram stores the EPC Gen2 RFID IDs.

Table 8 features the port map registers and Table 7 presents the ports interface description.

Table 7: Custom_IP_RFID_DECOCER port description.

Port Name	Type	Description
s00_axi	BUS	AXI Bus Interface
RFID_data_IN	IN	RFID baseband data
RFID_data_valid_IN	IN	RFID baseband data valid
DDC_clk_IN	IN	Clock from DDC
s00_axi_ack	IN	AXI Bus Interface asynchronous clock
s00_axi_aresetn	IN	AXI Bus Interface asynchronous reset
MIFARE_Bram	BUS	MIFARE BRAM Bus
EPCGEN2_Bram	BUS	EPCGEN2 BRAM Bus

Table 8: Custom_IP_RFID_DECOCER.

Address	Bit Organization	Description
0x00	[31 downto 31] → Define RFID MIFARE decoder Start/Stop.	Start or stop the RFID MIFARE decoder. Uses values 1 to start and 0 to stop.
0x00	[16 downto 25] → Define RFID MIFARE decoder decision threshold.	Set the MIFARE decoder decision threshold for a MIFARE Manchester-coded symbol.
0x00	[7 downto 0] → Define RFID decoder number of samples of a Manchester-coded symbol.	Set the number of samples for a MIFARE Manchester-coded symbol.
0x04	[31 downto 31] → Define RFID EPC Gen2 decoder Start/Stop.	Start or stop the RFID EPC Gen2 decoder. Uses values 1 to start and 0 to stop.
0x04	[23 downto 8] → Define RFID decoder number of samples of a violation symbol ¹ .	Set the number of samples for an EPC Gen2 violation symbol.
0x04	[7 downto 0] → Define RFID decoder number of samples of a FM0-coded symbol ¹ .	Set the number of samples for an EPC Gen2 FM0-coded symbol.

¹ The number of samples may change depending of the data-rate emitted by the RFID tag.

• Custom_IP_IIC

The *Custom_IP_IIC*, presented in Fig. 11 is responsible to program the Jitter Attenuated Clock Si5326. This clock generator allows to change, in real-time, the MGT F_{STX} .

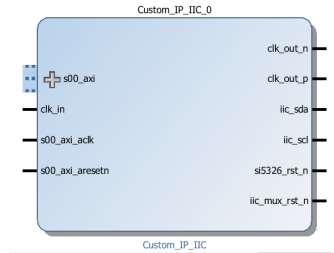


Figure 11: Custom_IP_IIC.

Table 9 presents the ports interface description.

Table 9: Custom_IP_IIC port description.

Port Name	Type	Description
s00_axi	BUS	AXI Bus Interface
clk_in	IN	Reference clock from the Si570 156.25 MHz clock
RFID_data_valid_IN	IN	RFID baseband data valid
s00_axi_aclck	IN	AXI Bus Interface asynchronous clock
s00_axi_aresetn	IN	AXI Bus Interface asynchronous reset
clk_out_n	OUT	Differential N clock to Si5326
clk_out_p	OUT	Differential P clock to Si5326
iic_sda	IN\OUT	IIC serial Data
iic_scl	IN\OUT	IIC serial clock
si5326_rst_n	OUT	Reset to Si5326
iic_mux_rst_n	OUT	Enable IIC bus transactions

2.4 Software Developed - MAC Layer

This layer refers to all the developed *C* functions, responsible for the International Standards Organization (ISO) 18000-6C and MIFARE protocol layer and was implemented at the FPGA's embedded processor (Microblaze).

The developed Software is divided in **four** major sections:

First The first section refers to platform initialization. This *C* function, whose name is *lwip_main()*, initializes and configures the third party open source software Lightweight IP (lwIP), the RFID commands, the RFID decoder, the Xilinx microprocessor timer, cache and finally, the All-Digital Receiver/Transmitter (ADR_xT_x);

Second The second section refers to the *C* function *ethernet_State_Machine()*. This state machine, as shown in Figure 12, is responsible for managing the host's Ethernet commands (*ethernet_communication()*) and the RFID protocols *inventory_round_StateMachine()* and *MIFARE_StateMachine()*. This section also controls the following aspects of the system: RF front-end switching -*RF_Switch()*; digital passband filter configuration -*Configure_RFID_PB_FILTER()*; Calculates the transmitter Fs -*clockSI_Chooser()*; Finally, it controls the stop and exit state of the system.

All the Ethernet commands, except the help command, follow the same structure formation:

```
[START_CHARACTER  TYPE_CHARACTER  PAYLOAD_CHARACTER  
END_CHARACTER].
```

There are **six** possible Ethernet commands (see Table 10) which can be grouped in four types:

- Change Frequency commands \implies These commands indicate the reader to initiate its operations for a given frequency and protocol. Those also enable the RF carrier, calculate the Fs for the transmission serializer, program the bandpass filter and finally enable the RFID decoder;
- Stop_command \implies This command disables the RF carrier;
- Exit_command \implies This command terminates all operations at the microblaze;
- Help_command \implies This command displays the command list;

Table 10: Ethernet commands.

Command	Structure	MHz	Description
Change Frequency EPC ETSI	@G104E @G107E @G110E @G113E	865.70 866.30 866.90 867.50	Change RFID operation to ETSI regulation
Change Frequency EPC FCC	@G201E @G202E ... @G250E	902.75 903.25 ... 927.25	Change RFID operation to FCC regulation
Change Frequency MIFARE	@G300E	13.56	Change RFID operation to MIFARE protocol
Stop	@1000E	————	Disable the RF carrier
Exit	@4000E	————	Terminate all the operations
Help	@h	————	Display help menu

Third The third section refers to the EPC Gen2 RFID protocol state machine (*inventory_round_StateMachine*). The state machine's block diagram is presented in Figure 13. Follows a brief description of the presented block diagram:

[**I/II/III**] The program begins to clear the decoder memory by invoking the *reset_decoder(EPC_TYPE)* function and sends two RFID commands

⇒ [*sendPreamble()+sendQuery()*];

[**IV**] The processor waits for a tag's RN16;

[**V**] The decoder is enabled, the tag's RN16 is stored in the decoder's memory and the processor accesses that information

⇒ [*decode_data(RN16_TYPE)+read_decoder_data_RN16()*];

[**VI/VII/VIII**] The processor generates an RFID acknowledgment command, sends the next two RFID commands and clears the decoder memory

⇒ [*generateACK()+ sendFrameSync() + sendACK() + reset_decoder(RN16_TYPE)*];

[**IX**] The processor waits for a tag's EPC;

[**X**] The decoder is enabled, the tag's EPC is stored in the decoder's memory and the processor accesses that information

⇒ [*decode_data(EPC_TYPE)+read_decoder_data_EPC()*];

[**XII/XIII**] If the tag's EPC is valid, the program sends that EPC via Ethernet and exits the state machine;

[**XI/XIII**] If the tag's EPC is not valid, the program exits the state machine;

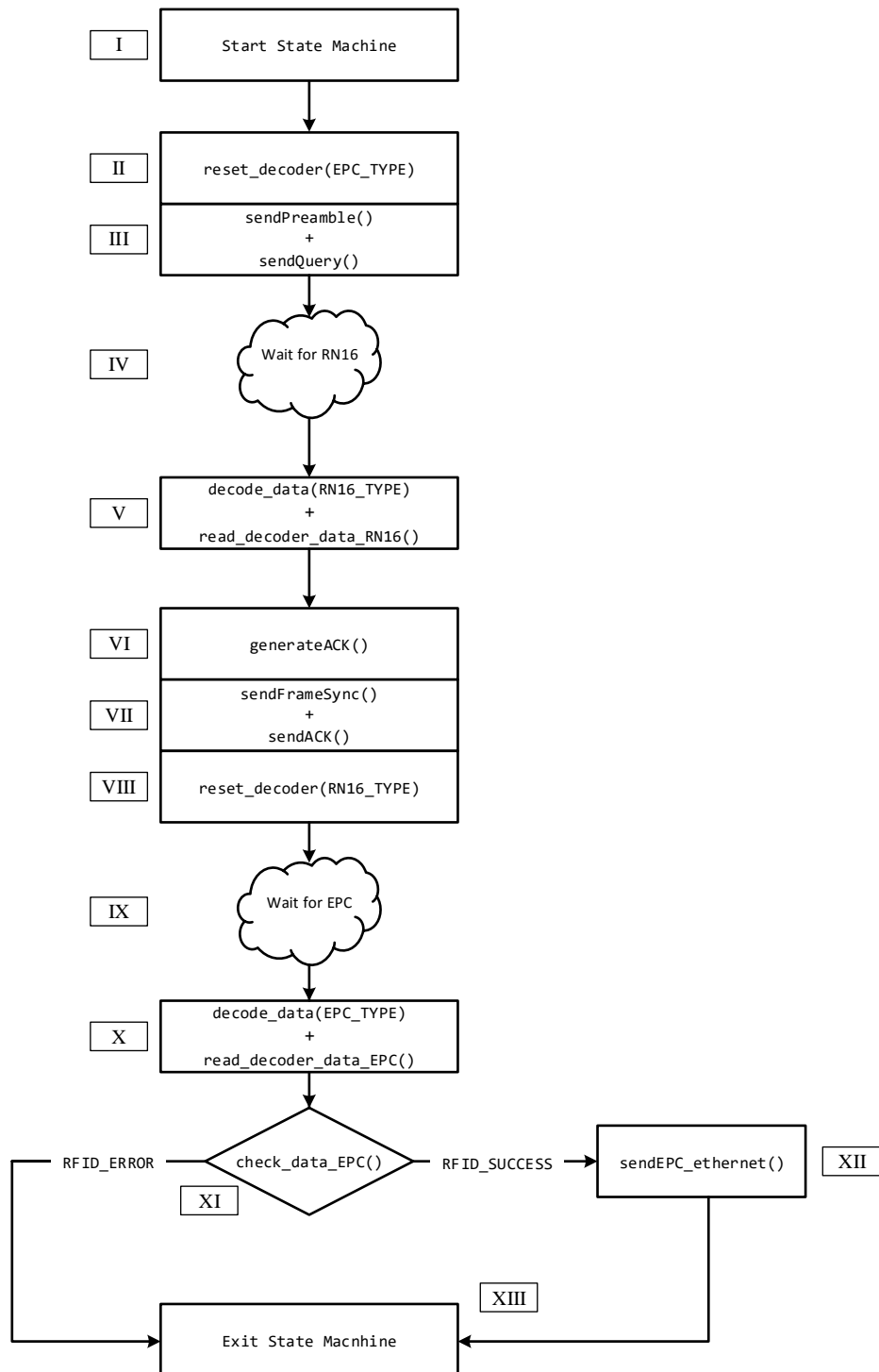


Figure 13: EPC Gen2 RFID protocol state machine.

Fourth The fourth section refers to the MIFARE RFID protocol state machine (*MIFARE_StateMachine*). The state machine's block diagram is presented in Figure 14. Next, follows a brief description of the presented block diagram:

[I/II/III] The program begins to clear the decoder memory and send an REQA RFID command

$\implies [\textit{reset_decoder}(\textit{UID_TYPE}) + \textit{Modified_Miller_Coddling}(\textit{REQA_COMMAND_TYPE})];$

[IV] The processor waits for a tag's ATQA;

[V] The decoder is enabled, the tag's ATQA is stored in the decoder's memory

$\implies [\textit{decode_data}(\textit{ATQA_TYPE})];$

[VI] The processor checks if as received any a valid ATQA

$\implies [\textit{check_ATQA}()];$

[VII] If the ATQA is valid, the processor sends an anti-collision level 1 RFID command

$\implies [\textit{Modified_Miller_Coddling}(\textit{ANTI_LEVEL1_COMMAND_TYPE})];$

[VIII] The processor waits for a tag's UID;

[IX] The decoder is enabled, the tag's UID is stored in the decoder's memory

$\implies [\textit{decode_data}(\textit{UID_TYPE})];$

[X] The processor checks if has received a valid UID $\implies \textit{check_UID}();$

[XI] If the UID is valid, the program sends that UID via Ethernet and exits the state machine $\implies \textit{sendUID_ethernet}();$

[XII] If the ATQA or UID is invalid, the program resets the decoder and exit the state machine;

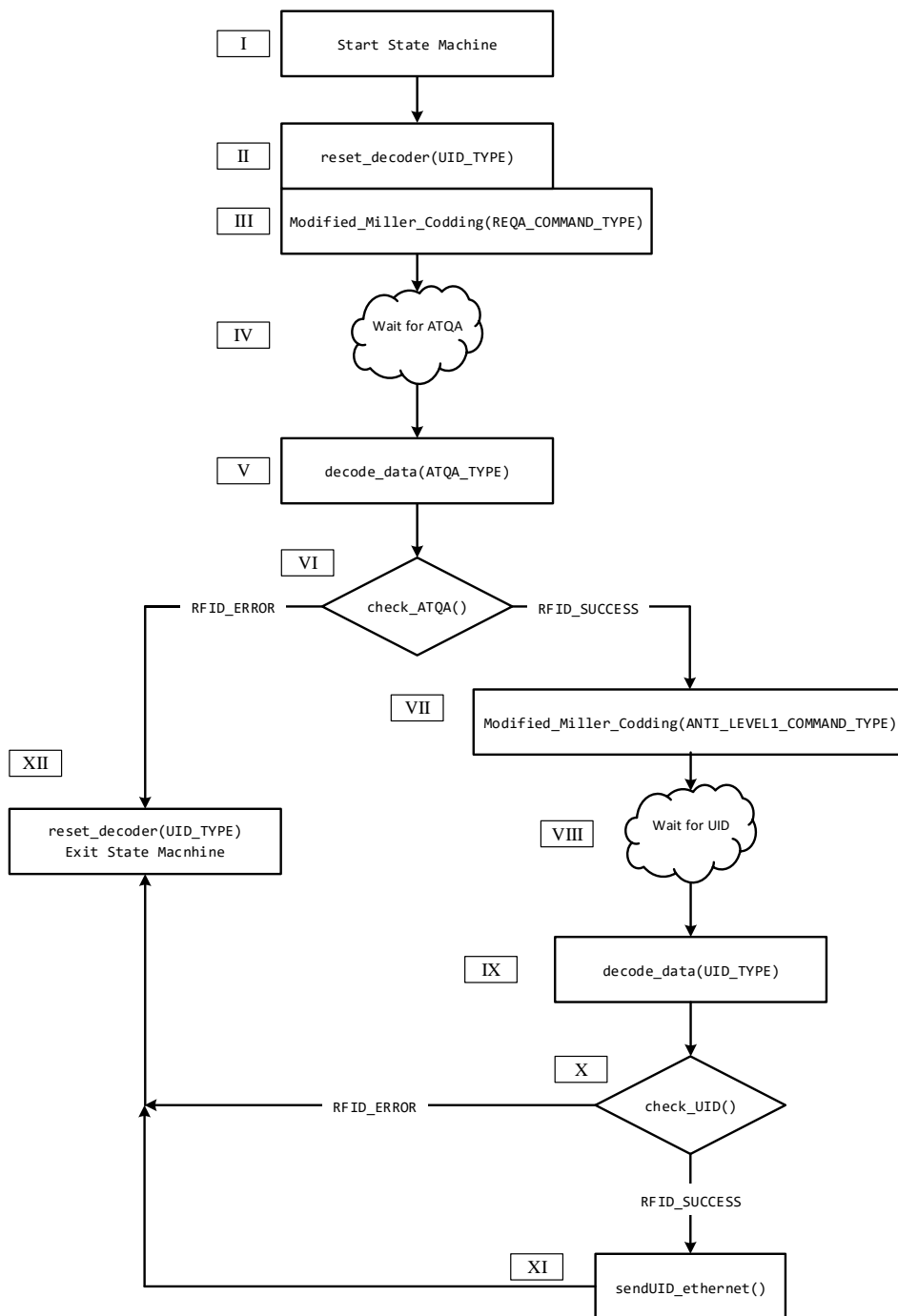


Figure 14: MIFARE RFID protocol state machine.

2.4.1 Description: Team Developed Functions

The functions implemented in *C* are now explained in more detail:

- **init_platform**

Brief: This function initializes and configures the RFID commands, the RFID decoder, the microprocessor timer and cache;

Param: None;

Return: None;

- **Modified_Miller_Codding**

Brief: This function generates Modified Miller Codding at 106 kbps;

Param: Command - Selects an REQA_COMMAND_TYPE to generate or an ANTI_LEVEL1_COMMAND_TYPE;

Return: None;

- **MGT_BUFFER_AMPLITUDE**

Brief: This function controls MGT buffer amplitude;

Param: amp - An uint8_t that control MGT Tx Buffer Amplitude. Values range from 0x0 to 0xF;

Return: None;

- **generateQuery**

Brief: This function generates a Query command;

Param: Query_pointer - A pointer to a query struct;

Return: 0 in case of success or negative error code;

- **generateNAK**

Brief: This function generates a Not Acknowledgment command;

Param: Nak_pointer - A pointer to a nak struct;

Return: 0 in case of success or negative error code;

- **reset_decoder**

Brief: This function resets decoder's memory by writing zeros;

Param: type - Defines if it is an RN16_TYPE, an EPC_TYPE, ATQA_TYPE or an UID_TYPE RFID signal to reset;

Return: 0 in case of success or negative error code;

- **generateMifare**

Brief: This function generates MIFARE type-A commands;

Param: mifare_pointer - A pointer to a RFID_MIFARE struct;

Return: 0 in case of success or negative error code;

• **init_ADRX_MGTXRX**

Brief: This function initialize Tx Bram and Tx MGT. Also initialize the DDC frequency and the Rx frequency;

Param: fc - uint32_t that defines the intended DDC frequency in kHz;

Param: nb - uint32_t that defines how many '1's are needed to generate a square wave. Square wave frequency is equal to $Tx_fs/(nb*2)$;

Return: None;

• **PulseShapeMGT**

Brief: Pulse shaping the MGT's output;

Param: sel - An uint16_t that selects if is a pulse shape at ascendent or descendent way. Values SHAPE_UP or SHAPE_DOWN;

Return: None;

• **wait_ms**

Brief: This function pauses the microprocessor for milliseconds;

Param: time2wait - milliseconds to wait;

Return: None;

• **ethernet_State_Machine**

Brief: State Machine for Ethernet Host Communication, see Fig 12;

Param: None;

Return: 0;

• **inventory_round_StateMachine**

Brief: State Machine for the UHF RFID protocol ISO 18000-6C, see Fig. 13;

Param: None;

Return: None;

• **MIFARE_round_StateMachine**

Brief: State Machine for the MIFARE for 13.56MHz RFID, see Fig. 14;

Param: None;

Return: None;

- **sendPreamble**

Brief: This function sends a preamble to a tag;

Param: None;

Return: 0 in case of success or negative error code;

- **sendQuery**

Brief: This function sends a pregenerated query to tag;

Param: query_pointer - A pointer to a query struct;

Return: 0 in case of success or negative error code;

- **decode_data**

Brief: This function starts the decoder, waits *time_to_wait* microseconds and stops the decoder;

Param: type - defines if it is an RN16_TYPE word, an EPC_TYPE word, an ATQA_TYPE word or an UID_TYPE word to decode;

Return: None;

- **timerWait**

Brief: This function enables timer 0 and pauses the microprocessor. Per approximately hundred *time2wait*, timer counts for 1 us.;

Param: time2wait - Time to wait time2wait/100 microseconds;

Return: None;

- **read_decoder_data_RN16**

Brief: This function reads RN16 data type from decoder memory;

Param: rn16_pointer - A pointer to a rn16 struct;

Return: 0 in case of success or negative error code;

- **generateACK**

Brief: This function generates an ACK command.

Param: ack_pointer - A pointer to a ack struct;

Param: rn16_pointer - A pointer to a rn16 struct;

Return: 0 in case of success or negative error code;

- **sendFrameSync**

Brief: This function sends a framesync to a tag;

Param: None;

Return: 0 in case of success or negative error code;

- **sendACK**

Brief: This function sends a pre-generated acknowledge to a tag;

Param: ack_pointer - A pointer to a ack struct;

Return: 0 in case of success or negative error code;

- **read_decoder_data_EPC**

Brief: This function reads the EPC data type from decoder memory;

Param: epc_pointer - A pointer to a epc struct;

Return: 0 in case of success or negative error code;

- **sendEPC_ethernet**

Brief: This function sends an EPC to the ethernet port;

Param: None;

Return: None;

- **sendUID_ethernet**

Brief: This function sends an UID to ethernet port;

Param: None;

Return: None;

- **start_decoder**

Brief: This function enables the RFID decoder;

Param: sel - uint32_t that defines if is to start decoder for EPC Gen2 protocol (MUX_UHF) or MIFARE protocol (MUX_MIFARE);

Return: None;

- **stop_decoder**

Brief: This function disables the RFID decoder;

Param: sel - uint32_t that defines if is to stop decoder for EPC Gen2 protocol (MUX_UHF) or MIFARE protocol (MUX_MIFARE);

Return: None;

- **PIE**

Brief: This function generates a Pulse Interval Encoder signal.

Param: symbol - A character that represents a symbol of a RFID command;

Return: None;

- **check_data_EPC**

Brief: This function checks if the received EPC is valid;

Param: None;

Return: 0 in case of success or negative error code;

- **WriteTxBram**

Brief: This function organizes data in memory to be used by the transmitter MGT;

Param: An uint32_t that defines how many '1's are needed to generate a square wave. The square wave frequency is equal to $MGT_rate / (counterLimit \times 2)$

Return: An uint32_t number of BRAM's memory positions occupied;

- **generateCRC5**

Brief: This function generates a CRC_5 for RFID UHF protocol;

Param: data_pointer - A pointer to a query struct;

Return: 0 in case of success or negative error code;

- **Channel_Chooser**

Brief: This function selects the frequency for a specific channel and the RFID norm;

Param: channel - uint8_t value that selects the frequency channel for a given norm;

Param: norm - uint8_t value that selects ETSI (1), FCC (2) or MIFARE (3);

Return: uint32_t Frequency in kHz;

- **clockFs_Chooser**

Brief: This function calculates the Fs clock to be the reference clock of the Tx MGT;

Param: freq - An uint32_t target Tx frequency kHz;

Param: n - An uint32_t that defines how many '1's are needed to generate a square wave;

Return: uint32_ Fs in kHz;

- **Configure_RFID_PB_FILTER**

Brief: This function configures the RFID passband digital filter for each protocol;

Param: type - An uint8_t that defines if is an RFID_MIFARE_FILTER_TYPE or RFID_UHF_FILTER_TYPE;

Return: None;

- **RF_Switch**

Brief: This function controls the RF switch front-end;

Param: type - An uint8_t that controls the RF switch front-end to change between the MIFARE path (RFID_MIFARE_FILTER_TYPE) and UHF path (RFID_UHF_FILTER_TYPE);

Return: None;

- **check_UID**

Brief: This function checks if UID from tag is valid;

Param: None;

Return: 0 in case of success or negative error code;

- **check_ATQA**

Brief: This function checks if ATQA from tag is valid;

Param: None;

Return: 0 in case of success or negative error code;

- **free_buffer_data**

Brief: This function frees the buffers: *buffer_data_ethernet* and *buf_aux*;

Param: None;

Return: 0 in case of success or negative error code;

- **IicFreqProg**

Brief: This function Program MGT F_{sTx} ;

Param: freq - An uint32_t that represents the target frequency in kHz;

Return: 1;

- **printHeaderRFID**

Brief: Displays help menu.;

Param: None;

Return: None;

2.4.2 Description: Third Party Developed Functions

- **crc_calculate_crc**

Brief: This function generates a CRC_16 for RFID UHF protocol;

Param: initial_crc - CRC Preset;

Param: buffer - pointer for PC+EPC+PacketCRC;

Param: length - length of data;

Return: Calculated CRC

develop by <http://bl.ocks.org/bewest/9559812>

- **TmrCtr_init**

Brief: This Xilinx function initializes the microprocessor timer for polled; Each device may contain multiple timer counters. The timer number is a zero based number with ranges from 0 to (XTC_DEVICE_TIMER_COUNT - 1);

Param: TmrCtrNumber is the timer counter of the device to be used.

Return: XST_SUCCESS if successful, XST_FAILURE if unsuccessful

- **LwIP functions**

It was used lwIP code provided by Xilinx to enable Ethernet communication.

- All the other functions, not described in this document but presented on the system, were developed by Xilinx.

2.5 How to assemble the RFID Reader

This section explains how to assemble the RF Front-End and how to start the RFID software.

First Assemble the RF Front-End as it is present in Fig. 26 and connect it to the respective SMA connectors presented at the Xilinx Kintex-7 FPGA KC705 Evaluation Board. In order to guarantee the FPGA integrity there are presented two Attenuators with 20 dB and 16 dB should be included (as depicted in Fig. 26). These values must guarantee a maximum value of 400 mVpp at the FPGA input SMA;

Second Turn on the FPGA;

Third Program the FPGA with the RFID_READER.bit file at directory
 \src\hw;

Fourth Open Xilinx SDK 2014.4 and open the workspace at the directory \src\c. In the main.c file change the `IP_ADDR` (Fig. 15) for a desired one, for example: `uint32_t IP_ADDR[4] = {192, 168, 1, 10};`

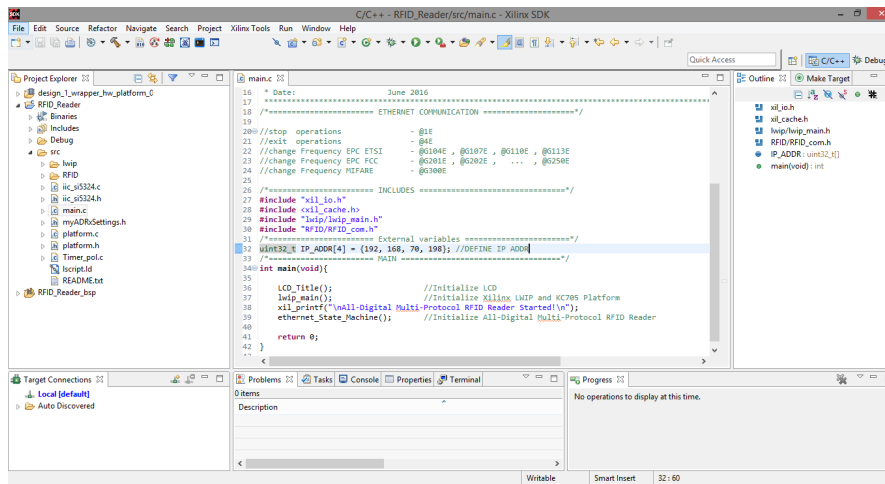


Figure 15: Change IP Address.

Fifth Open a serial terminal (*Termite* or *Tera Term*) and configure:

Port: 'COM3'

Baud Rate: 9600

Data bits: 8

Stop bits: 1

Sixth At the Xilinx SDK 2014.4 run the program and check if the platform was successfully initialized at the serial terminal. See Fig 16;

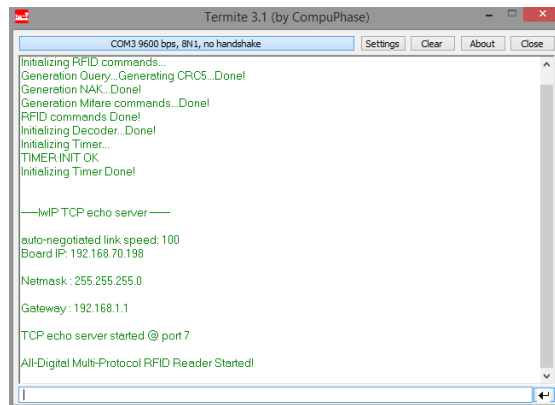


Figure 16: Platform was successfully initialized.

Seventh At the Xilinx board LED2 and LED4 should be on, see Fig 17;



Figure 17: Status LED after Xilinx board initialization.

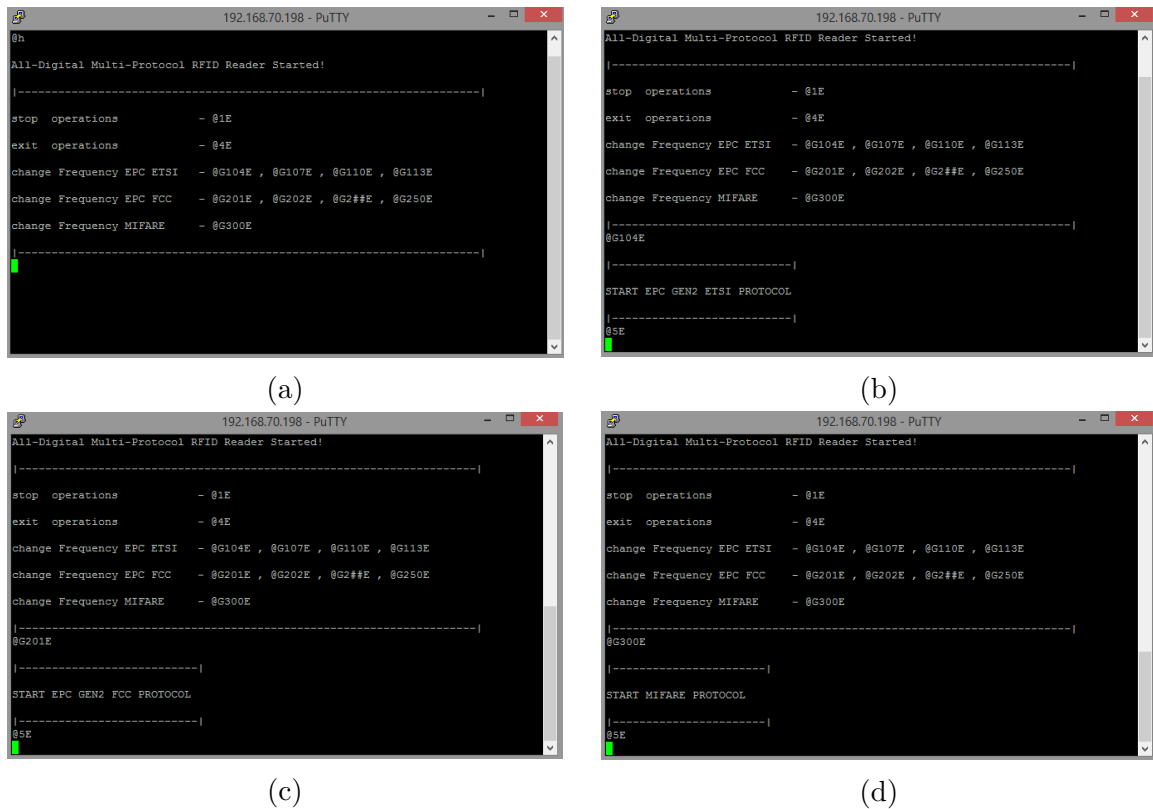
Eighth Open *Putty* and configure:

IP Address: defined at Fourth

Port: 7

Connectivity Type: RAW

Ninth At *Putty* program enter the commands presented in Tab. 10, as showed in Fig 18;



```

(a)
192.168.70.198 - PuTTY
@h
All-Digital Multi-Protocol RFID Reader Started!
|-----|
stop operations      - @1E
exit operations      - @4E
change Frequency EPC ETSI - @G104E , @G107E , @G110E , @G113E
change Frequency EPC FCC  - @G201E , @G202E , @G2##E , @G250E
change Frequency MIFARE   - @G300E
|-----|

(b)
192.168.70.198 - PuTTY
All-Digital Multi-Protocol RFID Reader Started!
|-----|
stop operations      - @1E
exit operations      - @4E
change Frequency EPC ETSI - @G104E , @G107E , @G110E , @G113E
change Frequency EPC FCC  - @G201E , @G202E , @G2##E , @G250E
change Frequency MIFARE   - @G300E
|-----|
@G104E
|-----|
START EPC GEN2 ETSI PROTOCOL
|-----|
@SE

(c)
192.168.70.198 - PuTTY
All-Digital Multi-Protocol RFID Reader Started!
|-----|
stop operations      - @1E
exit operations      - @4E
change Frequency EPC ETSI - @G104E , @G107E , @G110E , @G113E
change Frequency EPC FCC  - @G201E , @G202E , @G2##E , @G250E
change Frequency MIFARE   - @G300E
|-----|
@G201E
|-----|
START EPC GEN2 FCC PROTOCOL
|-----|
@SE

(d)
192.168.70.198 - PuTTY
All-Digital Multi-Protocol RFID Reader Started!
|-----|
stop operations      - @1E
exit operations      - @4E
change Frequency EPC ETSI - @G104E , @G107E , @G110E , @G113E
change Frequency EPC FCC  - @G201E , @G202E , @G2##E , @G250E
change Frequency MIFARE   - @G300E
|-----|
@G300E
|-----|
START MIFARE PROTOCOL
|-----|
@SE

```

Figure 18: RFID Ethernet commands. (a) Help command; (b) Change to EPC Gen2 ETSI protocol; (c) Change to EPC Gen2 FCC protocol; (d) Change to MIFARE protocol;

Tenth LED0 to LED6 should be on, see Fig 19;



Figure 19: Status LED after Xilinx board initialization.

Eleventh After those previous steps, the RFID reader is now fully operational.

2.6 Design Reuse

This section presents the implemented system reuse. For that all parts of system (Hardware and Software) were uploaded for an online repository. One can fully access the parts by clicking in the following link:

https://www.dropbox.com/s/2pqq14lm1zwa1bz/fpga_student_XIL-29448_Oliveira_20160615_1.zip?dl=0

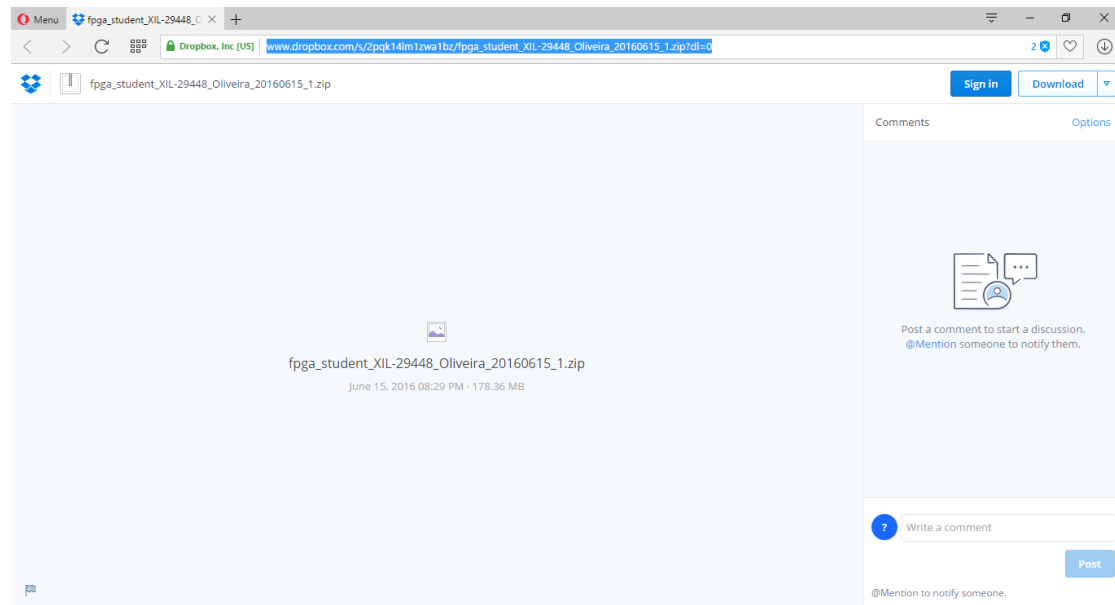


Figure 20: Project repository.

3 Results

This section presents the obtained results which is divided in three subsections. The first one refers to the reader agility to switch between multiple-bands/protocols. The second one refers to the reader's performance at UHF band. For the UHF band measurements of the reader's emitted powerpower are presented and compared with the protocol specifications. Finally, the last subsection presents the maximum distance that the implemented reader can obtain a tag's ID for each band.

The RFID reader setup and its key elements are illustrated in Fig. 21.

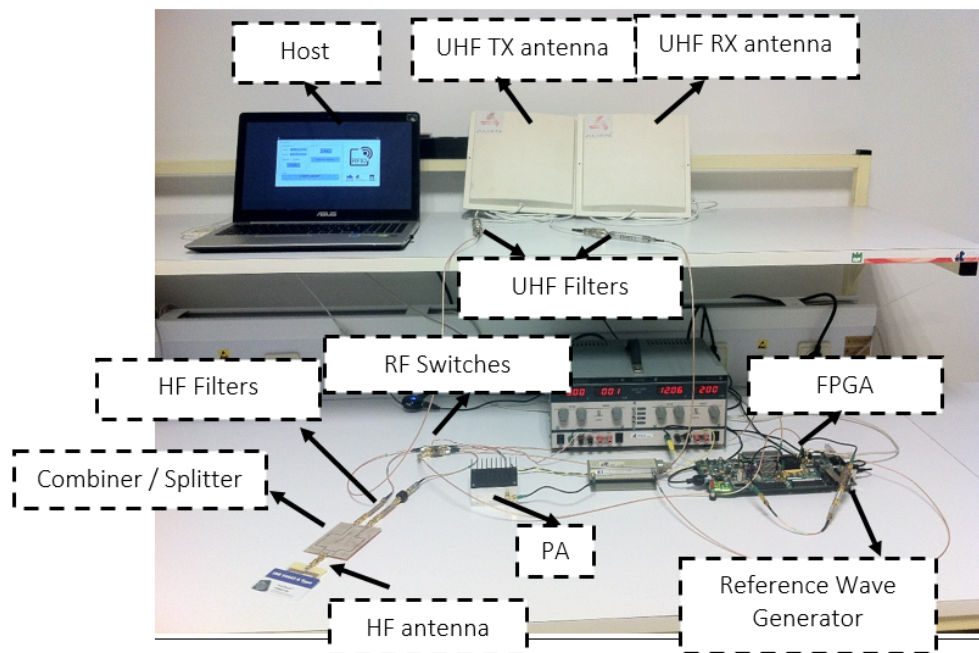


Figure 21: Experimental Setup.

Finally, a video of the working prototype is available at the following link:
<https://www.youtube.com/watch?v=xal4XwPjpWI>

3.1 Reader Multi-Standard Agility

The multi-standard agility of the implemented RFID reader was validated by measuring the signal transmitted spectrum at three different carriers: 13.56 MHz, 866.3 MHz and 915.25 MHz. Each carrier represents a different RFID protocol band. In this case, ISO 18000-3, ISO 18000-6C ETSI and ISO 18000-6C FCC respectively. The spectrum was obtained by connecting the PA to a Spectrum Analyzer (R&S FSH13) while changing the RFID carrier protocol. The results are present in Fig. 22 and were measured with a RBW and VBW of 3 kHz. As one can see, Fig. 22 features a typical ASK spectrum with a well defined carrier for each protocol.

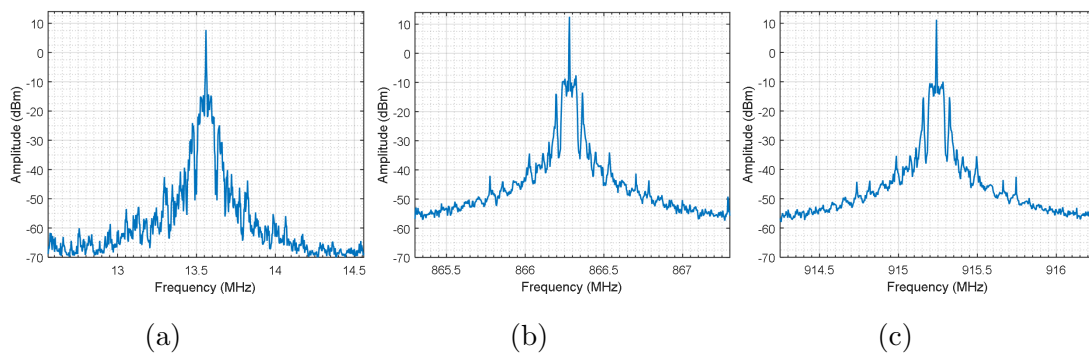


Figure 22: RFID signal transmitter power spectrum. (a) Carrier frequency centered at 13.65 MHz; (b) Carrier frequency centered at 866.3 MHz; (c) Carrier frequency centered at 915.25 MHz;

3.2 Transmitted power mask at UHF Band

This section features the RFID reader performance at UHF band (915.25 MHz). As stated in [7], RFID readers at UHF must meet two transmission power masks: Multiple-Interrogator and Dense-Interrogator Masks. Fig. 23 illustrates these two masks together with the implemented all-digital RFID reader transmitted power mask. Table. 11 sets forth the measured results.

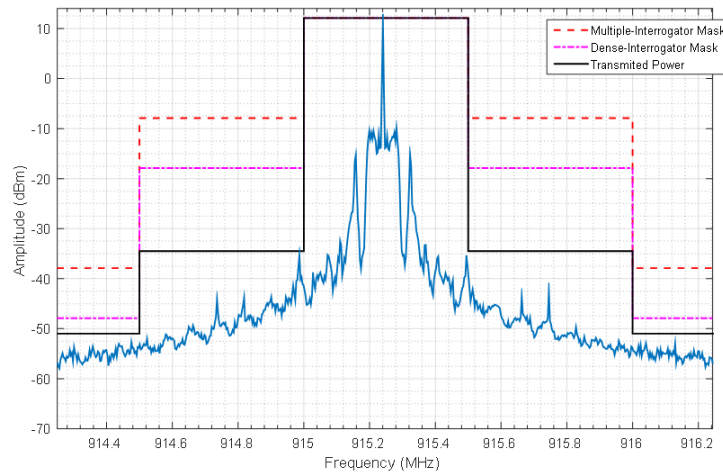


Figure 23: Implemented RFID reader UHF Power Mask.

Table 11: Transmission mask measured at UHF.

Channel	Multiple-Int. (dBch)	Dense-Int. (dBch)	Transmitted (dBch)
0	0	0	0
1	-20	-30	-46.6
2	-50	-60	-63.1
-1	-20	-30	-46.6
-2	-50	-60	-63.1

3.3 RFID reader maximum range

To measure the RFID reader maximum range, for the UHF scenario the reader was configured to transmit a maximum channel power of 11.7 dBm with a side-by-side antenna (Rx/Tx) configuration. The distance between the reader and the tag was changed until the reader stop detecting the tag's ID. It was obtained an approximately range of 2.10 m. For the HF case the maximum range was approximately 0.25 m.

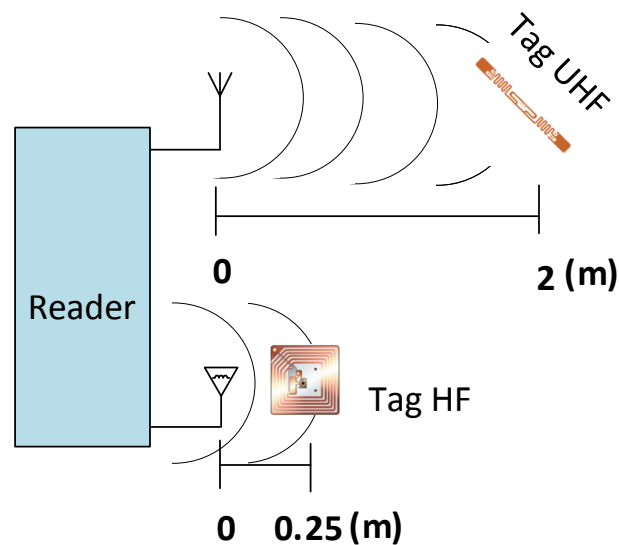


Figure 24: Implemented RFID reader maximum range.

3.4 FPGA resource occupation

This final subsection presents the FPGA's resources occupied by the developed system.

Figure 25 shows a graphic with the percentage of occupation for the different building blocks of the FPGA.

Has one can see, in the majority of the cases the occupation is bellow 50%. The major exception are the DSP48s that are at 74% because the digital filters implemented in the system.

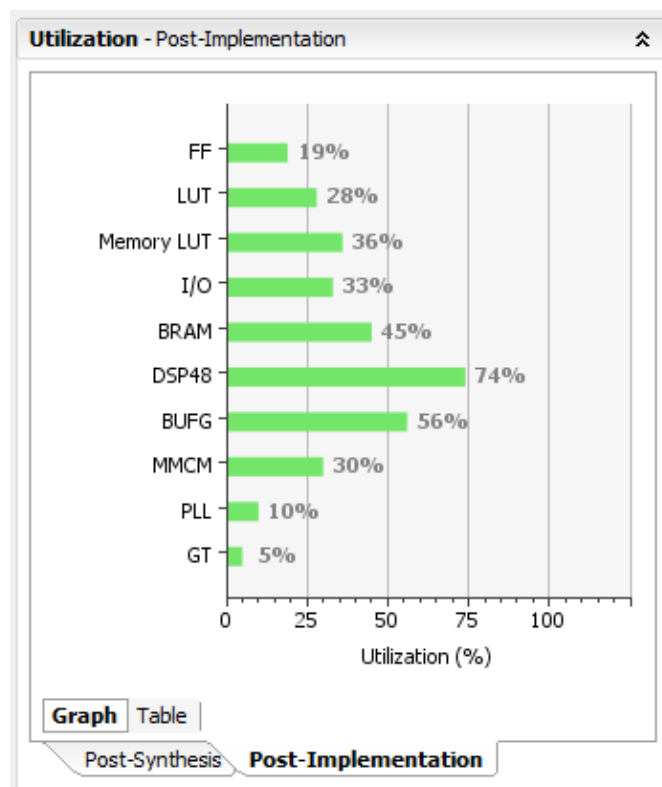


Figure 25: Percentage of occupation.

4 Conclusion

In this project an FPGA-Based multi-band and multi-protocol RFID reader was presented. This reader embraces an all-digital architecture that allows it to be versatile, flexible and adaptive to possible/future physical circuit and protocol changes. This multi-protocol reader is of high importance for IoT scenarios where everything and everyone is connected, because with a single centralized reader it is possible to handle different types of RFID connectivity.

In conclusion a multi-band and multi-protocol RFID reader was implemented with FPGA-based all-digital architecture with a minimal RF front-end.

Appendix A RF Front-End Schematic

This appendix presents the implemented RF Front-End schematic. This schematic is showed in Fig 26 and its commercial components are presented in Table 12.

Table 12: Implemented RF Front-End commercial components.

Component	Model	Manufacture
Power Amplifier	ZHL-1042J+	Mini-Circuits
RF Switch	AS169-73LF	Skyworks Solutions
Inverter Ports	74HC04D	NXP

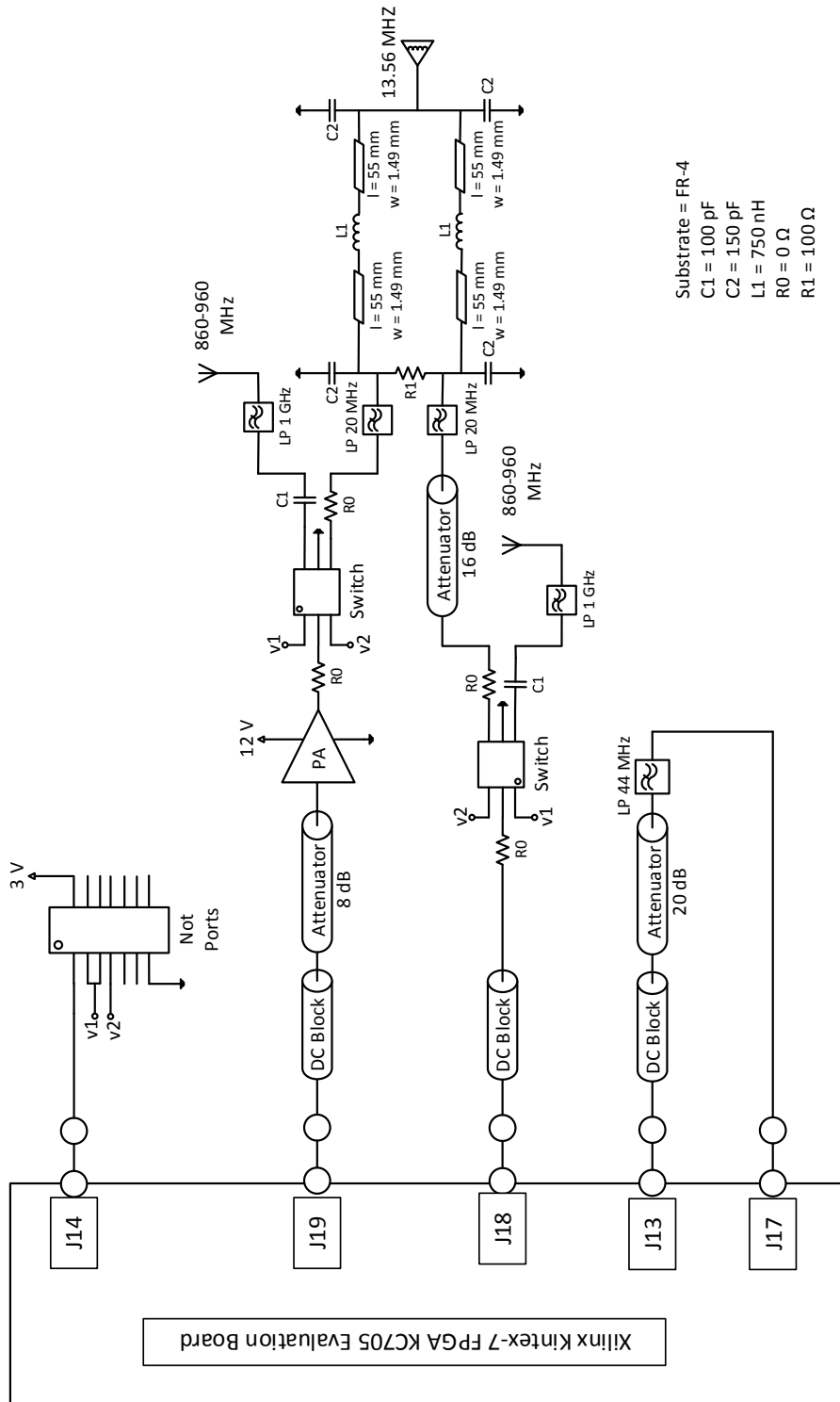


Figure 26: Implemented RF Front-End electric schematic.

References

- [1] J. A. Stankovic, “Research directions for the internet of things,” *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, Feb 2014.
- [2] N. Silva, A. Oliveira, and N. Carvalho, “Design and Optimization of Flexible and Coding Efficient All-Digital RF Transmitters,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 61, no. 1, pp. 625–632, January 2013.
- [3] R. Cordeiro, A. Oliveira, and J. Vieira, “All-digital Transmitter with Mixed-domain Combination Filter,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. PP, no. 99, pp. 1–1.
- [4] A. Prata, A. Oliveira, and N. Carvalho, “An Agile and Wideband All-Digital SDR Receiver for 5G Wireless Communications,” *18th Euromicro Conference on Digital System Design (DSD)*, August 2015.
- [5] S. Maier, Y. Xin, H. Heimpel, and A. Pascht, “Wideband base station receiver with analog-digital conversion based on RF pulse width modulation,” *IEEE MTT-S International Microwave Symposium Digest (IMS)*, pp. 1,3,2–7, June 2013.
- [6] B. Widrow, I. Kollar, and L. Ming-Chang, “Statistical theory of quantization,” *IEEE Transactions on Instrumentation and Measurement*, vol. 45, no. 2, pp. 353,361, April 1996.
- [7] GS1, “EPC Radio-Frequency Identity Protocols Generation-2 UHF RFID: Specification for RFID Air Interface Protocol for Communications at 860 MHz and 960 MHz version 2.0.1,” GS1, April 2015. [Online]. Available: <http://www.gs1.org/epcrfid/epc-rfid-uhf-air-interface-protocol/2-0-1>