Lime Microsystems Limited

Surrey Technology Centre

Occam Road

The Surrey Research Park

Guildford GU2 7YG

Surrey

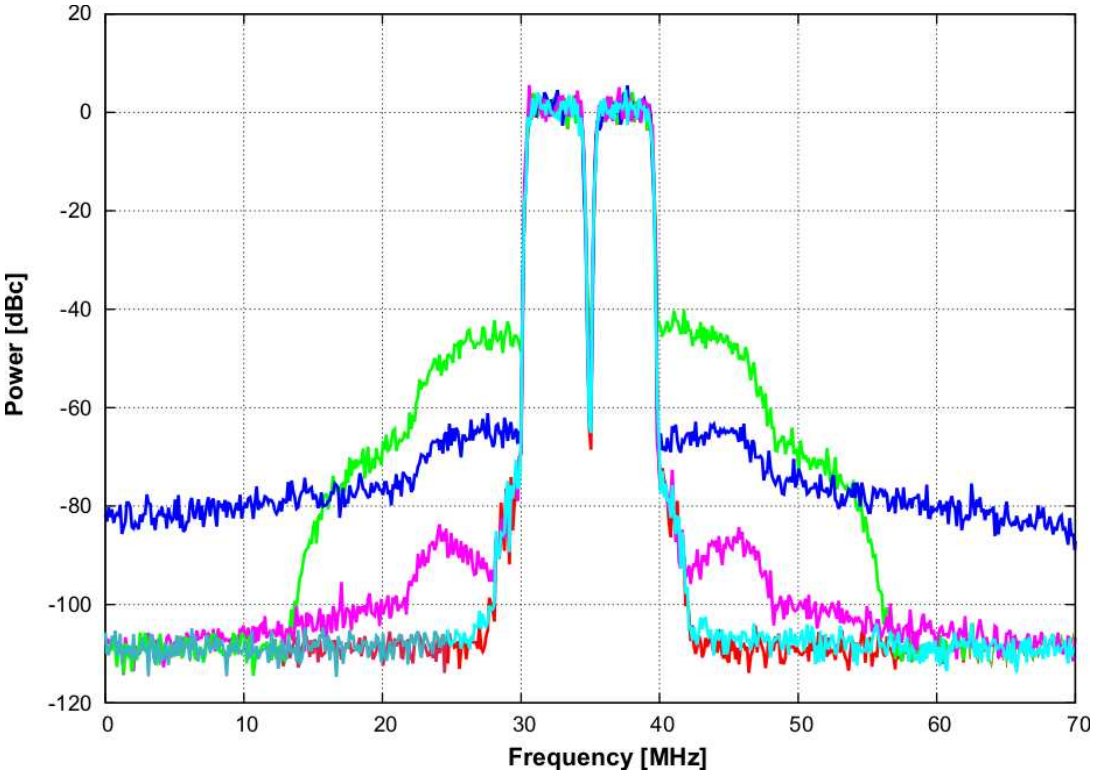
United Kingdom

Tel: +44 (0) 1483 685 063

e-mail: enquiries@limemicro.com

Adaptive Digital Predistortion of RF Power Amplifiers and Crest Factor Reduction

-Application Note-


20
0
-20
-40
-60
-80
-100
-120
Power [dBc]
0
10
20
30
40
50
60
70
Frequency [MHz]

Contents

1

Introduction

Power amplifiers (PA) are nonlinear devices and their linearization is highly desired for a number of reasons. In case of RF PAs, linearization improves power efficiency and subsequently reduces running cost of the wireless infrastructure.

Considering the PA performance for a given air interface, ACPR and EVM are the key considerations to provide support for sophisticated modulation schemes, multicarrier signals and high modulation bandwidths.

Here, we present the Lime solution for PA linearization based on adaptive digital predistortion (ADPD) and crest factor reduction (CFR).

2

Lime ADPD Structure

2.1 Indirect Learning Architecture

The simplified block diagram of an indirect learning architecture is given in Figure 1. Please note that RF part in both TX (up to PA input) neither in RX (back to base band frequency) paths is shown for simplicity.

Delay line compensates ADPD loop (*yp(n)* to *x(n)*) delay. Postdistorter is trained to be inverse of power amplifier. Predistorter is simple copy of postdistorter. When converged:

$\varepsilon(n)=0$, $yp(n)=y(n) \Rightarrow x(n)=xp(n)$,

hence, PA is linearized.

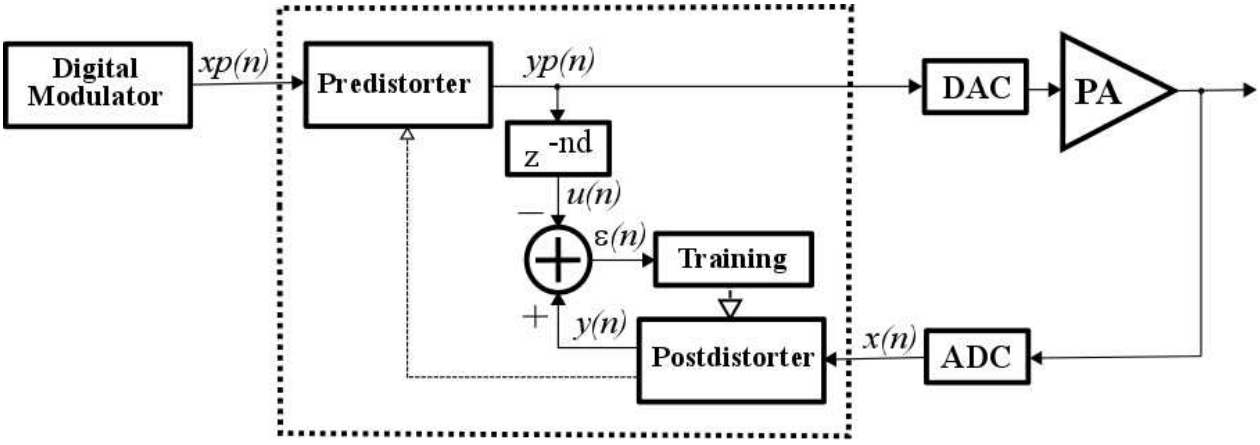


Figure 1: Indirect learning architecture

2.2 Complex Valued Memory Polynomial

LimeADPD algorithm is based on modelling nonlinear system (PA and its inverse in this case) by complex valued memory polynomials which are in fact cut version of Volterra series which is well known as general nonlinear system modelling and identification approach. In this particular case “cut version ” means the system can efficiently be implemented in real life applications.

For a given complex input:

$$x(n)=x_I(n)+j\,x_Q(n) \quad ,$$

complex valued memory polynomial produces complex output:

$$y(n)=y_I(n)+j\,y_Q(n) \quad ,$$

$$y(n)=\sum_{i=0}^{N}\sum_{j=0}^{M} w_{ij}\,x(n-i)e(n-i)^j$$

where:

$$w_{ij}=a_{ij}+j\,b_{ij} \quad ,$$

are the polynomial coefficients while *e(n)* is the envelop of the input. For the envelop calculation, two options are considered, the usual one:

$$e(n)=\sqrt{x_I(n)^2+x_Q(n)^2} \quad ,$$

and the squared one:

$$e(n)=x_I(n)^2+x_Q(n)^2 \quad .$$

Usually, squared one is used in ADPD applications since it is simpler to calculate and in most cases provides even better results.

In the above equations, *N* is memory length while *M* represents nonlinearity order. Hence, complex valued memory polynomial can be taken into account both system memory effects as well as the system nonlinearity.

2.3 LimeADPD Equations

Based on discussions given in previous sections and using signal notations of Figure 1, ADPD predistorter implements the following equations:

$$\mathbf{yp}(n)=\sum_{i=0}^{N}\sum_{j=0}^{M} w_{ij}\,\mathbf{xp}(n-i)ep(n-i)^j \quad ,$$

$$\mathbf{xp}(n)=xp_I(n)+j\,xp_Q(n) \quad ,$$

$$ep(n)=xp_I(n)^2+xp_Q(n)^2 \quad ,$$

while postdistorter does similar:

$$y(n)=\sum_{i=0}^{N}\sum_{j=0}^{M} w_{ij}\, x(n-i)e(n-i)^{j} \quad ,$$

$$x(n)=x_I(n)+j\,x_Q(n) \quad ,$$

$$e(n)=x_I(n)^2+x_Q(n)^2 \quad .$$

Note that predistorter and postdistorter share the same set of complex coefficients $\mathbf{w}_{ij}$. Delay line is simple and its output is given by:

$$u(n)=\mathbf{yp}(n-nd) \quad .$$

2.4 Training Algorithm

ADPD training algorithm alters complex valued memory polynomial coefficients $\mathbf{w}_{ij}$ in order to minimise the difference between PA input $\mathbf{yp}(n)$ and $\mathbf{y}(n)$, ignoring the delay and gain difference between the two signals. Instantaneous error shown in Figure 1 is calculated as:

$$\varepsilon(n)=\sqrt{\left|u_I(n)-y_I(n)\right|^2+\left|u_Q(n)-y_Q(n)\right|^2} \quad .$$

Training is based on minimising Recursive Least Square (RLS) *E(n)* error:

$$E(n)=\frac{1}{2}\sum_{m=0}^{n}\lambda^{n-m}\varepsilon(m)^2,\ \lambda<1 \quad ,$$

by solving linear system of equations:

$$\frac{\partial E(n)}{\partial a_{kl}}=0,\ \frac{\partial E(n)}{\partial b_{kl}}=0;\ k=0,1,\ldots,N;\ l=0,1,\ldots,M \quad .$$

Any linear equation system solving algorithm can be used. Lime ADPD involves LU decomposition. However, iterative techniques such as Gauss – Seidel and Gradient Descent have been evaluated as well. LU decomposition is adopted in order to get faster adaptation and tracking of the ADPD loop.

3

Peak to Average Power Ratio Reduction (Crest Factor Reduction)

3.1 Motivation for Peak to Average Power Ratio Reduction

Modern modulation schemes (LTE, 5G NR and similar) generate signals which inherently contain frequent and large peaks in time domain. In the presence of high Peak to Average Power Ratio (PAPR), PAs must be heavily backed off and/or further linearized by DPD. The efficiency of DPD algorithms themselves is also affected by high PAPR.

The solution which helps DPD to compensate PA distortions more efficiently is to deal with the signals with reduced PAPR. In this case, it is possible to increase transmitted signal average power, avoid PA operation in non-linear region and consequently improve PA energy and spectral efficiency.

The utilization of PAPR reduction techniques in modern telecommunication systems becomes a “must have” option, at least at BTS side.

NOTE: Peak to Average Power Ratio Reduction (PAPR) is also known as Crest Factor Reduction (CFR). The Crest Factor Ratio (CFR) is defined as a ratio between the signal magnitude maximum value and the signal average value:

$$CFR = \frac{\|s(n)\|_{\max}}{s_{rms}}$$

Peak to Average Power Ratio (PAPR) is defined as:

$$PAPR = \frac{\|s(n)\|_{\max}^2}{s_{rms}^2} \quad \text{or} \quad PAPR_{dB} = 10\log_{10}\frac{\|s(n)\|_{\max}^2}{s_{rms}^2}$$

3.2 Peak Windowing Method

Hard Clipping (HC) technique cuts the peaks when the envelope $|x(n)|$ of the complex signal $x(n)$ exceeds user selectable threshold level Th:

$$y(n)=c(n)x(n)$$

$$c(n)=\begin{cases}\dfrac{Th}{|x(n)|}, |x(n)|>Th\\ 1, |x(n)|\le Th\end{cases}$$

HC is quite simple. However it produces high signal distortion due to hard clipping. Undesirable side effects of HC include in-band signal distortion which is measured by Error Vector Magnitude (EVM) and out-band signal distortion, measured by Adjacent Channel Power Ratio (ACPR). For that reason, we have adopted Peak Windowing (PW) algorithm for PAPR reduction. In PW, the large signal peaks are multiplied with a windowing function to smooth the sharp edges at clipping points. In fact, the above clipping coefficients $c(n)$ are replaced by $b(n)$:

$$b(n)=1-\sum_{k=-\infty}^{k=\infty}(1-c(k))w(n-k)$$

where w(n) is some symmetrical windowing function (Hann's for example).

$$b(n)=1-\sum_{k=-\infty}^{k=\infty}(1-c(k))w(n-k)\le c(n)$$

The difference between $c(n)$ and $b(n)$ is minimized by choosing narrow window lengths which results in lower EVM degradation. However, if clipping operation happens frequently, neighboring correction windows overlap and the difference between $c(n)$ and $b(n)$ becomes larger.

For LTE standards, usually 30.72 MS/s sample rate is used. CFR block itself runs at 122.88 MHz clock frequency, i.e. four times the data rate in order to bust hardware DSP blocks processing power. Of course, these frequencies can easily be changed to conform to any other similar telecommunication standard.

The implementation of PW consists of several stages. The PW processing operations are depicted in Figure 2.a.

To determine the envelope $e(n)=|x(n)|$, complex I/Q input components are squared, summed and square-rooted. The envelope $e(n)$ is then compared to the threshold level Th.

If the magnitude of $e(n)$ is greater than threshold Th, clipping coefficient $c(n)$ is calculated as the value of Th divided by $e(n)$. Otherwise, $c(n)$ value is set to one.

Peak search block is introduced in PW processing stage to find local minimum values of the signal $c(n)$. If the input sample is not local minimum, then the output of Peak search block (signal $cp(n)$) is set to one.

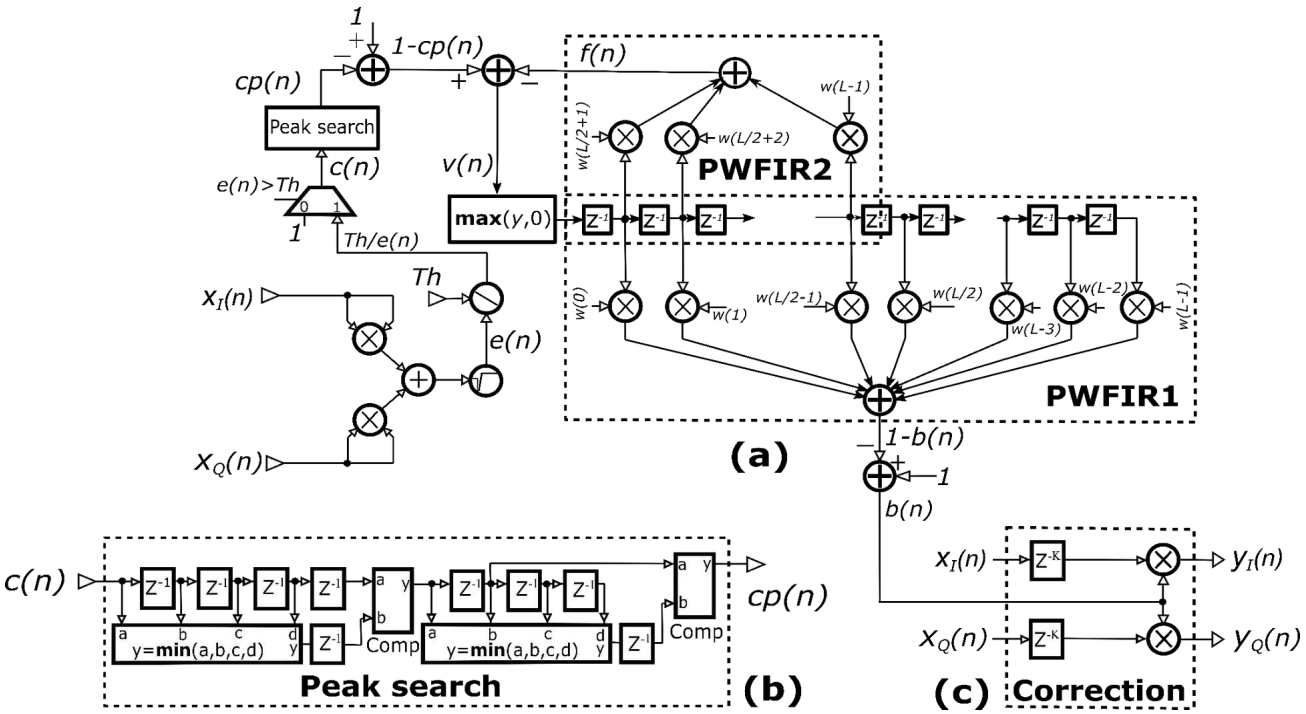


Figure 2: (a) Peak windowing architecture (b) Peak search block and (c) Correction bock

Figure 2.a shows the architecture of PWFIR filter. PWFIR consists of feed-forward (PWFIR1) and feedback (PWFIR2) sub-filters.

PWFIR takes input signal *v(n)* and generates 1-*b(n)*. Negative values of *v(n)* are replaced by zeros before driving the rest of the filter. The sequence *b(n)* is gain correction of the input sequences $x_I(n)$ and $x_Q(n)$.

Prior to applying the correction, $x_I(n)$ and $x_Q(n)$ are properly delayed to compensate for the delay (latency) introduced by the implementation of PWFIR and other CFR preprocessing stages. Hence, CFR output *y(n)* is constructed as shown in Figure 2.c.

In order to reduce overlapping, the feedback structure (PWFIR2) is introduced. The feedback path adjusts the next filter input value. Looking forward to when clipped input value reaches the center tap (Figure 2.a), the contribution of all previous input values (between first and center tap) are calculated and used for correction of the next input value. At the time when incoming clipped signal reaches the unity weighted center tap, the contributions from all previous values have already been compensated. This reduces the EVM degradation when successive peaks in *cp(n)* occur within the period shorter than half of the window length.

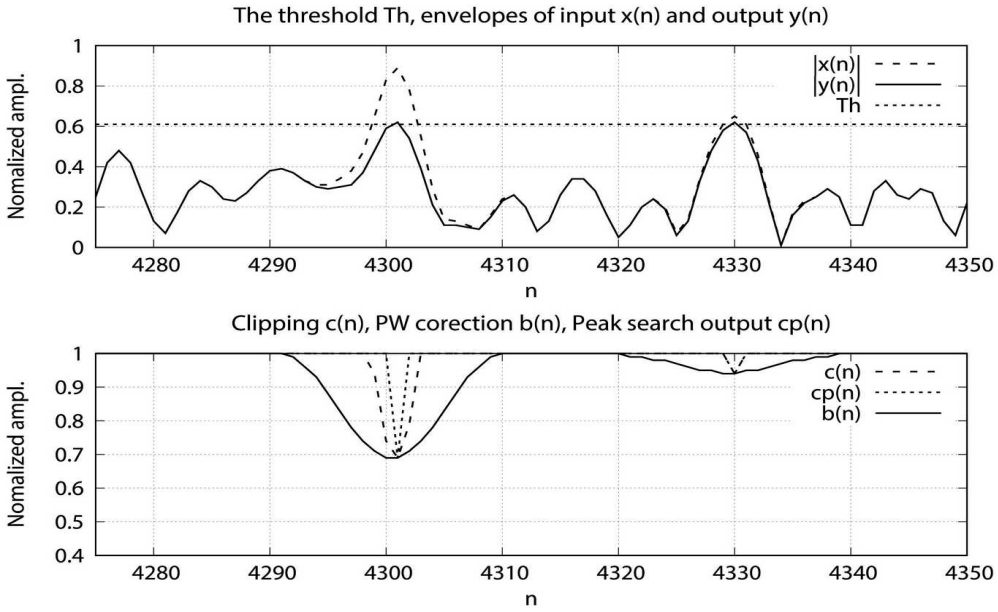


Figure 3: CFR algorithm in action. Bottom graph gives c(n), cp(n) and b(n), top graph shows Th, e(n)=|x(n)| and envelope of the CFR output |y(n)|.

As mentioned before, PWFIR filter structure is divided into two parts. PWFIR1 produces output signal 1-*b(n)* while PWFIR2 generates the feedback signal *f(n)* (Figure 2.a). For the implementation, we have chosen Hann windowing function:

$$w(k)=\frac{1}{2}\left(1-\cos\left(2\pi\frac{k}{L-1}\right)\right),0\le k\le L-1$$

PWFIR is designed to implement 1 <= *L* <= 40 tap filters where the filter length *L* and the filter coefficients *w(k)* are easily software programmable.

The architecture of PWFIR filter is based on multiply-and-accumulate (MAC) circuitry.

The architecture is area optimized. The number of utilized multipliers is reduced by multiplexing input data and operating the block at the clock frequency which is four times higher than the sample rate. PWFIR operates at the clock frequency of 122.88 MHz while input and output data rates are both equal to 30.72 MS/s.

The architecture of CFR FIR filter is further optimized by exploiting the fact that the filter has symmetrical window coefficients. The required number of multiplications is reduces by factor of 1/2. Consequentially, the number of FPGA DSP blocks used for the filter implementation is also reduced.

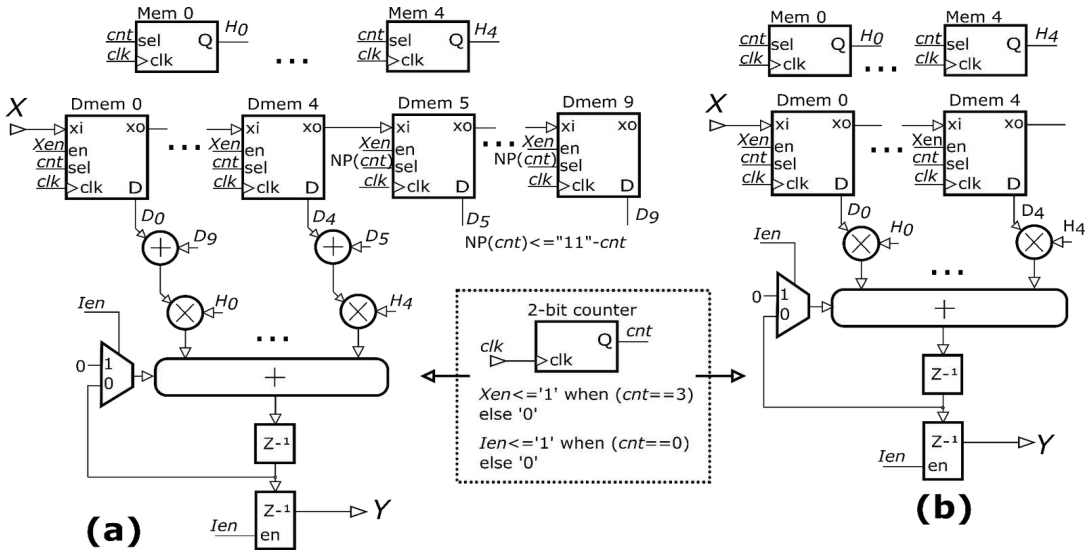


Figure 4: The architecture of PWFIR1 (a) and PWFIR2 (b) modules

The detailed architecture of PWFIR1 is given in Figure 4.a. The coefficients are indexed from 0 to 19. Whenever the condition is true, the coefficient at index j is determined by following equation. Otherwise, the coefficient is set to zero.

$$h_{PWFIR1}(j) = w\left(j - \left(20 - \left\lceil \frac{L+1}{2} \right\rceil\right)\right)$$

$$20 - \left\lceil \frac{L+1}{2} \right\rceil \le j \le 19$$

PWFIR2 architecture is given in Figure 4.b. It provides up to 20 programmable filter coefficients which are stored in the register array and indexed from 0 to 19. The coefficients of PWFIR2 are determined by following equation whenever condition is met. Otherwise, the coefficient value at index j is set to zero.

$$h_{PWFIR2}(j) = w\left(\left\lceil \frac{L+1}{2} \right\rceil + j\right)$$

$$0 \le j \le \left\lceil \frac{L}{2} \right\rceil - 1$$

4

Implementation Platform

ADPD and CFR algorithms are implemented using LimeSDR-QPCIe board, a high level block diagram of which is shown in Figure 5. LimeSDR-QPCIe board has a lot more options than shown (two LMS7002M chips, USB interface, GPS receiver, …). LMS7002M itself is 2T-2R RF IC. For clarity, Figure shows the minimum hardware options required to illustrate the LimeADPD implementation. Regarding the implementation, the same signal names as in Figure 1 are used here.

DPD operates at 122.88 MHz clock while its input/output sample rates are 61.44 MS/s. Interpolation block is used to double usual LTE sample rate of 30.72 MS/s before driving DPD block. DPD sample rate of 61.44 MS/s (or higher) is required if we want to cancel at least IMD3 products in case of 20 MHz modulation.

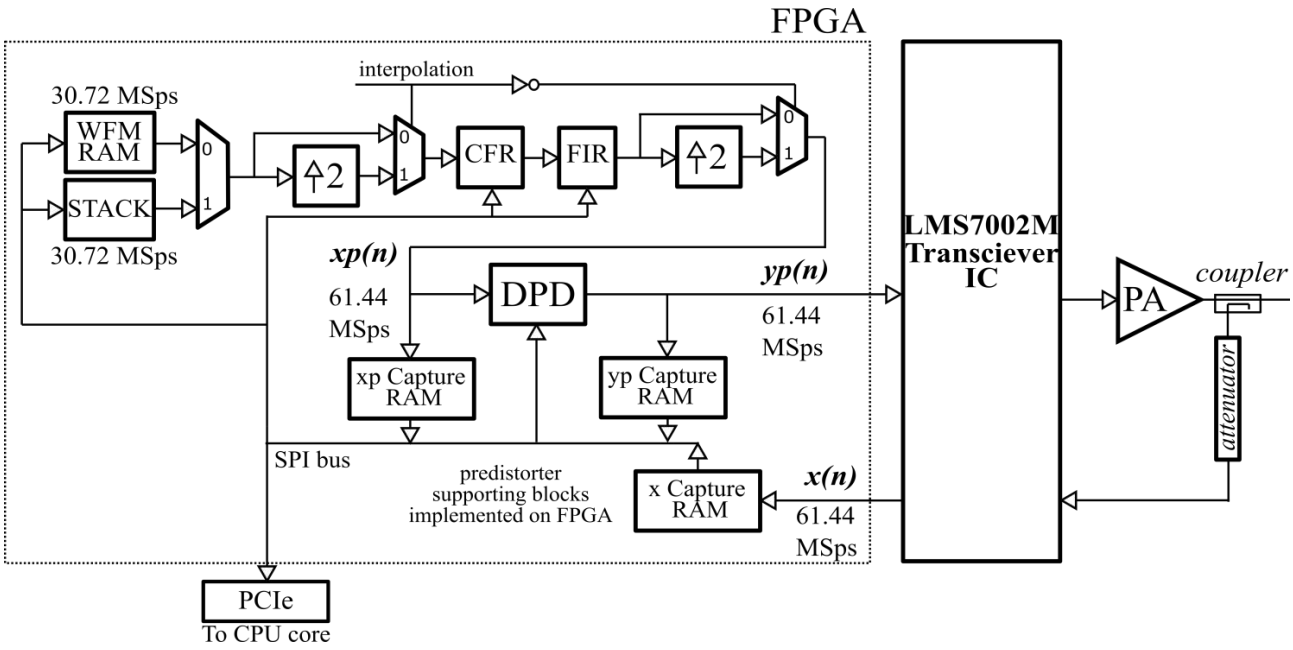


Figure 5: ADPD implementation based on LimeSDR-QPCIe board

For the development or demo, test waveform is uploaded first and played from the WFM RAM Block implemented using Altera Cyclone V FPGA resources. More importantly, the same FPGA also implements both DPD and CFR modules.

Initially, predistorter is bypassed i.e. $yp_I=xp_I$, $yp_Q=xp_Q$. Predistorter has provision for SPI in order to update the coefficients during the training. Signals **xp**, **yp** and **x** are captured using Data Capture RAM Blocks implemented also using FPGA resources.

Captured data is made available to CPU (Intel Motherboard) Core via PCIe interface. FPGA implements PCIe and other glue logic required to interconnect LimeSDR-QPCIe on board components including two LMS7002M ICs to the CPU Core.

CPU implements postdistorter block, delay line and the rest of training algorithm. After each adaptation step, CPU updates predistorter coefficients via SPI/PCIe interface.

Besides, CFR filter order, filter coefficients and CFR threshold are configured through the same SPI/PCIe interface.

PC/GUI implements graphical display for demo and debugging purposes. GUI is capable to show important ADPD signals in FFT (frequency), time and constellation (I vs Q) domains.

In the real applications, WFM and **xp** Capture RAM blocks are not required. The algorithm needs only **yp** and **x** as shown in Chapter 2. CPU Core performs both ADPD adaptation, as explained above, and base band (BB) digital modem functions which are application specific, LTE for example.

As shown in Figure 5, frequency conversion from BB to RF is performed by LMS7002M transmitter chains. Frequency down conversion from RF to BB is implemented by only one LMS7002M receive chain dedicated to ADPD, i.e. one receiver of the available RF RX chains is allocated as ADPD monitoring path. In case of MIMO applications, the same ADPD monitoring path is used as time sharing recourse to linearize multiple PAs which saves power consumption as well as on board RF resources.

Regarding data converters, LMS7002M on chip 12-bit DACs and ADCs are used. The data rate at *LimeLight* interface is 61.44MSps.

In order to increase the capacity of the radio link, 2x2 MIMO transceiver is implemented. One transceiver IC is used to implement two MIMO transmitters. In each of the transmit channels, separate DPD, CFR and low-pass FIR filter blocks are implemented. Another transceiver IC is used to implement two regular MIMO receivers.

In case of MIMO, single ADPD monitoring path is used as time shared resource to linearize multiple PAs which saves power consumption as well as on board RF resources.

CFR block has provision of changing the number of FIR filter taps L in the range 1 <= L <= 40.

Using the same interface, clipping threshold can be set to 0 <= Th <= 1. It is floating point number. The value of 0 is equivalent to “CFR power down” while 1 corresponds to “CFR bypass”.

The interpolation/decimation option can be enabled or disabled. The choice is related to the modulation bandwidth as shown later.

CFR output is filtered by on-FPGA digital low-pass post-CFR FIR filters (Figure 5). These filters are very frequency selective and efficiently remove out-of-band unwanted products generated by BB digital modem as well as CFR PW method itself.

We have implemented two CFR blocks for both transmitted channels. The 40-tap PWCFR module operates at 122.88 MHz clock frequency.

If interpolation control signal is zero (see Figure 5) the data rate of CFR input signal samples, which come (from WFM RAM) to CFR input is 30.72 MSps. Therefore, the PWCRF operates at the clock frequency level which is 4 times greater then processing data rate. In this case, the data interpolation (using up-conversion of factor of 2) is used after CFR and post-CFR FIR blocks (see Figure 5). The number of taps in these filters is 40. The order L is in the range 1 <= L <= 40. The predistorter, external ADCs and DACs operate at rate of 61.44 MHz.

If interpolation control signal is equal to equal to one (Figure 5), interpolation (using up-conversion of factor of 2) is used in front of CFR and post CFR FIR blocks. (see Figure 5). In this case, the data rate of signals processed by CFR and post-CFR FIR blocks is equal to 61.44 MSps. Therefore, the PWCFR operates at the clock frequency level which is only 2 times greater then processing data rate. The data interpolation, located after the CFR and post-CFR FIR blocks, is now bypassed. The number of taps in the CFR and post-CFR low-pass filters is limited to 20. Therefore, the L is in the range 1 <= L <= 20.

5

LimeSDR-QPCIe Board Programming

This Section describes how to program LimeSDR-QPCIe board.

5.1 Uploading FX3 Firmware to SPI FLASH Memory

The LimeSDR-QPCIe FX3 firmware source code, required USB drivers and software application *CyControl.exe* are available at:

https://github.com/myriadrf/LimeSDR-QPCIe_FX3_FW

The compiled FX3 firmware (the *LimeSDR-QPCIe_fx3_fw.img*) is at:

https://github.com/myriadrf/LimeSDR-QPCIe_FX3_FW/tree/master/src/Debug

In order to upload the compiled FX3 firmware into the board FLASH memory, please, follow the procedure described below.

- The procedure requires a computer and external 12V power supply for the LimeSDR-QPCIe. The LimeSDR-QPCIe board is positioned outside the PC.
- The Cypress drivers must be installed first on computer.
- The connector J28 (on LimeSDR-QPCIe board) is open and external power supply is provided to the board. The USB3 microcontroller boots-up into bootloader mode.
- Short the jumper J28 and connect LimeSDR-QPCIe board to the PC using USB 3.0 port.
- Start “CyControl.exe” application and select Cypress USB BootLoader.
- After entering into boot loader mode, there are two ways of uploading the firmware to USB3 microcontroller: using SPI FLASH memory or internal RAM memory. Choose SPI FLASH memory option by pressing the menu command Program -> FX3 -> SPI FLASH.
- In the status bar you will see Waiting for Cypress Boot Programmer device to enumerate.... and after some time window will appear.
- Select firmware image file (the *LimeSDR-QPCIe_fx3_fw.img*) and press Open. Status bar of the USB Control Center application will indicate *Programming of SPI FLASH in Progress*….

- This message will change to the *Programming succeeded* after FLASH programming is done. The USB3 microcontroller will boot from FLASH memory after every power-on.
- Disconnect the board from computer.

5.2 PCIe IP core generation

Before compiling the FPGA gateware bitstream, the PCIe Xillybus IP core has to be first generated and downloaded.

This chapter describes all steps and parameters required to generate Xillybus PCIe core.

- Xillybus requires filling up free registration form in order to download generated core. Go to link **http://xillybus.com/ipfactory/signup**, fill required fields and confirm registration via received email.
- After successful registration, go to IP core Factory page link and click *Add New Core*.
- Select option PCIe core and press *Next*.
- Choose the IP core's Name, for *Target device family* select *Intel Cyclone V*, select *Demo bundle settings*; for operating system select *Linux and Windows*. Press *Create!* button.
- After new core creation is done, change the settings as specified in the Table 1.
- After specifying all IP core parameters from Table 1 click *Generate core*.
- Check core status and download it when available.

Table 1: Xillybus PCIe IP core settings

Name	Direction	Data width	Expected BW	Auto set	Details
xillybus_stream0_read_32					
	Upstream (FPGA to host)	32 bits	395 MB/s	No	Asynchronous, 512 x 16 kB = 8 MB Data acquisition / playback
xillybus_stream0_write_32					
	Downstream (host to FPGA)	32 bits	395 MB/s	No	Asynchronous, 512 x 16 kB = 8 MB DMA acceleration: 8 segments x 512 bytes Data acquisition / playback
xillybus_control0_read_32					
	Upstream (FPGA to host)	32 bits	1 MB/s	Yes	General purpose
xillybus_control0_write_32					
	Downstream (host to FPGA)	32 bits	1 MB/s	Yes	General purpose
xillybus_mem_8					
	Upstream (FPGA to host)	8 bits	102.400 kB/s	Yes	Address/data interface (5 address bits)
	Downstream (host to FPGA)	8 bits	102.400 kB/s	Yes	Address/data interface (5 address bits)
xillybus_stream1_read_32					

	Upstream (FPGA to host)	32 bits	395 MB/s	No	Asynchronous, 512 x 16 kB = 8 MB Data acquisition / playback
xillybus_stream1_write_32					
	Downstream (host to FPGA)	32 bits	395 MB/s	No	Asynchronous, 512 x 16 kB = 8 MB DMA acceleration: 8 segments x 512 bytes Data acquisition / playback
xillybus_stream2_read_32					
	Upstream (FPGA to host)	32 bits	395 MB/s	No	Asynchronous, 512 x 16 kB = 8 MB Data acquisition / playback
xillybus_stream2_write_32					
	Downstream (host to FPGA)	32 bits	395 MB/s	No	Asynchronous, 512 x 16 kB = 8 MB DMA acceleration: 8 segments x 512 bytes Data acquisition / playback

5.3 FPGA gateware bitstream generation

The LimeSDR-QPCIe DPD gateware project is available at:

https://github.com/myriadrf/LimeADPD/

tag **v21.07.0**

Download the project from specified site. In order to generate *LimeSDR-QPCIe-lms7_trx_HW_1.2.jic* file follow the procedure described as:

- The Xillybus IP compressed file is first downloaded from Xillybus site. The compressed file contains files *xillybus.v* and *xillybus_core.qxp*.
- Place file *xillybus.v* to Quartus project directory *limesdr-qpcie_xillybus_core*/
- Place file *xillybus_core.qxp* to Quartus project directory *limesdr-qpcie_xillybus_core/*
- Open Quartus *LimeSDR-QPCIE_lms7_trx project*.
- To recompile project, press *Processing → Start Compilation*.
- When compilation is finished, the *LimeSDR-QPCIe-lms7_trx_HW_1.2.jic* file is located in gateware project directory */output_files*

5.4 Uploading FPGA gateware bitstream to FLASH memory

- This procedure requires two computers (LimeSDR-QPCIe board inserted into PCIe slot on computer #1 and Quartus software running on computer #2).
- Besides, Altera USB Blaster is required.
- Insert LimeSDR-QPCIe board into computer #1. Make sure that computer is turned off while inserting board.

- Board is programmed using JTAG header J26. Connect one end of download cable (e.g Altera USB Blaster) to LimeSDR-QPCIe board J26 connector and other end to USB port on the computer #2 running Quartus software.
- Turn on computer #1 and interrupt the boot sequence to bring up the BIOS System Setup interface.
- Run Quartus software in computer #2 and select *Tools → Programmer*
- Click *Hardware Setup*.. button and select your download cable, click *Close*
- Click *Add File*.. and select *.jic file
- Pre compiled bitstream can be found in *DPD/gw/LimeSDR-QPCIe-lms7_trx_HW_1.2.jic*
- If you have generated your own bitstream then your file is located in gateware project directory /*output_files*.
- Select *Program/configure* and click *Start.* After successful programming turn off computer #1.
- FPGA boots from programmed FLASH memory automatically when computer #1 is turned on.

5.5 LimeSuiteGUI installation

The LimeSuiteGUI source code dedicated to DPD demonstration is available at:

https://gitlab.com/myriadrf/lime-suite

branch **DPD_LimeSDR-QPCIe_Stable**

Please download LimeSuiteGUI source code from specified Gitlab branch and in order to install software follow instructions described at:

https://wiki.myriadrf.org/Lime_Suite

6

CFR and DPD User Guide

6.1 The Hardware Configuration

For DPD demonstration two transceiver channels are implemented in LimeSDR QPCIe board (named with channels A and B). Also, two power amplifiers are requited belonging different transmitting paths.

Follow the steps explained below:

- The LimeSDR QPCIe channel A output, the LimeSDR QPCIe LMS#1 TX1_1 port, is connected to channel A PA#1 input.
- For channel B, port LMS#1 TX2_1 port is used and it is connected to corresponding PA#2 input.
- The output of one of the PAs is via RF attenuator connected to spectrum analyzer RF input. The other PA output can be terminated with 50 Ohms.
- PA coupling outputs are over 10dB-20dB RF attenuators fed to two LimeSDR QPCIe receive inputs.
- The on-board analogue multiplexer is used for selection of PA coupling output signals. The multiplexer input, the U.FL RF1, is dedicated for channel A receive input, while the U.FL RF3 is used as channel B input.
- The analogue multiplexer output U.FL port RFC is connected to the U.FL LMS#1 RX1_W, which is used as DPD monitoring input.

6.2 LimeSuiteGUI settings

The CFR and DPD control is implemented in LimeSuiteGUI application. Follow the steps 1 to 8:

1. Copy the content of folder *DPD/sw* (the subfolders and *QADPDconfig.ini*) into folder that belongs to LimeSuiteGUI installation:

`<LimeSuite install folder>/LimeSuite/build/bin`

2. Open the terminal in this folder.

3. Start the LimeSuiteGUI application **with sudo**:

```
sudo ./LimeSuiteGUI
```

4. Make the connection with the LimeSDR QPCIe board *Options -> Connection settings*. Select the LimeSDR QPCIe board.

5. Read the LMS7002M .ini configuration file *LMS1settings/LMS1settings_20_751.ini*.

6. In LimeSuiteGUI open the *Calibrations* tab, press buttons *Calibrate Tx*, then *Calibrate Rx* for static I/Q calibration.

7. Open the window *Board related controls* through *Modules -> Board Controls*. When window is opened, read the FPGA configuration file (with extension .ini2) which contains the CFRs settings and post-CFR FIR filter configuration. To do this press *Read settings* button and choose the file dedicated to 10MHz LTE waveform – *FPGAsettings/FPGAsettings_10MHz.ini2*.

8. Now, select the test waveform by *Modules->FPGA controls*, then select the 10MHz LTE waveform *lms7suite_wfm/LTE_DL_TM31_10MHZ.wfm*. Check *MIMO* option and press button *Custom* to start the waveforms.

6.3 Board Related Controls

The Crest factor reduction (CFR) controls have been implemented in the *Board related controls* window, which is the part of LimeSuite GUI (Figure 6). The window provides:

- Selection of the transmit channels A or B
- Change of PWFIR filter order, in the range from 1 to 40.
- Setting the clipping threshold
- To change the coefficients of post-CFR FIR filter.
- To turn on/off the LimeNET internal PAs and DCDCs (only if LimeNET internal PAs are used)

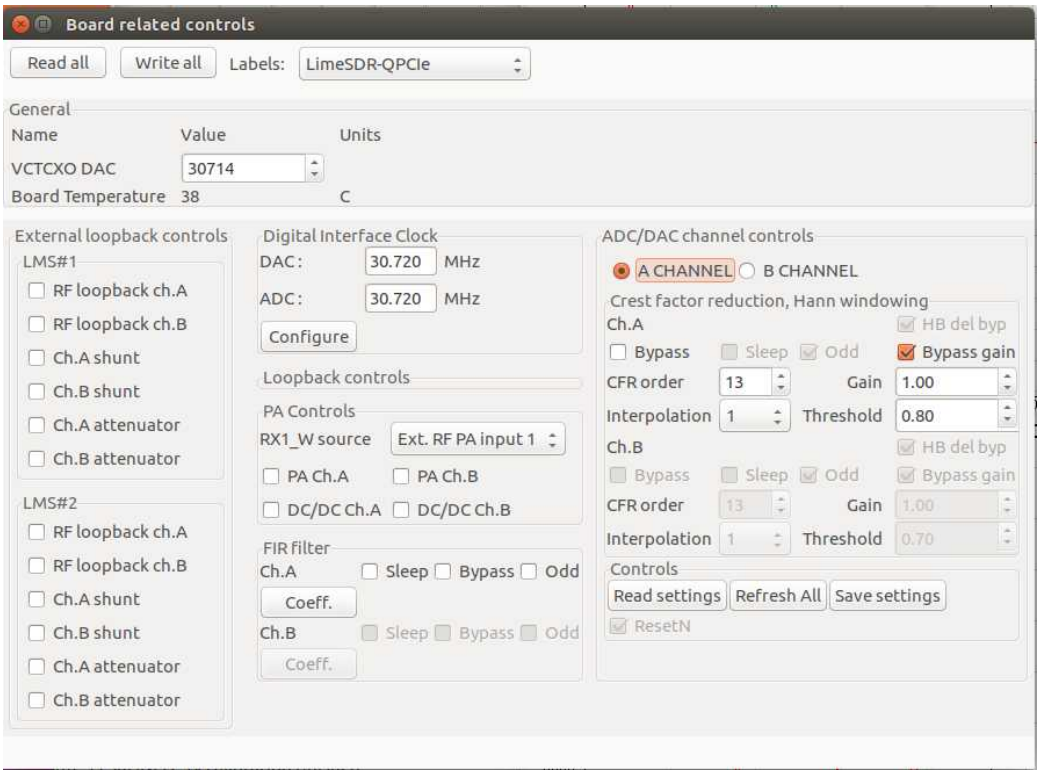


Figure 6: The *Board related controls* dialog

The radio buttons A_CHANNEL and B_CHANNEL select one of the transmit paths: A or B (Figure 6).

Two different CFR blocks and accompanying post-CFR FIR filters in the FPGA gateware are dedicated to different transmit paths A and B. Therefore, before any modification of CFR parameters is made, the transmit path must be selected using the previously specified radio buttons.

CFR parameters for each of transmit paths include:

- *Bypass* – when is checked, the CFR is bypassed
- *Interpolation* has possible values 0 and 1 (Figure 6). The value 1 selects the interpolation in front of CFR block. (see Figure 5). In this case the data rate of signals entering the CFR is 61.44 MSps. Otherwise, when 0 value is chosen, the interpolation is used after CFR and post-CFR FIR blocks. In this case the data rate of signals is 30.72MSps.
- *CFR order* is the integer value representing the CFR PWFIR order. When Interpolation=0 the CFR order maximum is 40, otherwise, when control signal interpolation = 1, maximum PWFIR order is 20.
- *Threshold* is the floating point number in the range from 0.0 to 1.0, determining the clipping threshold. The value is normalized to input signal amplitude maximum. The parameter Threshold determines the amount of PAPR reduction. For example, the value of 0.7 reduces the input signal PAPR by 3dB. When value of 1.0 is chosen, the clipping operation is bypassed.
- *Gain* is the digital gain following CFR block. The default value is set to 1.0.

When interpolation or CFR order values are changed in the window, the new Hann windowing coefficients are automatically calculated and are programmed to the dedicated CFR registers located in FPGA gateware.

The recommended CFR configuration for different LTE bandwidths is given in the Table 2.

Table 2: The recommended CFR configuration for different LTE bandwidths

LTE bandwidth [MHz]	CFR order	Interpolation	Threshold
05	21	0	0.75*
10	17	0	0.75*
15	17	1	0.75*
20	13	1	0.75*

*Note: If the power of the input signal is additionally backed-off by LTE stack settings, the threshold given in the Table 2 should be re-calculated and modified.

As previously mentioned, the low-pass post-CFR FIR filter follows the CFR block (Figure 5). The options for filter coefficients reading end programming are provided. When *Coeff.* button in the *Board related controls* window is pressed, the post-CFR FIR filter coefficients are read from FPGA gateware registers and displayed in the new window. New post-CFR FIR coefficients can be loaded from .fir file and displayed in the window. For different LTE waveforms (5MHz, 10MHz, 15MHz and 20MHz) the corresponding .fir files are provided in folder in folder *<LimeSuiteGUI install folder>/LimeSuite/build/bin/FIRcoefficients.* After pressing OK button, the window is closed and new coefficients are programmed into the FPGA gateware registers. Since different post-CFR filters exist for different channels, it is required to select the transmitting channel before changing filter coefficients. For this purpose the radio buttons *A_CHANNEL/B_CHANNEL* are used.

The post-CFR filter length depends on Interpolation. When interpolation is 0, the data rate of post-CFR FIR signals is 30.72MSps (see Figure 5). In this case the post-CFR FIR order is 40. Otherwise, when value 1 is chosen, the interpolation is done before the CFR and post-CFR FIR blocks. In this case, the data rate of signals is 61.44 MSps and filter order maximum is equal to 20.

Before waveform bandwidth is changed it is required to change both post-CFR filter coefficients and CFR parameters, including the CFR filter length and interpolation.

To save or read gateware configuration the *Board related control* window (Figure 6) provides three buttons:

- *Read settings* which reads the .ini2 file, updates the configuration shown in the window and also, automatically programs the FPGA gateware (the CFR blocks and post-CFR FIR filters),
- *Refresh* button reads the configuration which is already programmed in the FPGA and updates the configuration in the window,
- *Save settings* is used to read the configuration from FPGA and save it into the .ini2 file

Beside the CFR and post-CFR FIR configuration, the *Board related controls* window controls the internal LimeNET Base station PAs and DC/DC convertors. Namely, the LimeNET Base station PAs and DCDCs can be turned on/off programmatically.

The check buttons *DC/DC ChA and ChB* (Figure 6) are used to switch on/off the LimeNET BS DC/DC convertors, which provide power supply to PAs (only if LimeNET internal PAs are used). Additionally, the LimeNET BS PAs can be turn on/off using *PA ChA and ChB* check buttons. Note that when the control is checked, the DCDC or PA is turned on.

6.4 DPD Viewer Window

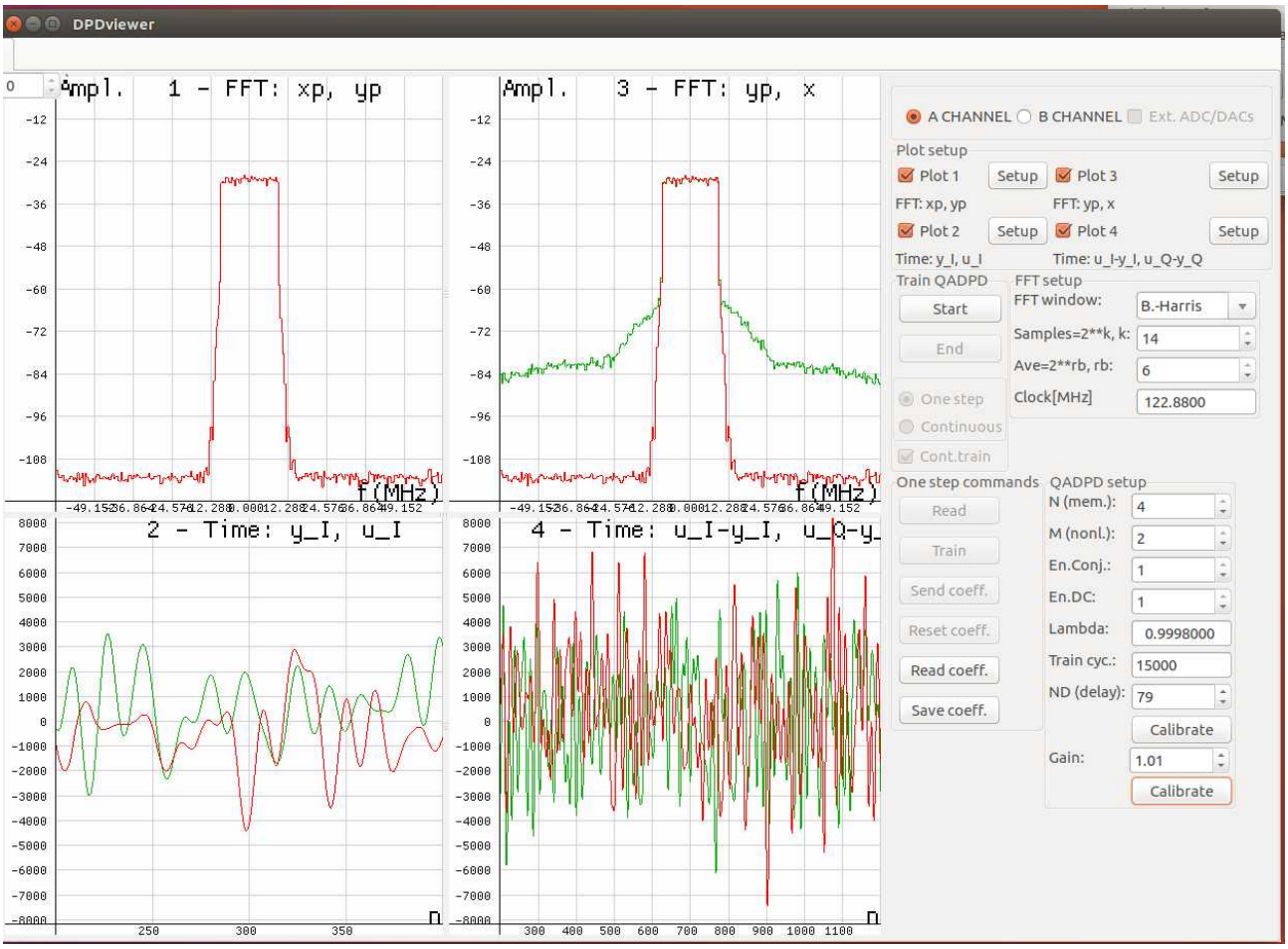


Figure 7: DPDViewer: ADPD signals before training

PC/GUI implements graphical display for demo and debugging purposes. GUI is capable to show important ADPD signals in FFT (frequency), time and constellation (I vs Q) domains. The DPD viewer window is displayed through *Modules -> DPD Viewer*.

Figures 7 and 8 show important ADPD signals before and after the algorithm convergence. Signals are captured by GUI executed by CPU Core.

ADPD parameters given in the QADPD setup part of the window are:

- *N(mem.)* - the DPD model memory order, maximum value N=4.
- *M (nonl.)* – the nonlinearity order, maximum value M=3,
- *Lambda* – the RLS forgetting factor. It is real value less than 1.0.
- *Train cycles* number of train cycles before new DPD coefficients are programmed
- *ND delay* - the DPD delay line length (in range from 74-80)
- *Gain* – floating point number representing the DPD digital gain. When Gain is obtained by gain calibration process, the PA output power is maintained at the save power level after DPD linearization process is performed compared to initial power. When Gain value is chosen to be less than the value derived after Gain calibration, the power at PA output is increased, as well the amount of distortion.

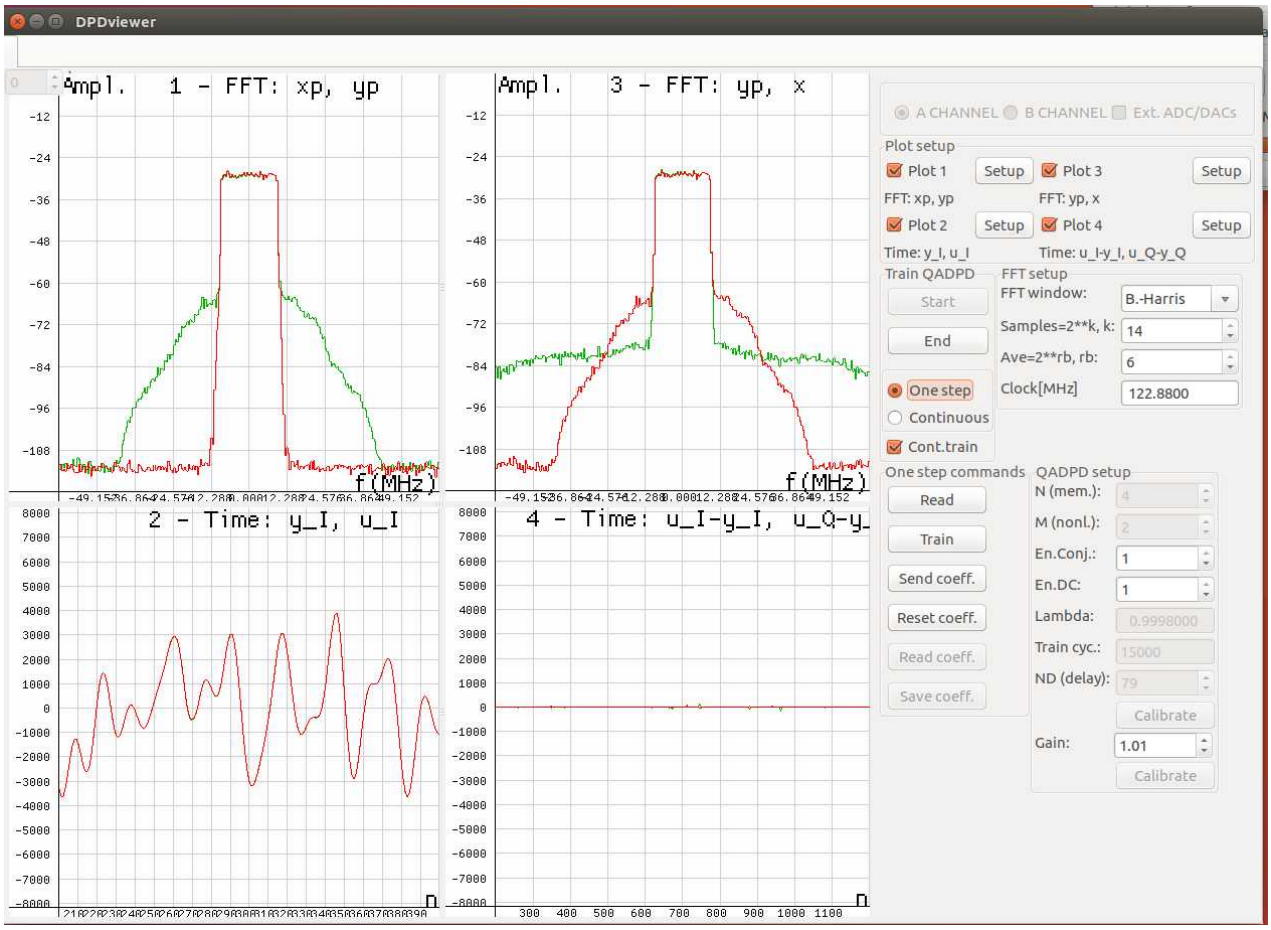


Figure 8: DPDViewer: ADPD signals after training

Before training (Figure 7), predistorter signals *yp* and *xp* are equal (plot 1). Signal *x* as a measure of PA output is distorted (plot 3). Waveforms *y* and *u* are very different (plot 2) which results in huge error (plot 4) which ADPD has to minimize.

After ADPD training (Figure 8), signal *yp* (plot 1) is predistorted in order to cancel PA distortion components. *x* as a measure of PA output is now linearized (plot 3). Excellent match between *y* and *u* waveforms in both time and amplitude scale (plot 2). ADPD error (plot 4) is minimized. Improvement in PA linearization can be seen by comparing *yp* and *x* spectra of plot 3.

The basic operations describing the DPD operations from LimeSuite GUI are as follows:

1. Start the waveforms (running the LTE stack, or loading the test waveform)
2. Select the transmitting channel (A or B)
3. Press *Calibrate ND delay* button.

Note: Expected values for delay ND are in the range [74-80].

Note: If in consecutive DPD calibration procedures, different, random values for ND are obtained, which are out of specified range, there is a RF reflection or interference. To solve this, check the RF cables. The cable dedicated for DPD monitoring path (from PA's coupling output to LimeSDR QPCIe board) should have strong shield. Else, 10dBm-20dBm RF attenuator should be placed at LimeSDR QPCIe board receive input, rather than at PA's coupling output.

4. Press *Calibrate gain* to determine DPD digital gain.

Note: If LTE stack is running, the DPD calibration procedure requires the data payload, generated by connecting mobile phone(s) to BTS and executing Magic Iperf application on both sides.

Note: The DPD digital gain should be in range [1.0-3.0], otherwise, LMS7002M channel A receiver gain settings must be modified.

5. In the part of the window *Train DPD*, press the *Start* button, check *Cont. train* option and then select *Continuous* option.
6. To stop the DPD training process, first press *One step*, then *End* button, above.
7. Repeat steps 2-6 for the other channel

Note: For DPD coefficient reset use *resetCoeff* button. The result of this operation is the same as DPD is bypassed.

When LTE stack is running there is a possibility to just monitor the signals without performing the DPD training. In this case, the sequence of operations is as follows:

1. Select the channel first (A or B)
2. In the part of the window *Train DPD*, press the *Start* button, uncheck *Cont. train* option and select *Continuous* option.
3. To stop monitoring operation, first press *One step*, then *End* button.
4. Repeat steps 1-3 for the other channel

7

Application DPDcontrol

The application is dedicated to LimeSDR QPCIe board. Before starting the command-line application, named *DPDcontro*l, the LMS7002 transceiver chip should be initialized and modulation waveforms started.

One option to do this is to start Amarisoft LTE stack. The other option is, using LimeSuiteGUI application, to load LMS7002M configuration files and run test waveforms.

In the first option, during LTE start-up procedure, two LMS7002M .ini files are automatically loaded into two transceiver ICs. Also, the .ini2 FPGA configuration file is loaded, containing on-board FPGA gateware configuration, including information regarding CFRs and post-CFR FIR filter coefficients.

In the second option, used for development or demo, test waveform is uploaded and played from the on-board WFM RAM Blocks. The LimeSuiteGUI application is used in this case.

1. Open the terminal in the folder which belongs to LimeSuiteGUI installation:

   <LimeSuiteGUI installation folder>/LimeSuite/build/bin

2. Start the LimeSuiteGUI application with sudo:

   `> sudo ./LimeSuiteGUI`

3. Make the connection with the board *Options->Connection settings*. Find and select LimeSDR QPCIe board.

4. Read the LMS7002M .ini configuration file *LMS1settings/LMS1settings_20_751.ini*.

5. Open the window *Board related controls* through *Modules -> Board Controls*. When window is opened, read the FPGA configuration file (with extension .ini2) which contains the CFRs settings and post-CFR FIR filter configuration. To do this press *Read settings* button and choose the file dedicated to 10MHz LTE waveform –

FPGAsettings/FPGAsettings_10MHz.ini2. When FPGA is initialized, close the *Board related controls window*.

6. In LimeSuiteGUI open the *Calibrations tab*, press *Calibrate Tx*, then *Calibrate Rx*.
7. Now, select the test waveform by *Modules->FPGA controls*, then select the 10MHz LTE waveform *lms7suite_wfm/LTE_DL_TM31_10MHZ.wfm*. Check *MIMO* option and press button *Custom* to start the waveform.

Note: if it is required to modify CFR or post-FIR CFR settings, LimeSuiteGUI must be used. Again, go to *Modules-> Board Controls*, open *Board related controls* window. After the CFR settings are modified, save new configuration into FPGA configuration .ini2 file or replace the existing FPGA configuration .ini2 file.

Once the test waveforms are played, the *DPDcontrol* application can be started.

It is not allowed to use the *DPDcontrol* application and *LimeSuiteGUI* at the same time. Therefore, before starting the *DPDcontrol*, close the LimeSuiteGUI.

It is still possible to linearize PAs using *DPDcontrol*, and then, after closing the *DPDcontrol*, open LimeSuiteGUI, its DPDViewer window, and check the spectrum of the PA output signals. The relevant signal is signal **x** which is a measure of PA output.

The very basic *DPDcontrol* operations are explained through steps 1-7.

1. Open the terminal in following folder, which belongs the *LimeSuiteGUI* installation:
   `<LimeSuite install folder>/LimeSuite/src/commandmode/`

2. Compile the *DPDcontrol* application:
   `>make`

3. Start the application with sudo:
   `>sudo ./DPDcontrol`

Note: If the application *DPDcontrol* is started without any argument, the DPD nonlinearity order *QADPD_M* is defined by the value which is last stored in *DPDcontrol* configuration file.
Please, find the description of **`storeConfigDPD`** command below.

Note: If the application is started with an argument, the argument represents the DPD nonlinearity order - *QADPD_M*, which is an integer value in the range from 1 to 3. This parameter should be stored into DPDcontrol configuration file. Use **`storeConfigDPD`** command after DPD is being calibrated.

4. To calibrate DPD parameters (calculate DPD digital gain and ND delay):
   `>calibrateDPD {1, 2, all}`

Note: the argument **`all`** refers to both transmitting channels; available arguments are 1, 2 or all, particularly for first channel A, second channel named B, or both channels.

Note: Expected values for delay ND are in the range [74 – 80].

If in consecutive DPD calibration procedures, different, random values for ND are obtained, which are out of specified range, there is a RF reflection or interference. To solve this, check the RF cables. The cable dedicated for DPD monitoring path (from PA's coupling output to board) must have strong shield. Else, place 10dBm-20dBm RF attenuator at LimeSDR QPCIe board receive port, dedicated to DPD monitoring input, rather than at PA's coupling output.

Note: The DPD digital gain should be in range [1.0-3.0], otherwise change the LMS7002M receiver gain settings. Open *LimeSuiteGUI*, in tab *RFE* modify *LNA*; in tab *RBB* modify *PGA gain* settings.

Note: when running the LTE stack, the DPD calibration procedure requires that the data payload is generated by connecting mobile phones to BTS and executing MagicIperf application on both phones.

5. When DPD is calibrated, the DPD training operation is started by:
   >startDPD {1, 2, all}

Note: Again, like in previous commands, the argument **all** refers to both transmitting channels; available arguments are 1, 2 or all, particularly for first channel A, second channel B or both channels.

Note: DPD training operation is performed periodically for both transmitting channels, the calculation period is equal to four seconds, just in a few iterations PAs get linearized.

Note: The information about DPD calculation errors obtained by DPD training process can be useful. The information is displayed or disabled by successive entering the character "**l**" in command line**.**

6. To stop DPD training operation use:
   >stopDPD {1, 2, all}

7. To stop the application:
   >quit

The application *DPDcontrol* has some additional useful commands which are explained below:

1. The entire command set provided by:
   >help

2. To turn on the DCDCs and PAs (only if LimeNET internal PAs are used):
   >startDCDC {1, 2, all}
   >startPA {1, 2, all}

3. To turn off the DCDCs and PAs (if LimeNET internal PAs are used):
   >stopPA {1, 2, all}
   >stopDCDC {1, 2, all}

4. To store the DPD parameters into DPDcontrol configuration file (DPD digital gain and ND delay, which are determined by **calibrateDPD;** and DPD nonlinearity order – QADPD_M**,** defined by *DPDcontrol* application argument), use:
   >storeConfigDPD {1, 2, all}

5. The DPD parameters (*DPD digital gain*, *ND delay* and *QADPD_M*) are loaded from configuration file using:
   >loadConfigDPD {1, 2, all}

Note: When the application *DPDcontrol* is started, the parameters *DPD digital gain* and *ND delay* are automatically loaded from *DPDcontrol* configuration file. Also, when application is started without arguments, the DPD nonlinearity order *QADPD_M* is read from configuration file. When it is started with argument, it represents value of *QADPD_M***.**

6. There is an option to store all calculated DPD coefficients (after training process is stopped with *stopDPD* command) into application's configuration file.
   >storeCoeffDPD {1, 2, all}

7. To read the DPD coefficients from configuration file:
   >loadCoeffDPD {1, 2, all}

8. To read current status of DPD parameters (*DPD digital gain*, *ND delay* and *QADPD_M*), or status of the PAs and DCDCs for both transmitting channels, use the following command:
   >readConfigDPD {1, 2, all}

9. To reset all DPD coefficients:
   >resetDPD {1, 2, all}

Note: The result of this command is the same as DPD is bypassed.

8

DPD Measured Results

Hardware setup:

- Rigol spectrum analyzer.
- Ubuntu PC equipped by LimeSDR-QPCIe board.

For channel A, LimeSDR QPCIe TX1_1 output is connected to power amplifier input; for channel B port TX2_1 port is used. The output of power amplifier is via RF attenuator connected to spectrum analyzer RF input. Power amplifier coupling outputs for both transmitter channels are over attenuator fed to LimeSDR QPCIe inputs. RX1_W is used as DPD monitoring input and it is connected to an on-board analog multiplexer output U.FL RFC port. The U.FL RF1 multiplexer input is channel A receive input, while U.FL port RF3 is channel B input.

Data payload is generated by connecting mobile phone(s) to BTS and executing Magic Iperf application on both sides. The case when CFR is bypassed is used as the reference point for the final comparison. In the further measurement points, the CFR block is enabled

Before implementation and measurements, ADPD algorithm has been thoroughly simulated. Simulation results are omitted from this document for clarity.

Most importantly, ADPD performance has been measured and the results for two cases are presented in this Chapter.

Test Case 1:

Moderate output power amplifier device Maxim Integrated MAX2612.Psat ~ 19dBm.

10MHz LTE

RF centre frequency 751MHz.

Test Case 2:

Moderate output power amplifier device Maxim Integrated MAX2612.Psat ~ 19dBm.

20MHz LTE

RF centre frequency 751MHz.

Test Case 3:

10W modulated output power amplifier.

10MHz LTE

RF centre frequency 751MHz.

8.1 Test Case 1: 10MHz LTE, Maxim Integrated MAX2612 PA

Power amplifier: Psat ~ 19dBm. RF centre frequency 751MHz. Test signal: 10 MHz LTE waveform .

ADPD Parameters: Nonlinearity order: M_1=2, M_2=0. Memory order: N_1=N_2=4.

CFR parameters: L=17, Th=0.75, Int/Dec=1, PAPR is reduced from 10.32 to 8.34 dB

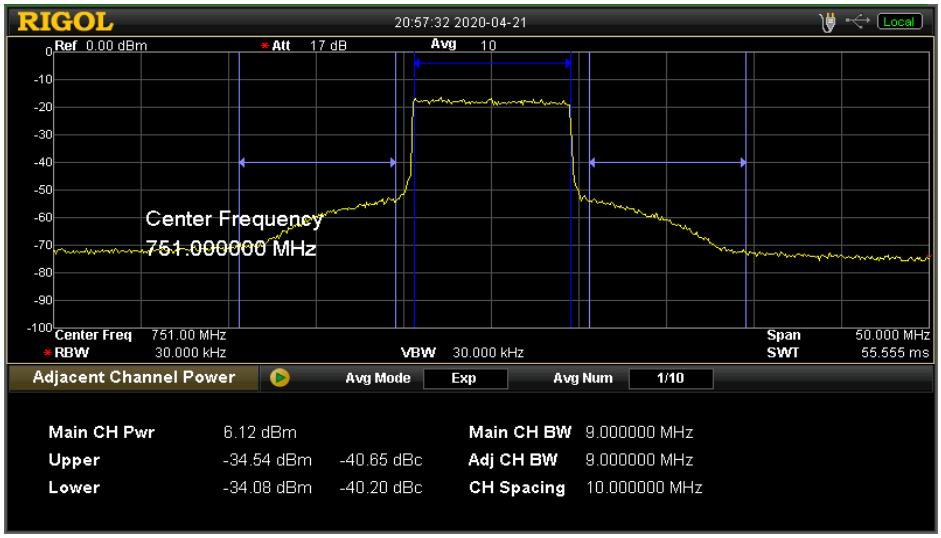


Figure 9: Signal spectrum before linerization

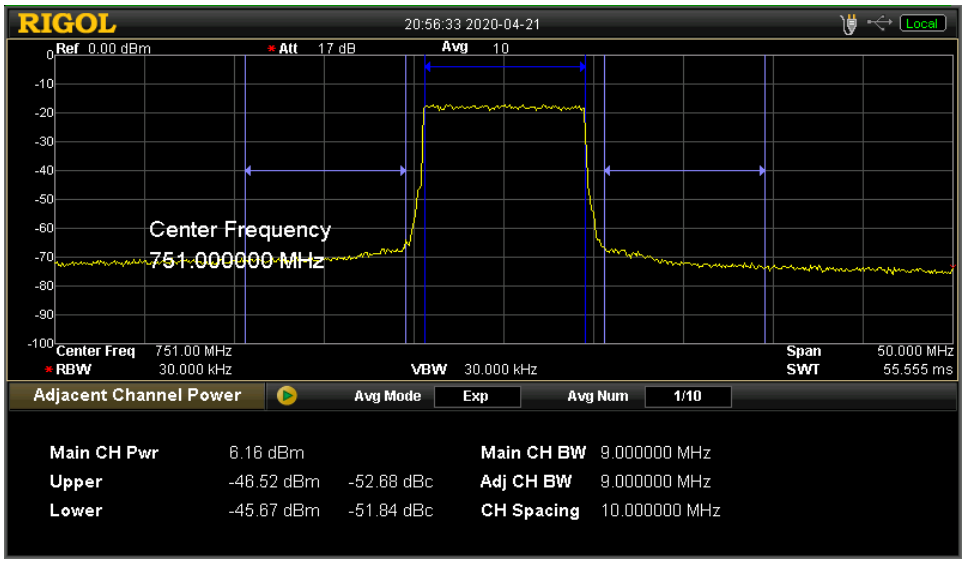


Figure 10: Signal spectrum after linerization

Before linearization: ACPR = -40.2 dBc; EVM = 3.2%

After linearization: ACPR = -51.8 dBc; EVM = 2.2%

Improvement: Δ ACPR = 11.6 dB; ΔEVM = 1.0%

PA output power is preserved at P_{out} = 6.1 dBm

8.2 Test Case 2: 20MHz LTE, Maxim Integrated MAX2612 PA

Power amplifier: Psat ~ 19dBm. RF centre frequency 751MHz. Test signal: 20 MHz LTE waveform

ADPD Parameters: Nonlinearity order: M_1=2, M_2=0. Memory order: N_1=N_2=4.

CFR parameters: L=13, Th=0.75, Int/Dec=2, PAPR is reduced from 10.6 to 8.3 dB

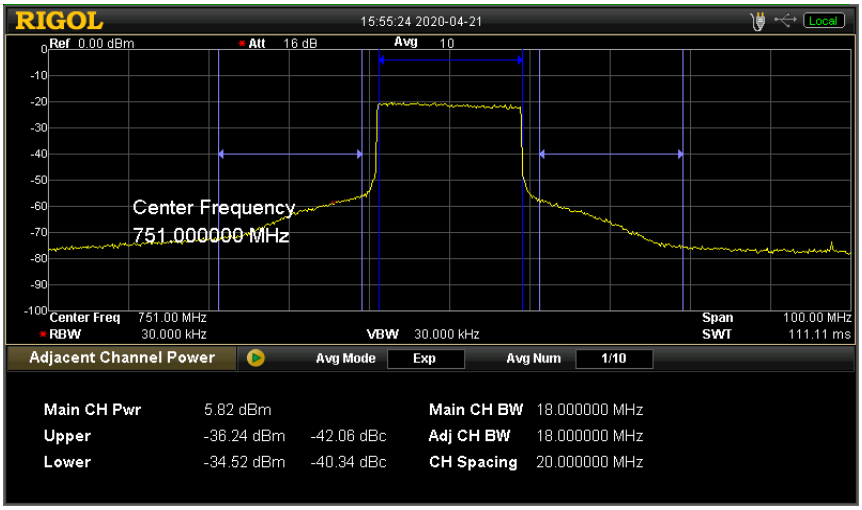


Figure 11: Signal spectrum before linerization

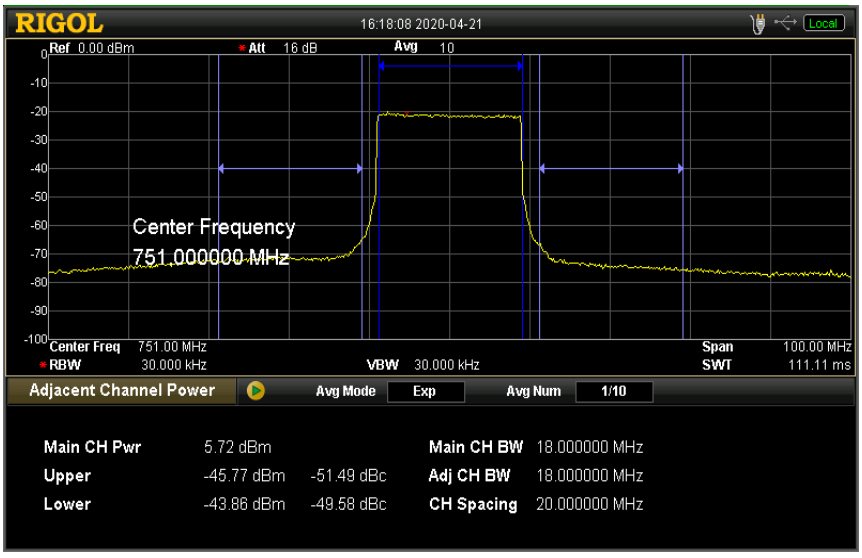


Figure 12: Signal spectrum after linerization

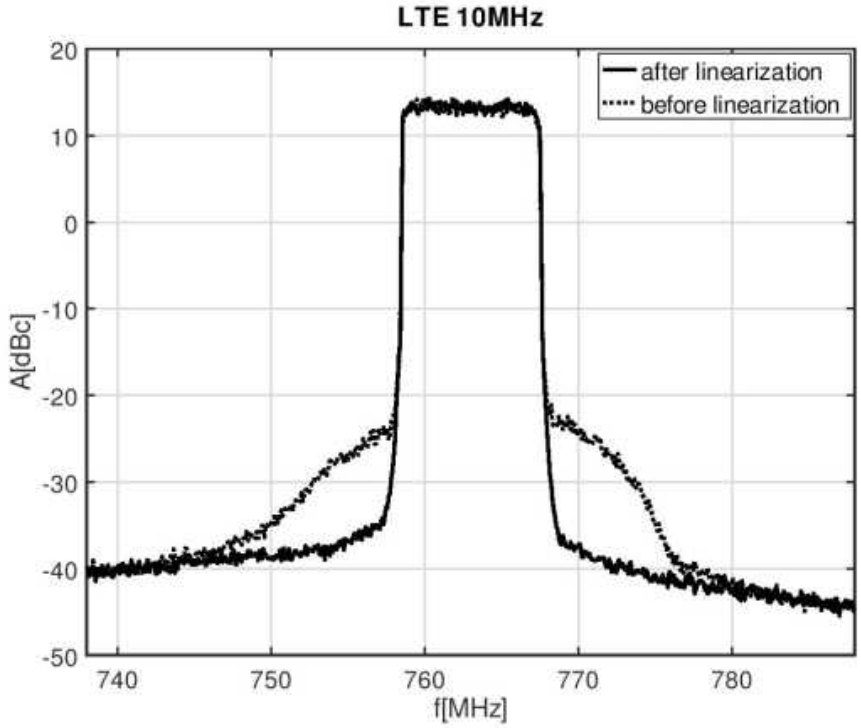
Before linearization: ACPR = -40.3 dBc; EVM = 3.6%

After linearization: ACPR = -49.6 dBc; EVM = 2.2%

Improvement: Δ ACPR = 9.3 dB; Δ EVM = 1.4%.

PA output power is preserved at P_{out}=5.8 dBm.

8.3 Test Case 3: 10MHz LTE, 10W modulated output power amplifier



The bandwidth of PA is 700-850 MHz. The average output power at 1 dB compression point is 40 dBm at the frequency of 750 MHz. In the measurement setup, the 30 dB attenuator is put between the output of PA and the RF input of spectrum analyzer.

Before linearization: ACPR=-37.5 dBc, EVM= 3.32 %.

The measured PA output power was Pout=39.7 dBm and the PAPR=10.3 dBm.

After the PAPR is decreased by 2 dBm, by choosing L=19 and Th=0.76, and the PA is being linearized by DPD: EVM = 2.42 %. ACPR = -50.51 dBc.

Therefore, ACPR and EVM are improved by 13.01 dBc and 0.9 % respectively, compared to the results of uncompensated PA.

9

CFR Measured Results

Hardware setup:

- Agilent PSA spectrum analyzer.
- Ubuntu PC equipped by LimeSDR-QPCIe board. Runs either Lime Suite GUI or real life BTS based on Amarisoft LTE stack, depending on test case scenario.
- Win 10 PC running Keysight Vector Signal Analyzer (VSA) software for modulation analyses.

In the test cases 1-4 the CFR algorithm is checked against 5MHz, 10MHz, 15MHz and 20MHz 64 QAM LTE modulated signals, with 20MHz being the most challenging one.

In test cases presented below, the signals are generated in different ways:

- The waveforms are generated using the Test Model 3.1 (E-TM 3.1) test specification, which applies to most LTE modulation schemes. The test waveform is upload into WFM RAM block of LimeSDR-QPCIe board and played. One of the TX RF outputs is connected to PSA for the modulation analysis. In the reference point, the CFR block is bypassed. In the further measurement points, CFR block is enabled. Different CFR options are considered: CFR order L, threshold Th and interpolation.
- Real BTS is constructed using Linux Ubuntu PC running Amarisoft LTE stack in FDD mode. This scenario checks the performance of Amarisoft LTE stack working together with LimeSDR-QPCIe hardware. Antennae are connected to LimeSDR-QPCIe RF ports. One of the TX outputs is connected to PSA via RF coupler. Data payload is generated by connecting mobile phone(s) to BTS and executing Magic Iperf application on both sides. The case when CFR is bypassed is used as the reference point for the final comparison. In the further measurement points, the CFR block is enabled.
-

9.1 Test Case 1: LTE 5MHz

Table 3: CFR Test case 1 results for E-TM 3.1 LTE 5MHz

E-TM 3.1 LTE 5MHz Wfm -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
no CFR	/	/	/	10.38	0.91	-56.00	0.00	0.00
CFR	**21**	**0**	**0.75**	**8.12**	**2.18**	**-55.00**	2.26	1.27
	19	0	0.75	8.12	2.12	-55.00	2.26	1.21
	17	0	0.75	8.20	2.05	-55.00	2.18	1.14
	15	0	0.75	8.42	1.96	-55.00	1.96	1.05

Table 4: CFR Test case 1 results for Amarisoft LTE 5MHz stack

Stack LTE 5MHz LTE Stack -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
no CFR	/	/	/	10.62	0.61	-56.00	0.00	0.00
CFR	**21**	**0**	**0.66**	**8.48**	**2.06**	**-55.00**	2.14	1.45

9.2 Test Case 2: LTE 10 MHz

Table 5: CFR Test case 2 results for E-TM 3.1 LTE 10MHz

E-TM 3.1 LTE 10MHz Wfm -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
no CFR	/	/	/	10.32	0.98	-54.00	0.00	0.00
CFR	19	0	0.75	8.28	2.28	-54.00	2.04	1.30
	17	**0**	**0.75**	**8.34**	**2.23**	**-54.00**	1.98	1.25
	15	0	0.75	8.38	2.19	-54.00	1.94	1.21
	13	0	0.75	8.52	2.12	-54.00	1.80	1.14

Table 6: CFR Test case 2 results for Amarisoft LTE 10MHz stack

LTE Stack 10MHz LTE Stack -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
no CFR	/	/	/	10.74	0.55	-54.00	0.00	0.00
CFR	**17**	**0**	**0.66**	**8.42**	**2.30**	**-54.00**	2.32	1.75

9.3 Test Case 3: LTE 15 MHz

Table 7: CFR Test case 3 results for E-TM 3.1 LTE 15MHz

E-TM 3.1 LTE 15MHz Wfm -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
no CFR	/	/	/	10.54	1.03	-52.00	0.00	0.00
CFR	19	1	0.75	8.34	2.22	-52.00	2.20	1.19
	17	**1**	**0.75**	**8.34**	**2.17**	**-52.00**	2.20	1.14
	15	1	0.75	8.48	2.11	-52.00	2.06	1.08
	13	1	0.75	8.48	2.06	-52.00	2.06	1.03

Table 8: CFR Test case 3 results for Amarisoft LTE 15MHz stack

Stack LTE 15MHz LTE Stack -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
No CFR	/	/	/	10.94	0.62	-52.00	0.00	0.00
CFR	**17**	**1**	**0.66**	**8.44**	**2.19**	**-52.00**	2.50	1.57

9.4 Test Case 4: LTE 20 MHz

Table 9: CFR Test case 4 results for E-TM 3.1 LTE 20MHz

E-TM 3.1 LTE 20MHz Wfm -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
no CFR	/	/	/	10.54	1.15	-51.00	0.00	0.00
CFR	17	1	0.75	8.24	2.33	-52.00	2.30	1.18
	15	1	0.75	8.24	2.29	-52.00	2.30	1.14
	13	**1**	**0.75**	**8.24**	**2.22**	**-52.00**	2.30	1.07
	11	1	0.75	8.38	2.15	-52.00	2.16	1.00

Table 10: CFR Test case 4 results for Amarisoft LTE 20MHz stack

LTE Stack 20MHz LTE Stack -> QPCIe -> PSA	**CFR L**	**CFR Int.**	**CFR Th**	**PAPR (dB)**	**EVM (%)**	**ACPR (dBc)**	**Δ PAPR (dB)**	**Δ EVM (%)**
no CFR	/	/	/	11.20	0.60	-52.00	0.00	0.00
CFR	**13**	**1**	**0.66**	**8.42**	**2.30**	**-52.00**	2.78	1.70

10

Conclusion

Lime ADPD algorithm has been implemented and verified by measured results.

ADPD is capable of cancelling any distortion above system noise floor,
DACs -> TX -> PA -> Coupler -> RX -> ADCs.

Improvements in ACPR and EVM have been achieved in all cases as shown in Table 11.

Table 11: DPD results summary

Configuration	**Modulation**	**Psat [dBm]**	**RF centre frequency [GHz]**	**ACPR [dBc]**		**EVM [%]**	
				No ADPD	**With ADPD**	**No ADPD**	**With ADPD**
Case 1	10 MHz LTE	19	0.751	-40.2	-51.8	3.2	2.2
Case 2	20 MHz LTE	19	0.751	-40.3	-49.6	3.6	2.2
Case 3	10 MHz LTE	39	0.75	-37.5*	-50.5*	3.3	2.4

Compared to the original Peak Windowing algorithm, time-multiplexing is implemented reducing the number of utilized multipliers. The CFR block operates at 122.88 MHz while data sample rate is 30.72 MS/s.

Booth multipliers, which were originally realized by FPGA flip-flops and combinatorial logic, are replaced by FPGA embedded multipliers, reducing the occupied area and increasing the processing speed.

The architecture of CFR FIR filter is further optimized having in mind that PWFIR coefficients are symmetrical. This reduces the number of multiplication operations, and consequentially, the number of used FPGA DSP blocks. FPGA resources are saved in this way leaving the room for some other DSP blocks to be added, DPD for example

The novelty not seen in the published literature so far is Peak search block which is introduced in CFR preprocessing stage to find local minimum values of the signal *c(n)*. Compared to the original

PW, the difference between local minimum values of the gain correction *b(n)* and the clipping signal *c(n)* is minimized. With this circuit, the peaks of the output signal envelope are more accurately constrained to the threshold *Th*, resulting in lower EVM degradation.

Another important novelty here is utilization of the interpolation and decimation blocks. These are placed in front of and after the CFR block respectively. Interpolation and decimation helps in getting better EVM results for the wider modulation formats (15MHz and 20MHz LTE, for example). In other words, this approach helps the cases when the modulation edge approaches the Nyquist frequency. With this option enabled the clipping operation becomes more precise since peaks are seen better. Adding interpolation/decimation required some modifications in FIR filter architecture and also in the method of coefficients programming.

PAPR is increasing as the modulation bandwidth is getting wider. Real LTE stack signal has higher PAPR than the test model one. Both facts are well known and expected. Due to higher PAPR, digital gain of LTE stack is backed off by 3 dB not to have digital overload. Consequently, CFR threshold *Th* is changed from 0.75 to 0.66.

Interpolation/decimation makes CFR algorithm almost insensitive to the modulation bandwidth.

If we take 20 MHz real life LTE stack test as the target and the most challenging case, we can say that CFR block reduced PAPR from 11.2 dB down to 8.42 dB while degrading EVM from 0.6% to 2.3%. In other words, PAPR is reduced by 2.78 dB while EVM is degraded by only 1.7%.

ACPR is not affected at all neither by BB modem nor CFR algorithm thanks to digital filtering implemented by FIR blocks.

11

Revision History

DPD revision history		
Date	Version	Description of Revisions
Jun 25, 2017	1.0	Lime ADPD implemented on LimeSDR-USB board. GUI control has been developed. The algorithm functionality confirmed by measurements.
Sep 07, 2017	2.0	Input data is multiplexed to reduce the number of utilized DSP blocks. Data rate increased to 61.44 MS/s by adding interpolation block.
Dec 24, 2017	3.0	Lime ADPD is implemented on LimeSDR-QPCIe board. External on board ADC/DACs are used.
May 25, 2018	4.0	ADPD on LimeSDR-QPCIe switched to use internal LMS7002M ADC/DACs and LimeLight digital interface..
Jan 12, 2019	5.0	CFR block added.
Apr 01, 2019	6.0	Command line DPD control with dedicated DPD library functions has been developed.
May 12, 2019	7.0	Command line CFR control with dedicated library functions has been implemented.
Oct 07, 2019	8.0	Automatic DPD calibration. New methods for calculating DPD digital gain and ND delay line length.
Mar 30, 2020	9.0	Missed measurement results collected and added to the document.
Apr 22, 2020	10.0	Some test cases remeasured.
Crest Factor Reduction revision history		
Date	Version	Description of Revisions
Mar 20, 2018	1.0	Initial release. Feasibility study.
Dec 28, 2018	2.0	System C simulation. CFR algorithm behaviour checked.
Feb 10, 2019	3.0	VHDL coding, simulation. Results confirmed to agree with System C simulation.
Mar 17, 2019	4.0	Compiling RTL VHDL code for Altera Cyclone V. First measurements

		using LimeSDR-QPCIe board.
Jul 23, 2019	5.0	Booth multipliers (based on flip-flops and combinatorial logic) replaced by embedded FPGA DSP blocks to save space and bust the processing power.
Sep 02, 2019	6.0	Peak search pre-processing block added to the structure. Improved behaviour confirmed by both simulations and measurements.
Apr 10, 2020	7.0	Interpolation/decimation added. Obtained measured results accepted and documented as the best and final so far. Figures re-imported to look better.