引き続き資料

DPD シミュレーションと実験

Yuelin Ma
corterma@hotmail.com
# Contents

1 Overview........................................................................................................................................0

1.1 About the Files .........................................................................................................................0

1.2 Getting Start .............................................................................................................................1

1.3 Others .......................................................................................................................................2

1.4 Note ..........................................................................................................................................2

2 Digital Processing & Virtualization..............................................................................................3

2.1 Signal Generation .......................................................................................................................3

2.2 Effect of Nonlinearity ...............................................................................................................7

3 LTE Generation and Measurement ..............................................................................................10

4 DPD Examples ................................................................................................................................0

4.1 QRD-Based DPD ......................................................................................................................0

4.2 Orthogonal Model .....................................................................................................................2

5 Crest Factor Reduction ...............................................................................................................5

6 Model Reduction Using Subspace Selection ..............................................................................0

6.1 Representing the Model with Orthogonal Basis ......................................................................0

6.2 Representing the Model with Orthogonal Basis ......................................................................1

7 Concurrent Dual-Band Predistorter ............................................................................................5

8 Instrumental Experiments using MATLAB Instrument Toolbox ............................................8

8.1 How to Control the Instruments ..............................................................................................8

8.2 Example: Control the M8190A .................................................................................................9

8.2.1 Connect with the M8190A using Instrument Control Toolbox ........................................9

8.2.2 Set Basic Parameters for the M8190A ............................................................................10

8.2.3 Upload Signals to the M8190A .......................................................................................10

8.2.4 Processing the Signal ........................................................................................................11

8.2.5 Programming Examples ....................................................................................................11
Example 1......................................................................................................................... 12
Example 2............................................................................................................................. 13
8.3 DPD Implementation with VSG and VSA................................................................. 14
9 DPD Design with Xilinx System Generator ................................................................. 18
  9.1 Adaptive LUT DPD.................................................................................................... 19
  9.2 Delay Estimator ........................................................................................................ 21
  9.3 Address Process ........................................................................................................ 22
10 FPGA Implementation ................................................................................................... 25
  10.1 Before the Experiment............................................................................................ 25
  10.2 Generate Files.......................................................................................................... 27
1 Overview

1.1 About the Files

This document contains the introductions for example codes/files for digital predistortion (DPD) simulation and instrumental measurements in MATLAB/Simulink, as well as the source codes (Verilog/NGC) and design files (Xilinx ISE/EDK/SDK) for an actual DPD system implemented with Kintex7 KC705 board and ADC/DAC FMCs.

The subfunctions supporting the simulation in MATLAB/Simulink are contain in the folder “MyfuncSource”, in which they are further categorized into 11 folders:

- **3GPP**: 3GPP conforming WCDMA signal generation and measurement
- **DPD**: PA/DPD model construction and simulations
- **DSM**: delta-sigma modulator simulation
- **DSP**: useful subfunctions for digital signal processing
- **Figure**: various functions for signal plotting and viewing
- **Identification**: linear regressive model identification algorithms such as LMS, RLS
- **InstrumMeas**: instrument controls through SCPI
- **LTE**: LTE signal generation and measurement
- **Math**: mathematical functions
- **Misc**: other useful functions
- **Signal**: various signal generation and processing

There are another 2 Simulink library files, which are

- **fpga_lib**: blocks for FGPA design, only available when Xilinx System Generator is properly installed
- **mylib**: blocks for Simulink simulation
1.2 Getting Start

- First, copy “MyfuncSource” to the following directory
  C:\Program Files\MATLAB

- Then add “MyfuncSource” to the search path, see the figure below.

- Restart the MATLAB, all the folders should be added to the searching path, if not, add all the folders manually.

- Open the Simulink library, as shown below

one can find two libs “FPGA” and “My Blockset” are added to the Simulink.
1.3 **OTHERS**

- To simulate and design with System Generator, Xilinx ISE higher than 14.4 should properly installed.

- Proper visa drivers are required to perform instrument control. One can use such as Agilent (KeySight) IO or NI visa. To use M8190a, Agilent IO is required.

1.4 **NOTE**

All the Simulink models were only validated under MATLAB 2012b Japanese Edition. For MATLAB in higher version, however, the Simulink may have different rules for defining the vectors. In such situation, the models may not be executed properly without corrections.
2 \textbf{DIGITAL PROCESSING \\ \\ \& \\ \\ VIRTUALIZATION}

2.1 \textbf{SIGNAL GENERATION}

In the procedure of source encoding, the randomness of the codes has been maximized to increase the information they can carry. Therefore, we can see the baseband signal processed in wireless communication systems is a random sequence. Most of the time we may not be able to find a clue from viewing the time domain waveform, as it comes out like the noise. Instead, we are more interested in the frequency domain power spectral density (PSD) plot, distribution and constellation, as they can provide us intuitive views of the properties of the signal. In what it follows, a 16-QAM signal is generated and virtualized with the provided example file \texttt{ex2_1.m}

```matlab
close all
clear all
clc
[x, xs] = qamgen(2^13, 16, 16, 40, 0.4); % generate 16-QAM signal x and symbol xs
% the signal is interpolated by a factor of 16, and pulse shaped with a
% raised cosine filter with a rolloff factor of 0.4
plot(x)
hold on
plot(xs, 'r.' )
axis square
xlabel('Real')
ylabel('Imag')
```

a figure should appear like
If we input

```python
figure
psdplot(x)
```

a figure should appear like

![Figure 1](image1.png)

which is the PSD of the signal. Try to change the parameter of psdplot, e.g.

```python
psdplot(x, 'r', 10)
```

we get a smoothed plot. Here 10 is the smooth order (default 1).
**Hint:** By inputting `help psdplot` in the prompt, one can view the help document for this function.

When we plot multiple plots in the same figure, we need special markers to distinguish the plots, for instance:

```matlab
x1=qamgen(2^13,16,8,40,0.1);
x2=qamgen(2^13,16,8,40,0.3);
x3=qamgen(2^13,16,8,40,0.5);
psdplot(x1,'r',10)
hold on
psdplot(x2,'b',10)
psdplot(x3,'m',10)
legend('Rolloff= 0.1','Rolloff= 0.3','Rolloff= 0.5')
```

we have the result that
To make the different plots more distinguishable we can use the “markplot” function:

```matlab
[X1,f1]=psdplot(x1,'r',10);
[X2,f2]=psdplot(x2,'b',10);
[X3,f3]=psdplot(x3,'m',10);
figure
markplot(f1,X1,14,{'k','-','m','s'})
hold on
markplot(f2,X2,15,{'b','--','b','o'})
markplot(f3,X3,16,{'r','-','g','>'})
legend('Rolloff= 0.1','Rolloff= 0.3','Rolloff= 0.5')
```
2.2 EFFECT OF NONLINEARITY

The program for this part can be find in \texttt{ex2_2.m}.

This section gives an intuitive view for the effect of nonlinearity.

```matlab
close all
clear all
clc
[x,xs]=qamgen(2^13,16,16,40,0.3);

xn=x/max(abs(x));  \% Normalize the signal with its max magnitude, as
\hspace{1cm}the saleh's model only accept signal smaller than 1

yn=salehs_model(xn);

y=yn/mean(abs(yn))*mean(abs(x));  \% rescale the signal

[yr]=rerrcf(y,16,0.3,640,'sqrt');  \% Matching filtering
[xr]=rerrcf(x,16,0.3,640,'sqrt');

figure
psdplot(x,'k')
hold on
psdplot(y,'b')
legend('Original','With Nonlinearity')
```

This would produce the PSD plots for the signals without and with the effect of nonlinearity

Very clearly, the nonlinearity expands the spectrum that should be strictly restricted inside the band determined by the pulse-shaping filter.
We can also see the effect on the signal constellation

```matlab
figure
plot(xs,'kx')
hold on
plot(yr,'b.'
plot(xr,'mo')
axis square
xlabel('Real')
ylabel('Imag')
legend('Original','With Nonlinearity','Without Nonlinearity')
```
The effect of nonlinearity includes amplitude compression, amplitude warping, constant phase rotation, amplitude-dependent phase rotation and noise leading.

AM-AM and AM-PM plots are frequently used to characterize the amplitude-dependent nonlinearity.

```plaintext
def amp(x, y):
    # Code for the function
```

![AM-AM and AM-PM plots](image_url)
We can now generate a more complicated signal, e.g. the LTE signal, this can be found the file ex3_1.m.

Before that, let us have a brief overview of the physic layer of the LTE. First of all, the LTE employs OFDM that assembles a number of subcarriers in the frequency domain to maximize the frequency utilization. For LTE, the bandwidth of 1 subcarrier is 15kHz, and 12 subcarriers form a resource block (RB) which is 180kHz, as seen below

The LTE supports 6 different modes of RB configuration, as tabulated below

<table>
<thead>
<tr>
<th>Channel Bandwidth (MHz)</th>
<th>1.4</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission Bandwidth (MHz)</td>
<td>1.08</td>
<td>2.7</td>
<td>4.5</td>
<td>9</td>
<td>13.5</td>
<td>18</td>
</tr>
<tr>
<td>RB</td>
<td>6</td>
<td>15</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
</tbody>
</table>

Therefore, for different RB configurations, the LTE signal contains different numbers of samples for 1 time slot, and this has been summarized below
We see that, for instance, the 20MHz LTE signal has 15360 samples for 1 time slot, and the sampling frequency is 30.72MHz if no over sampling is performed.

For more information, please find semi_LTE.pdf in the Doc folder.

ex3_1.m contains a simple example for LTE signal generation and measurement

close all
clear all
clc
x=ltegen(1:2,16,5,4); % generate 20MHz LTE signal oversampled 4 times
% (with length of 2 time slots)
fs=30.72*4;

% to see the PSD
figure
psdplot(x,'b',1,2^13,fs)
xlabel('Frequency (MHz)')

% to see the CCDF
xg = randn(length(x), 1) + j*randn(length(x), 1);
figure
ccdf(abs(x)/sqrt(mean(abs(x).^2)))
hold on
ccdf(abs(xg)/sqrt(mean(abs(xg).^2)), 'k')
xlim([-3 12])
ylim([1e-4, 1])
legend('LTE', 'Gaussian Noise')

We also can use
papr(x)
papr(xg)
to measure the signal’s PAPR (peak-to-average power ratio)

Using

\[ \text{lteevm}(x,16,5,4,1:2) \]

can measure the EVM for the LTE signal

If we add linear distortion to the LTE signal

\[
[b,a]=\text{cheby1}(5,3,0.5);
\]
\[
y=\text{filter}(b,a,x);
\]
\[
\text{lteevm}(y,16,5,4,1:2)
\]

we have degraded EVM of 10%
We can turn on the automatic equalizer to compensate linear distortion by

\texttt{lteevm(y,16,5,4,1:2,1)} \% With equalizer

Then we can have correct measured EVM:
Mean EVM: 0.76% (-42.4 dB)
Max EVM: 0.86% (-41.3 dB)
Min EVM: 0.66% (-43.6 dB)
Reference Error: 64.04% (-3.9 dB)
Frequency Error: -21.1 dB
DC Error: -116.2 dB
IQ Error: -\infty dB
Frequency Shift: 8 Hz
4 DPD EXAMPLES

4.1 QRD-BASED DPD

This section provide an example for DPD simulation. As shown below, the DPD simulation contains 5 elemental factors which are: PA model, feedback path, error calculation, adaption and DPD model.

The PA model used in this simulation is a Saleh’s model which has the expression of

\[ y = \frac{a_1 |x|^{d_2} e^{i1k|x|}}{1 + b_1 |x|^2} \]

We can realize such equation in the Simulink, say...
The DPD model is a 5th order polynomial that can be written as

\[ y = \sum_{n=1}^{5} c_n |x|^{n-1} x \]

with its diagram

The adaption used in this example is QRD-based RLS algorithm. It can be realized with very efficient systolic structure as shown below.
Please find the folder QRD_DPD that contains the source files for more details.

4.2 **ORTHOGONAL MODEL**

If the signal is wide sense stationary, and its distribution is known *a priori*, we can find a matrix to orthogonalize the model. Using orthogonalized model can be very desirable, in terms of both numerical stability and convergent rate. For more information, read my Ph.D thesis (Section 3.1.3).

For perturbation based DPD, we can find that LMS has the lowest hardware overhead, but has difficulties to converge to the global optimum solution. This chapter demonstrates a DPD using a so-called Hammerstein model (a polynomial followed by an FIR filter). With orthogonal mapping, we can achieve rapid convergence even employing two LMS blocks that independently update the polynomial and FIR filter, respectively.

Please find the folder Orth_DPD that contains the source files for more details.
The polynomial is mapped to orthogonal basis with a matrix multiplexing, we can see the triangular structure of the mapping matrix from the following figure.
The LMS block is shown below
5 Crest Factor Reduction

The commonly used OFDM, CDMA and multi-carrier signals have large PAPR, as they generally follow Rayleigh or exponential distribution. That the occurrence of large peak has low probability. This is very adverse to the PA’s efficiency. If we have constant power supply, and the output signal reaches the peak out power, we see the shaded area can represent the wasted power.

After DPD, the signal is expanded, by adjusting its average power to match the peak power, we see that the shaded area is larger than the that in the left. If we can properly reduce the peaks, and match the peak the power we will get smaller wasted power. But we need to notice that larger the power larger the distortion. Therefore, a balance should be taken between the distortion and efficiency. The figure shown below has demonstrated this concept.
We can see that with only DPD, the achieved efficiency may be even lower than that without DPD; while with only CFR, we get larger portion of the signal being distorted.

The most common CFR is clipping and filtering (CF), and there is an alternative method known as peak cancelling (PC) that has much lower complexity. This chapter will adopt these two methods to reduce the PAPR of the LTE signal, and gives comparisons in the aspects of distortion and efficiency with and without the presence of DPD. The source code can be found in `ex5_1.m` and `ex5_2.m`.

close all
clear all
x=ltegen(1:2,16,5,4);
x=awgn(x,65,'measured');
iter=3;
bw=18.5;fs=3.84*8*4;
L=30; % 2L+1 is the filter length
Nt=5000; % the number of points used for DPD training
BO=7; % Backoff
x=x/max(abs(x));
Nnl=7; % predistorter order
h=sinc((-L:L)*bw/fs);
h=h.*kaiser(length(h),1)';
xcf=cfr(x,7,h,'cf',iter); % reduce the PAPR to 7 dB
xpc=cfr(x,7,h,'pc1',iter);

we first is to see the CCDF

figure
ccdf(x/sqrt(mean(abs(x).^2)),'k')
hold on
ccdf(xcf/sqrt(mean(abs(xcf).^2)),'r')
ccdf(xpc/sqrt(mean(abs(xpc).^2)),'b--')
xlim([2 12])
ylim([10e-5 1])
We see PC gets the desired PAPR (almost 7 dB), while CF has an obvious PAPR regrowth due to the effect of filtering.

We also need to see the spectrum of the signals after CFR, as CFR also introduces out-of-band distortion.

```matlab
figure
psdplot(x,'k')
hold on
psdplot(xcf,'r')
psdplot(xpc,'b--')
ylim([-100 20])
legend('LTE','CF','PC')
```
See that, with properly designed filter, the out-of-band distortion can be suppressed to a very low level.

Now we perform a rigorous comparison:

close all
clear all
x=ltegen(1:2,16,5,4);
iter=3;
bw=20;fs=3.84*8*4;
L=30; % 2L+1 is the filter length
Nt=5000; % the number of points used for DPD training
G=2; % desired linearization gain
BO=7; % Backoff
x=x/max(abs(x))/G;
Nnl=7; % predistorter order
h=sinc((-L:L)*bw/fs);
h=h.*kaiser(length(h),3)';
num=0;
par_iter=3:0.5:8.5;
% figure(1)
% ccdf(x,'r')
% hold on
% xlim([-5 12])
for thld=par_iter
num=num+1;

xcf=cfr(x,par_iter(num),h,'cf',iter);xpc=cfr(x,par_iter(num),h,'pc1',iter);
xcf=xcf/max(abs(xcf))/G;xpc=xpc/max(abs(xpc))/G;
%  ccdf(xcf)
ycf=salehs_model(xcf,0);ypc=salehs_model(xpc,0);
ycf_pa=salehs_model(xcf/max(abs(xcf)),-BO);ypc_pa=salehs_model(xpc/max(abs(xpc)),-BO);
%------ DPD Applies Below------
Ycf=MemPolyn(ycf(1:Nt)/G,Nnl,1);Ypc=MemPolyn(ypc(1:Nt)/G,Nnl,1); % polynomial
% model applied to the PA output
Wcf=pinv(Ycf)*xcf(1:Nt);Wpc=pinv(Ypc)*xpc(1:Nt); % compute the coefficients of the
% post-compensator (post-inverse). this is the LS solution for Y*W=x
% which means after applying the post-inverse, the signal equals to the
% original input signal
Xcf=MemPolyn(xcf,Nnl,1);Xpc=MemPolyn(xpc,Nnl,1); % polynomial for input signal
xcf_pd=Xcf*Wcf; xpc_pd=Xpc*Wpc; % the DPD output signal (the post-inverse
  % is used as the DPD)
ycf_pd=salehs_model(xcf_pd,0);ypc_pd=salehs_model(xpc_pd,0); % Apply the predistorted signal to the PA
%------ DPD Applies Above------
ecf(num)=pi/4*sqrt(mean(abs(ycf_pa).^2));
epc(num)=pi/4*sqrt(mean(abs(ypc_pa).^2));
par_cf(num)=papr(xcf);par_pc(num)=papr(xpc);

ycf_pa=adjust(x,ycf_pa);ypc_pa=adjust(x,ypc_pa);xcf=adjust(x,xcf);xpc=adjust(x,xpc);
ecf_pd(num)=pi/4*(mean(abs(ycf_pd).^2))/mean(abs(ycf_pd));
epc_pd(num)=pi/4*(mean(abs(ypc_pd).^2))/mean(abs(ypc_pd));
ycf_pd=adjust(x,ycf_pd); ypc_pd=adjust(x,ypc_pd);

acp_cf(:,num)=aclr(ycf_pa,fs,18,20); acp_pc(:,num)=aclr(ypc_pa,fs,18,20);

acp_cf_pd(:,num)=aclr(ycf_pd,fs,18,20); acp_pc_pd(:,num)=aclr(ypc_pd,fs,18,20);
ncf(num)=nmse(x,xcf); npc(num)=nmse(x,xpc);
ncf_pa(num)=nmse(x,ycf_pa); npc_pa(num)=nmse(x,ypc_pa);
ncf_pd(num)=nmse(x,ycf_pd); npc_pd(num)=nmse(x,ypc_pd);

end

close all
figure('Position',[1 21 400 300])
hold on
plot(par_cf,ncf_pa,'--s','MarkerFaceColor','b')
plot(par_pc,npc_pa,'r--o','MarkerFaceColor','r')
plot(par Cf,ncf_pd,'g--s')
plot(par_pc,npc_pd,'k--o')
xlabel('Achieved PAPR (dB)')
ylabel('NMSE (dB)')
legend('C&F W O DPD','PC W O DPD','C&F W DPD','PC W DPD','C&F','PC',0)

figure('Position',[401 21 400 300])
plot(par Cf,ecf*100,'-s','MarkerFaceColor','b')
hold on
plot(par_pc,epc*100,'r--o','MarkerFaceColor','r')
plot(par Cf,ecf_pd*100,'g--s')
plot(par_pc,epc_pd*100,'k--o')
xlabel('Achieved PAPR (dB)')
ylabel('Efficiency (%)')
legend('C&F','PC','C&F W DPD','PC W DPD',0)

figure('Position',[1 401 400 300])
This result tells us the lower achieved PAPR, the larger distortion.
However, lower PAPR can result to higher efficiency.

The above figure shows the compromising between NMSE and efficiency. With DPD+PC, we can achieve higher efficiency under the same distortion restriction.
Similar results apply for the ACLR.
6 MODEL REDUCTION USING SUBSPACE SELECTION

This section is copied from my Ph.D thesis, you can find more details from:

*Advanced Digital Signal Processing Technology for Nonlinear Compensation in Wireless Communication Systems*

[https://www.academia.edu/10229574/Advanced_Digital_Signal_Processing_Technology_for_Nonlinear_Compen...](https://www.academia.edu/10229574/Advanced_Digital_Signal_Processing_Technology_for_Nonlinear_Compen...)

6.1 REPRESENTING THE MODEL WITH ORTHOGONAL BASIS

If the signal is wide sense stationary, and its distribution is known a priori, we can find a matrix to orthogonalize the polynomial. Define the covariance matrix

\[
R = E[X^H X]
\]

(6.1)

It can be transformed to a unit matrix \( I \), if a projection matrix is introduced,

\[
\]

(6.2)

The projection matrix \( U \) recasts the basis space \( X \) in a space that all the subspaces are orthogonal to each other. Here, we use

\[
Ψ = XU = [φ_1, φ_2, \ldots, φ_N]
\]

(6.3)

to denote the orthogonalized space, where

\[
φ_1 = u_{11}x_1
\]

\[
φ_2 = u_{12}x_1 + u_{22}x_2
\]

\[
\vdots
\]

\[
φ_N = u_{1N}x_1 + u_{2N}x_2 + \cdots + u_{NN}x_N
\]

(6.4)

is the operators of the orthogonal model and
is a upper triangular matrix.

The pseudo code based on classic Gram-Schmidt orthogonalization is summarized as follows

### Initialization:

\[
\Psi = X \quad U = \begin{bmatrix} 1 & 0 \\ \vdots & \ddots \\ 0 & 1 \end{bmatrix}_{L \times L}
\]

### Update:

**Loop 1:**

\[
G = \begin{bmatrix} 1 & 0 \\ \vdots & \ddots \\ 0 & 1 \end{bmatrix}_{L \times L}
\]

**Loop 2:**

\[
G(k, m) = \frac{\phi_k \cdot \phi_m}{\phi_k \cdot \phi_k}
\]

\[
\phi_m = \phi_m + G(k, m) \phi_k
\]

\[
U = UG
\]

\[
\phi_k = \frac{\phi_k}{\|\phi_k\|}
\]

U(k,:) = \frac{U(k,:)}{\|\phi_k\|}

**Adjust:**

\[
U = \frac{U}{u_{11}} \quad \Psi = \frac{\Psi}{u_{11}}
\]

### 6.2 Representing the Model with Orthogonal Basis

Model pruning is intrinsically a subspace selection problem, i.e. selecting dominant columns from the whole basis space X, and several mathematical methods have been developed to accomplish this task, which are based on either singular value decomposition (SVD) or QR factorization.
Now we first return to the result shown in the last subsection where we have developed the orthogonalized basis space:

$$\Psi = XU.$$  \hspace{1cm} (6.6)

If we use the above orthogonal space (orthogonal model) to perform the estimation with LS we will have:

$$\tilde{W} = (\Psi^H\Psi)^{-1}\Psi^Hd.$$  \hspace{1cm} (6.7)

Recalling (6.2), there yields:

$$\tilde{W} = \Psi^Hd,$$  \hspace{1cm} (6.8)

which reveals a merit of using orthogonal model that the matrix inverse can be removed. The residual error of LS, namely mean square error (MSE), can be written as

$$\zeta = \|\Psi\tilde{W} - d\|^2
= (\Psi\tilde{W} - d)^H(\Psi\tilde{W} - d)
= \tilde{W}^H\Psi^H\tilde{W} - \tilde{W}^H\Psi^Hd - d^H\Psi\tilde{W} + d^Hd
= \tilde{W}^H\tilde{W} - \tilde{W}^H\Psi^Hd - d^H\Psi\tilde{W} + d^Hd.$$  \hspace{1cm} (6.9)

Substituting (6.6) into the above equation it yields:

$$\zeta = (\Psi^Hd)^H\Psi^Hd - (\Psi^Hd)^H\Psi^Hd + d^Hd
= d^Hd - \sum_{i=1}^{N}\|\phi_i^Hd\|^2.$$  \hspace{1cm} (6.10)

Therefore, the $$\phi_i, (i = 1, 2, \ldots, N)$$ that result in largest $$\|\phi_i^Hd\|^2$$ are the dominant terms with the largest contribution to MSE.

Our task is to select the dominate subspaces from the basis space $$X$$ instead of $$\Psi$$, thus there needs some extra efforts to refactor the $$X$$ from the reduced $$\Psi$$. This is a matrix factorization updating problem and an example is given below to select
the most significant spaces from a memory polynomial. The code can be found in `ex_6.1.m`.

```matlab
load DPA_Sig
y=tdelay(y,0.1);
x=x(1:2:end); y=y(1:2:end);
N=7; M=4; % define the memory polynomial with nonlinear order of 7
% and memory depth of 4
X=MemPolyn(x,N,M);
[Xs,redind,redpros]=krllsel(X,y,N*M); % perform subspace selection
% the resulting Xs is the reordered space of X with the most
% significant
% subspaces in the left hand

We can see the result

```matlab
%%
close all
figure('Position',[401 100 600 300])
err=diff([-0;redpros(:,2)]);
[sind,ll]=sort(redind);
serr=(err(ll));
serr=reshape(serr,N,M);
bar(serr,'grouped')
xlabel('Nonlinear Order')
ylabel('Contribution to MSE (dB)')
legend ('Memoryless','1st Memory Depth','2nd Memory Depth','3rd Memory Depth')
%xlim([0.5 21.5])

figure('Position',[1 100 400 300])
plot(redpros(:,2),'k-s','MarkerFaceColor','b')
xlabel('Number of Selected Subspaces')
ylabel('NMSE (dB)')
grid on
```
Seeing the plots below, we know that leaving only 10 terms, the memory polynomial can yield good performance as the full model.
7 Concurrent Dual-Band Predistorter

This section demonstrates a simple simulation of Spectra-Folding Feedback (SFFB) concurrent dual-band DPD that has been proposed in chapter 9 of my Ph.D thesis. The code can be found in ex7_1.m.

```matlab
clear all
close all
osr=16; Gimb=0; f=50; bw=15*6; fc=150;
fs=3.84*8*osr; Ns=2^15;
Nl=2^14;
x1=ltegen(1,16,3,osr*2);
x2=ltegen(1,16,5,osr); x2=x2/max(abs(x2));
Ns=length(x1);
G1=0.5*db2mag(-Gimb); G2=0.5*db2mag(Gimb);

t=(1:Ns)'/fs;
x1=G1*x1; x2=G2*x2;
x=x1.*exp(-j*pi*2*f*t)+x2.*exp(j*pi*2*f*t);
y=salehs_model(x); y=awgn(y,70,'measured');

f1=figure('Position',[1 1 600 450]);
figure(f1)
hold on
psdplot2(y,'b',fs,5)

N=5; onlyodd=0; mu=1;
Wl=zeros(sum(1:N),1); Wl(1)=1; W2=Wl;
indbw=round(bw/fs/2*Nl);
indif=round(f/fs*Nl);

xt1=x1; xt2=x2;
```
Mx1=xpolyn(xt1,xt2,N,onlyodd);Mx2=xpolyn(xt2,xt1,N,onlyodd);
XX=[conj(Mx1),Mx2];

yf=idealfilt(y,(f+2*bw)/fs);
y1=idealfilt(y.*exp(j*pi*2*f*t),bw/fs);
y2=idealfilt(y.*exp(-j*pi*2*f*t),bw/fs);
ytx=real(yf.*exp(-j*pi*2*fc*t));
yrx=ytx.*cos(-pi*2*(fc)*t);
yt =idealfilt(yrx.*exp(-j*pi*2*f*t),bw/fs);
WW=pinv(XX)*yt;

nmse(XX*WW,yt)
ym2=XX(:,end/2+1:end).*WW(end/2+1:end);
ym1=conj(XX(:,1:end/2).*WW(1:end/2));
ym1=adjust(x1,ym1);ym2=adjust(x2,ym2);
y1=adjust(x1,y1);y2=adjust(x2,y2);
My1=xpolyn(ym1,ym2,N,onlyodd);My2=xpolyn(ym2,ym1,N,onlyodd);
Y1=xpolyn(y1,y2,N,onlyodd);Y2=xpolyn(y2,y1,N,onlyodd);
Wp1=pinv(My1)*x1;Wp2=pinv(My2)*x2;
Wd1=pinv(Y1)*x1;Wd2=pinv(Y2)*x2;
xpd1=Mx1*Wp1;xpd2=Mx2*Wp2;
xpdd1=Mx1*Wd1;xpdd2=Mx2*Wd2;
xpd=xpd1.*exp(-j*pi*2*f*t)+xpdd2.*exp(j*pi*2*f*t);
xpdd=xpdd1.*exp(-j*pi*2*f*t)+xpdd2.*exp(j*pi*2*f*t);
ypd=salehs_model(xpd); ypd=awgn(ypd,60,'measured');
ypdd=salehs_model(xpdd); ypdd=awgn(ypdd,60,'measured');
figure(f1)
psdplot2(y,'g',fs,5)
hold on
psdplot2(ypd,'r',fs,5)
psdplot2(ypdd,'k:',fs,5)

legend('Wo DPD','W SFFB DPD','W DPD')
ylim([-110 -10])

ypd1=idealfilt(ypd.*exp(j*pi*2*f*t),bw/fs);
ypd2=idealfilt(ypd.*exp(-j*pi*2*f*t), bw/fs);
ypd1=adjust(x1,ypd1); ypd2=adjust(x2,ypd2);
nmse(x1,ypd1)
nmse(x2,ypd2)
8 Instrumental Experiments Using MATLAB Instrument Toolbox

8.1 How to Control the Instruments

There are several standards for the interface used for communications between the instruments and computer, which are GPIB (general purpose interface bus), VISA (virtual instrument software architecture), TCP/IP (local area network abbrv. LAN) and serial port. The instruments are controlled by software compatible with SCPI (Standard Commands for Programmable Instruments), which is a universal syntax and commands for programming the test. There are various selections for instrument control software, including the ones from first-part instrument makers such as Agilent VSA81900, as well as third-part software such as LabView and MATLAB instrument toolbox.
8.2 Example: Control the M8190A

M8190A uses PCI-E cable to communicate with the PC, to invoke the communication, M8190A driver should be installed to the computer. Instrument Control Toolbox should also be installed for MATLAB which provides the SCPI (standard commands for programmable instruments) commands support. Follow these steps to start the measurement:

1. Switch on the M8190A (first turn on the power switch behind and push the power bottom on the front panel).
2. Launch the computer that is connected with the M8190A. If the computer is already on, reboot it.
3. The PC should recognize the instrument. To see if it is properly installed, check the device manager. The instrument should be visible as Agilent-M8190.
4. Start the driver: just double click the M8190A icon on the desktop. If the connection fails, reboot the computer.
5. Start MATLAB and get ready for test.

There is a simple software provided by Agilent to control the instrument. It can be found in Start->Agilent->M8190->Control Panel. Please note this software only provide basic functions such as voltage control and path routing, signal uploading is not supported. To fully access the M8190A, the only way is to use SCPI programming, and this can be accomplished in the MATLAB. This chapter only provides some basic orders for controlling the M8190A using Instrument Control Toolbox in MATLAB. For full guideline to the use of SCPI for M8190A, refer to the M8190A manual.

8.2.1 Connect with the M8190A using Instrument Control Toolbox

To set up a connection with the instrument use the following order:

\[
f = \text{visa}('agilent', 'TCPIPO::localhost::inst0::INSTR');
\]

Note: Only TCP/IP protocol can successfully communicate with the M8190A in MATLAB, and PXI protocol may fail to control the instrument in MATLAB.

Then set basic parameters for the connection and start the communication.
f. OutputBufferSize = 20000000;
f. InputBufferSize = 640000;
f. Timeout = 20;
fopen(f);

**Hint**: to view the instruments that are communicating with MATLAB, input `tmtool` in the MATLAB prompt to launch the Instrument Control Toolbox user interface. Please ensure only one connection with M8190A is open.

### 8.2.2 Set Basic Parameters for the M8190A

**Examples:**

- Set the sampling rate to 8Gsaps (M8190A supports maximum sampling rate of 12Gsaps)
  ```matlab
  fprintf(f, sprintf(':freq:rast %.12g', 8e9));
  ```

- Set the voltage of the 1st channel to be 0.5V
  ```matlab
  fprintf(f, sprintf(':volt%d:ampl %g', 1, 0.5));
  ```

- Reset the instrument
  ```matlab
  fprintf(f, '*rst');
  ```

- Switch off the instrument
  ```matlab
  fprintf(f, ':outp%d off', 1));
  ```

- Set the 1st channel mode to be ‘DAC’
  ```matlab
  fprintf(f, ':outp%d:rout %s', 1, 'DAC'));
  ```

### 8.2.3 Upload Signals to the M8190A

We suppose **data** is the signal which exists in the MATLAB working space.

The following commands upload the **data** to the instrument.
offset = 0;

while (offset < segm_len)
    len = min(segm_len - offset, 524160);
    cmd = sprintf(':\trac%d\:data %d,%d,\: ', chan, segm_num, offset);
    binblockwrite(f, data(1+offset:offset+len), 'int16', cmd);
    fprintf(f, ' ');
    offset = offset + len;
end

where segm_len is the length of the segment, normally it can be set to 1, and len is the length of data.

**8.2.4 Processing the Signal**

*data* needs to be normalized since the DAC only receives the data smaller than 1.

```
data = data / (max(max(abs(real(data))), max(abs(imag(data)))));
```

Note that the M8190A in our lab only support 12bits DAC, to maximize the dynamic range of the signal, quantization of the signal need to be done.

Since **Fixed Point Toolbox** is not installed, we can only quantize the signal with the closest form, which is 16bits

```
data = int16(round(8191 * data) * 4);
```

If a *marker* is required, we can use the following commends to apply it

```
data = data + int16(bitand(uint16(marker), 3));
```

**8.2.5 Programming Examples**

The M8190A is not so easy to use, especially for beginners. I have made some useful sub-functions in MATLAB, and the users can directly apply these functions to generate
signal, control the instrument and perform the measurement. Here are two useful examples for the users to understand the basics for controlling M8190A.

**Example 1**: Generate a multi-sine signal and upload it to the M8190A. The signal is centered at 3.3GHz, it has 221 tones and the tone spacing is 0.5MHz.

```matlab
clear all
close all

fs=12e9;
fc=3.3e9;
Ns=2^6*95*fs/150e6; M=221; Bw=110; fs1=150;

fcar=-12.5;
MultiToneF=linspace(-Bw/2,Bw/2,M); MultiToneF=MuliToneF(:);
InitialPhase=mod((1:M).^2/2/M,2)*pi;
InitialPhase=InitialPhase(:);
t=(0:Ns-1)';
f=MultiToneF/(fs/1e6);
wd=(f(2)-f(1))/2*pi*2;
xb=zeros(size(t));
for ii=1:M
    xb=xb+exp(j*2*pi*f(ii)*t+j*InitialPhase(ii));
end
xb=xb(:)/M;
t=(0:Ns-1)';
x=xb.*exp(j*pi*2*Fc*t/fs);
```
Example 2: Generate an LTE-A downlink signal which contains five 20MHz LTE carriers and upload it to the M8190A. The modulation scheme is QPSK, 1200 subcarriers OFDM is involved.

clear all

close all

fs=12e9;
fc=3.0e9;

osr=round(fs/20e6);

Mq=4:

xs = randint(1000,1,Mq);

xs=modulate(modem.qammod('M',Mq,'PhaseOffset', 0),xs);

xb=ofdm(xs,1200,2048,fs/20e6,20,0.2):

Nc=5:

ff=linspace(-(Nc*20/2-10),(Nc*20/2-10),Nc):

xb=mcgen(xb,fs,ff*1e6):

xb=xb(:):

Ns=length(xb):

t=(0:Ns-1)';

x=xb.*exp(j*pi*2*fc*t/fs);

awgset('sig',x);
Here, let's have a look at an example of DPD instrumental experiment using R/S FSW/FSV VSA and Agilent N5182a VSG.

This experiment uses FSW to collect the PA output signal, and the spectrum is displayed in the FSV. The code can be found in `ex8_1.m`.

clear all
close all
clc
x=ltegen(1,64,5,4);
x=x/max(abs(x));
L=30; att=3; attref=48.7;
fs=3.84*8*4;fc=2.1; % fs is sampling frequency in MHz, ...
% fc is the center frequency in GHz
h=sinc((-L:L)*18.5/fs).*kaiser(L*2+1,2.5)';
xcfr=cfr(x,7.5,h,'pctl',3); % reduce to PAPR to 7.5 dB
disp(['PAPR: ',num2str(papr(x))])
disp(['PAPR after CFR: ',num2str(papr(xcfr))])
pow=1; % set the VSG output power to 1 dBm
n5182set('fc',fc,'pow',pow,'fs',fs);
n5182set('sig',xcfr); % upload the signal to VSG

%%
fswset('mode','Spectrum') % set the measure mode to spectrum
fswset('fc',fc,'rbw',100) % set the resolution bandwidth
fsvset('fc',fc,'rbw',100)
fsvset('acp',[1 2 18.015 20]) % set the channel (1 channel, 2 adjacent channel)
% spaced by 20MHz, and the occupied bandwidth 18.015MHz
fswset('acp',[1 2 18.015 20])
fswset('span',150,'av',5,'att',5)  \% set the span, average number and attenuation

fswset('tracr','av')
fswset('trac2','av')
fsvset('tracr','av'))\%
fsvset('trac2','av')\%
\%
xc=xcfr;
ysp{1}=fswset('read',1);
cc={'k' 'r' 'b' 'm' 'g'};
y1=fswread(length(x),fs,fc,att,0);
y2=talign(xc,y1,256);
y(:,1)=adjust(xc,y2);
ampm(xc,y(:,1))
\%
f1=figure(1);
set(f1,'Position',[401 1 800,600])
subplot(2,2,[2 4])
hold on
psdplot(y(:,1),cc{1},20)
subplot(221)
hold on
plot(abs(xcfr),abs(y(:,1)),[cc{1} '.'])
subplot(223)
hold on
plot(abs(xcfr),angle(y(:,1)./xcfr)*180/pi,[cc{1} '.'])
P_pa=pwsens;

clear WW
N=7;M=3;Nls=2^14;
iter=2;
fsvset('trac2','view')
fsvset('tracr','av')\%
disp('----------------DPD on----------------')

for ii=1:iter
    ind=mod(Nls*(ii-1)+1:Nls*ii,length(x)+1);
    yt=y(ind,ii);xt=xc(ind);
    Y=Exten_Mod(yt,N,M,M-1);X=Exten_Mod(xcfr,N,M,M-1); % using extended memory polynomial
    W=pinv(Y)*xt;
    WW(:,ii)=W;
    xc=X*W;
    n5182set('sig',xc);
    disp('Push Enter to continue')
    pause(1.5)
    ysp{ii+1}=fsvset('read',1);
    P_dpd(ii)=pwsens; % read the PA output power
    y1=fswread(length(x),fs,fc,att,0);
    fswset('mode','Spectrum')
    y(:,ii+1)=talign(xc,y1,128); % time alignment both integer delay and fractional delay
    y(:,ii+1)=adjust(xc,y(:,ii+1));% power alignment
    fi=figure(1);
    subplot(2,2,[2 4])
    hold on
    psdplot(y(:,ii+1),cc{ii+1},20)
    subplot(221)
    hold on
    plot(abs(xcfr),abs(y(:,ii+1)),[cc{ii+1} '.'])
    subplot(223)
    ylim([-50,50])
    hold on
    plot(abs(xcfr),angle(y(:,ii+1)./xcfr)*180/pi,[cc{ii+1} '.'])
end

%%
% y=y/max(abs(y));
ff=fname(['LTE20M_EMP_']);
save([ff,'.mat'], 'x', 'ysp', 'y', 'WW', 'att', 'fc', 'xcfr', 'P_pa', 'P_dpd', 'pow', ... '
' 'N', 'M', 'Nls', 'attref')

fsvset('sc', ff) % capture the screen from FSV

We have the figure stored in FSV

![Spectrum](image_url)
9  DPD DESIGN WITH XILINX
SYSTEM GENERATOR

To run this demo, Xilinx ISE higher than 14.3 is required. The files are included in the folder SG_LUTDPD.

This is a LUT based DPD. For the sake of simplicity, only a memoryless DPD (one LUT unit) is shown here. Refer to my paper for more information.


We see the whole system is constituted with 3 blocks: Adaptive LUT DPD, Delay Estimator and Address Process. Double click the blue block in the right hand, parameters relating to this simulation can be set there.
9.1 **Adaptive LUT DPD**

The Adaptive LUT DPD, needless to say, contains the DPD model and DPD parameter adaption (LMS).
We can open the block in green to take a closer look at the DPD, as shown in the figure below. There are 3 RAMs used. The RAM in the left is to receive the updated contents. It is then passed over the two RAMs in the right side. The two RAMs’ output signals are used to produce the interpolated DPD output.
9.2 **Delay Estimator**

As the feedback signal has a loop delay, it is crucial to align it with the input signal to produce the correct error signal. The largest block in the following figure contains the delay estimator, which is implemented as a black box.
The delay estimator is written in Verilog, its ports are defined here

module dlyest_1024len( 
  clk,
  ce,
  x,
  delayed,
  rst,
  nd,
  delay_est,
  corr_est,
  rdy,
  thld,
);

9.3 **Address Process**

Address process contains the functions to produce correct addressing signal.
Among these, it also contains a phase adjuster, as the constant phase shift may cause the direct learning DPD unstable.
Two CODIC cores are used here to estimate the phase and rotate the input signal. The instantaneous phase shift is smoothed with two exponential moving average filters.
10 FPGA IMPLEMENTATION

Here, we turn to the actual FPGA implementation of a DPD with up to 500MHz linearization bandwidth, which can be applied to LTE-Advanced signal with 100MHz bandwidth. The files relating to this chapter are included in the folder LTE5c_DPD_Demo_Kc705. A video showing actual experiment is also included in the folder Example.

10.1 BEFORE THE EXPERIMENT..

Use the KC705 board in the test. If you wish to do the experiment on VC707, please use the files in LTE5c_DPD_Demo_Vc707. Note some unresolved bugs exist in the VC707 version, which may make the DPD unstable.

The boards are configured as follows. If you are using KC705, connect the DAC board with the low pin connector (LPC) and ADC board with the high pin connector (HPC). If you are using VC707, connect DAC board with HPC1 and ADC board with HPC2. The DAC output should be passed with a BPF to eliminate undesired harmonics. We have two filters in our lab, each for 1.7 GHz and 2.1 GHz with 500+ MHz bandwidth. The DAC output is less than 0 dBm, please use proper driving amplifier when doing the experiment.

The diagram of the whole system is shown below
A simplified DSP (digital signal processor) MicroBlaze running at 100 MHz is also available to take up complicated computing tasks. The MicroBlaze also controls and monitors the peripheral chips (ADC, DAC, clock synthesizer and demodulator, etc.) with SPI (serial peripheral interface).

The DAC is 14-bit with rate of 2 GSa/s (giga samples per second), which removes the use of up converter as the DAC suffices to generate digital RF signal. A switch serving as the mixer is integrated in the DAC to extend the RF frequency coverage. Since the DAC has a data rate of 2GSa/s (max 2.5 GSa/s), the programmable transmission frequency is from 0 to 1 GHz. By using the mixer, this coverage is extended to 1~3 GHz if the DUC (digital up converter) is design to support frequency shift between 1~1 GHz. Accordingly, by setting bypass of the mixer and programming the DUC, the DAC can provide a frequency range of 0~3 GHz, which covers all the bands used for nowadays mobile communication. The DUC implemented in the FPGA is processed in parallel with 8 concurrencies, and each concurrency is under a clock of 250 MHz, thus the speed of the overall DUC is 2 GSa/s. The input to the DUC is a 250 MHz 2-parallelizing signal, yielding a maximum usable bandwidth of 500 MHz.
In the receiving path, a broadband quadrature demodulator is used to convert the RF signal to IF (intermediate frequency), and an ADC of 250 MSa/s is used to sample the IF signal. As the ADC is AC-coupled, zero-IF is not supported thereby the maximum usable bandwidth is only 125 MHz. Moreover, the I/Q impairment compensation needs not to take DC shift into account because of the AC-coupling. The digital IF signal is transferred to baseband with the DDC (digital down converter), which is followed with an equalizer to extend the distortionless usable bandwidth.

10.2 Generate Files..

By double clicking the system in the folder the EDK tool is started.

The IP core **duc_8p_bb2p** contains the whole transmission path, including CFR, DPD, DU, other DAC configuration drivers, etc. A DDR3 memory (2GB) can be used to store the signal. You can also use **romtx2p** as the signal source. **fmc150** contains the DDC, equalizer, I/Q balancer, etc. By clicking **Export Design**, the bit file will be generated. After the generation finished, the SDK tool would be automatically started.
The sw contains the C files used to configure the FPGA.

```c
/* @brief Main function. */
extern int main(void)
{
    int32_t mu_status;
    int32_t rc_status;
    int32_t ret;
    int32_t error = 0;

    Xil_ICacheEnable();
    Xil_DCacheEnable();

    /* Initialize the SPI peripheral */
    ret = SPI_Init(SPI_BASEADDR, 0, 0, 0);
    if(ret < 0)
    {
        return ret;
    }
    delay_ms(100);

    /* Configure ADF4350 device. */
    ret = adf4350_setup(SPI_BASEADDR, 2);
    /* Configure AD9739A device. */

    struct main_param *duc_param;
    duc_param->dac_current=0;
}
```

Connect both the UART and USB of the FPGA board with the PC. Push **Configure FPGA** to download the bit file to the FPGA. Then, **just take a drive!**