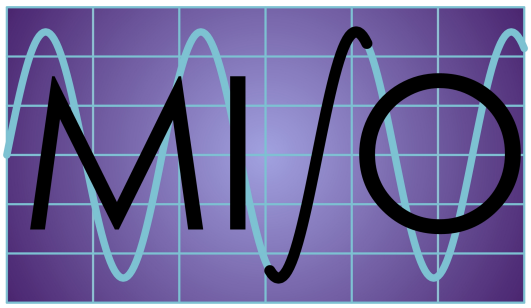
MILLIMETER WAVE SDR-BASED OPEN EXPERIMENTATION PLATFORM



USER MANUAL

June 11, 2019

Adriana Moreno
Jesus O. Lacruz
Joerg Widmer

Contents

Chapter 1

Design Methodology

Main objective of MISO Project is to propose a mixed hardware-software design for millimeter wave (mm-wave) experimentation on software-defined radios (SDR). Specifically, the basic blocks required to process the preamble (Short Training Field, STF and Channel Estimation Field, CEF) of 802.11ad compliant single carrier frames have been designed allowing mm-wave experimentation with USRP boards at a scaled-down bandwidth, compared to the one defined in the standard (1.76GSPS) [1].

The preamble processing blocks have been integrated in the GNU-Radio + RFNoC framework, allowing easy integration with other signal processing blocks from RFNoC / GNURadio community, debugging and upgrading due to the open-source nature of the project.

This document includes a description of the signal processing blocks developed for MISO project using RFNoC and integrated in the GNU-Radio framework, including an explanation of the configuration parameters of each one of the blocks, input/output signal characteristics and functionalities.

1.1 Hardware and Software setup

The final goal of the project is to test the developed system in the laboratories that the ORCA Consortium has available for remote experimentation using X310 USRP devices and GNU-Radio/RFNoC framework. Despite that, the development and testing process were conducted at IMDEA Networks Institute facilities using on-site X310 USRP devices performing over-the-wire experiments and the host PC which main characteristics are listed in Table 1.1.

From the software perspective, GNU-Radio and RFNoC tools were installed following

Component	Type
CPU	Intel Corei7-7820X CPU @3.60 GHz
RAM	128GB DDR4 @2400MHz
Ethernet Card	10GB Full Duplex card with PCIe x8 interface
SSD	Samsumg SM961 1TB

Table 1.1: Characteristics of the host PC used in the development and on-site testing at IMDEA facilities

the instructions from [2] using PyBOMBS on a Ubuntu 18.04 LTS operating system. After following the commands from [2], GNU-Radio, UHD, FPGA and gr-ettus sources were cloned into the rfnoc folder. Besides, Xilinx Vivado development tools are required for design and verification of the hardware blocks and also for FPGA image builder. Characteristics of the software tools used are summarized in Table 1.2.

Software	Version
Ubuntu OS	18.04 LTS
GNU-Radio	3.7.13.4
UHD	3.15.0
gr-ettus	–
Xilinx Vivado/Vivado HLS	2017.4

Table 1.2: Characteristics of the host PC used in the development and on-site testing at IMDEA facilities

1.2 Design Considerations

The general design and verification process described here was used for each one of the blocks developed within the MISO project unless explicitly indicated.

1. A software **floating point model** of the model is developed, considering the input and outputs required, depending on the characteristics of each block. The code is intended to be similar to the behaviour that it will have in the hardware implementation. Validation is performed over this model using real experimental data captured in over-the-air channels under different channels conditions. The hardware available at IMDEA facilities is capable of operates at the symbol rates from IEEE 802.11ad standard [1].

2. **Fixed point modelling** is introduced in the software model. 16-bit complex datapath is considered for input / output of the block in normal operation. Despite

this, special configuration modes can be used for the blocks to allow outputs with different formatting, which will be explained in detail for each block in the following chapter.

3. **Hardware Description** of the block. For MISO project some of the blocks were described using VHDL and others using C/C++ by means of Vivado HLS tools. AXI interface is considered for each block to ease integration with the RFNoC framework and the target board is the Kintex7-410T device which is the one included in the X310 USRP SDR.

4. **RFNoC packaging**. Using the `rfnocmodtool` tool (Described in [2]) all necessary files for RFNoC integration are generated. `noc_block_XX.v` is the verilog skeleton code for the block with name `XX` included on the `rfnoc/fpga-src` folder which is used to instantiate the developed block from Step 3. User registers for the block are also defined in `noc_block_XX.v` and connected to the instantiated block allowing configuration from the host PC.

5. **Verification** is conducted using a testbench skeleton code generated by means of the `rfnocmodtool` and input / output stimulus generated from the corresponding software model. The testbench architecture generated by the `rfnocmodtool` allows testing each block in the exact same environment that the one used once the FPGA image is created. Besides, multiple blocks could be tested using a single testbench and also simple functions could be used to communicate data going to/from a block under test [2].

6. **FPGA Implementation** is made using the `image_builder` script included with the RFNoC tools. The FPGA image is created with the RFNoC blocks necessary to test the system. After implementation, timing and area constraints must be met before loading the image on the X310 device. If any constraint is not met, redesigning and verification must be conducted again until all constraint be met.

7. **GNU-Radio Integration and Validation**. The .xml files generated with the `rfnocmodtool` must be updated to include reference and appropiate interface for the user registers defined in the `noc_block_XX.v` necessary to proper functioning of the corresponding block. GNU-Radio framework can be used for physical validation of the block under test, connecting it with the proper signal processing blocks, sources and sinks and using 802.11ad compliant frames generated offline (for over-the-wire experimen-

tation) and also validated with frames captured from over-the-air experiments using the available hardware at IMDEA facilities.

Steps on the hardware design enumerated above conforms an iterative process, where validation in different steps of the implementation are made in order to ease possible redesigning.

1.3 Architecture Overview

High-level overview of the system designed during the MISO project is explained in this section, including the transmitter and receiver side of the communication link.

1.3.1 Transmitter side

Since MISO project is focused mainly on the design of the receiver preamble processing blocks, the transmitter side was highly simplified. To this end, multiple frames with different Modulation Coding Skin (MCS) and packet length were generated to be used in the GNU-Radio framework as source files to be transmitted continuously using the X310 USRP and then captured and processed with the developed RFNoC blocks.

Moreover, frame files were also generated using real over-the-air experimental data for different channel conditions captured using a full bandwidth hardware available at IMDEA facilities.

Is important to remark that the frame files generated including shaping filtering using a Square Root Raised Cosine (SRRC) Filter with a oversampling rate of two and a roll-off factor ($\beta = 0.25$).

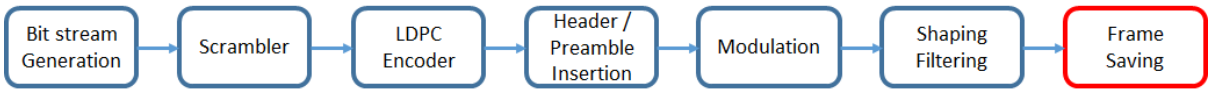


Figure 1.1: Baseband Tx Block Diagram

The rest of the blocks composing the transmitter side are the Digital Up-conversion (DUC) and Radio blocks from the RFNoC library, allowing sampling rate conversion and instantiate of the Tx RF Front-end using the X310 device and also a FIFO block intended for possible variations on samples streaming from host PC to the FPGA. Figure 1.2 shows and screenshot of GNU Radio including the building block of the simple transmitter side used in the project.

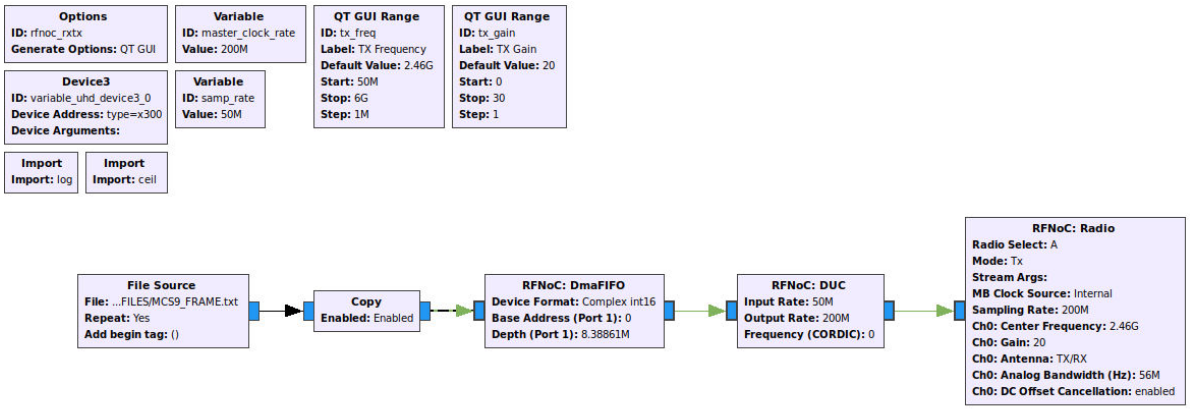


Figure 1.2: Screenshot of the GNU-Radio Tx block diagram

1.4 Receiver side

Preamble of 802.11ad Single Carrier frames are composed by to main fields: i) Short Training Field (STF) composed by multiple repetitions of Golay sequences that are intended to allow the receiver to detect the incoming packets and compensate impairments such as Carrier Frequency Offset (CFO), symbol timing deviations and also detect the channel estimation field (CEF); ii) the CEF is also formed by a special composition of Golay sequences and it is intended to compute the Channel Impulse Response (CIR) of the communication link to be used later in the channel equalization block and to determine the Channel State Information (CSI). Figure 1.3 shows the main fields that compose a single carrier 802.11ad frame and the major signal processing tasks that have to be performed over them.

Following list include the blocks designed and implemented for the MISO project. Details about each block will be explained in the following chapter. Figure 1.4 includes a data flow diagram for the proposed system, where it can be distinguish the FPGA domain where the preamble processing blocks are implemented and also the host domain which can be used for result visualization, data saving or more processing blocks could be added to the system.

Besides, from Figure 1.4 it can be seen that packet detector outputs data to the CFO estimation and correction blocks. Furthermore, CFO Correction block requires data coming from both, packet detector and CFO Estimation blocks. Taking these into account we decided to comprise the CFO estimation and correction in a single block, which will be detailed in the following chapter.

1. Packet detector

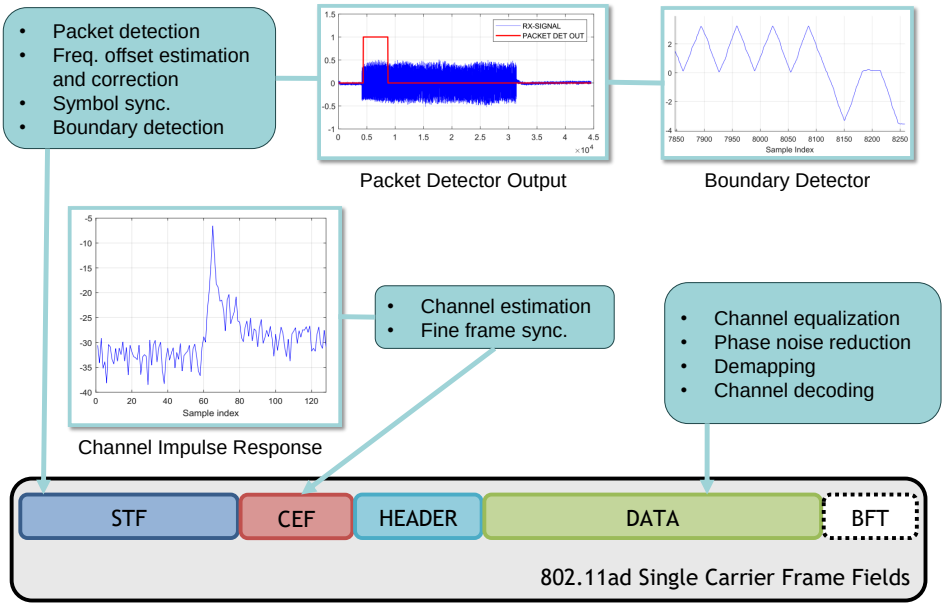


Figure 1.3: 802.11ad Single Carrier Frame Fields

2. Carrier Frequency Offset (CFO) Estimation
3. CFO Correction
4. Boundary Detector
5. Channel Impulse Response (CIR) Computation

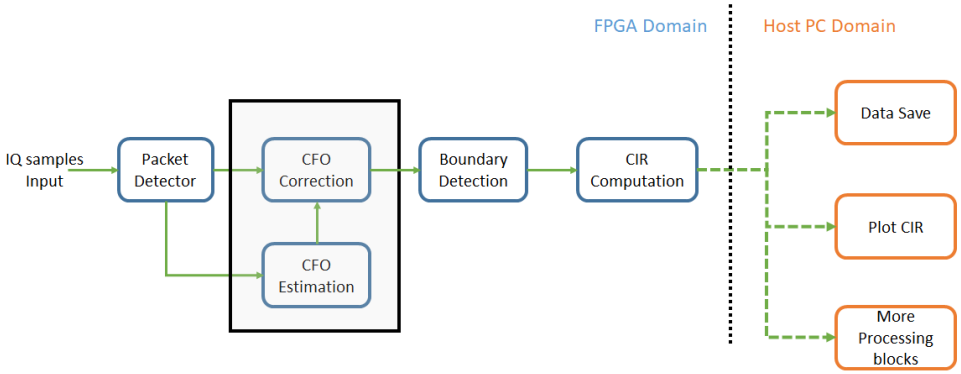


Figure 1.4: Data flow of the baseband preamble processing system

Processing blocks in the host PC are left open to the potential users of the developed system. For example, row data can be saved to a file for offline post-processing, or simply add available sink plots from the GNU-Radio library to check the output of a certain

block. Furthermore, GNU Radio signal processing blocks could be added to build more complex mixed hardware-software systems.

Chapter 2

Preamble Processing RFNoC Blocks

Each one of the blocks designed for MISO project are explained in this chapter. Explanations include basic functioning, user registers, input / output formatting and special design considerations to be taken into account when using these block in more complex designs.

Design steps described in Section 1.2 are followed to design each one of the blocks, considering the special characteristics for each block.

2.1 Packet Detector

This block computes the Normalized Auto Correlation (NAC) algorithm [3] over the complex input streaming, looking for the repetition of Golay sequences used in the STF in 802.11ad [1] and output a high-value when the NAC is greater than a certain value (threshold). Besides this well-known algorithm for packet detection, further processing have been added to increase the robustness of the block and reduce the number of false packets in certain channel conditions.

State machine diagram for the packet detector operation is shown in Figure 2.1. Basic functioning is as following:

1. **WAIT STATE:** a comparison between the NAC output and a `noise_TH` (which is configurable from a user register) is continuously done. If NAC is higher than the `noise_TH` and the NAC value is higher than the previous one, an increment over a internal counter is done. If the counter reach a predefined value (`N_COUNT` , defined

as an user register), input is considered as a valid packet and the state machine jumps to PD_STATE.

2. **PD_STATE:** state machine will remain in this state a number of predefined cycles (`PD_HIGH_TIME` , defined as an user register) and then it returns to the WAIT state, looking for a new valid packet.

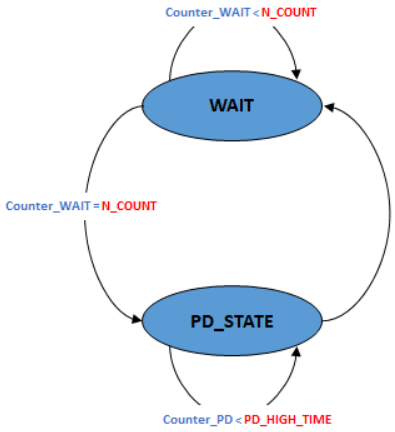
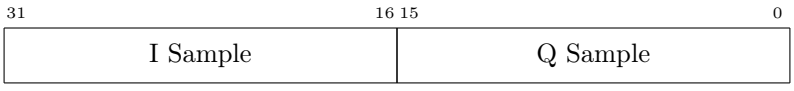


Figure 2.1: Packet detector State Machine

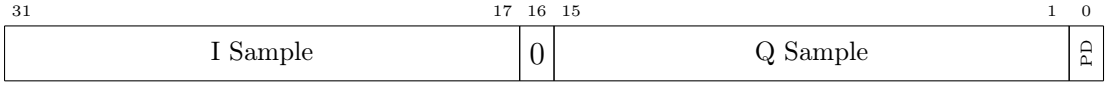
2.1.1 Input / Output Characteristics

Packet detector block is intended to input complex IQ samples (16-bit per channel) coming directly from the RFNoC Radio (RX) or RFNoC DDC blocks, depending on the bandwidth and sampling rate requirements. In any case, the block meet 200MHz timing requirements for operation with the X310 USRP devices. Besides, it is possible to input complex IQ samples coming from a source file which could be interesting for certain applications. 32-bits data going into the block is formatting as follows, assuming two samples per complex symbol:

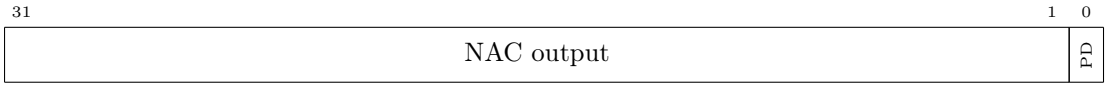


The block can generate up to four different outputs depending on a `SEL_OUT` user register value (ranging from 0 to 3, both inclusively).

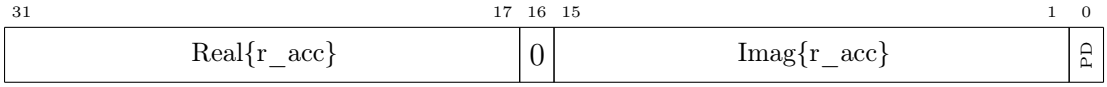
1. **SEL OUT = 0**: in this case the block outputs a 128-cycles delayed version of the input IQ samples together with the packet detected flag (PD) from the state machine. 32-bits output signal with this mode is as follow:



2. **SEL OUT = 1**: in this case the block outputs the 31 most significant bits (MSB) of the result of the NAC algorithm together with the packet detected flag (PD) from the state machine. 32-bits output signal with this mode is as follow:

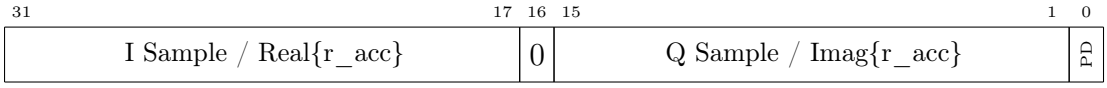


3. **SEL OUT = 2**: this mode is intended to test the CFO estimation block by means of output the results of the cross-correlation of two consecutive Golay sequences from the preamble of a frame (r_acc) [3, 4]. This corresponds to an intermediate result of a NAC computation and could be useful in certain situations where CFO estimation block requires some extra debugging or updating.



4. **SEL OUT = 3**: this is the mode or regular operation of the block when it is used together with the CFO estimation/correction block. From Figure 1.4, it can be seen that CFO estimation/correction block (surrounded by a black box) requires two inputs coming from the packet detector, one corresponding with the IQ samples along with the PD flag and another with the cross-correlation result (similar to the one from SEL OUT = 2). With the aim of maintain blocks having single input / output streams, reducing the traffic overhead over the RFNoC AXI crossbar, output of the packet detector switch from IQ samples + PD flag to r_acc when a packet is detected, then maintain this output for a time equivalent to 128*nsps*N_PER, being nsps the number of samples per symbol (nsps = 2 for this project), 128 is the length of the Golay sequence used in the preamble and N_PER is a configurable user register used to set the number of Golay sequence repetitions used to compute

the CFO (detailed in the corresponding section). After completing this number of clock cycles, the output is switched back to the IQ samples + PD Flag.



Is important to remark, that for normal operation of the blocks and with the aim of meeting area/timing constrained designs, this multiple output modes can be suppressed, keeping only the one intended for normal operation of each one of the blocks.

Figure 2.2 shows examples of each one of the output modes for the packet detector. Files containing frames from over-the-air experiments where used as sources for most of the experiments showed in this document. Figure 2.3 shows the GNU-Radio flow-graph used to validate the block and capture the data used to generate Figure 2.2.

Figure 2.2.a shows four consecutive 802.11ad frames (blue and red curves) and the PD flag (black curve). High-time or the PD flag is configured by means of the corresponding user register (Table 2.1). Besides, PD Flag has been scaled in the graph for the sake of readability. Rest of parameters values used to plot the figures can be seen in block diagram from Figure 2.3.

Figure 2.2.b shows the typical NAC output for 802.11ad frames. It can be seen how the signal rises from a low value (almost zero in the figure) to a value that depends on the received signal strength when a preamble of a valid packet is detected. Then, its value remains almost stable during the time that the STF takes and then it shows consecutive peaks before reducing to a negative value during the rest of the packet.

From Figure 2.2.c it can be seen the region where the complex correlation values remains almost stable, which allows CFO estimation in the corresponding block. FInally, Figure 2.2.d include an example of the output in normal operation mode. When the packet is detected (PD Flag rise to a high value), output change from IQ samples to complex correlation values. Then, after the pre-configured time, it jumps again to the IQ samples until a new packet is detected.

Matlab scripts was written to plot saved GNU Radio data files for each one of blocks in each one of the operations modes. These are included along with the source code of the developed blocks[1].

[1]Scripts from `https://github.com/adamgann/matlab_utils` where used to proper read GNU Radio output files from Matlab.

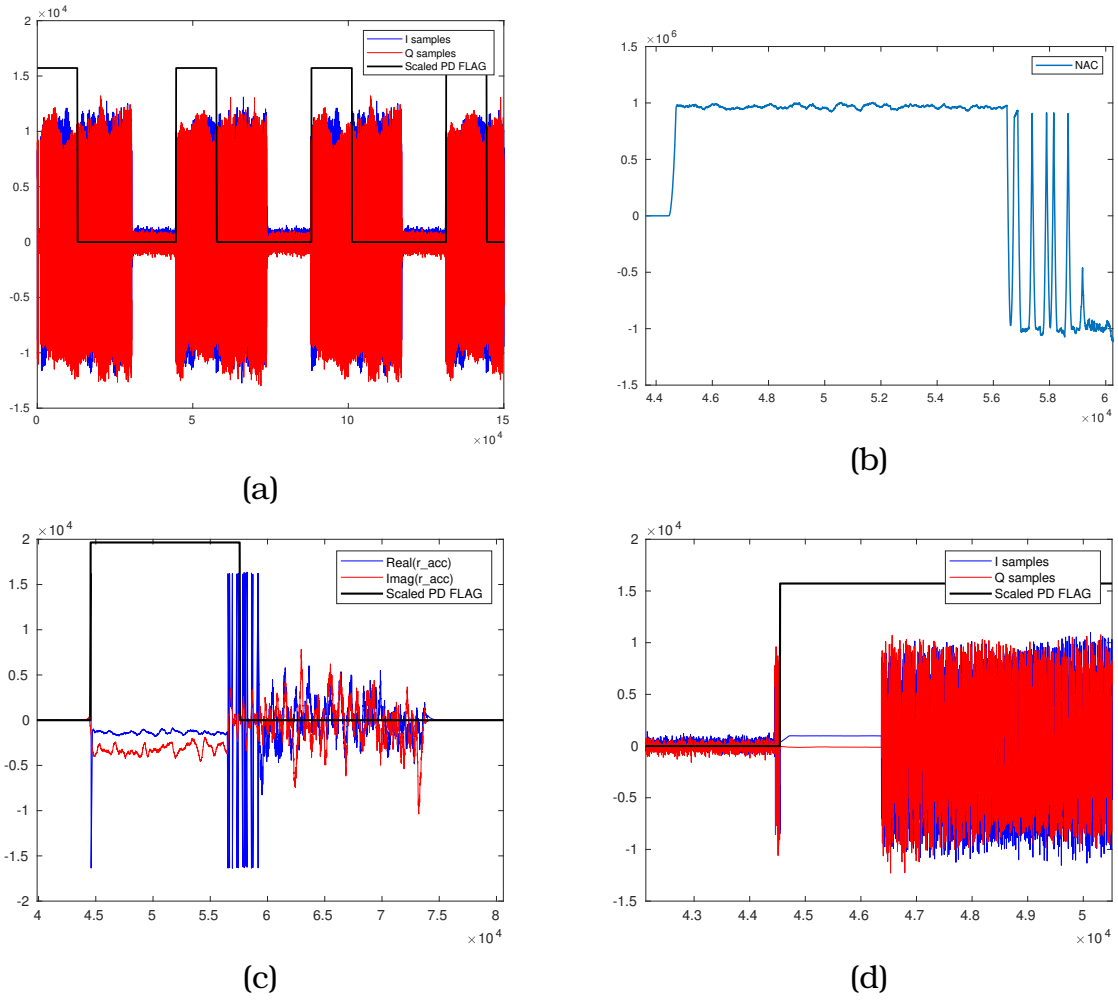


Figure 2.2: Examples of captured data for the different output modes in the packet detector block. a) SEL OUT = 0, b) SEL OUT = 1, c) SEL OUT = 2, d) SEL OUT = 3

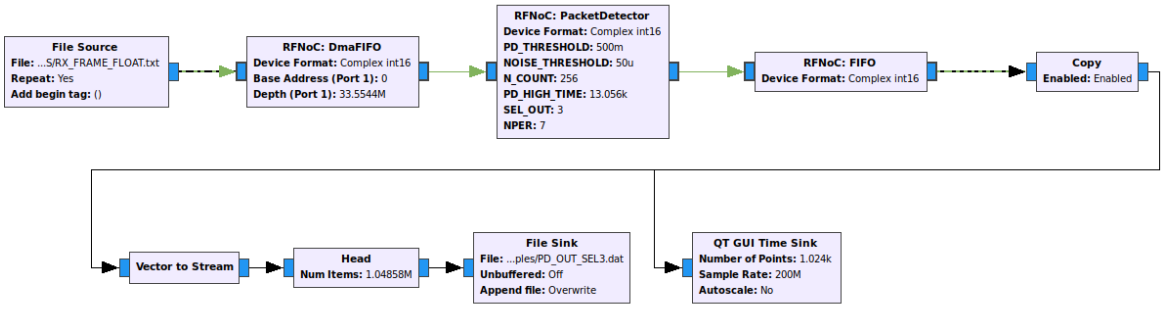


Figure 2.3: GNU-Radio flow-graph for packet detector block testing

2.1.2 User Registers

User registers for this block are listed in Table 2.1. Selection of the N_PER value must be made according to the corresponding value used in the CFO estimation/correction block (detailed in the following section).

Name	Address	Valid Values	Function
PD_THRESHOLD	133	0 < x < 1	Threshold for NAC computation [3]
NOISE_THRESHOLD	134	0 < x < 1	Noise Threshold to validate NAC output
N_COUNT	135	0 < x < 32768	Number of valid NAC values over the threshold to detect a valid packet
PD_HIGH_TIME	136	0 < x < 32768	Number of clock cycles the PD flag is high after detecting a valid packet
SEL_OUT	137	0 ≤ x ≤ 3	Output selection mode

Table 2.1: User registers for packet detection block

2.2 Carrier Frequency Offset Estimation / Correction

This block is used to estimate the possible differences in the local oscillator frequencies of transmitter up-conversion and receiver down-conversion stages and compensate its value by means of modulate the incoming IQ samples with the conjugate complex sine wave with frequency equal to the estimated one.

State machine diagram for the Carrier Frequency Offset Estimation / Correction (CFOEC) block operation is shown in Figure 2.4. Basic functioning is as following:

1. **WAIT STATE:** state machine remain in this state waiting for a PD flag = 1. When the condition is met, it jumps to the ACC STATE.

2. **ACC STATE:** Once a packet is detected, it accumulate N_PER input complex cross-correlation values (One per Golay sequence period = 128*nsps clock cycles[2]). If the number of values accumulated is equal to N_PER, it jumps to CORDIC state, else it jumps to COUNT state.

[2]Explanations on why a single value per repetition of the Golay sequence is accumulated, can be found in literature, e.g. [3, 4]

3. **COUNT STATE:** it counts 128*nsps clock cycles and then jumps to ACC state to accumulate a new cross-correlation value. Works as an intermediate state serving as a delay between each cross-correlation addition.

4. **CORDIC STATE:** it pass the accumulated cross-correlation value to the internal CFOE block responsible of compute the estimated CFO of the incoming packet. It jumps to the WAIT PD LOW state.

5. **WAIT PD STATE:** it remains in this state until PD flag goes low and then it jumps to the WAIT state. It is intended to avoid multiple computations of CFO inside a single packet.

It is important to remark than CFOEC block assumes than it is connected to the packet detector block operating with SEL OUT = 2 or 3 and N_PER has the same value in both blocks.

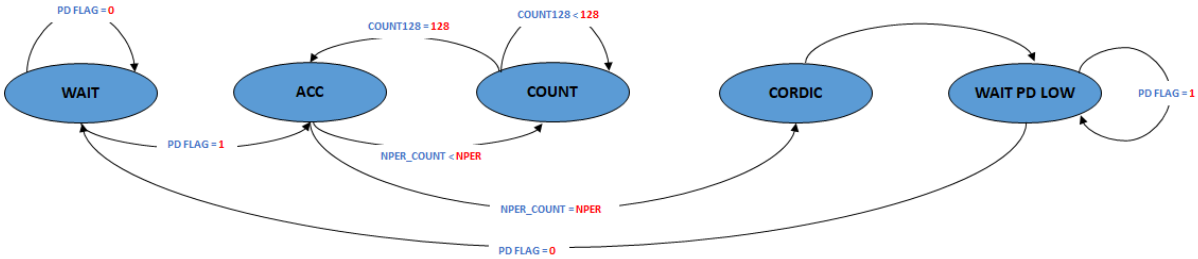


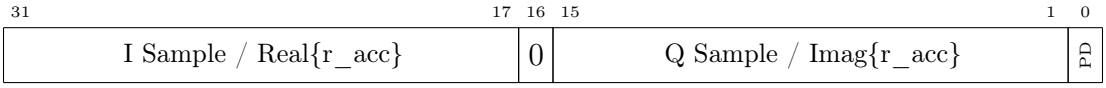
Figure 2.4: CFOEC State Machine

An internal **CFO Estimation** block computes the CFO by means of an implementation of CORDIC algorithm [5] described using Vivado HLS. The number of iterations for the algorithm can be adjusted by means of a dedicated user register ranging from 1 to 22 iterations. arctan values are stored in a Look-Up Table (LUT) considering 22-bits per position of the table. The block waits for a valid accumulated cross-correlation value to start computing during N iterations the estimated CFO which is used as input in the internal CFO Correction block.

CFO Correction is built by means of an implementation of a Direct Digital Frequency Synthesizer (DDFS) architecture based on a dual port ROM memory storing a quarter of period of a sine wave and a phase accumulator. The block multiplies the generated complex sine wave with the incoming IQ samples, besides, frequency information is updated each time a new valid packet arrives and a new frequency value is available.

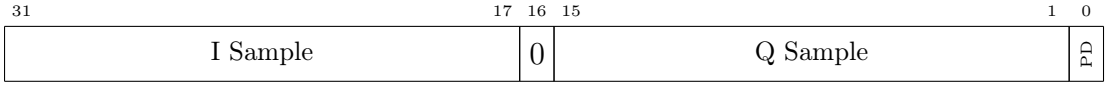
2.2.1 Input / Output Characteristics

Normal operation of the CFOEC block requires that the packet detector's output mode be configured to be SEL OUT = 3. This means that the CFOEC block's input will be changing between the IQ samples and the cross-correlation values along with the PD flag signal, as explained in the previous section. Therefore, IQ samples will be complex with 15 bit for the real and imaginary parts, considering two samples per symbol.

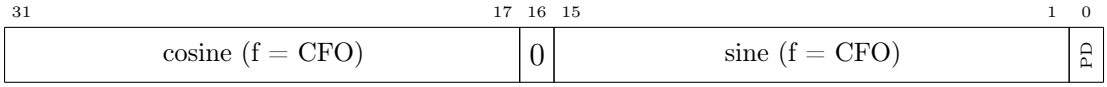


Similar to the packet decector, CFOE can generate up to three different outputs depending on the `SEL_OUT` user register value (ranging from 0 to 2, both inclusively).

1. **SEL OUT = 0**: this is the mode or regular operation of the block when it is used together with the packet detector block. Its output correspond to the complex IQ samples multiplied by the complex sine wave from the DDFS. PD flag is also included in the LSB of the output, since it is needed by the boundary detection block.



2. **SEL OUT = 1**: in this case the DDFS input (CFO Correction block) is selected to be a constant amplitude signal with the aim of generate a cosine/sine wave which frequency would correspond to the CFO present in the system. 32-bits output signal with this mode is as follow:



3. **SEL OUT = 2**: output of the block will corresponds to the the estimated angle (proportional to the CFO) from the CORDIC block (CFO Estimation). Fixed point format for the angle is 23 bits word-length with 22 fractional bits. 32-bits output signal with this mode is as follow:

31 — 24	23 — 0
Sign Extension	Estimated Angle (Q(23.22)

Figure 2.5 shows examples of each one of the output modes for the CFOEC block. It can be noted from Figure 2.5.c that the estimated angle from the CORDIC implementation change once per valid packet. In this example, it jump between two different values. Since the difference between both angle values is not high, frequency variations in the generated cosine/sine waves cannot be noted easily from Figure 2.5.c.

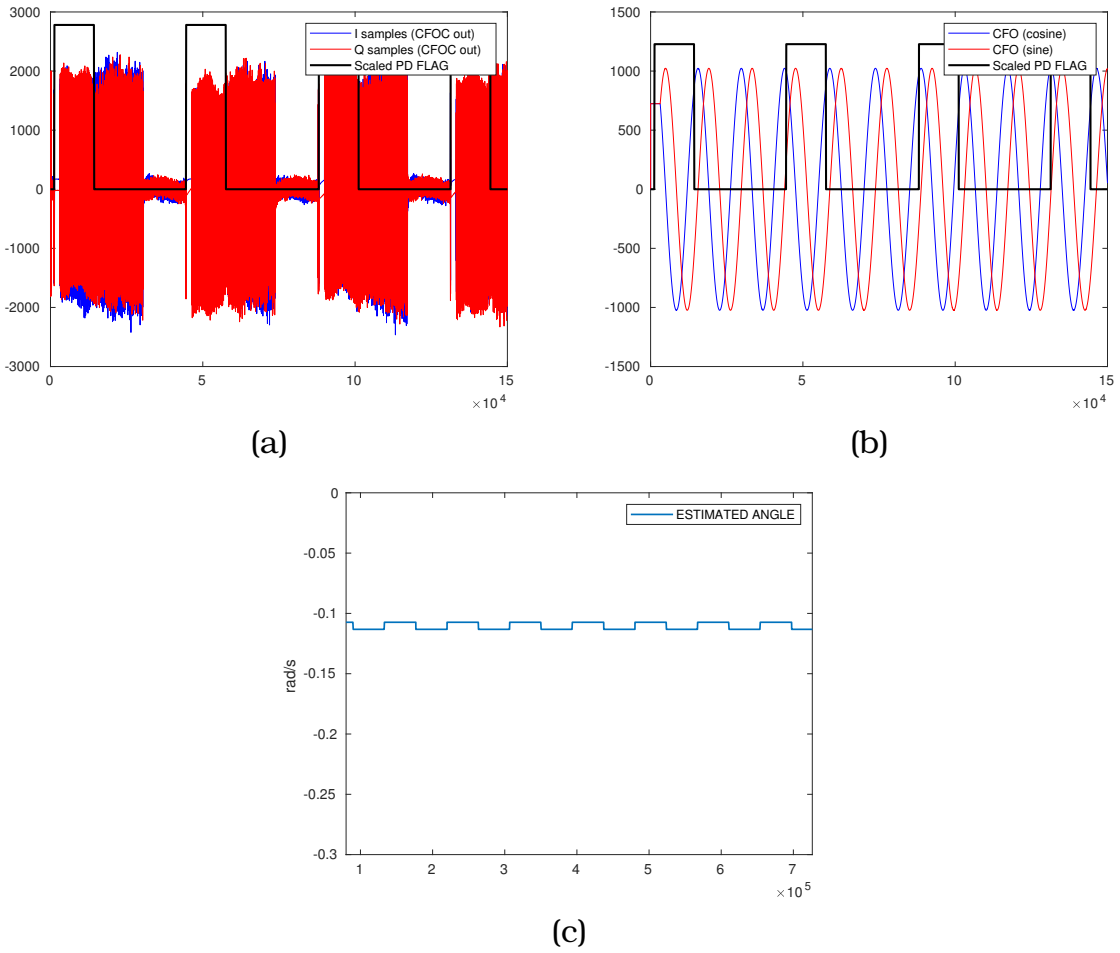


Figure 2.5: Examples of captured data for the different output modes in the CFOEC block. a) SEL OUT = 0, b) SEL OUT = 1, c) SEL OUT = 2

Figure 2.6 shows the GNU-Radio flow-graph used to validate the block and capture the data used to generate Figure 2.5. Parameters used for the validation can be also noted from figure.

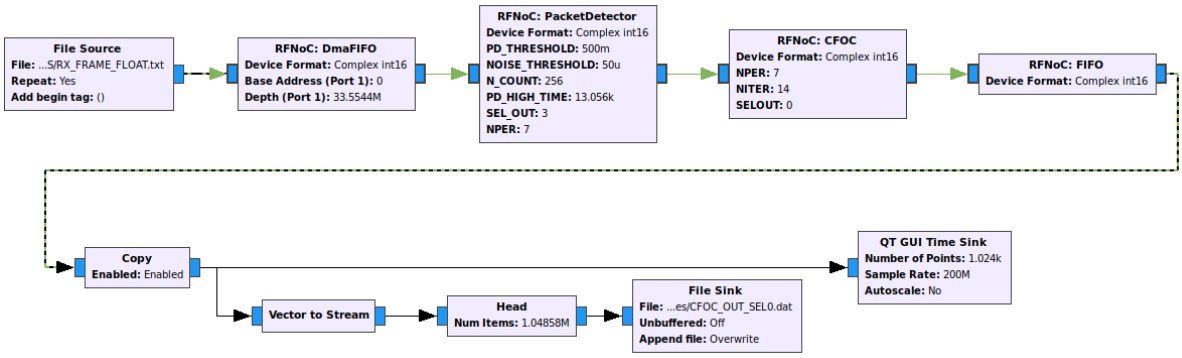


Figure 2.6: GNU-Radio flow-graph for CFOEC block testing

2.2.2 User Registers

User registers for this block are listed in Table 2.2. Selection of the N_PER value must be made according to the corresponding value used in the packet detector block.

Name	Address	Valid Values	Function
NPER	132	(0 < x < 18)	Number of repetitions of Golay sequences used to compute the CFO in the CORDIC block
NITER	133	(0 < x < 23)	Number of iterations for the CORDIC algorithm
SEL_OUT	134	0 ≤ x ≤ 2	Output selection mode

Table 2.2: User registers for CFOEC block

2.3 Boundary Detection

This block is used to determinate the last sample of the STF, which serve to notify to the block responsible to compute the channel impulse response (CIR) when it has to start its processing. Computation is based on correlation (similar to the packet detector), and usually this is implemented in the same block that the packet detection for area-saving porpoises [3]. For MISO project, we decided to implement separated blocks for packet and boundary detectors for the sake on simplify connection and exchange of information between RFNoC blocks.

Basic operation is as follows: block is waiting for a PD flag and once a rising edge occurs in this signal, boundary detection (BD) block waits for a time longer than the time required to compute and update the CFO (which is programmed by means of an

user register), giving enough time to switch the output of the packet detector back to stream the IQ samples.

Once the wait time is met, output of the correlation block pass though a 96-length shift register (SR). The block looks for 48 consecutive negative samples in the newest 48 samples in the SR. Once the pattern is met, it looks for the maximum in the oldest 48 samples samples in the SR. The index of this sample will correspond to the last sample of the STF.

State machine diagram for the BD block is shown in Figure 2.7. Basic functioning is as following:

1. **WAIT STATE:** state machine remain in this state waiting for a PD flag = 1. When the condition is met, it jumps to the PD STATE.

2. **PD STATE:** Once a packet is detected, it counts DELAY clock cycles and then jumps to WAIT PHASE CHANGE state.

3. **WAIT PHASE CHANGE STATE:** it checks continuously the condition for the 48 negative samples as explained before. Once it is met, it jumps to a LOCK OUT state. A timeout condition is included in this state to avoid system blocking when a unlikely false packet occurs and the condition for the pattern does not occur. In this unlikely case, the state machine jumps to the WAIT PD LOW STATE.

4. **LOCK OUT STATE:** it serves as a synchronization state, waiting for a predefined number of clock cycles to generate a one-cycle pulse indicating the first sample of the CEF. After that, it jumps to the WAIT PD LOW STATE.

5. **WAIT PD STATE:** it remains in this state until PD flag goes low and then it jumps to the WAIT state.

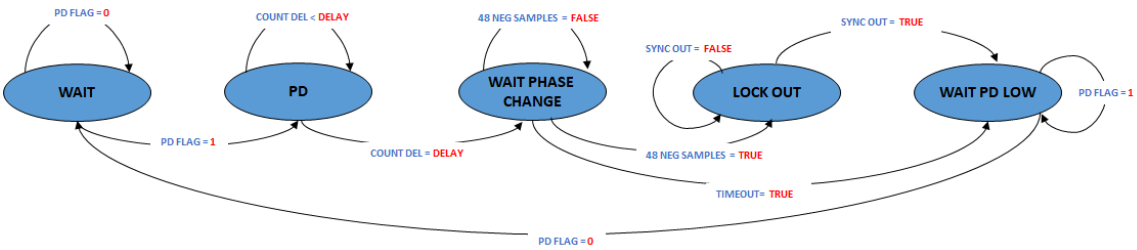
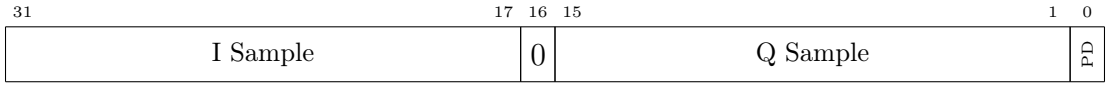


Figure 2.7: BD State Machine

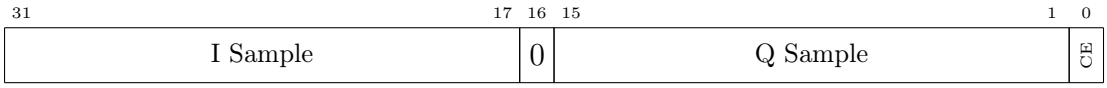
2.3.1 Input / Output Characteristics

Boundary Detector block's input expects 15-bit complex IQ samples with one sample per complex symbol and the PD flag with the following format:



Boundary Detector block can generate up to two different outputs depending on the `SEL_OUT` user register value (0 or 1).

1. **SEL OUT = 0**: this is the mode or regular operation of the block. Its output correspond to the complex IQ samples unmodified from the input of the block and the CE flag in the LSB, indicating the first sample of the CEF.



2. **SEL OUT = 1**: output will corresponds to the 32 MSB bits of the correlation output.

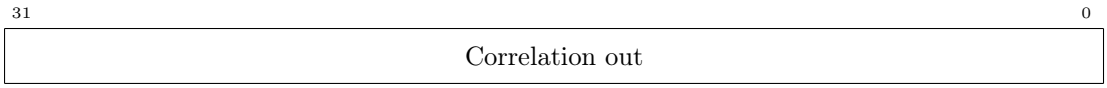


Figure 2.8 shows examples of each one of the two output modes for the BD block. Figure 2.8.a shows the IQ samples signals (unmodified in this block) and the one-cycle CE flag indicating the first sample of the CEF. Figure 2.8.b shows the typical output of the correlation used in the boundary detection block, showing the first phase-change with is used to find the last positive peak (maximum value), which corresponds to the first sample of the CEF.

Figure 2.9 shows the GNU-Radio flow-graph used to validate the block and capture the data used to generate Figure 2.8. It is really important to remark, that the Packet Detector and Carrier Frequency Offset Estimation/Correction blocks were designed to operate with an oversampling rate of two samples per symbol, but the boundary detector operates at symbol-rate (one-sample per symbol). Real systems include symbol synchronization blocks that align the symbols and at the same time downsample the signal. This

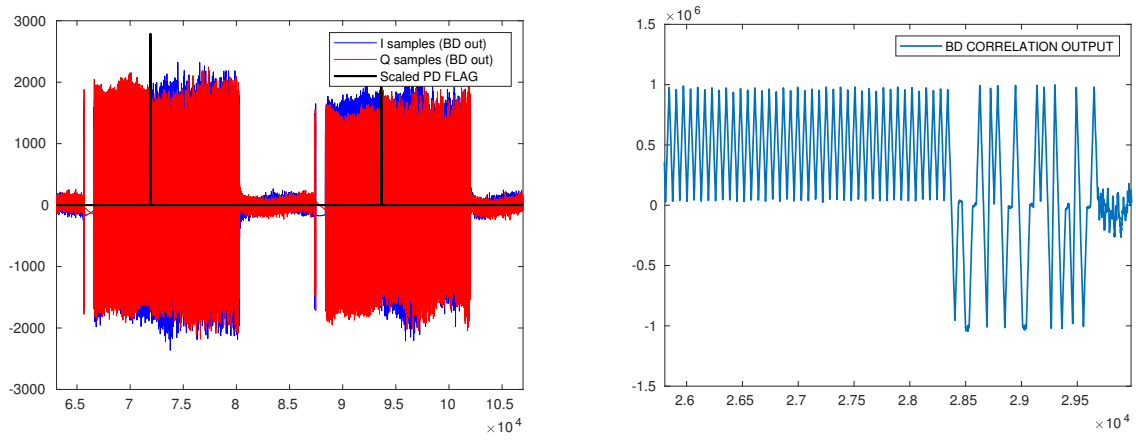


Figure 2.8: Examples of captured data for the two output modes in the BD block. a) SEL OUT = 0, b) SEL OUT = 1

block was not implemented for MISO project. Despite this, a skeleton for this block was included, which function is simply to take one out of two incoming samples. VHDL code is based on a simplified version of the KEEP 1 IN N block from RFNoC library.

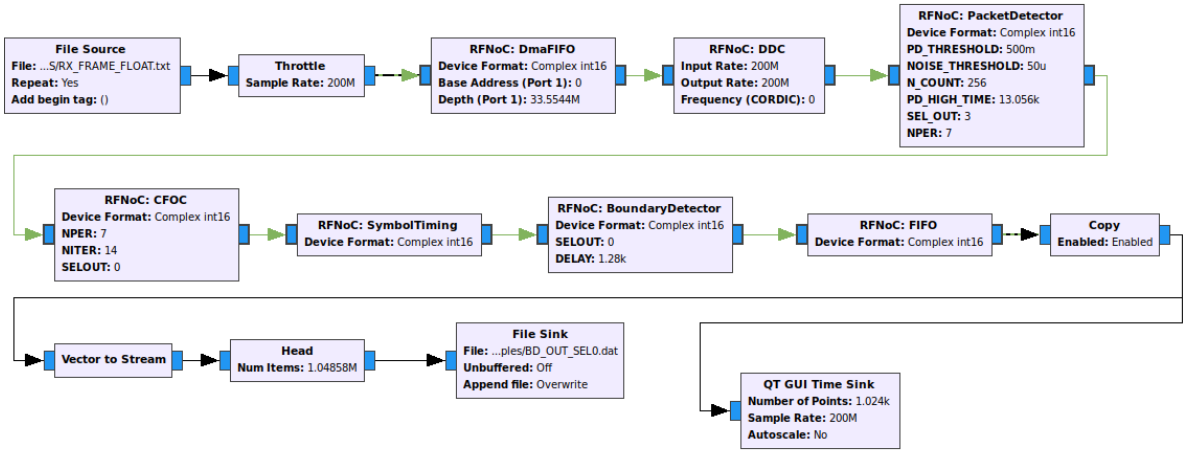


Figure 2.9: GNU-Radio flow-graph for BD block testing

2.3.2 User Registers

User registers for this block are listed in Table 2.3. Selection of the DELAY value must take into account the N_PER value from packet detector block (DELAY > 128*N_PER).

Name	Address	Valid Values	Function
DELAY	131	(0 < x < 32768)	Delay time from the PD Flag to start looking for the last sample of the STF.
SEL_OUT	132	0 ≤ x ≤ 1	Output selection mode

Table 2.3: User registers for BD block

2.4 Channel Impulse Response

This block is used to compute the channel impulse response (CIR) of the communication channel applying cross-correlation of the CEF samples of the received frame with a training samples formed by Golay sequences. Single Carrier 802.11ad CEF is composed by 1152 samples with the following structure:

-Gb128	-Ga128	Gb128	-Ga128	-Gb128	Ga128	-Gb128	-Ga128	-Gb128

CIR is computed dividing the input CEF in groups of 128 samples and then performing the cross-correlation with the corresponding Golay sequence following the structure from above. Individual results are added and finally the 128-length CIR is sent to the output. Golay sequences are stored in registers taking into account the $\pi/2$ rotation of the modulated incoming samples. Implementation of the block could be easily used with different binary training sequences due to the modular estruture used in the VHDL implementation. State machine diagram for the CIR block is shown in Figure 2.10. Basic functioning is as following:

1. **WAIT STATE:** state machine remain in this state waiting for a CE flag = 1. When the condition is met, it jumps to the COUNT128 STATE.

2. **COUNT128 STATE:** state machine remain in this state 128 valid clock cycles buffering the input samples needed for each cross-correlation computation. When the counter reach 128, it jumps to the COUNTER9 STATE.

3. **COUNT9 STATE:** take counts for each individual cross-correlation computed. While it reach 9, it jumps again to the COUNT128 STATE to buffer a new set of 128 samples. When counter reach 9, it jumps to a GEN OUT state indicating that a valid CIR has been computed and a output have to be generated.

4. **GEN OUT STATE:** used to generate a flag used to generate the 128-length CIR at the output port of the block. It jumps to the WAIT state after one clock cycle.

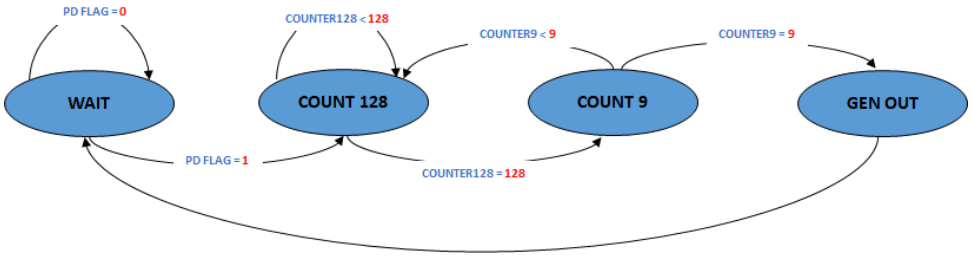
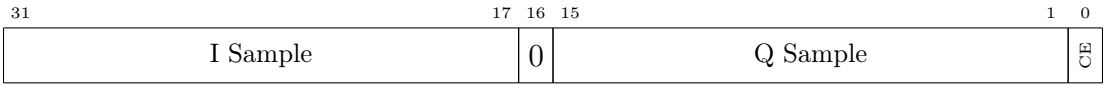


Figure 2.10: CIR State Machine

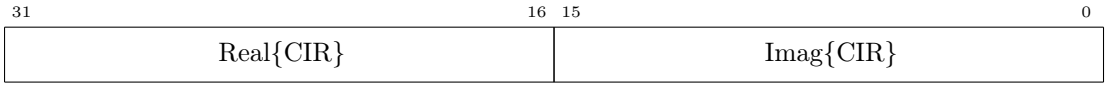
2.4.1 Input / Output Characteristics

CIR block's input expects 15-bit complex IQ samples with one sample per complex symbol and the CE flag with the following format:



For the block's output it is important to remark that it only generate 128 samples per valid packet corresponding to the CIR estimated. Taking this into account, two different outputs depending on the `SEL_OUT` user register value (0 or 1) can be generated:

1. **SEL OUT = 0**: in this case, 16 MSB bits per sample of the complex impulse response is produced.



2. **SEL OUT = 1**: output will corresponds to the 32 MSB bits of the absolute value of the computed CIR, hardware implemented using the complex CIR samples.

Figure 2.11 shows examples of each one of the two output modes for the CIR block. Each sub-figure show five consecutive computed CIR.

Figure 2.12 shows the GNU-Radio flow-graph used to validate the block and capture the data used to generate Figure 2.11.

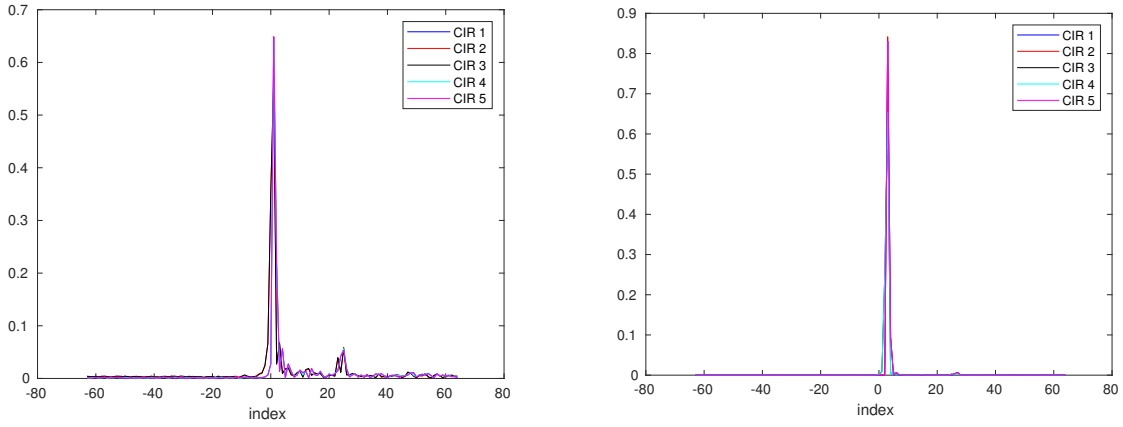


Figure 2.11: Examples of captured data for the two output modes in the CIR block. a) SEL OUT = 0, b) SEL OUT = 1

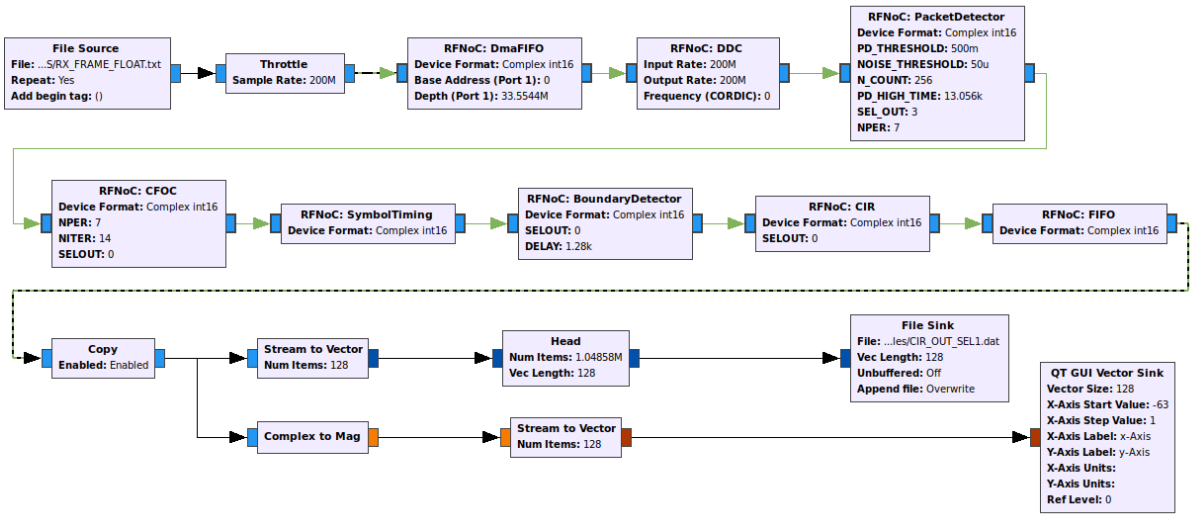


Figure 2.12: GNU-Radio flow-graph for CIR block testing

2.4.2 User Registers

For this block, only the SEL OUT user register is needed to proper configuration of the CIR block.

Name	Address	Valid Values	Function
SEL_OUT	131	$0 \leq x \leq 1$	Output selection mode

Table 2.4: User registers for CIR block

Chapter 3

Useful information

Compressed `MISO_PROJECT.zip` file accompanies this document. It includes the main folders listed below:

1. **FRAMES:** it includes IEEE 802.11ad compliant single carrier frames that could be used to validate the proper functioning or perform experiments with the blocks developed in this project. Noiseless frame files for different MCS (ranging from 1 to 12) are included.

   Besides, two files captured from over-the-air experiments were also included. Each one of these files includes frames for each MCS sweeping continuously from 1 to 12.

2. **rfnoc-ORCA_BLOCKS:** includes the folder structure for the out-of-tree (OOT) module created using the `rfnocmodtool` to accommodate all the blocks developed during the project.

   - `rfnoc/fpga-src`: includes the HDL files for each one of the developed blocks. Each `noc_block_name_of_the_block.v` block serve as the AXI-RFNoC shell wrapper for the VHDL files which actually implement the block's functionality.

     It is easy to follow the VHDL files corresponding to each designed block, since their name begin with the initials of the block that they belong. PD for the packet detector, CFO for the carrier frequency offset, BD for the boundary detector and CIR for the channel impulse response.

     Although some block functionalities (from the CFOEC block) were implemented using vivado HLS, the VHDL files from synthesis are copied to this folder.

- `rfnoc/testbench`: it includes a list of sub-folders (one per designed RFNoC block). Inside each sub-folders there is a `.sv` file which corresponds to the top-level testbench file for each block. Besides, a `FIXED_POINT_MODEL` folder included the software model for each one of the blocks. The software model was coded trying to mimic the behavior of its corresponding hardware implemented block.
- `examples`: it includes GNURadio block diagram to test each one of the blocks. They match the block diagrams from Chapter 2. Besides, `.m` scripts are also included to process the outputs files generated when running each one of the examples provided and also the `.dat` recorded files used to generate the figures from Chapter 2.

3. **VIVADO_HLS_FILES:** it includes the source files necessary to synthesize the blocks for the cordic and DDFS implementations. Besides, scripts were also included to generate the Vivado HLS projects, synthesize and implement the designs. They can be executed from command line typing:

```
vivado_hls -f script_CFOXXX.tcl
```

   where `XXX` corresponds to the specific name of the script depending on the block. After implementation, all VHDL files from `CFOXXX/solution1/syn/vhdl` folder must be copied to the `rfnoc-ORCA_BLOCKS/rfnoc/fpga-src` folder and included in the Makefile from the same location.

4. **FPGA IMAGES:** this folder contain a FPGA image built with the software versions listed in Table 1.2. It contain the following RFNoC blocks:

   - **PacketDetector**
   - **CFOC**
   - **SymbolTiming**
   - **BoundaryDetector**
   - **CIR**
   - DDC
   - DUC

- FIR
- DmaFIFO
- FIFO (x2)

FPGA image can be built using the scripts provided within the RFNoC framework [2]:

```
./uhd_image_builder.py PacketDetector CFOC ddc duc fir_filter SymbolTiming
BoundaryDetector CIR -I [USER PREFIX]/src/rfnoc-ORCA_BLOCKS/ -d x310
-t X310_RFNOC_HG -m 10 --fill-with-fifos}
```

Where in this case, `[USER PREFIX]` refers to the rfnoc installation folder. FPGA image loading is made as indicated in [2] web-page and it is replicated here to ease reference.

```
uhd_image_loader --args "type=x300,addr=XXX.XXX.XXX.XXX" --fpga-path
[USER PREFIX]/FPGA_IMAGES/MISO_IMAGE_x300.bit
```

Bibliography

[1] IEEE 802.11 working group, "Standard for Information technology-telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 Ghz Band." Dec. 2012.

[2] Ettus Research. (2017) Getting Started with RFNoC Development. [Online]. Available: https://kb.ettus.com/Getting_Started_with_RFNoC_Development

[3] W. Liu, T. Wei, Y. Huang, C. Chan, and S. Jou, "All-Digital Synchronization for SC/OFDM Mode of IEEE 802.15.3c and IEEE 802.11ad," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 2, pp. 545–553, Feb 2015.

[4] Y. Huang, W. Liu, and S. Jou, "Design and implementation of synchronization detection for IEEE 802.15.3c," in *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*, April 2011, pp. 1–4.

[5] P. K. Meher, J. Valls, T. Juang, K. Sridharan, and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009.