WHITE PAPER

Modeling RF Power Amplifiers and Increasing Transmitter Linearity with DPD Using MATLAB

This paper discusses an approach for modeling and simulating RF power amplifiers (PAs) and digital predistortion algorithms (DPDs) using MATLAB® and Simulink®. These tools let you rapidly develop ideas, validate designs, and build robust RF systems.

MathWorks®

This paper discusses an approach for modeling and simulating RF power amplifiers (PAs) and digital predistortion algorithms (DPDs) using MATLAB® and Simulink®. These tools let you rapidly develop ideas, validate designs, and build robust RF systems.

Introduction

RF power amplifiers lie at the front end of most RF systems, including wireless communications and radar systems, and are critical in ensuring the appropriate range of wireless systems. However, these amplifiers contain non-idealities that can cause performance degradation of the systems they drive when deployed in the real world. To mitigate this performance degradation, you can apply linearization algorithms including DPD as part of the transmitter or receiver chain. DPD algorithms aim to preemptively distort the waveform to be transmitted to counter the nonlinear effects of the power amplifier. Since these waveforms are often wideband, they must typically be implemented on FPGA/ASIC to achieve real-time performance.

DPD algorithms should be designed in the context of the full system because performance is largely dependent on the system in which it is placed. Simulation provides an early-stage behavioral model of the full system and makes it possible to develop the DPD algorithm early without dependency on hardware. System design and simulation with Simulink also provides a framework to validate the algorithm and other components later in development and can produce repeatable results that are hard to achieve with real RF hardware in the lab. In addition, you can use Simulink to create bit- and cycle-accurate models of algorithms as they would run on firmware. You can also generate production-quality HDL code from these models.

In this paper, we discuss the challenges around RF power amplifier design with a focus on RF imperfections, as well as a workflow for modeling PAs and increasing linearity with DPD (Figure 1).

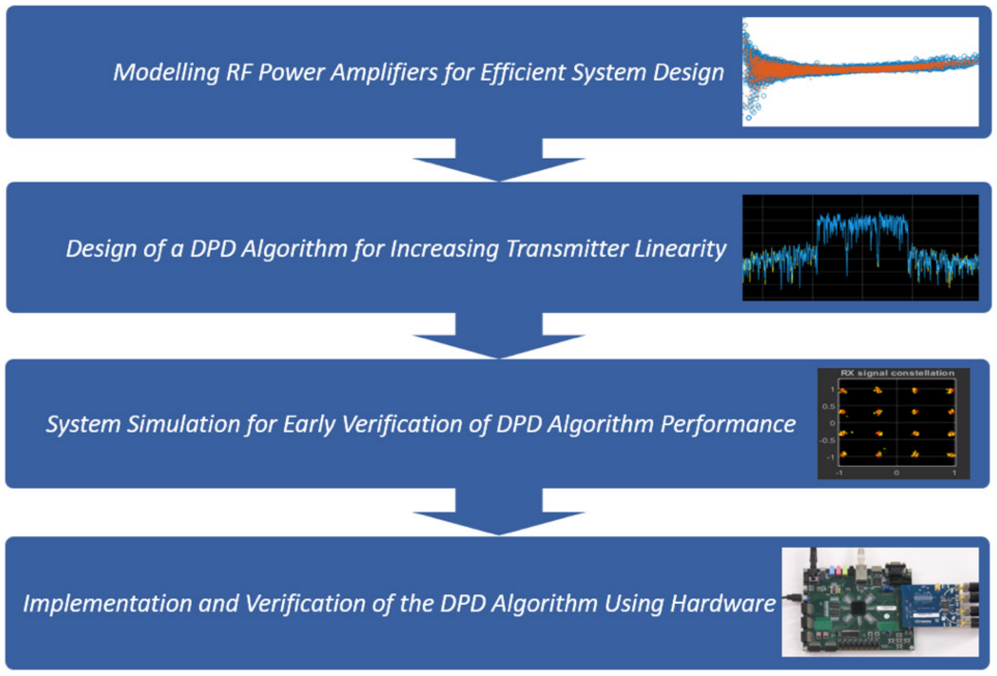

Modelling RF Power Amplifiers for Efficient System Design
Design of a DPD Algorithm for Increasing Transmitter Linearity
System Simulation for Early Verification of DPD Algorithm Performance
RX signal constellation
Implementation and Verification of the DPD Algorithm Using Hardware

Figure 1. DPD development workflow.

Challenges in RF Power Amplifier Design

The design of RF transmitters is becoming more challenging. The emergence of new standards such as 5G and the demand for multistandard transmitters present a range of contrasting design decisions. This includes tradeoffs on:

- **Size:** The transmitter might be confined to a small portable device or a limited real estate.
- **Weight:** Portability of the design might be key in ensuring its usability.
- **Power:** The device may be solar or battery powered.
- **Bandwidth:** Transmitters must work with increasingly larger signal bandwidths.
- **Linearity:** Amplifier gain will increase nonlinearly with input power.
- **Frequency:** Systems may need to operate in a range of frequencies from 10 Hz to 100 GHz.
- **Integrated technology:** Materials such as gallium-arsenide have improved high-frequency performance when used for fabricating transistors.
- **MIMO architecture:** Hybrid beamforming architectures are considered for good tradeoff of power and flexibility, for example.
- **Configurability:** Systems must be designed to be reconfigurable through the use of embedded algorithms to be able to operate under different conditions.

The design strategy you choose will have an impact on more than one of these tradeoffs. For example, increasing the bandwidth of the transmitter will typically draw more power and require the use of larger and heavier components and more specialized IC technologies. Since the job of these components is to ensure that the signal power is large enough, their requirements can tend to dominate the design.

PAs contain a number of non-idealities including nonlinearity and noise, which must also be assessed. Linearity describes the behavior of input power vs. output power. Ideally this should be perfectly linear, but in reality, this isn't the case, as shown in Figure 2.

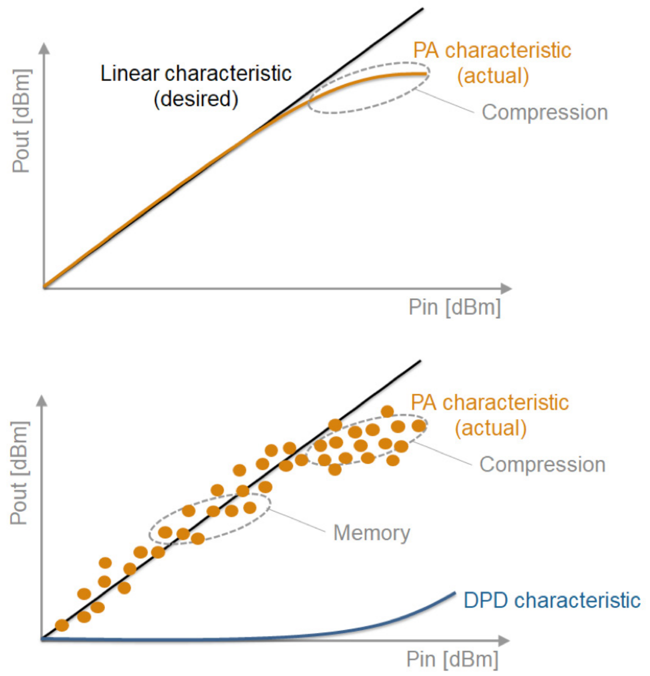


Figure 2. Top: Input/output power characteristic (orange) of a nonlinear power amplifier showing compression, and the ideal linear characteristic (black). Bottom: Input/output power characteristic (orange) of a nonlinear power amplifier showing memory effects, the static DPD characteristic (blue) to achieve the ideal linear characteristic (black).

When input power increases beyond a certain point, output power begins to increase in a nonlinear fashion through an effect known as *compression*. In general, this begins to happen at lower input power as the target carrier frequency of the system increases, causing high-frequency amplifiers to be less efficient. More crucially, waveform distortion also takes place. A further complication here is that power amplifiers contain memory effects such that output power will depend on input power at the current time as well as previous points in time. This is shown in the lower area of the two plots in Figure 2. Various techniques, including DPD, can be used to linearize the behavior in the PA to make it more deterministic.

Rapid prototyping is key to reaching an optimal amplifier and linearization design that can meet the many requirements listed above. However, this number of complex system requirements causes the design space to explode, which makes physical prototyping too time-consuming and impossible to scale. Simulation can help in producing and testing many prototypes to reduce development time and reach an optimal RF transmitter design.

Modeling RF Power Amplifiers for Efficient System Design

Power amplifier behavior is affected by nonlinear behavior and memory effects, and hence depends on input signal characteristics such as signal bandwidth, frequency, peak-to-average power ratio, modulation, and loading conditions. Most PA models that capture these effects are derived from Volterra series, often using complex identification procedures. Proprietary (“black box”) models of power amplifiers exist, but these tend to be problematic as they lack transparency, are hard to customize, and are difficult to understand when used in complex system simulations.

MATLAB offers a simple, fast, and customizable ("white box") fitting procedure to create a model of a PA that can then be integrated into a larger system simulation. The model is based on [1]:

$$y_{\mathrm{MP}}(n) = \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} a_{km} x(n-m) \left| x(n-m) \right|^{k} .$$

where:

a are the PA model coefficients

k is the degree of nonlinearity, K is the number of degrees to which the PA is modeled

m is the memory depth value, M is the memory depth of the PA model

Note that when M is equal to 1, the above model describes purely static characteristics such as AM-AM/AM-PM.

The fitting procedure uses input and output data from the PA to extract the PA model coefficients based on chosen values of K and M. Once fitting has been performed and a PA model created, the quality of fit is assessed. Values of polynomial order K and memory depth M are then iterated on until the model performs as desired. Iteration can be a manual process, or it can be automated within MATLAB.

As an example, consider the following complex IQ data capture collected at the input and output of the PA when an LTE modulated test signal is injected. The input and output signal magnitudes as a function of time are shown in Figure 3.

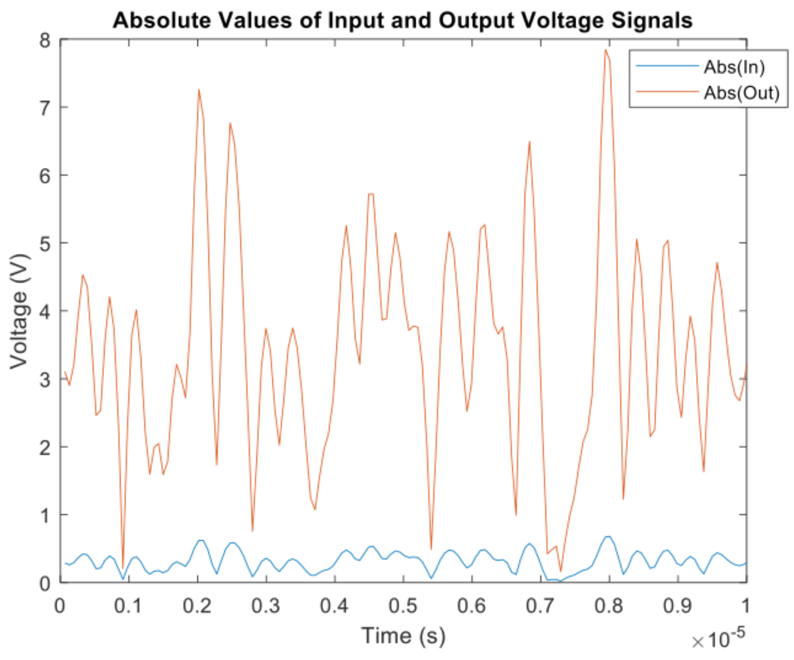


Figure 3. Input/output PA characteristics plotted as a function of time, showing the absolute value of the input waveform (blue), and the absolute value of the output waveform (red).

The power transfer characteristic plotted from this data is shown in Figure 4. Here, the x-axis shows the input power, and each point represents the instantaneous power gain. A straight horizonal line would imply that the PA is perfectly linear, and points spreading from this line would imply memory effects in the PA. Hence, in this case it can be deduced that the PA is mildly nonlinear and has memory effects.

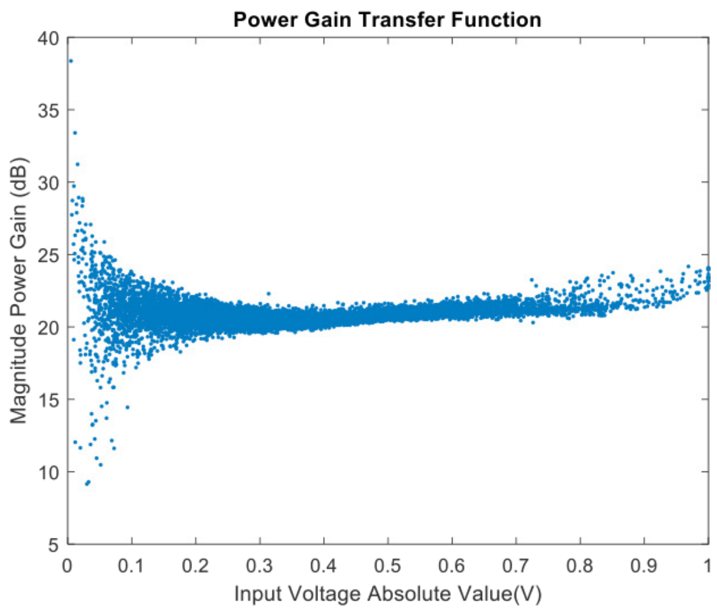
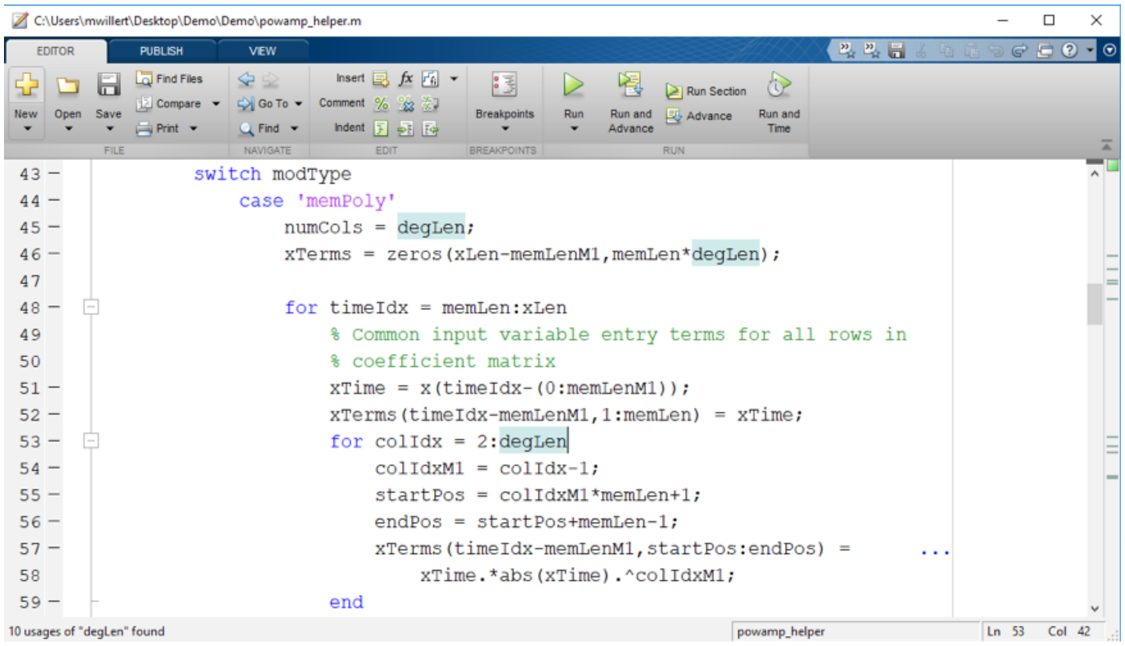


Figure 4. Input/output power characteristics of the PA. The X-axis is the absolute value of the input voltage, and the Y-axis is the magnitude of the PA power gain.

The MATLAB based fitting procedure is now used to extract coefficients to feed a PA model and measure the quality of fit. The white-box MATLAB function `poweramp _ helper.m` is used to compute a least-squares fit solution with the aid of the MATLAB backslash operator. A snippet of the code is shown in Figure 5.



```
        switch modType
            case 'memPoly'
                numCols = degLen;
                xTerms = zeros(xLen-memLenM1,memLen*degLen);

                for timeIdx = memLen:xLen
                    % Common input variable entry terms for all rows in
                    % coefficient matrix
                    xTime = x(timeIdx-(0:memLenM1));
                    xTerms(timeIdx-memLenM1,1:memLen) = xTime;
                    for colIdx = 2:degLen
                        colIdxM1 = colIdx-1;
                        startPos = colIdxM1*memLen+1;
                        endPos = startPos+memLen-1;
                        xTerms(timeIdx-memLenM1,startPos:endPos) =      ...
                            xTime.*abs(xTime).^colIdxM1;
                    end
```

Figure 5. Example of the identification procedure for the PA model coefficients implemented with MATLAB.

Half of the dataset is used to create the model, and the full dataset is then used to evaluate the quality of the fit by feeding input data into the PA model and comparing the output data from the model with that recorded from the real PA. Starting with a relatively low order and memory depth equal to K=3, M=3 results in the fit shown in Figure 6.

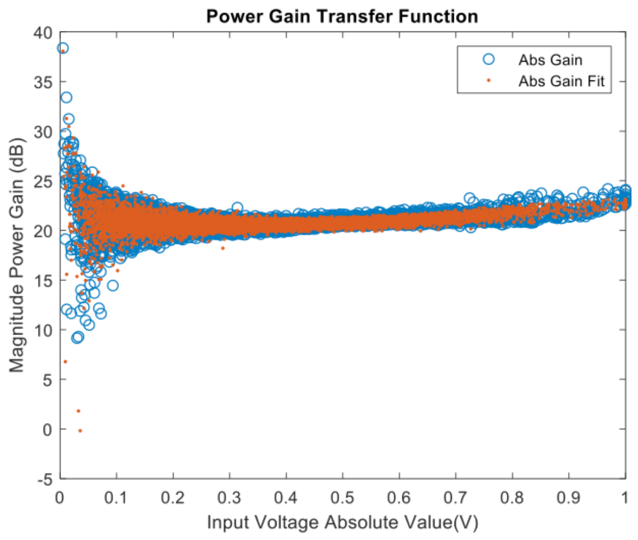


Figure 6. Input/output power characteristics of the PA, showing the measured characteristic (blue), the result of the fitting with memory depth equal to three and polynomial order equal to three (red).

The model points (red) show fewer memory effects compared with the real data (blue) and a little more linearity. The signal standard deviation, which measures the quality of the fit, returns 5.148%. This captures both nonlinear behavior and memory effects.

Increasing the polynomial length to K=4 while keeping memory depth to M=3 improves the PA model, reducing the signal standard deviation to 4.2384% as shown in Figure 7.

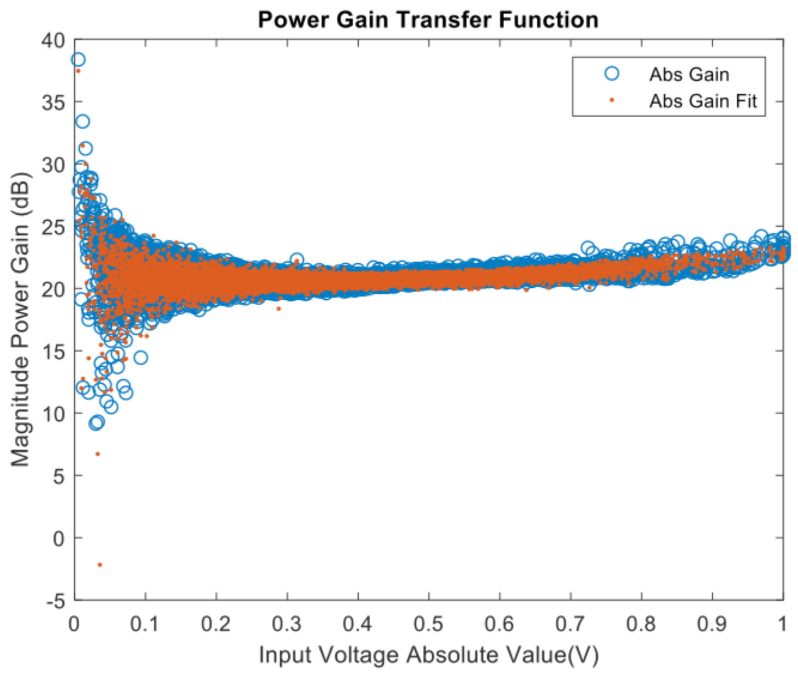


Figure 7. Input/output power characteristics of the PA, showing the measured characteristic (blue), and the result of the fitting with memory depth equal to three and polynomial order equal to four (red).

However, higher values of K and M don't always mean an improved fit. To the contrary, an overfitted model might cause numerical instability. This is especially the case for this data set since it comes from a quite linear PA. For example, increasing the polynomial length and memory depth to 10 causes the signal standard deviation to rise to 4.6972% and a weaker fit to a more complex model.

When considering the quality of fit, it is important to analyze quality metrics in the frequency domain as well as the time domain. Figure 8 shows the spectrum of the predicted signal from the fitted PA model alongside the measured signal when memory depth is M=3 and polynomial length is K=4.

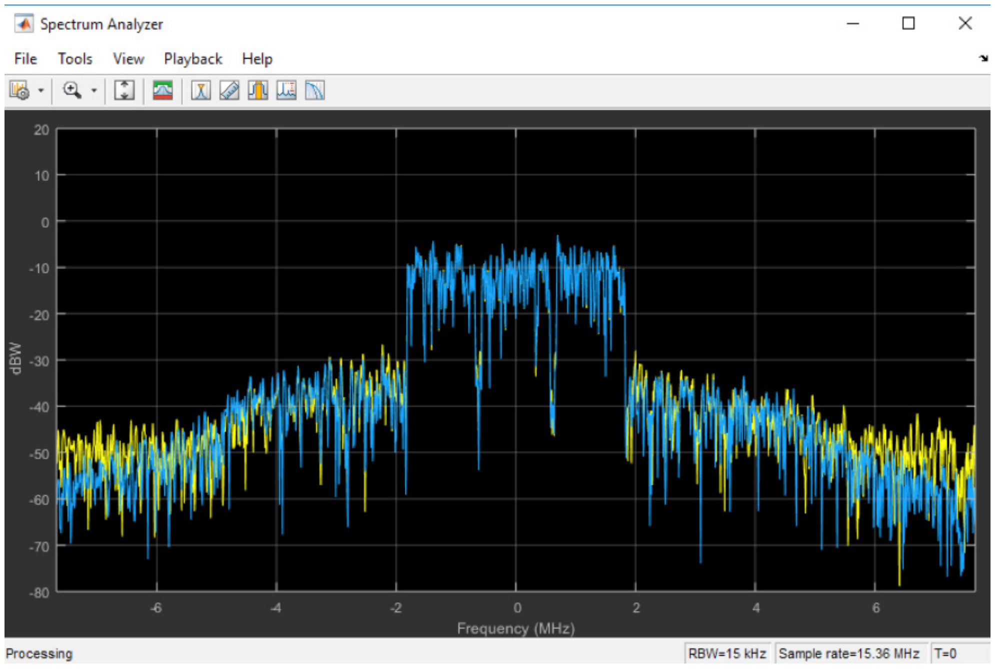


Figure 8. Power spectrum of the output characteristics of the PA, showing the measured characteristic (yellow) and the result of the fitting with memory depth equal to three and polynomial order equal to four (blue).

Here the yellow line represents the frequency spectrum of the captured PA output data, and the blue line is the spectrum of data collected from the PA model output when injected with the captured PA input data. Plotting the difference between these two spectra gives the signal shown in Figure 9.

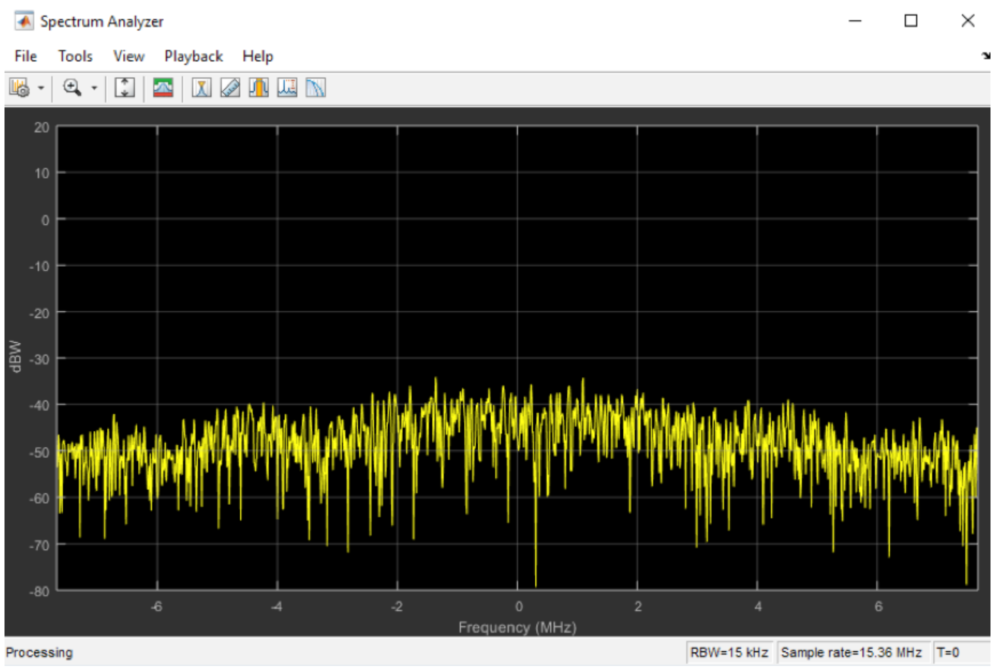


Figure 9. Power spectrum of the error signal representing the difference between the measured and predicted output of the PA using a memory polynomial model with memory depth equal to three and polynomial order equal to four.

As can be seen, the fit is generally good in the frequency domain for this model. However, it has suboptimal performance at the main passband and spectral edges, which could be improved.

To further improve the quality of the fit, a PA model that includes leading and lagging memory cross terms could be used:

$$\begin{aligned} y_{\mathrm{GMP}}(n) = & \sum_{k=0}^{K_a-1} \sum_{l=0}^{L_a-1} a_{kl} x(n-l) \left| x(n-l) \right|^k \\ & + \sum_{k=1}^{K_b} \sum_{l=0}^{L_b-1} \sum_{m=1}^{M_b} b_{klm} x(n-l) \left| x(n-l-m) \right|^k \\ & + \sum_{k=1}^{K_c} \sum_{l=0}^{L_c-1} \sum_{m=1}^{M_c} c_{klm} x(n-l) \left| x(n-l+m) \right|^k \end{aligned}$$

This model and associated fitting procedure are also described in more detail in [1] and provided as white-box functions in the `poweramp _ helper.m` script. It helps to better characterize memory effects while still providing a causal response. Using a memory depth of M=5 and a polynomial length of L=7 produces a model with a much reduced signal standard deviation of 2.3833% (Figure 10). Once again, the yellow line represents the spectrum from the captured PA output data, and the blue line represents the spectrum from the modeled PA when injected with captured PA input data.

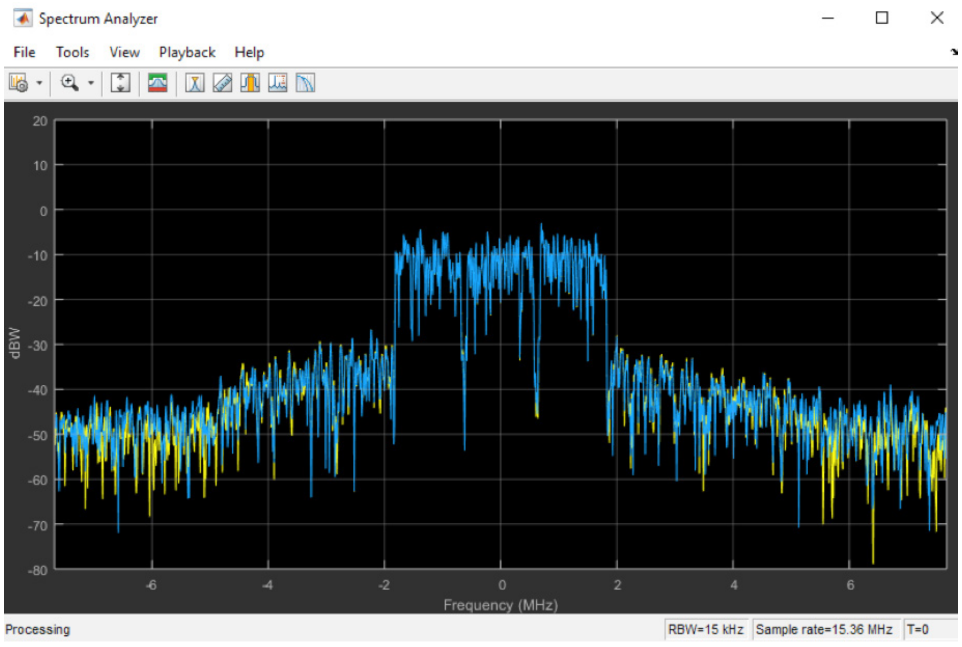


Figure 10. Power spectrum of the output characteristics of the PA, showing the measured characteristic (yellow)and the result of the fitting using a memory polynomial model (yellow) including cross-terms with memory depth equal to five and polynomial order equal to seven.

In the frequency domain, the maximum error in the predicted signal is just -50 dB, as shown in Figure 11.

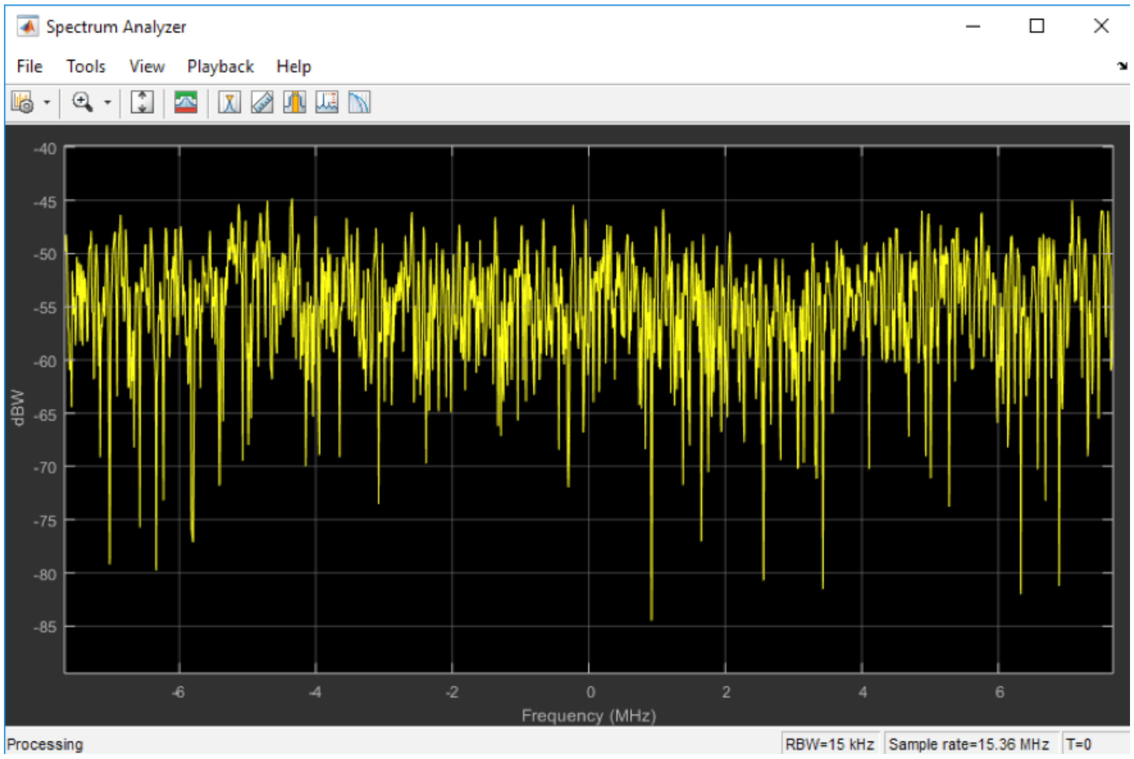


Figure 11. Power spectrum of the error signal representing the difference between the measured and predicted output of the PA using a memory polynomial model including cross-terms with memory depth equal to five and polynomial order equal to seven.

Hence, memory cross terms improve the quality of the PA model at the cost of additional complexity. Since the MATLAB code used here is white-boxed, it can be reviewed and modified. The same framework could also be used to model a completely different PA modeling and fitting procedure such as those based on Doherty models [2,3].

In these examples, the data used was collected by passing a waveform through a PA. The waveform used here must be representative in order to create a good PA model. It is possible to create these waveforms synthetically in MATLAB if required. Specifically, for LTE, WLAN, and 5G systems, signal generators exist within LTE System Toolbox™ and WLAN System Toolbox™ to generate standard compliant waveforms. Instrument Control Toolbox™ can also be used to pass synthetically generated or prerecorded waveforms to and from lab-based test and measurement hardware to exercise the PA for the purposes of automated test.

Designing a DPD Algorithm to Increase Transmitter Linearity

Digital predistortion (DPD) algorithms can be placed before the PA to predistort the signal being transmitted. The DPD characteristic is designed to be the inverse of the PA characteristic and as such creates a behavior that is linear overall because the two effects cancel one another. As a result, understanding and characterizing the behavior of the PA is a crucial part of DPD design.

However, the practical design and implementation of a DPD algorithm is far more complex than inverting the non-linear characteristic (Figure 12). Since the PA is affected by memory, the DPD algorithm must be adaptive and operate in a closed feedback-loop configuration to compensate for PA effects. In addition, DPD algorithms are digital algorithms implemented in baseband, whereas PAs are RF components operating at high frequency. Upconversion and downconversion stages are required to make the two systems operate together. Non-idealities associated with doing this will affect the DPD behavior. Timing imperfections are particularly important as they can make the system unstable. Loading effects of the antenna on the PA must also be considered as well as the DPD strategy being used to adaptively compute the DPD coefficients.

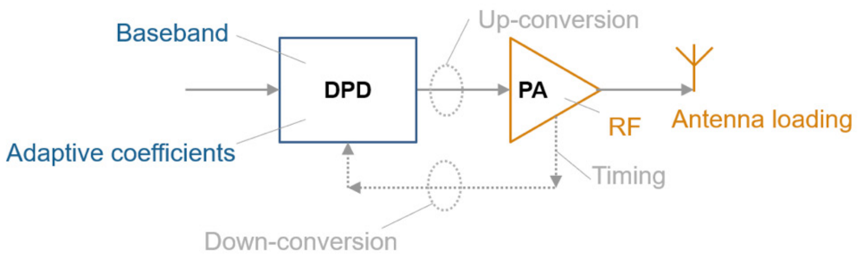


Figure 12. Schematic representation of the closed-loop system including the power amplifier and the digital predistortion algorithm.

Once a suitable PA model has been determined, the DPD algorithm can be developed. When multiple signal domains are being modeled (RF, digital, and analog), Simulink lends itself nicely as the modeling environment. However, some blocks contain MATLAB code for design clarity. A DPD approach based on [4] is modeled. This approach describes an indirect learning architecture implemented by a recursive predictive error method (RPEM). The top-level hierarchy consists of a Simulink block to adaptively calculate the coefficients based on their current values and the current output from the PA and another Simulink block to apply them to the incoming signal before feeding to the PA (Figure 13).

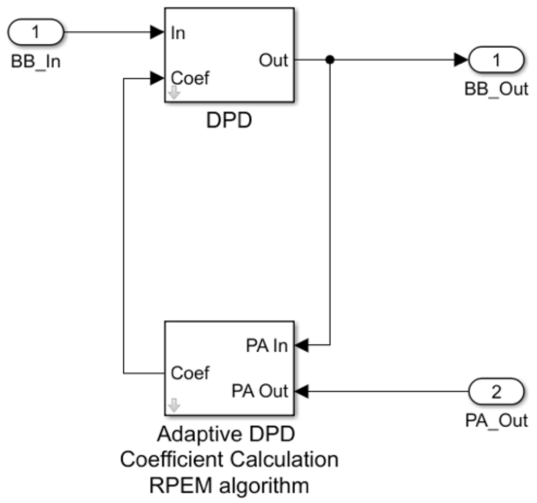


Figure 13. Schematic representation of indirect learning architecture used for DPD implemented by a recursive predictive error method (RPEM). The bottom block calculates the DPD coefficients based on their current values and the current output from the PA; the top block applies the coefficients and predistorts the incoming signal before feeding to the PA.

The coefficient calculation block contains the RPEM core, as shown in Figure 14.

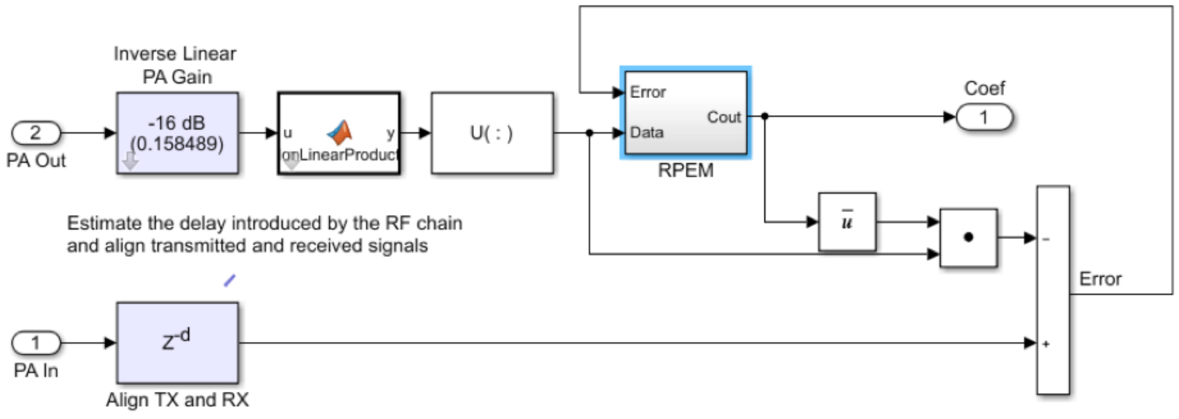
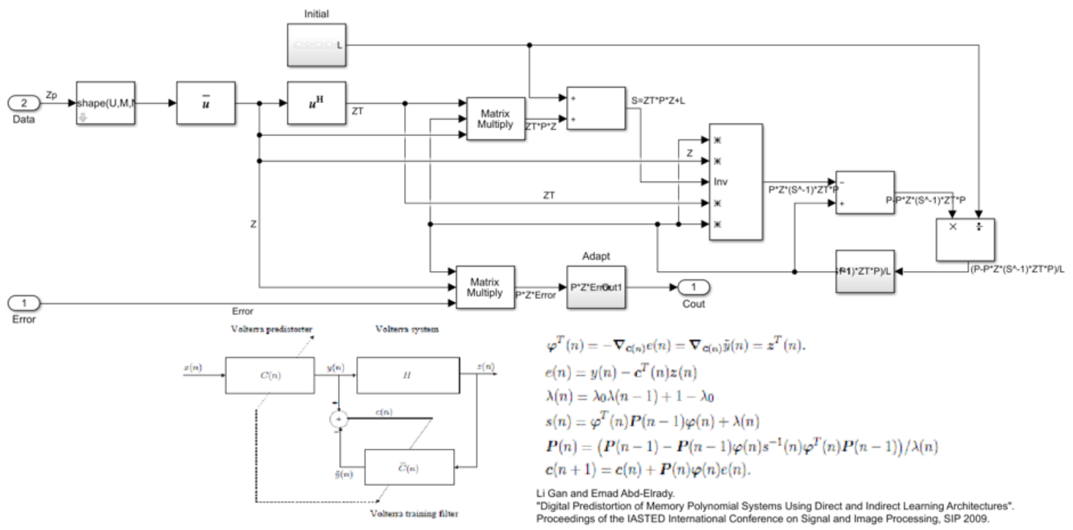


Figure 14. Simulink block diagram for adaptively computing the DPD coefficients.

The RPEM block in Figure 14 computes the coefficients based on [2] as shown in Figure 15.



$$\varphi^T(n) = -\nabla_{c(n)} e(n) = \nabla_{c(n)} \tilde{y}(n) = z^T(n).$$
$$e(n) = y(n) - c^T(n) z(n)$$
$$\lambda(n) = \lambda_0 \lambda(n-1) + 1 - \lambda_0$$
$$s(n) = \varphi^T(n) P(n-1) \varphi(n) + \lambda(n)$$
$$P(n) = \left(P(n-1) - P(n-1)\varphi(n) s^{-1}(n) \varphi^T(n) P(n-1)\right)/\lambda(n)$$
$$c(n+1) = c(n) + P(n)\varphi(n)e(n).$$

Li Gan and Emad Abd-Elrady.
"Digital Predistortion of Memory Polynomial Systems Using Direct and Indirect Learning Architectures".
Proceedings of the IASTED International Conference on Signal and Image Processing, SIP 2009.

Figure 15. Simulink block diagram for adaptively computing the DPD coefficients following the RPEM algorithm. An annotation describes the implemented math.

The RPEM core computes coefficients based on a memory length M and polynomial depth K in a similar way to the strategy used for computing the PA polynomial coefficients. The higher the value of M and K, the higher the level of computation. This generally results in larger resource and latency requirements to implement it in real hardware.

The DPD Simulink block that applies the computed coefficients to predistort the incoming signal is described in Figure 16.

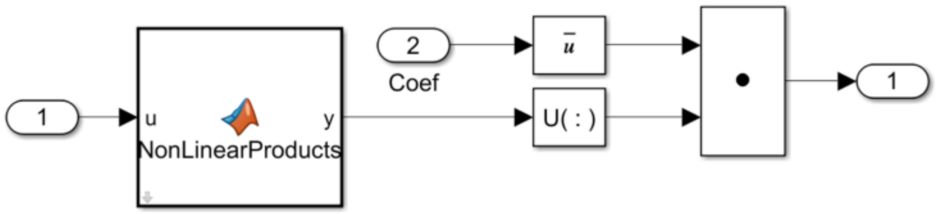


Figure 16. Simulink block diagram of the DPD algorithm used to predistort the PA input signal.
A MATLAB function is used to perform the matrix multiplication between the incoming signal and the DPD coefficients.

Basic Simulink blocks are used in this design. Simulink could be used to create alternative strategies for computing coefficients and applying them to the incoming signal using a similar framework.

System Simulation for Early Verification of DPD Algorithm Performance

The PA and DPD algorithm models should now be integrated and tested together within a system simulation model. For this, a circuit envelope simulation is performed using RF Blockset™ [5]. This allows for a fast RF simulation while accounting for nonlinear effects, impedance mismatches, and timing imperfections [6].

Figure 17 shows a simple behavioral model of a transmitter that is fed with two sine wave tones.

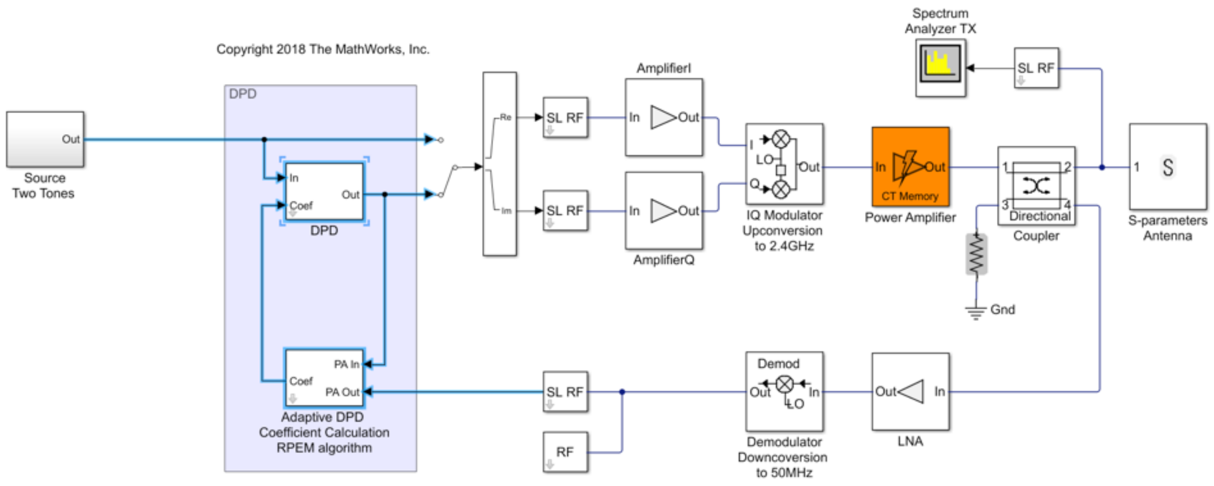
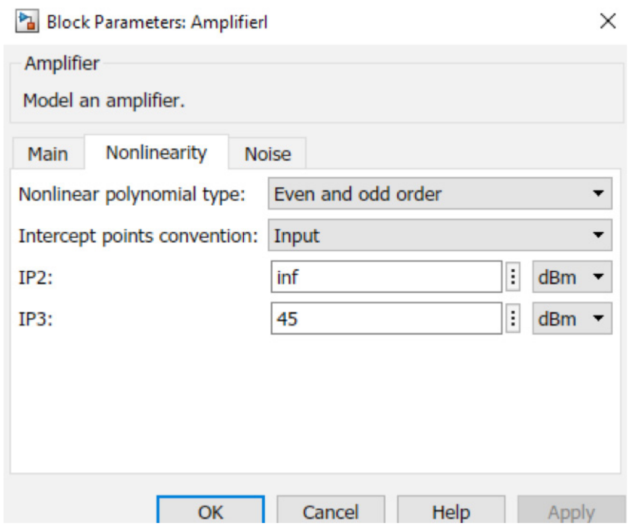


Figure 17. Simulink block diagram of a simple behavioral model of an RF transmitter,
an observer receiver path, and an adaptive DPD algorithm tested with a two-tone signal.

The transmitter model contains different amplification stages, a quadrature modulator, and a PA model (orange) that uses the PA coefficients calculated previously. The PA is loaded by an antenna that is described using S-parameters. A coupler is used to “observe” the output of the PA, and downconvert it to an intermediate frequency. All RF component blocks in the transmitter model come from the RF Blockset library and use circuit envelope simulation. In addition, component models contain RF imperfections such as the nonlinearities defined within the input amplifier as described in Figure 18.

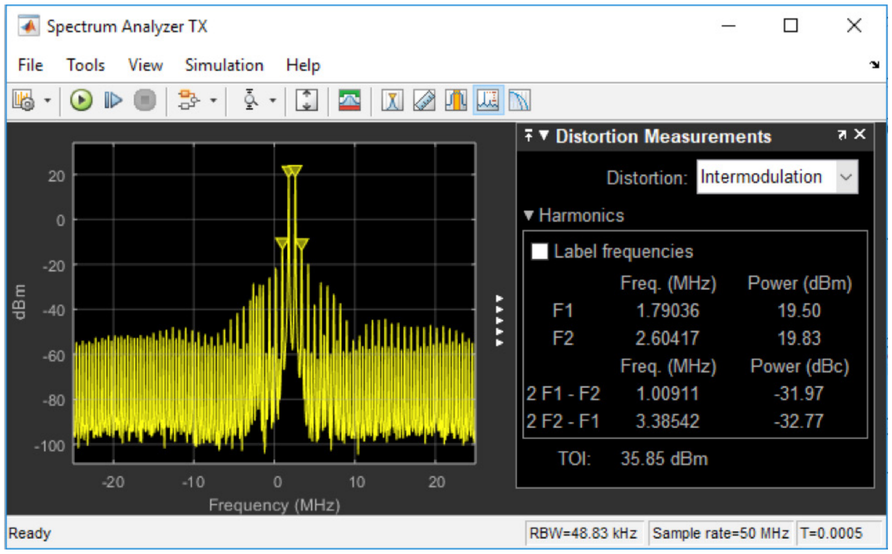


Nonlinear polynomial type: Even and odd order
Intercept points convention: Input
IP2: inf dBm
IP3: 45 dBm

Figure 18. Mask parameters of the nonlinear input amplifier.

The S-parameter antenna model could be captured from a real antenna that has been characterized by a VNA or using data available from the manufacturer. Alternatively, Antenna Toolbox™ could be used within MATLAB to build a representative model of the antenna and extract the S-parameters to be integrated in to this model [7]. Antenna Toolbox provides a full EM solver and uses Method of Moments to analyze the performance of antennas from a prebuilt library as well as custom built antenna structures.

The surrounding testbench injects a two-tone sine wave into the transmitter, and the spectrum analyzer is used to visualize the output from the PA and compute the third-order output IP coefficient (OIP3). Prior to enabling the DPD algorithm, the OIP3 value is around -36 dBm with the third-order tone is -32 dBc, as shown in Figure 19.



Distortion: Intermodulation

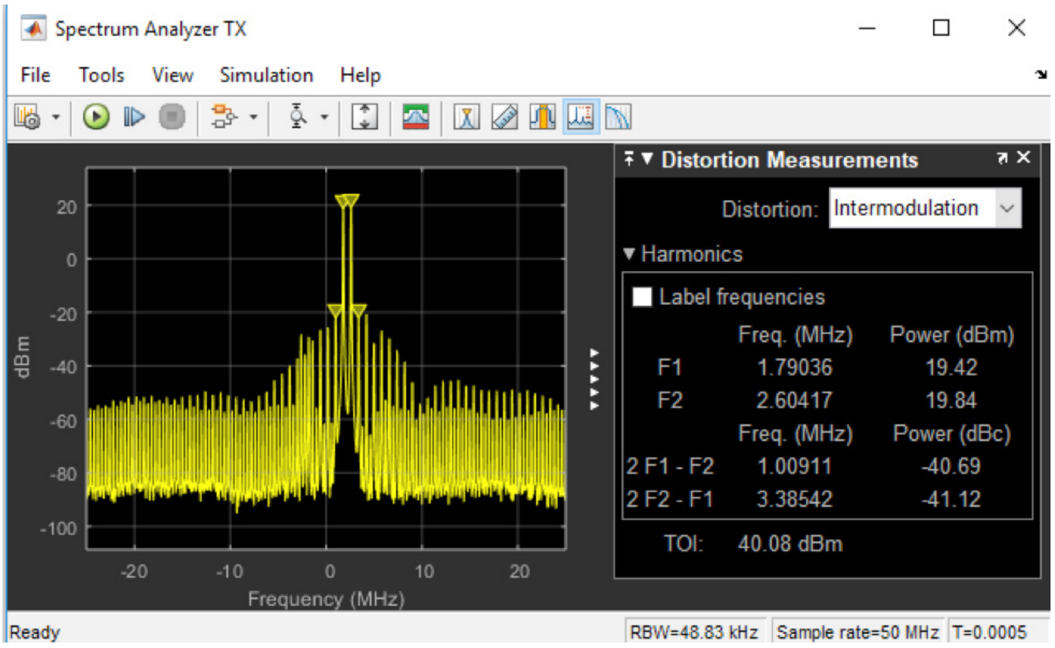
	Freq. (MHz)	Power (dBm)
F1	1.79036	19.50
F2	2.60417	19.83
	Freq. (MHz)	Power (dBc)
2 F1 - F2	1.00911	-31.97
2 F2 - F1	3.38542	-32.77

TOI: 35.85 dBm

RBW=48.83 kHz | Sample rate=50 MHz | T=0.0005

Figure 19. Power spectrum of the output signal of the RF transmitter tested with a two-tone excitation. When the DPD is disabled, the third-order intercept point is approximately 36 dBm.

However, after enabling the DPD algorithm, the OIP3 value drops to -40 dBm with the third-order tone at -41 dBc, as it shown in Figure 20.



Distortion Measurements

Distortion: Intermodulation

Harmonics

Label frequencies

	Freq. (MHz)	Power (dBm)
F1	1.79036	19.42
F2	2.60417	19.84
	Freq. (MHz)	Power (dBc)
2 F1 - F2	1.00911	-40.69
2 F2 - F1	3.38542	-41.12

TOI: 40.08 dBm

Ready | RBW=48.83 kHz | Sample rate=50 MHz | T=0.0005

Figure 20. Power spectrum of the output signal of the RF transmitter tested with a two-tone excitation. When the DPD is enabled, the third-order intercept point is approximately 40 dBm.

In a similar way to the PA modeling exercise, the order and memory length of the DPD algorithm could be modified and the effect on performance of the DPD strategy at the system level observed. This simulation model facilitates fast iterative testing of different DPD configurations to quickly and more easily optimize the algorithm. It also provides an environment for early-stage verification of the DPD approach, perhaps before any physical prototype of the PA even exists.

The model can also be tested using a modulated waveform. In Figure 21, a QAM signal is used and the effects of the DPD algorithm are assessed by measuring the difference in EVM before and after the DPD algorithm is applied.

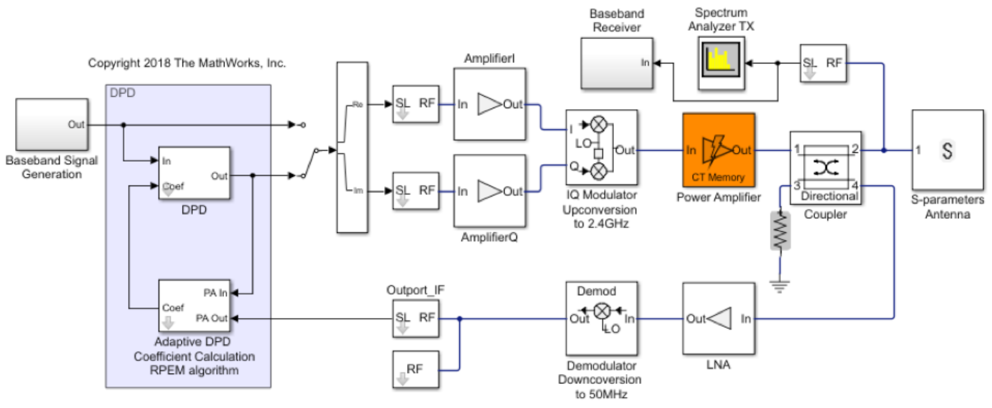
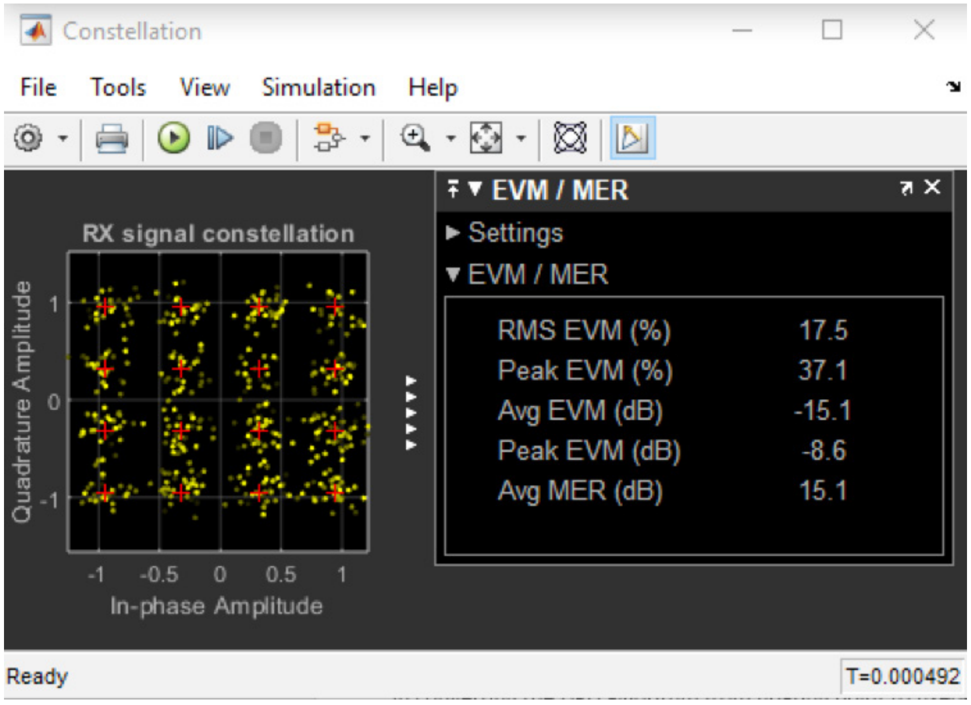

Copyright 2018 The MathWorks, Inc.
Baseband Signal Generation
DPD
In
Coef
Out
Adaptive DPD Coefficient Calculation RPEM algorithm
PA In
PA Out
AmplifierI
AmplifierQ
IQ Modulator Upconversion to 2.4GHz
Baseband Receiver
Spectrum Analyzer TX
CT Memory
Power Amplifier
Directional Coupler
S-parameters Antenna
Outport_IF
Demod
Demodulator Downconversion to 50MHz
LNA
SL
RF

Figure 21. Simulink block diagram of a simple behavioral model of an RF transmitter, an observer receiver path, and an adaptive DPD algorithm tested with QAM signal.

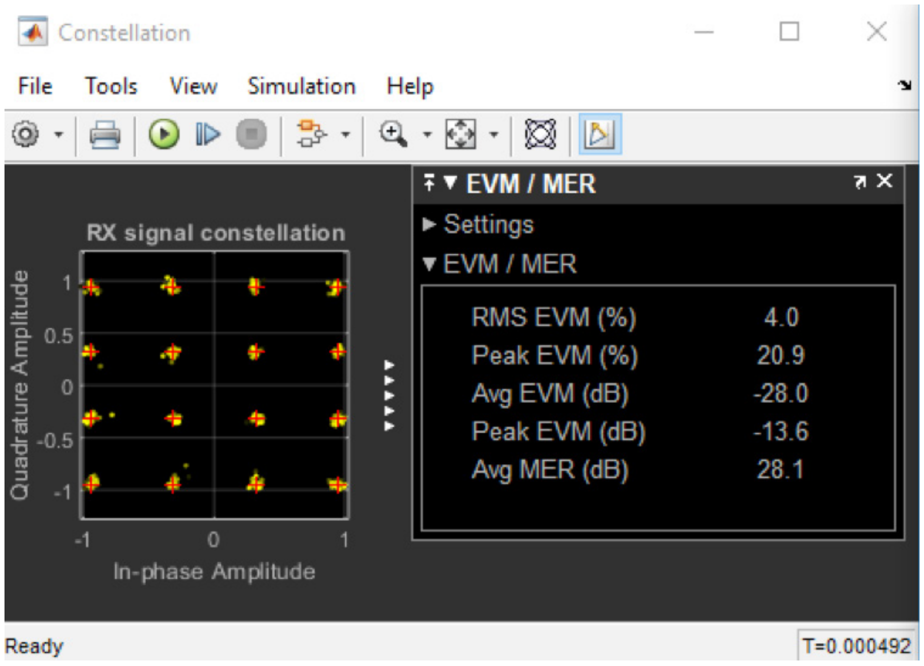
Before the DPD algorithm is enabled, the EVM has an RMS value of 17.5% (Figure 22).



RMS EVM (%)	17.5
Peak EVM (%)	37.1
Avg EVM (dB)	-15.1
Peak EVM (dB)	-8.6
Avg MER (dB)	15.1

Figure 22. Constellation of the output signal of the RF transmitter tested with a QAM signal. When the DPD is disabled, the EVM is 17.5%.

After the DPD algorithm is enabled, this drops to 4.0% (Figure 23).



RMS EVM (%)	4.0
Peak EVM (%)	20.9
Avg EVM (dB)	-28.0
Peak EVM (dB)	-13.6
Avg MER (dB)	28.1

Figure 23. Constellation of the output signal of the RF transmitter tested with a QAM signal. When the DPD is enabled, the EVM is 4%.

The constellation points are more compacted after the DPD algorithm is applied.

Once the DPD algorithm has been developed and tested in simulation, a next step could be to test it with a real PA. This could be done by keeping the DPD algorithm in Simulink and interfacing it with test and measurement hardware to drive a baseband signal through the DPD, out to the real PA and then back in to Simulink for analysis and DPD adaptation. This can be achieved using Instrument Control Toolbox [8]. This approach was employed as a part of the workflow for DPD design at Huawei using Simulink. In [9] the authors describe the benefits of using MATLAB and Simulink for DPD design, which enabled them to:

- Perform closed-loop simulation of designs containing both analog/RF and digital components, such as digital predistortion (DPD) for RF power amplifiers
- Quickly develop a flexible, high-performance hardware development platform at the beginning of the R&D process using a seamless interface to RF instruments
- Quickly build an automatic verification platform between software and hardware
- Use a single platform for hardware development, including reference models, fixed-point conversion, and automatic C and RTL code generation
- Reuse models for bit-true verification of floating-point, fixed-point, and RTL code

Implementation and Verification of the DPD Algorithm Using Hardware

After experimenting with different DPD strategies in simulation, the next step is to prototype the algorithm outside of Simulink. As transmitter bandwidth increases, the DPD algorithm must operate at a faster rate to achieve real-time performance and meet latency requirements. Because of this, FPGAs are common prototyping architectures for DPD algorithm testing and verification. Algorithms are generally deployed to FPGA as fixed-point, so the first step to performing rapid prototyping is to convert the DPD algorithm from floating point to fixed point and to analyze the impact of fixed-point precision on the performance of the algorithm.

Fixed-Point Designer™ [10] provides a guided workflow to assist in converting a model from floating point to fixed point. A bit-true simulation model can be created, and the performance tested in simulation, in advance of deployment to FPGA. This is particularly significant in models with feedback configurations, such as the DPD model under discussion here, since using finite precision arithmetic can affect stability and locking time. This is notably difficult to detect and overcome when the algorithm is on FPGA. Simulation can help diagnose these issues more easily and earlier in the design cycle.

To convert the DPD algorithm modeled within the system simulation in the previous section, a data type conversion block is placed in advance of the DPD algorithm (Figure 24).

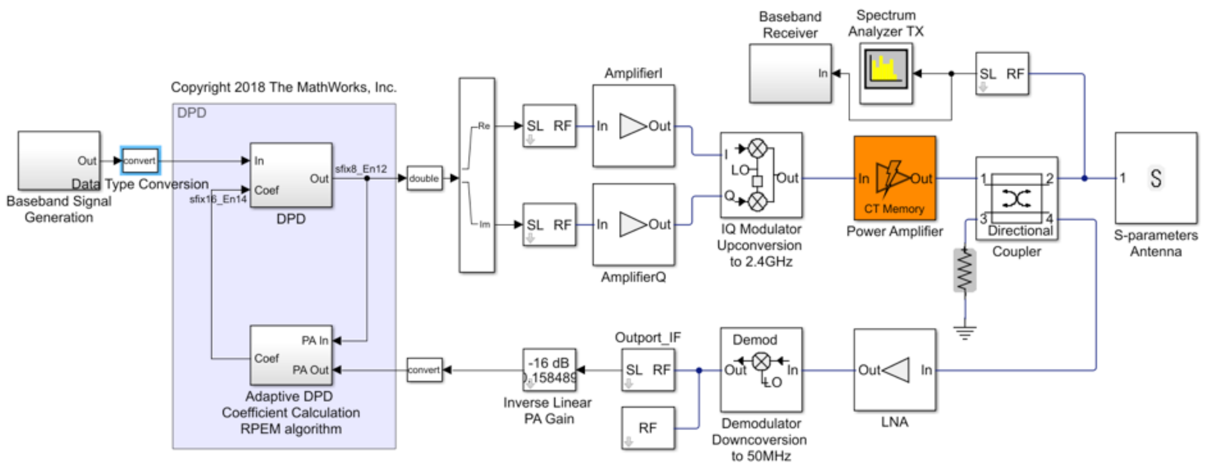
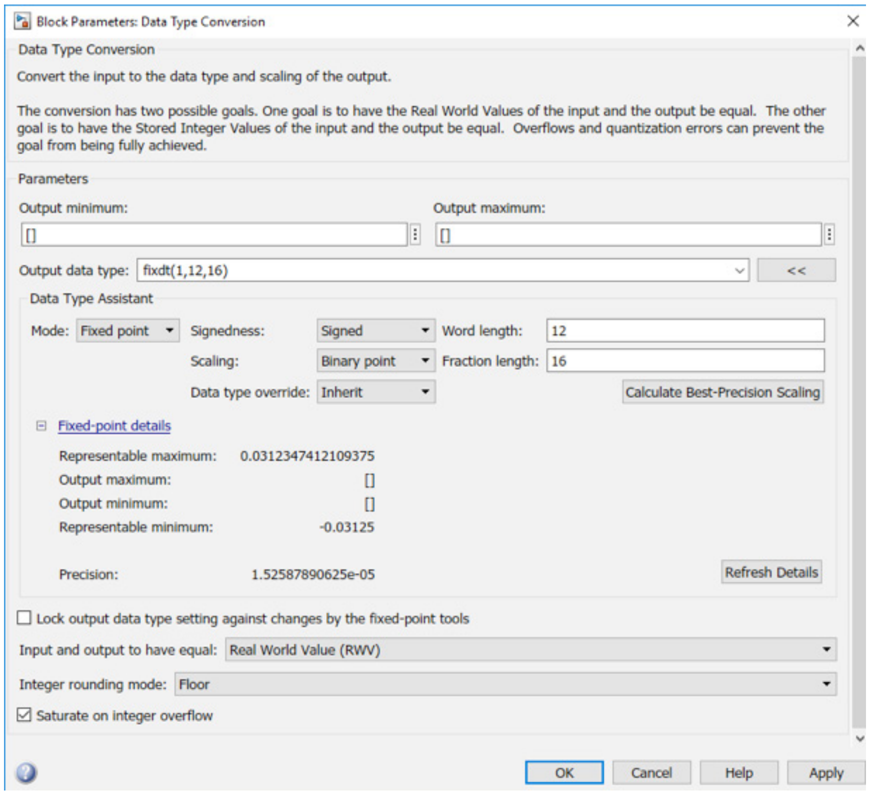


Figure 24. Simulink block diagram of a simple behavioral model of an RF transmitter, an observer receiver path, and an adaptive DPD algorithm using finite precision arithmetic.

The Data Type Conversion block parameters are configured to convert the input signal from floating point to a 12-bit signed signal with a fractional length of 16 bits (Figure 25).


Block Parameters: Data Type Conversion

Data Type Conversion

Convert the input to the data type and scaling of the output.

The conversion has two possible goals. One goal is to have the Real World Values of the input and the output be equal. The other goal is to have the Stored Integer Values of the input and the output be equal. Overflows and quantization errors can prevent the goal from being fully achieved.

Parameters

Output minimum: []

Output maximum: []

Output data type: fixdt(1,12,16)

Data Type Assistant

Mode: Fixed point

Signedness: Signed

Word length: 12

Scaling: Binary point

Fraction length: 16

Data type override: Inherit

Calculate Best-Precision Scaling

Fixed-point details

Representable maximum:	0.0312347412109375
Output maximum:	[]
Output minimum:	[]
Representable minimum:	-0.03125
Precision:	1.52587890625e-05

Refresh Details

☐ Lock output data type setting against changes by the fixed-point tools

Input and output to have equal: Real World Value (RWV)

Integer rounding mode: Floor

☑ Saturate on integer overflow

Figure 25. Parameters of the Data Type Conversion block to convert the input signal from double floating-point precision to fixed-point finite precision arithmetic.

As shown in Figure 25, this implies a representable range of approximately ±0.03 and signals can be represented to the nearest 1.5e-5. The floor operator will be used to achieve the rounding required to do this. Floor is a good operator for FPGA implementation since it requires only simple logic to implement. In addition, signals exceeding the limits of the maximum and minimum values will saturate rather than wrap. This is a more complex behavior to implement but is safer when there are multiple feedback loops. These two strategies to rounding and saturation will be adopted throughout the DPD design.

The recursive predictive error method uses the transmitter input and PA output to compute the coefficients. The error between the PA input and the predistorted PA output is closed in a feedback loop to compute future DPD coefficients, as shown in Figure 26.

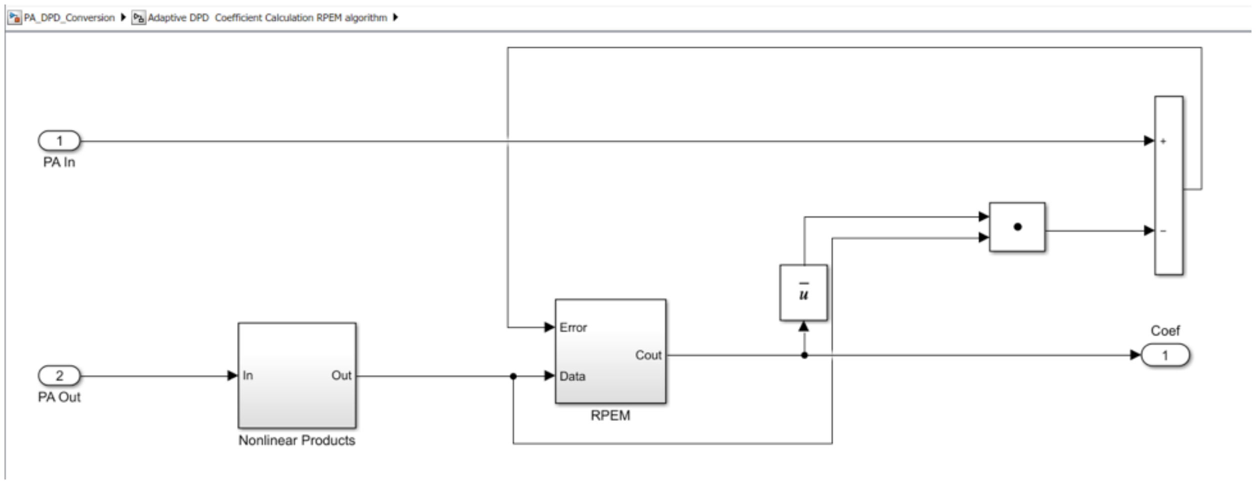
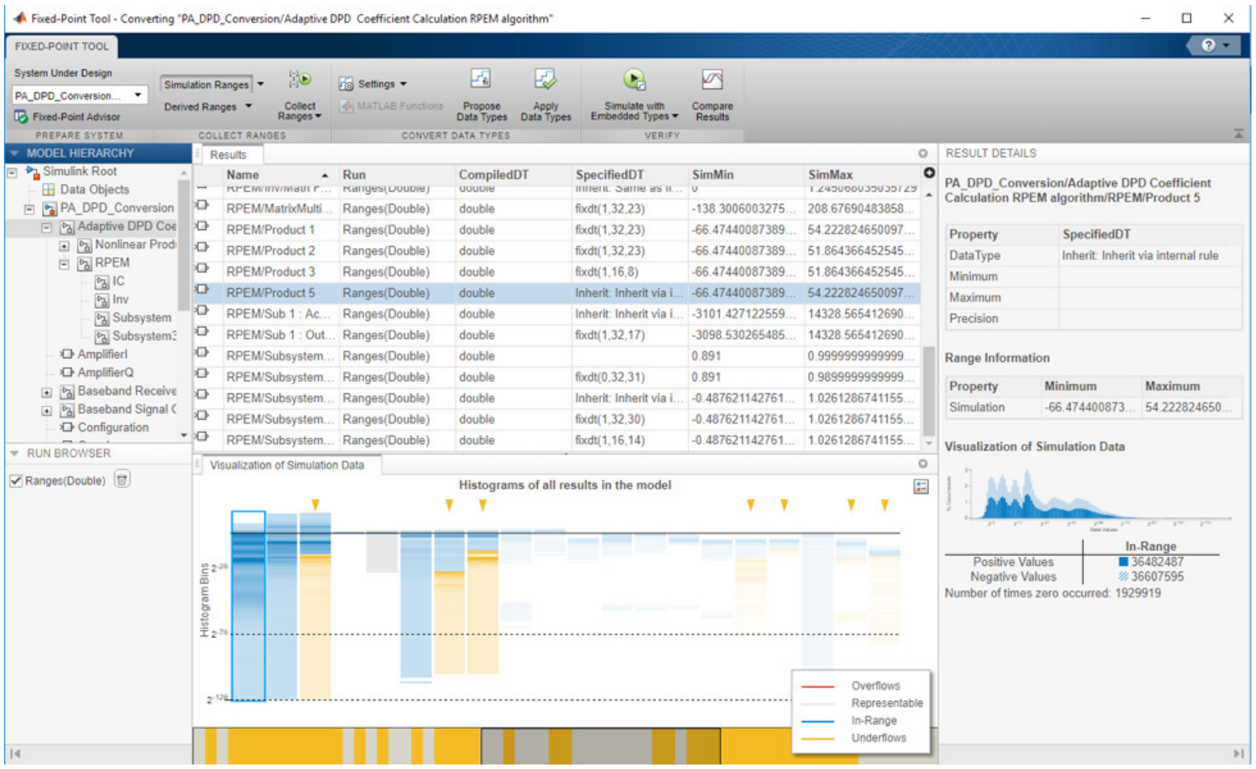

PA_DPD_Conversion ▸ Adaptive DPD Coefficient Calculation RPEM algorithm ▸
1
PA In
2
PA Out
In
Out
Nonlinear Products
Error
Data
Cout
RPEM
ū
Coef
1
+
-

Figure 26. Fixed-point implementation of the RPEM algorithm for computing the DPD coefficients using finite precision arithmetic.

The DPD coefficients are computed using multiple matrix multiplications, inversions, and feedback loops, so converting the algorithm to fixed point isn't straightforward. Specifically, it isn't clear what word length and fractional lengths are required for each of these internal signals to allow the algorithm to work correctly and efficiently without using excessive resources. The Fixed-Point Tool in Simulink will be used to help in the conversion of the coefficient estimation subsystem illustrated above.

Initially, all signal lines are set to override as double. The Fixed-Point Tool then runs the whole model and collects and analyzes data from the part of the system to be converted at each signal line, as shown in the report in Figure 27.



Name	Run	CompiledDT	SpecifiedDT	SimMin	SimMax
RPEM/MatrixMulti...	Ranges(Double)	double	fixdt(1,32,23)	-138.3006003275...	208.67690483858...
RPEM/Product 1	Ranges(Double)	double	fixdt(1,32,23)	-66.47440087389...	54.222824650097...
RPEM/Product 2	Ranges(Double)	double	fixdt(1,32,23)	-66.47440087389...	51.864366452545...
RPEM/Product 3	Ranges(Double)	double	fixdt(1,16,8)	-66.47440087389...	51.864366452545...
RPEM/Product 5	Ranges(Double)	double	Inherit: Inherit via i...	-66.47440087389...	54.222824650097...
RPEM/Sub 1 : Ac...	Ranges(Double)	double	Inherit: Inherit via i...	-3101.427122559...	14328.565412690...
RPEM/Sub 1 : Out...	Ranges(Double)	double	fixdt(1,32,17)	-3098.530265485...	14328.565412690...
RPEM/Subsystem...	Ranges(Double)	double		0.891	0.9999999999999...
RPEM/Subsystem...	Ranges(Double)	double	fixdt(0,32,31)	0.891	0.9899999999999...
RPEM/Subsystem...	Ranges(Double)	double	Inherit: Inherit via i...	-0.487621142761...	1.0261286741155...
RPEM/Subsystem...	Ranges(Double)	double	fixdt(1,32,30)	-0.487621142761...	1.0261286741155...
RPEM/Subsystem...	Ranges(Double)	double	fixdt(1,16,14)	-0.487621142761...	1.0261286741155...

PA_DPD_Conversion/Adaptive DPD Coefficient Calculation RPEM algorithm/RPEM/Product 5

Property	SpecifiedDT
DataType	Inherit: Inherit via internal rule
Minimum	
Maximum	
Precision	

Range Information

Property	Minimum	Maximum
Simulation	-66.474400873...	54.222824650...

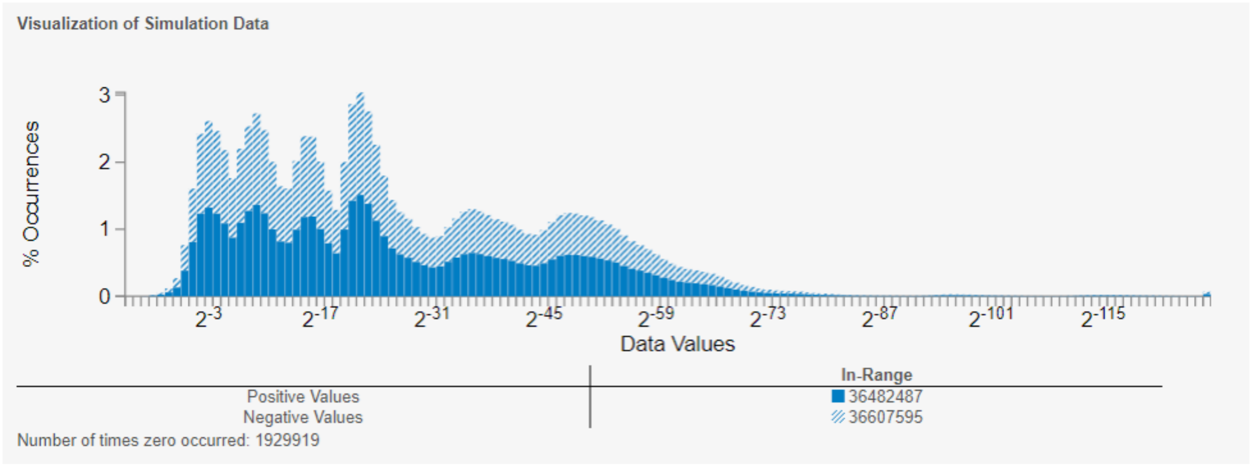
Visualization of Simulation Data

	In-Range
Positive Values	36482487
Negative Values	36607595

Number of times zero occurred: 1929919

Figure 27. Fixed-Point Tool report of the simulation dynamic range of all the signals in the DPD subsystem.

As can be seen for the output of the block "Product 5," the range of outputs produced during the simulation was around -66.5 to 54. A histogram (Figure 28) of the data values that were represented during the simulation run is also shown, to give an idea of the dynamic range required by the signal line.



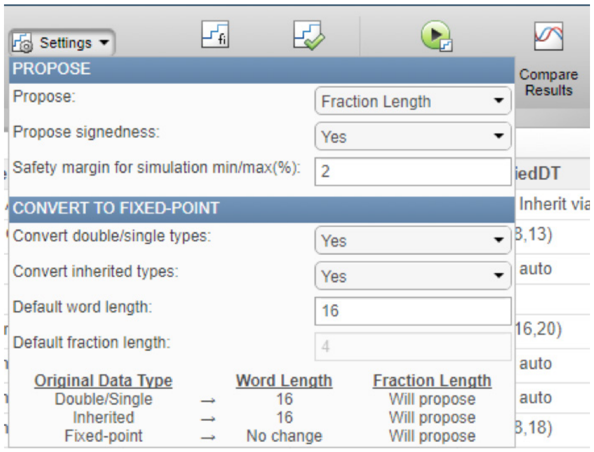
	In-Range
Positive Values	🟦 36482487
Negative Values	▨ 36607595

Number of times zero occurred: 1929919

Figure 28. Histogram of the simulation dynamic range of one of the internal signals used to compute the DPD coefficients.

Here, the signal dynamic range is large, and the output is both positive and negative. This analysis can help to inform decisions on appropriate data type choice.

Given a fixed word or fractional length, the tool can recommend a suitable fractional or word length for each of the signal lines in the system under design. For example, you can configure word length to 16 bits and request an optimal fractional length to be computed, as shown in Figure 29.



PROPOSE

Propose: Fraction Length

Propose signedness: Yes

Safety margin for simulation min/max(%): 2

CONVERT TO FIXED-POINT

Convert double/single types: Yes

Convert inherited types: Yes

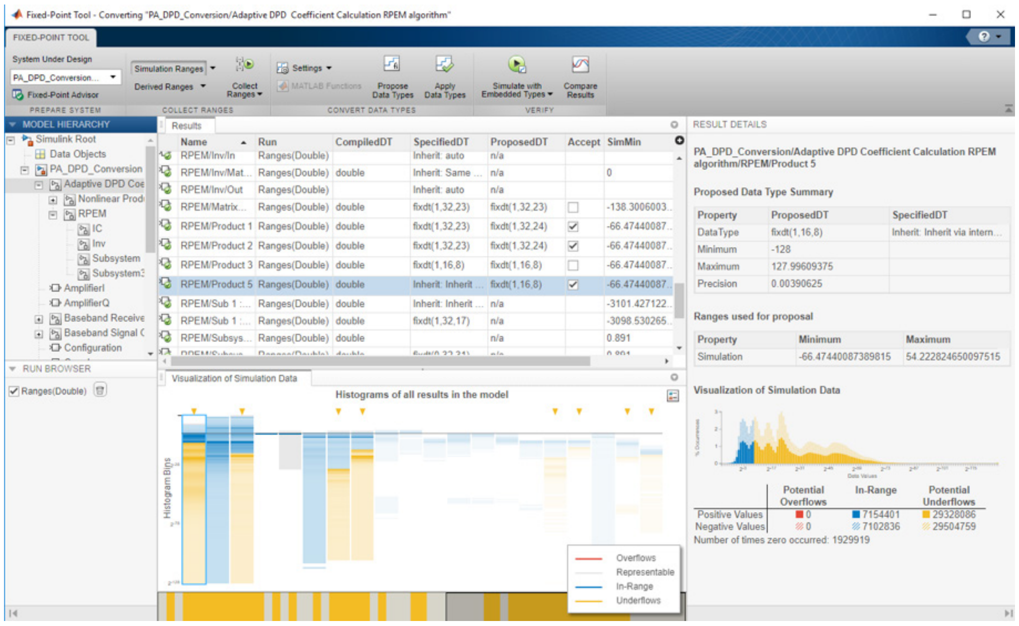
Default word length: 16

Default fraction length: 4

Original Data Type		Word Length	Fraction Length
Double/Single	→	16	Will propose
Inherited	→	16	Will propose
Fixed-point	→	No change	Will propose

Figure 29. Automatically calculating fraction length in Fixed-Point Tool.

Requesting a proposal under this configuration results in the proposed data types shown in Figure 30.



Name	Run	CompiledDT	SpecifiedDT	ProposedDT	Accept	SimMin
RPEM/Inv/In	Ranges(Double)		Inherit: auto	n/a		
RPEM/Inv/Mat...	Ranges(Double)	double	Inherit: Same ...	n/a		0
RPEM/Inv/Out	Ranges(Double)		Inherit: auto	n/a		
RPEM/Matrix...	Ranges(Double)	double	fixdt(1,32,23)	fixdt(1,32,23)	☐	-138.3006003...
RPEM/Product 1	Ranges(Double)	double	fixdt(1,32,23)	fixdt(1,32,24)	✓	-66.47440087...
RPEM/Product 2	Ranges(Double)	double	fixdt(1,32,23)	fixdt(1,32,24)	✓	-66.47440087...
RPEM/Product 3	Ranges(Double)	double	fixdt(1,16,8)	fixdt(1,16,8)	☐	-66.47440087...
RPEM/Product 5	Ranges(Double)	double	Inherit: Inherit ...	fixdt(1,16,8)	✓	-66.47440087...
RPEM/Sub 1 :...	Ranges(Double)	double	Inherit: Inherit ...	n/a		-3101.427122...
RPEM/Sub 1 :...	Ranges(Double)	double	fixdt(1,32,17)	n/a		-3098.530265...
RPEM/Subsys...	Ranges(Double)	double		n/a		0.891

PA_DPD_Conversion/Adaptive DPD Coefficient Calculation RPEM algorithm/RPEM/Product 5

Proposed Data Type Summary

Property	ProposedDT	SpecifiedDT
DataType	fixdt(1,16,8)	Inherit: Inherit via intern...
Minimum	-128	
Maximum	127.99609375	
Precision	0.00390625	

Ranges used for proposal

Property	Minimum	Maximum
Simulation	-66.47440087389815	54.222824650097515

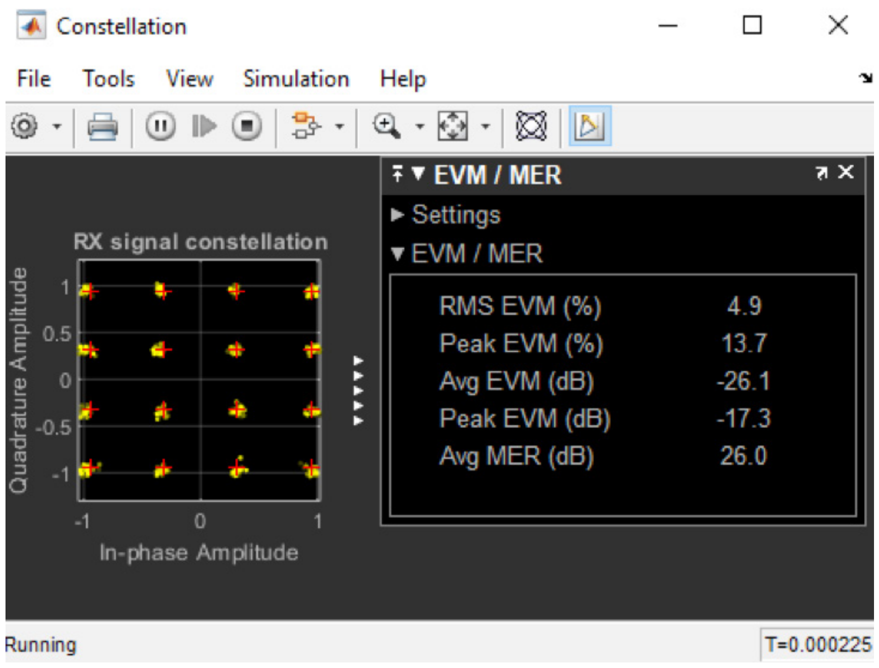
Visualization of Simulation Data

	Potential Overflows	In-Range	Potential Underflows
Positive Values	🟥 0	🟦 7154401	🟨 29328086
Negative Values	▨ 0	▨ 7102836	▨ 29504759

Number of times zero occurred: 1929919

Figure 30. A Fixed-Point Tool report after automatic calculation of the signal fractional length for a specified word length.

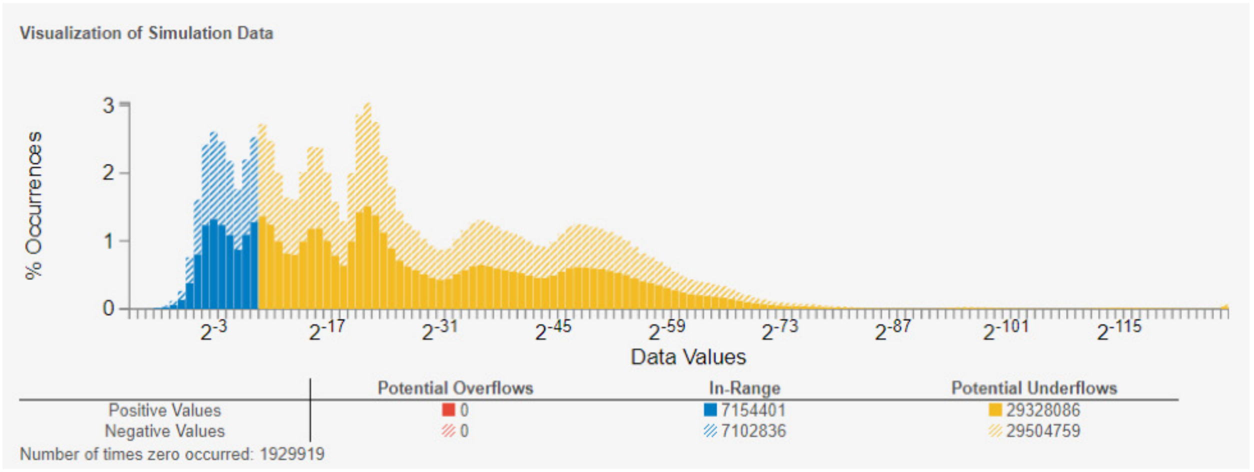
The Fixed-Point Tool proposes that the output from the "Product 5" block should be signed with 8 fractional bits. This results in a representable range of ±128 with a precision of around 0.004. All data types proposed can be accepted and then the simulation rerun with these new types. In this case, the EVM now increases to 4.9% (Figure 31) from the 4% that was achieved when using double precision.



RMS EVM (%)	4.9
Peak EVM (%)	13.7
Avg EVM (dB)	-26.1
Peak EVM (dB)	-17.3
Avg MER (dB)	26.0

Figure 31. Constellation of the output signal of the RF transmitter tested with a QAM signal using finite precision arithmetic for the DPD algorithm.

In the histogram of simulation ranges captured for the "Product 5" block, a number of underflows occur, as shown in Figure 32.

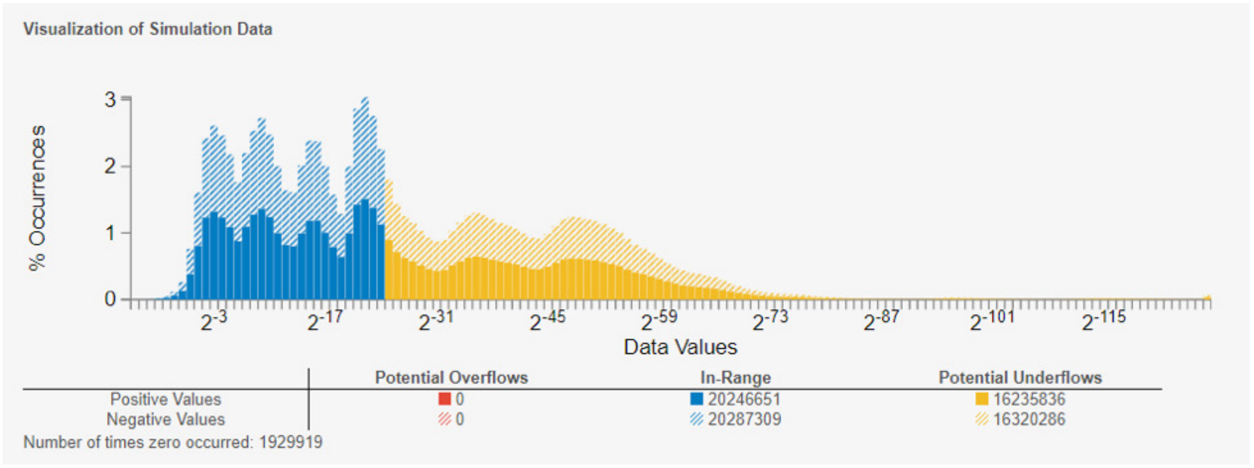


	Potential Overflows	In-Range	Potential Underflows
Positive Values	🟥 0	🟦 7154401	🟨 29328086
Negative Values	▨ 0	▨ 7102836	▨ 29504759

Number of times zero occurred: 1929919

Figure 32. Histogram of the simulation dynamic range of one of the internal signals used to compute the DPD coefficients, showing the dynamic range covered by a 16-bit word length (blue) and the data affected by underflows occurring due to lack of sufficient precision (yellow).

The fixed-point configuration can be easily modified within the Fixed-Point Tool and the system respecified. The word length is now increased to 32 bits, which provides more coverage of the simulation data. The fractional length of the output of the “Product 5” block proposed by the Fixed-Point Tool is now 24, and the precision now drops to almost 6e-8. The updated coverage on the histogram of logged data points is shown in Figure 33.



	Potential Overflows	In-Range	Potential Underflows
Positive Values	🟥 0	🟦 20246651	🟨 16235836
Negative Values	▨ 0	▨ 20287309	▨ 16320286

Number of times zero occurred: 1929919

Figure 33. Histogram of the simulation dynamic range of one of the internal signals used to compute the DPD coefficients, showing the dynamic range covered by a 32-bit word length (blue)and the data affected by underflows occurring due to lack of sufficient precision (yellow).

In addition, the EVM also reduces to 4.3%. As well as designing signal line data types for the entire design, individual signal lines can be isolated and analyzed. Through this process, the whole model can be converted by selectively focusing on critical blocks and estimating the impact on system performance. By selecting a larger word length, even if just for one of the internal blocks, the EVM can reduce and the algorithm converge more rapidly. A decision must then be made as to whether the additional cost is worth the improved performance.

The Simulink subsystem to compute the DPD coefficients has now been converted to a fixed-point equivalent. The Simulink subsystem to apply the DPD coefficients to the incoming signal must also be converted since this will also be targeted to the FPGA. For this, an alternative, new and fully automated approach to the assisted conversion workflow described above will be used. Here, the tool will suggest an optimum word length and fractional length for each signal line by using an optimization approach to meet an acceptable conversion error defined by the user.

The workflow is MATLAB based but can drive a Simulink model. First, an acceptable range of word lengths to be used in the system under design must be defined as well as the maximum relative and absolute errors that are acceptable at the output of the system. These constraints feed the optimization algorithm, which suggests appropriate data types. For the DPD subsystem, acceptable word lengths are specified to be between 8 and 32:

```
%% make the options object

options = fxpOptimizationOptions;

options.AllowableWordLengths = 8:32;
```

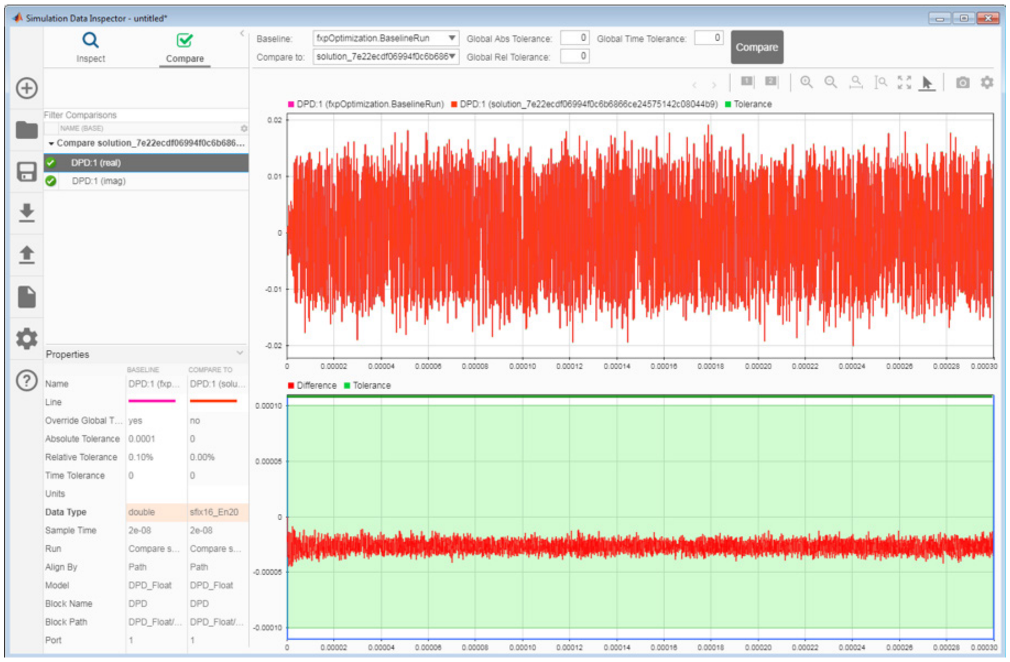
Acceptable relative and absolute tolerances at the subsystem output are specified as 1e-3 and 1e-4, respectively:

```
%% set the behavioral tolerances to be used when comparing
% fixed-point designs to the original baseline model
addTolerance(options, sud, 1, 'RelTol', 1e-3);
addTolerance(options, sud, 1, 'AbsTol', 1e-4);
```

The optimization is then started:

```
%% set max time to 30 minutes
options.MaxTime = 60 * 30;
options.Verbosity = 'High';
%% run optimization
result = fxpopt(model, sud, options);
%% explore the best fixed-point design
bestSolution = explore(result);
```

Successive iterations allow the proposal of fractional and word lengths for each signal line until the desired absolute and relative tolerances are achieved. For the DPD subsystem, this takes approximately 10 minutes. After conversion, the Simulation Data Inspector shows the simulation outputs from the DPD block before and after fixed-point data types are applied as well as the difference between the two signals (Figure 34).


Simulation Data Inspector - untitled*
Inspect
Compare
Compare
Filter Comparisons
NAME (BASE)
Compare solution_7e22ecdf06994f0c6b686...
DPD:1 (real)
DPD:1 (imag)
DPD:1 (fxpOptimization.BaselineRun)
DPD:1 (solution_7e22ecdf06994f0c6b6866ce24575142c08044b9)
Tolerance
Difference
Tolerance

Baseline: fxpOptimization.BaselineRun
Compare to: solution_7e22ecdf06994f0c6b686
Global Abs Tolerance: 0
Global Rel Tolerance: 0
Global Time Tolerance: 0

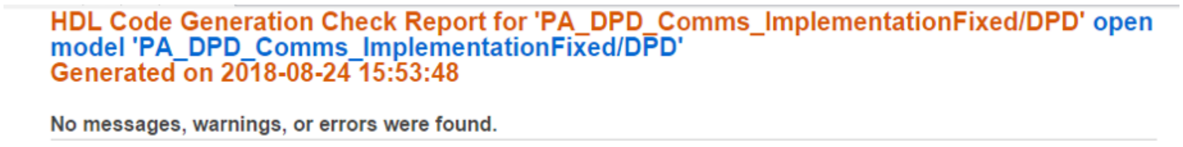
Properties

	BASELINE	COMPARE TO
Name	DPD:1 (fxp...	DPD:1 (solu...
Line		
Override Global T...	yes	no
Absolute Tolerance	0.0001	0
Relative Tolerance	0.10%	0.00%
Time Tolerance	0	0
Units		
Data Type	double	sfix16_En20
Sample Time	2e-08	2e-08
Run	Compare s...	Compare s...
Align By	Path	Path
Model	DPD_Float	DPD_Float
Block Name	DPD	DPD
Block Path	DPD_Float/...	DPD_Float/...
Port	1	1

Figure 34. Plot of the error signal representing the difference between the simulation results of the output signal of the DPD block computed with floating-point and fixed-point accuracy.

Figure 34 shows that the differences between the final fixed- and floating-point results are well within the acceptable tolerances. Following this, Fixed-Point Tool could be used for fine-tuning of the conversion.

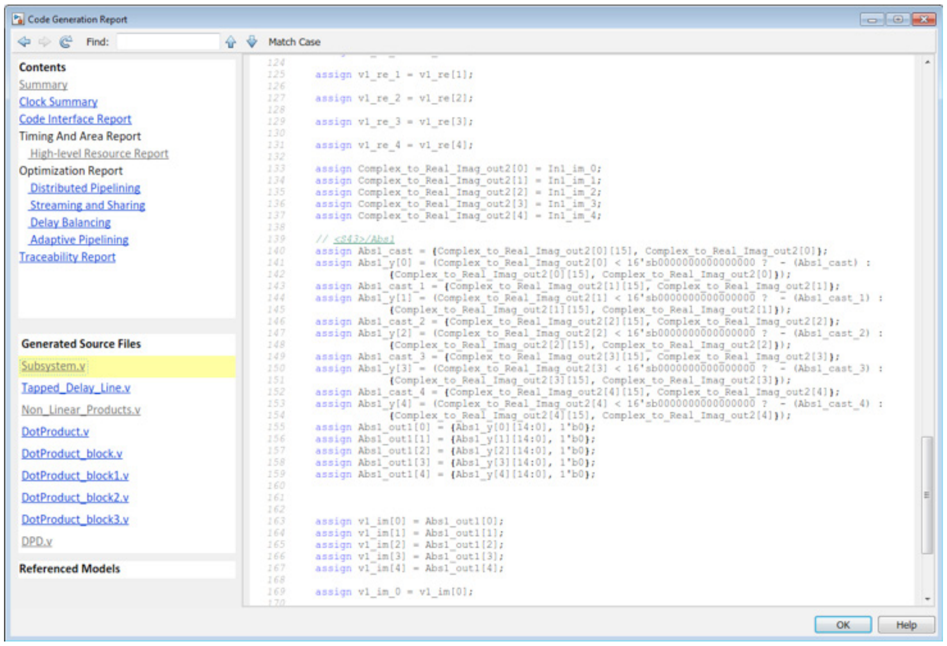
Both the DPD coefficient calculation subsystem and the DPD subsystem to apply these coefficients to the incoming signal have now been converted to fixed-point, and it has been verified in simulation that these subsystems still operate as expected within a system-level simulation. With HDL Coder™, it is now possible to generate readable and synthesizable VHDL® or Verilog® code that could then be deployed to FPGA. The example below will do this for the DPD subsystem. The first step is to ensure that the Simulink subsystem being converted only contains blocks that support HDL code generation. This can be achieved by running the HDL compatibility checker, as shown in Figure 35.


HDL Code Generation Check Report for 'PA_DPD_Comms_ImplementationFixed/DPD' open model 'PA_DPD_Comms_ImplementationFixed/DPD'
Generated on 2018-08-24 15:53:48

No messages, warnings, or errors were found.

Figure 35. Report of the HDL compatibility checker for the DPD block.

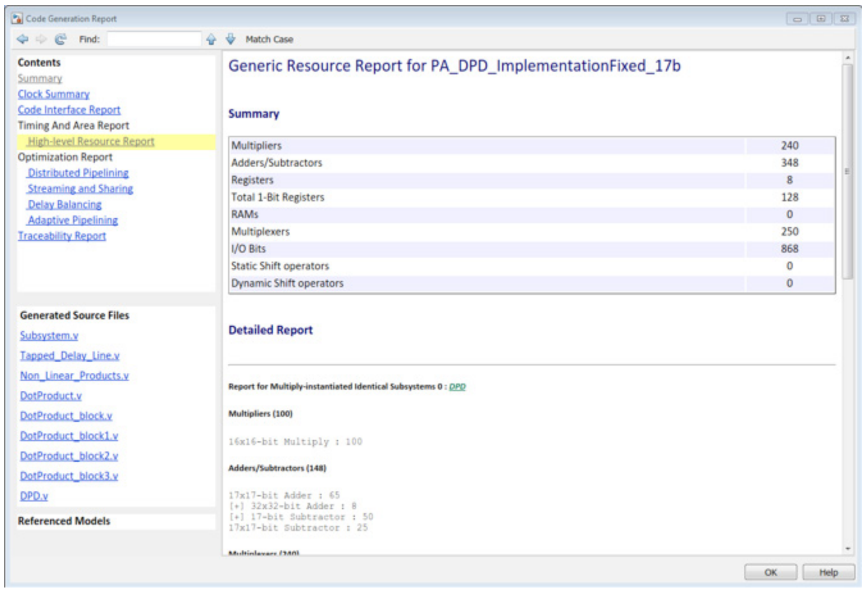
HDL code can then be generated by right-clicking on the subsystem and selecting "Generate HDL Code for Subsystem." Generated code is displayed in an HTML code generation report (Figure 36).



```
  assign v1_re_1 = v1_re[1];

  assign v1_re_2 = v1_re[2];

  assign v1_re_3 = v1_re[3];

  assign v1_re_4 = v1_re[4];

  assign Complex_to_Real_Imag_out2[0] = In1_im_0;
  assign Complex_to_Real_Imag_out2[1] = In1_im_1;
  assign Complex_to_Real_Imag_out2[2] = In1_im_2;
  assign Complex_to_Real_Imag_out2[3] = In1_im_3;
  assign Complex_to_Real_Imag_out2[4] = In1_im_4;

  // <S43>/Abs1
  assign Abs1_cast = {Complex_to_Real_Imag_out2[0][15], Complex_to_Real_Imag_out2[0]};
  assign Abs1_y[0] = (Complex_to_Real_Imag_out2[0] < 16'sb0000000000000000 ?  - (Abs1_cast) :
              {Complex_to_Real_Imag_out2[0][15], Complex_to_Real_Imag_out2[0]});
  assign Abs1_cast_1 = {Complex_to_Real_Imag_out2[1][15], Complex_to_Real_Imag_out2[1]};
  assign Abs1_y[1] = (Complex_to_Real_Imag_out2[1] < 16'sb0000000000000000 ?  - (Abs1_cast_1) :
              {Complex_to_Real_Imag_out2[1][15], Complex_to_Real_Imag_out2[1]});
  assign Abs1_cast_2 = {Complex_to_Real_Imag_out2[2][15], Complex_to_Real_Imag_out2[2]};
  assign Abs1_y[2] = (Complex_to_Real_Imag_out2[2] < 16'sb0000000000000000 ?  - (Abs1_cast_2) :
              {Complex_to_Real_Imag_out2[2][15], Complex_to_Real_Imag_out2[2]});
  assign Abs1_cast_3 = {Complex_to_Real_Imag_out2[3][15], Complex_to_Real_Imag_out2[3]};
  assign Abs1_y[3] = (Complex_to_Real_Imag_out2[3] < 16'sb0000000000000000 ?  - (Abs1_cast_3) :
              {Complex_to_Real_Imag_out2[3][15], Complex_to_Real_Imag_out2[3]});
  assign Abs1_cast_4 = {Complex_to_Real_Imag_out2[4][15], Complex_to_Real_Imag_out2[4]};
  assign Abs1_y[4] = (Complex_to_Real_Imag_out2[4] < 16'sb0000000000000000 ?  - (Abs1_cast_4) :
              {Complex_to_Real_Imag_out2[4][15], Complex_to_Real_Imag_out2[4]});
  assign Abs1_out1[0] = {Abs1_y[0][14:0], 1'b0};
  assign Abs1_out1[1] = {Abs1_y[1][14:0], 1'b0};
  assign Abs1_out1[2] = {Abs1_y[2][14:0], 1'b0};
  assign Abs1_out1[3] = {Abs1_y[3][14:0], 1'b0};
  assign Abs1_out1[4] = {Abs1_y[4][14:0], 1'b0};



  assign v1_im[0] = Abs1_out1[0];
  assign v1_im[1] = Abs1_out1[1];
  assign v1_im[2] = Abs1_out1[2];
  assign v1_im[3] = Abs1_out1[3];
  assign v1_im[4] = Abs1_out1[4];

  assign v1_im_0 = v1_im[0];
```

Figure 36. Report including the automatically generated HDL code for the DPD block.

Additional HDL specific options can be selected within the model to optimize resource utilization, speed, and area of the design being implemented. Configuration options can be modified and the design and resulting generated code iterated on. Figure 37 shows resource utilization, which might be of interest when iterating on the design to ensure it fits on the FPGA being used.



Generic Resource Report for PA_DPD_ImplementationFixed_17b

Summary

Multipliers	240
Adders/Subtractors	348
Registers	8
Total 1-Bit Registers	128
RAMs	0
Multiplexers	250
I/O Bits	868
Static Shift operators	0
Dynamic Shift operators	0

Detailed Report

Report for Multiply-instantiated Identical Subsystems 0 : DPD

Multipliers (100)

```
16x16-bit Multiply : 100
```

Adders/Subtractors (148)

```
17x17-bit Adder : 65
[+] 32x32-bit Adder : 8
[+] 17-bit Subtractor : 50
17x17-bit Subtractor : 25
```

Multiplexers (240)

Figure 37. Report of the resources required for the automatically generated HDL code for the DPD block.

The generated HDL code could be used for rapid prototyping on FPGA but is also production quality and could be used for FPGA or ASIC implementation of the final design. The generated HDL code can be verified by generating a cosimulation testbench using HDL Verifier™. This product interfaces with a third-party HDL simulator such as Mentor Graphics® ModelSim® or Cadence® Incisive® and reuses the Simulink testbench to verify that the HDL code running in the HDL simulator and the Simulink model have the same behavior. This is particularly useful when the system being converted is part of a closed-loop system because the interface allows the input being passed to the HDL simulator to be changed adaptively over the course of the simulation dependent on the output from the HDL simulator in previous time steps. This wouldn't be possible if the HDL testbench stimulus was prerecorded. Figure 38 shows the automatically generated cosimulation testbench, which is run from Simulink.

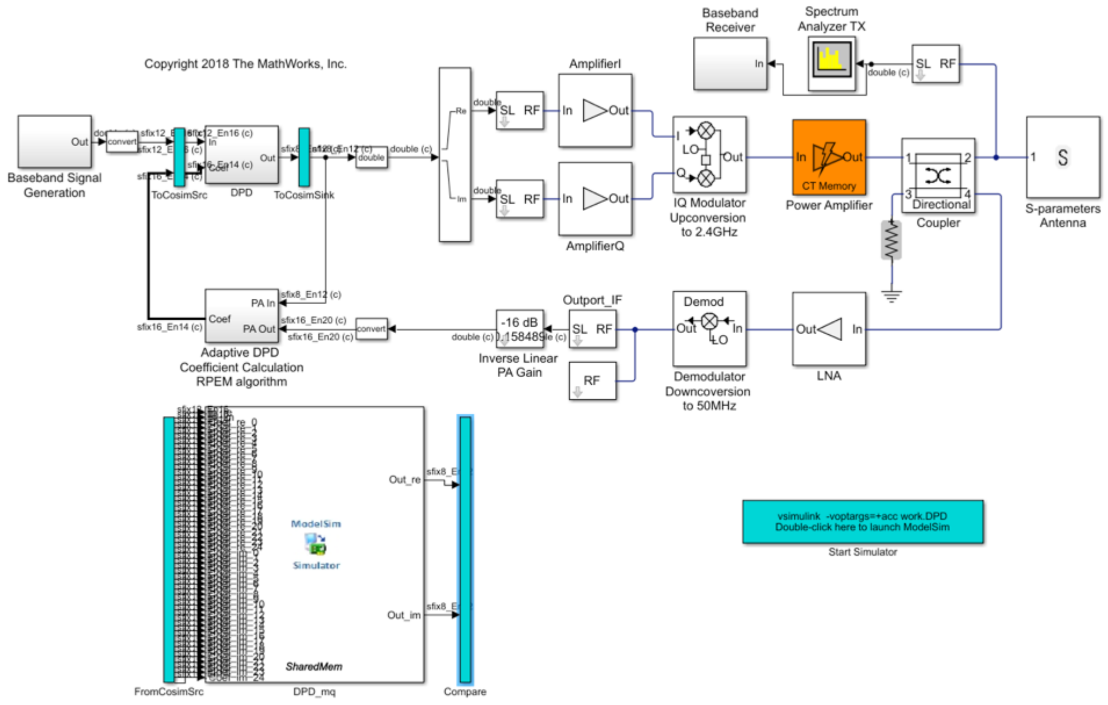
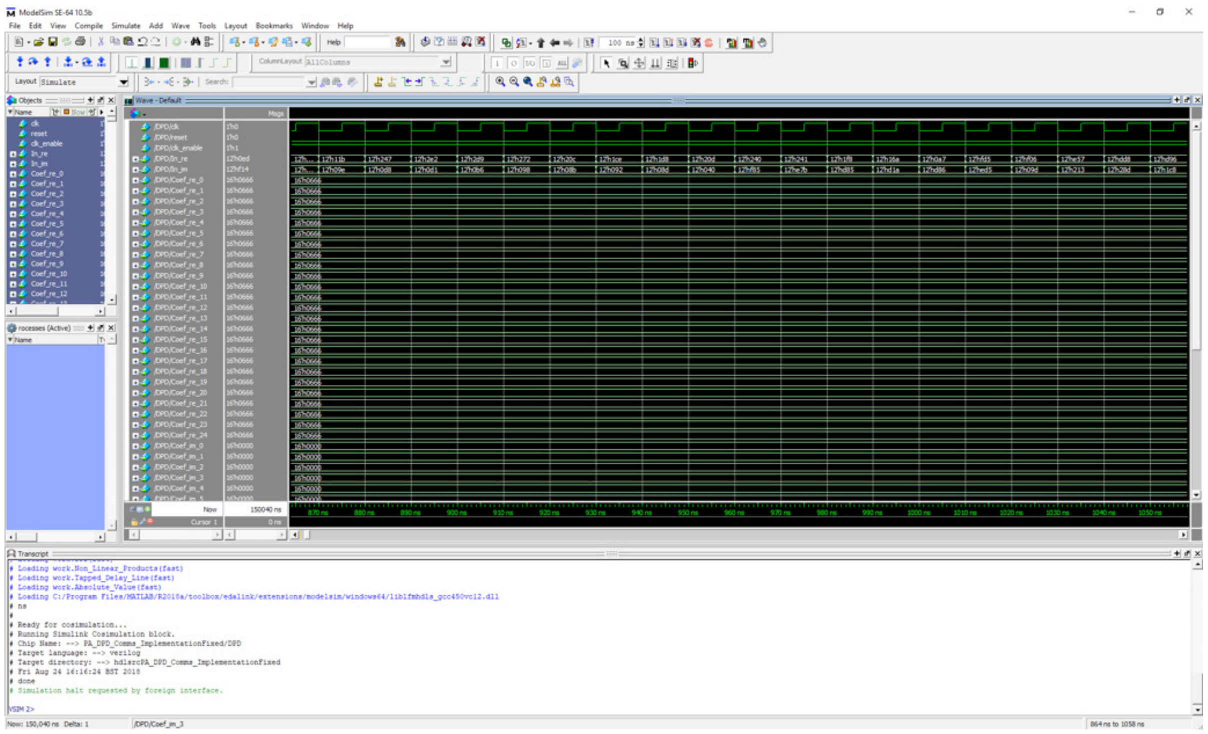


Figure 38. Simulink testbench for cosimulating the HDL implementation of the DPD algorithm with Mentor Graphics ModelSim.

Clicking the blue box launches the HDL simulator, which is ModelSim in this case. It then automatically loads in the HDL code and configures the simulator for cosimulation. Once this is done, hitting the play button in Simulink starts to stream data to the ModelSim environment and sends outputs back to Simulink on each timestep.



```
# Loading work.Non_Linear_Products(fast)
# Loading work.Tapped_Delay_Line(fast)
# Loading work.Absolute_Value(fast)
# Loading C:/Program Files/MATLAB/R2018a/toolbox/edalink/extensions/modelsim/windows64/liblfmhdls_gcc450vc12.dll
# ns
#
# Ready for cosimulation...
# Running Simulink Cosimulation block.
# Chip Name: --> PA_DPD_Comms_ImplementationFixed/DPD
# Target language: --> verilog
# Target directory: --> hdlsrcPA_DPD_Comms_ImplementationFixed
# Fri Aug 24 16:16:24 BST 2018
# done
# Simulation halt requested by foreign interface.

VSIM 2>
```

Figure 39. Simulation results of the cosimulation testbench as shown in Mentor Graphics ModelSim.

Following the simulation, results can be analyzed within the HDL simulator (Figure 39) or within Simulink. The difference between the Simulink model and the generated HDL code can also be compared in Simulink by using the scopes inserted by HDL Verifier as part of the testbench generation process. The scope outputs in Figure 40 below show the real and imaginary signal outputs from the DPD subsystem. In each case, the top plot shows the Simulink model output, the middle plot shows the output from the HDL code running in ModelSim, and the bottom plot shows the difference between the two. In both cases, it is clear that the Simulink model and HDL code are equivalent.

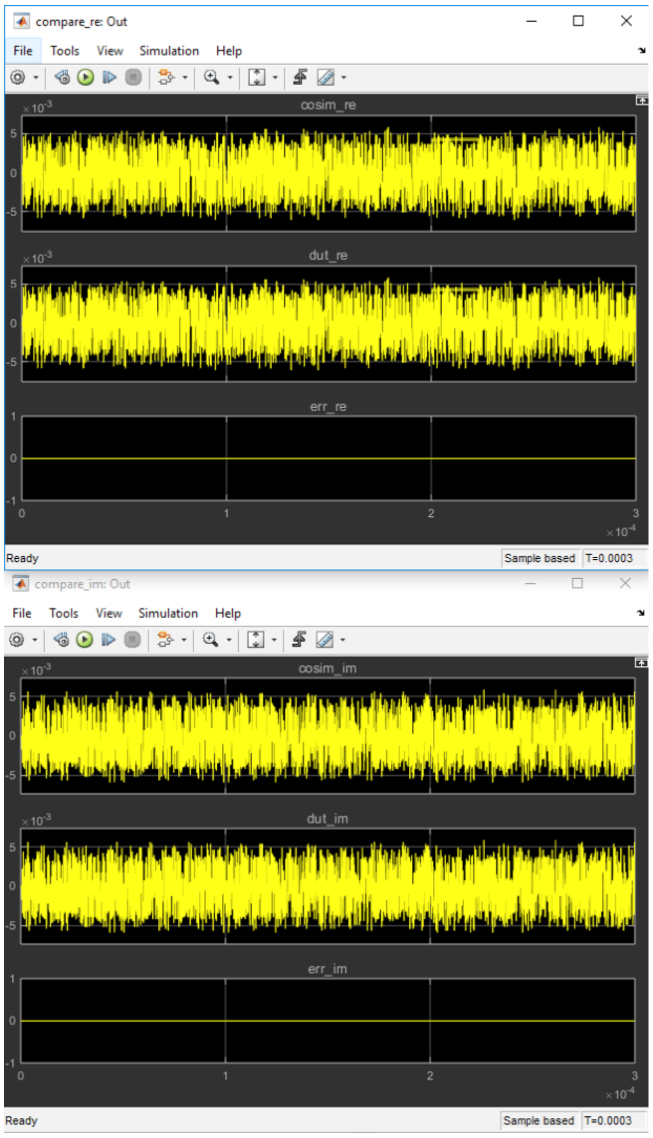


Figure 40. Comparison of the output of the reference Simulink model and the HDL implementation of the DPD block. The top scope plots the real part of the output data, and the bottom scope plots the imaginary part. In each scope, the top trace is the cosimulation output, the middle trace is the Simulink output, and the bottom trace is the error signal representing the difference between the two.

HDL Verifier can also be used for HIL simulations involving an FPGA by following a similar workflow. This validates that the algorithm behavior is the same following HDL synthesis.

In summary, the Simulink model is used to directly create and verify the implementation. With Model-Based Design, the same environment can be used to design, test, prototype, and implement your algorithms. More details and examples and videos of HDL workflows can be found on the HDL Coder and HDL Verifier product pages [11].

Conclusion

In this paper, we have discussed the top-down design process of a DPD algorithm and shown how you can use simulation and Model-Based Design to accelerate this process and reduce risk by providing early-stage understanding of the performance of the RF system and a workflow from simulation to implementation. This starts by modeling the RF power amplifier using actual component characteristics. We explored RF component imperfections as well as strategies to create a model by fitting to collected RF power amplifier data.

The PA model was then placed alongside a model of a DPD algorithm within a closed loop, and we performed system-level simulation and analysis using OIP3 and EVM metrics to assess the quality of the linearization technique.

The DPD algorithm was then converted from a floating- to a fixed-point implementation. The effect on quantization, timing, and other system imperfections was then assessed. Finally, we generated HDL code from the DPD algorithm, and we covered techniques to optimize and verify the generated HDL code.

MATLAB and Simulink can be used to develop DPD algorithms, using the same models throughout the development process to maintain consistency, reduce risk, accelerate design and implementation, and complement lab testing and prototyping. You can find more information about this on the ***RF Blockset*** product page.

References

[1] Morgan, et.al., "A Generalized Memory Polynomial Model for Digital Predistortion of RF Power Amplifiers," IEEE Transactions on Signal Processing, Vol. 54, No. 10, October 2006.

[2] A. Zhu, "Decomposed Vector Rotation-Based Behavioral Modelling for Digital Predistortion of RF Power Amplifiers," IEEE Transactions on Microwave Theory and Techniques, Vol. 63, No. 2, February 2015.

[3] O. Hammi, "Augmented Twin-Nonlinear Two-Box Behavioral Models for Multicarrier LTE Power Amplifiers," The Scientific World Journal, Volume 2014, Article ID 762534.

[4] L. Gan, "Adaptive Digital Predistortion of Nonlinear Systems," Ph.D. Thesis, Electrical and Information Engineering, Graz University of Technology, Austria, April 2009.

[5] RF Blockset. ***mathworks.com/products/simrf.html***

[6] Circuit Envelope Fundamentals: Simulate High Frequency Components with RF Blockset. ***mathworks.com/help/simrf/gs/minimize-computations-for-rf-simulations.html***.

[7] Antenna Toolbox. ***mathworks.com/products/antenna.html***.

[8] Instrument Control Toolbox. ***mathworks.com/products/instrument.html***.

[9] E. Zhu, "Developing a Radio Frequency System for Wireless at Huawei", September 2018. ***mathworks.com/content/dam/mathworks/case-study/huawei-customer-case-study-portrait.pdf***.

[10] Fixed-Point Design Optimization. ***mathworks.com/products/fixed-point-designer/features.html#data-type-exploration-and-optimization***.

[11] HDL Code Generation and Verification. ***mathworks.com/solutions/hdl-code-generation-verification.html***.