APPLICATION NOTE

Measurement-Based Artificial Neural Network Simulation Models for RF Power Amplifiers

KEYSIGHT

Introduction

Engineers working on Power Amplifiers (PAs) collect a large amount of measured load pull data over the design cycle. This measured data accumulates on hard disks or cloud drives and development teams sometimes forget about it after finishing a design. However, new AI-based modeling techniques have the potential to turn this measured data into a gold mine because engineers can now extract advanced simulation models from measured data that is just “lying around.” These models can breathe new life into old data and create exciting possibilities for product design and development.

This Application Note will describe how to build measurement-based simulation models for Power Amplifiers using Keysight’s Artificial Neural Network (ANN) engine. The Application Note will also cover validation and use of these models in Keysight’s Advanced Design System (ADS) environment.

Power Amplifiers and Load Pull Measurement

Power Amplifiers are nonlinear circuits whose performance depends primarily on the load impedance presented to a biased transistor or set of transistors. Since the nonlinear behavior of the circuit is difficult to predict, engineers perform characterization of power amplifier devices using a tunable RF load. PA Designers call such a measurement a “Load Pull” Characterization because different loads pull varying amounts of power from the nonlinear device(s). Figure 1 shows a block diagram of a simple load pull measurement system.

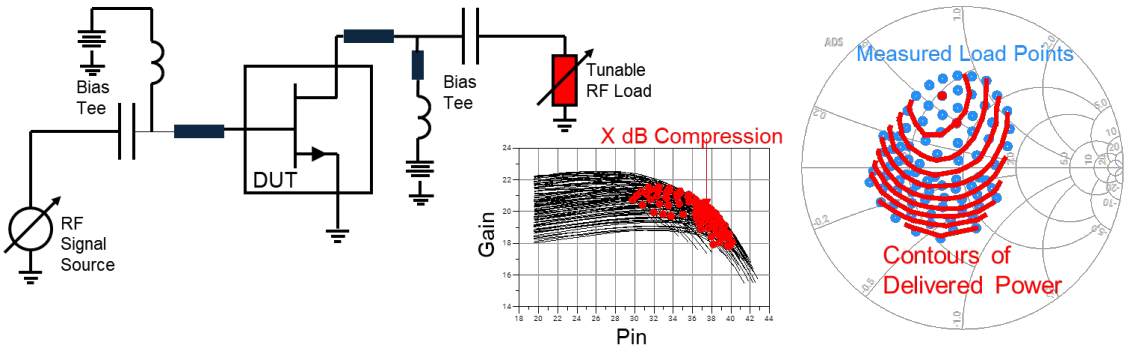


Figure 1. Block diagram of a Load pull measurement system, with contours plotted at X dB gain compression.

Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of the human brain.[1] The basic unit of an ANN is the artificial neuron. Figure 2 depicts an ANN structure to model the Load Pull measurement shown in Figure 1. The system has four inputs: real and imaginary load (gamma), input power, and fundamental frequency. The network has two layers with five hidden neurons each, plus an input and output layer. There is one output term, “Gain.” The input to each neuron/layer is a weighted sum of outputs from the previous layer transformed through an activation function to model nonlinearity in the system. Training the network involves adjusting weights, producing an output, and evaluating the error between input and output. Typically, the training process continues until either the error drops below a desired tolerance or a maximum number of iterations complete.

1 https://www.keysight.com/blogs/en/tech/sim-des/2023/12/4/ai-in-device-modeling

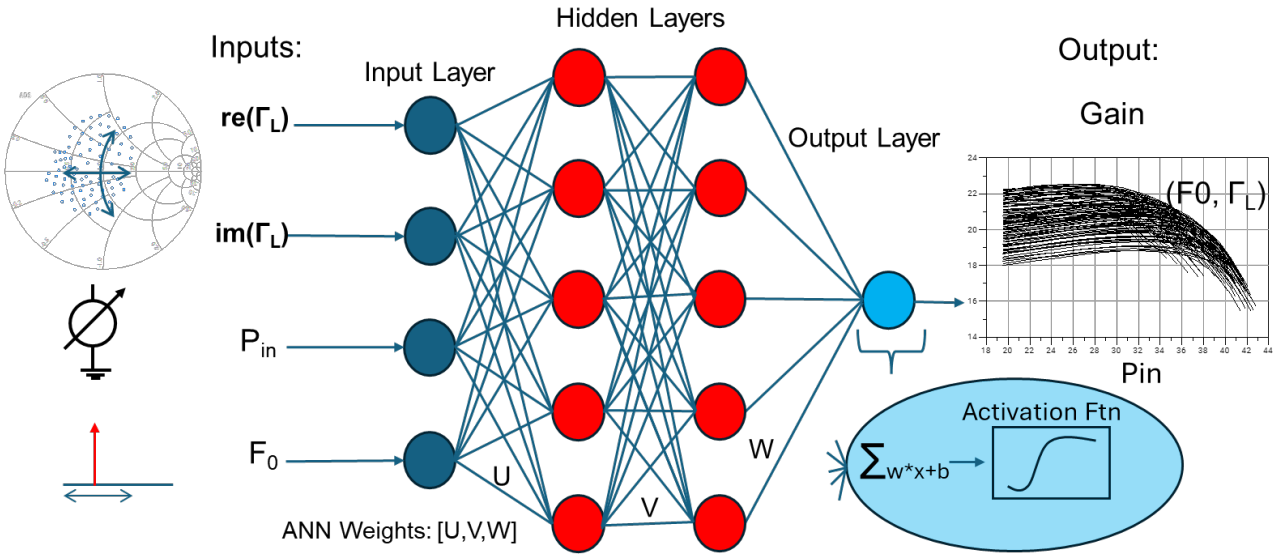


Figure 2. Artificial Neural Network model of a power amplifier characterized with load pull shown in Figure 1.

Configuring and training an ANN involves skill and experience. For example, too few hidden layers can result in an overly simple model (underfitting), while too many hidden layers can result in a model which matches the data well but performs poorly outside the training range (overfitting). Different software programs may express ANNs as a function, however, these tools often obscure the details of the underlying function itself, which can make the network difficult to apply to general circuit modeling applications in EDA software.

Keysight has been applying ANN's to high frequency modeling for a long time; Keysight's first ANN implementation was published in 2006.[2] This implementation evolved to have unique capabilities such as training from either (x,y) or (x, dy/dx) types of datasets, and the ability to express the output conveniently in a format which is consumable by circuit simulation tools. Popular nonlinear device models such as DynaFET make extensive use of Keysight's homegrown ANN engine.[3] This engine is now accessible directly inside of Keysight ADS for general purpose modeling/data fitting. This Application Note utilizes the Keysight ANN engine for all the modeling work described in the upcoming sections.

For more information

For more insights on Artificial Neural Network Modeling, check out: IMS2023: Artificial Intelligence & Machine Learning for RF & Microwave Design

2 J. Xu, D. Gunyan, M. Iwamoto, A. Cognata, and D. E. Root, "Measurement-Based Non-Quasi-Static Large-Signal FET Model Using Artificial Neural Networks," 2006 IEEE MTT-S Int. Microwave Symp. Dig., San Francisco, CA, USA, June 2006.

3 J. Xu, S. Halder, F. Kharabi, J. McMacken, J. Gering and D. E. Root, "Global dynamic FET model for GaN transistors: DynaFET model validation and comparison to locally tuned models," *83rd ARFTG Microwave Measurement Conference*, Tampa, FL, USA, 2014, pp. 1-6

Building ANN-Based PA Simulation Models

Data formatting challenges

The first step in building a useful simulation model is to access and condition the measured data. Load Pull data comes in a variety of formats output by hardware test systems. For example, Maury Microwave systems produce data in cst, spl, lp and mat formats; Focus Microwaves systems produce data in lpc, lpcwave, and lpd formats; Keysight Wideband Active Load Pull systems produce data in mdf format. Most ANN engines, including Keysight's, require input training data formatted as a continuous set. The challenge is: none of the raw formats above happen to produce data with such a structure. Consider a datafile from a Qorvo GaN device measured on a Focus test system shown in Figure 3.[4]

```
! Power Sweep Load Pull Measurement Data
!-------------------------------------------------------
! File =  L:\DATA\ID20180105075456_QPD1025_FLP1\LP1_final_QPD1025_E1_0LPR_W1745156_sn030_bot_1743-2_3_Ohm_50V_750mA_915MHz.lpc
! Comment =
! Date = Thursday, January 18, 2018   Time: 09:09:54
!-------------------------------------------------------
! Frequency = 0.915 GHz
!-------------------------------------------------------
Point  Gamma  Phase[deg]  Pin[dBm]  Pout[dBm]  Gain[dB]  Psource[dBm]  V1[V]  I1[uA]  V2[V]  I2[mA]  OutEff[%]  PAEff[%]  R_S@F0  PowerIndex
!-------------------------------------------------------
# 001  0.280  97.4
    19.53     41.21     21.68     -30.00     -2.412     -5.133     50.071  1770.675     14.91     14.80     0.72     -1.74     1.     1
    20.54     42.26     21.73     -29.00     -2.412     -5.016     50.071  1978.777     16.99     16.88     0.72     -1.74     1.     2
    21.53     43.37     21.84     -28.00     -2.412     -4.736     50.071  2225.923     19.49     19.36     0.72     -1.74     1.     3
    22.49     44.39     21.90     -27.00     -2.412     -4.628     50.071  2488.467     22.07     21.93     0.72     -1.74     1.     4
    23.50     45.46     21.96     -26.00     -2.412     -4.232     50.071  2799.805     25.07     24.91     0.72     -1.74     1.     5
# 040  0.311  25.7
    19.54     39.99     20.45     -30.00     -2.413     -5.474     50.071  1638.079     12.18     12.07     0.72     -1.74     1.     1
    20.53     40.99     20.46     -29.00     -2.413     -5.283     50.071  1827.907     13.73     13.61     0.72     -1.74     1.     2
    21.53     42.03     20.49     -28.00     -2.412     -5.111     50.071  2043.083     15.60     15.46     0.72     -1.74     1.     3
    22.51     43.02     20.51     -27.00     -2.413     -4.865     50.071  2277.137     17.57     17.41     0.72     -1.74     1.     4
    23.50     44.07     20.57     -26.00     -2.413     -4.609     50.071  2554.635     19.98     19.80     0.72     -1.74     1.     5
# 005  0.455  -107.3
    19.51     37.53     18.01     -30.00     -2.412     -7.529     50.035  2299.502      4.92      4.84     0.72     -1.74     1.     1
    20.50     38.59     18.08     -29.00     -2.412     -7.247     50.071  2585.852      5.58      5.49     0.72     -1.74     1.     2
    21.49     39.66     18.17     -28.00     -2.412     -7.212     50.071  2908.920      6.35      6.25     0.72     -1.74     1.     3
    22.46     40.71     18.25     -27.00     -2.412     -7.164     50.071  3261.074      7.21      7.10     0.72     -1.74     1.     4
    23.47     41.80     18.33     -26.00     -2.412     -7.161     50.071  3677.927      8.22      8.10     0.72     -1.74     1.     5
```

Figure 3. Raw Focus Load Pull datafile (lpc) for Qorvo QPD1025 GaN Power Transistor at 915 MHz, 50V drain bias.

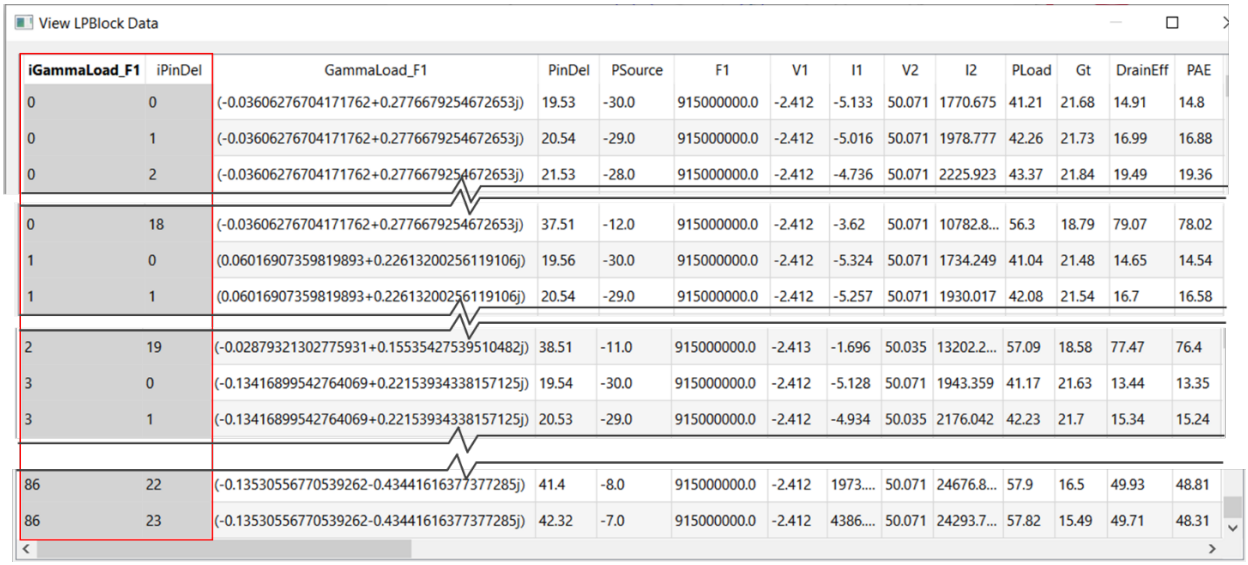
Modeling engineers cannot pass data from Figure 3 into the ANN engine directly because the file structures measurement data into subblocks. These subblocks break up the data into smaller subsets divided along specific indexed dependencies like Gamma Load or Input Power. To extract an ANN model, the data must exist in a single continuous set rather than in separated subblocks.

Fortunately, a Keysight-provided Python library called PathWave Data Tools (pwdatatools) can read common file formats from Maury, Focus, and Keysight and fit the data to a structure which enables easy model extraction. This library is available to download for free, and it also ships as part of Keysight Advanced Design System.[5]

The pwdatatools library reads measured data files into a structure called a "LoadPullBlock" object.[6] Referencing Figure 4: pwdatatools assigns the swept variables (GammaLoad_F1 and PinDel) a unique

4 https://www.qorvo.com/products/p/QPD1025#documents
5 https://docs.keysight.com/pages/viewpage.action?pageId=748497656
6 M. Ozalas and J. Boh, "A Modern Framework to Incorporate Measured Load Pull Data in Power Amplifier Design," *2024 IEEE Wireless and Microwave Technology Conference (WAMICON)*, Clearwater, FL, USA, 2024, pp. 1-3.

index, which allows representation of the data as one single set. This makes it possible to create the type of input file required for ANN model extraction.



iGammaLoad_F1	iPinDel	GammaLoad_F1	PinDel	PSource	F1	V1	I1	V2	I2	PLoad	Gt	DrainEff	PAE
0	0	(-0.03606276704171762+0.2776679254672653j)	19.53	-30.0	915000000.0	-2.412	-5.133	50.071	1770.675	41.21	21.68	14.91	14.8
0	1	(-0.03606276704171762+0.2776679254672653j)	20.54	-29.0	915000000.0	-2.412	-5.016	50.071	1978.777	42.26	21.73	16.99	16.88
0	2	(-0.03606276704171762+0.2776679254672653j)	21.53	-28.0	915000000.0	-2.412	-4.736	50.071	2225.923	43.37	21.84	19.49	19.36
0	18	(-0.03606276704171762+0.2776679254672653j)	37.51	-12.0	915000000.0	-2.412	-3.62	50.071	10782.8...	56.3	18.79	79.07	78.02
1	0	(0.06016907359819893+0.22613200256119106j)	19.56	-30.0	915000000.0	-2.412	-5.324	50.071	1734.249	41.04	21.48	14.65	14.54
1	1	(0.06016907359819893+0.22613200256119106j)	20.54	-29.0	915000000.0	-2.412	-5.257	50.071	1930.017	42.08	21.54	16.7	16.58
2	19	(-0.02879321302775931+0.15535427539510482j)	38.51	-11.0	915000000.0	-2.413	-1.696	50.035	13202.2...	57.09	18.58	77.47	76.4
3	0	(-0.13416899542764069+0.22153934338157125j)	19.54	-30.0	915000000.0	-2.412	-5.128	50.071	1943.359	41.17	21.63	13.44	13.35
3	1	(-0.13416899542764069+0.22153934338157125j)	20.53	-29.0	915000000.0	-2.412	-4.934	50.035	2176.042	42.23	21.7	15.34	15.24
86	22	(-0.13530556770539262-0.43441616377377285j)	41.4	-8.0	915000000.0	-2.412	1973....	50.071	24676.8...	57.9	16.5	49.93	48.81
86	23	(-0.13530556770539262-0.43441616377377285j)	42.32	-7.0	915000000.0	-2.412	4386....	50.071	24293.7...	57.82	15.49	49.71	48.31

Figure 4. Load Pull data reconfigured for model extraction showing explicit dependencies on GammaLoad, PinDel.

Load Pull Block structure

Figure 5 shows an example of the Python command to do a PathWave Data Tools read operation, creating a ‘LoadPullBlock’ object. Figure 5 also shows a printout of the contents inside a ‘LoadPullBlock’.

```
from keysight import pwdatatools as pwdt

lpblock = pwdt.read_file_as_loadpullblock(
    "LP1_final_QPD1025_E1_0LPR_W1745156_sn030_bot_1743-2_3_Ohm_50V_750mA_915MHz.lpc"
)

LoadPullBlock(
    <'V1', 'I1', 'V2', 'I2', 'PLoad', 'GainT', 'PSource', ... with 1848
     observations>,
    name='LP1_final_QPD1025_E1_0LPR_W1745156_sn030_bot_1743-
     2_3_Ohm_50V_750mA_915MHz',
    gamma_ivarname='GammaLoad_F1',
    power_ivarname='PinDel',
    attrs={'Measurement_Type': ..., 'Sweep_Type': ..., 'Date': ..., 'Re ...},)
```

Figure 5. PathWave Data Tools LoadPullBlock object read operation shown in Python Command Line.

Referring to Figure 5, The terms (“V1”,” I1”,) are types of response measurements, indexed by name (i.e., lpblock[“V1”]). Each observation is a measurement made under a different test condition. The ivarnames represent the measurement stimulus. For example, this test system swept load gamma

('GammaLoad_F1') and input power ('PinDel'), and measured DC bias ('V1','I1','V2','I2'), power levels ('PLoad','PSource'), Gain ('GainT') and other variables ('…').

Extracting ANN models

Figure 6 shows an example of how to train an ANN model using Keysight's Python implementation.

```
from keysight.ads import ann
setup = ann.AnnSetup(num_inputs, num_outputs)
setup.output_format = setup.output_format.TEXT_FORMULA
ann.configure_setup(setup)
ann.extract_model(input_data_file, output_data_file)
```

Figure 6. Python code snippet showing a model training operation using Keysight's ANN engine.

For the example loadpull block in Figure 5, the inputs would be the complex gamma split into real and imaginary parts, and input power. The output could be power delivered to the load or another response measurement. The resulting ANN function would then have three inputs and one output. Referencing Figure 6: the script configures the ANN engine (inputs, outputs, format), then the model trains on input data contained in a tab- delimited CSV file (input_data_file pointer). In this case, the engine outputs the result as a formula contained in a text file (output_data_file.equation), because the output format is set to a text formula. Figure 7 shows the input and output file contents. Note that the ANN equation in the output file has three input terms (_v1, _v2, _v3) and one output term (y1). To use this equation directly in an EDA tool, these terms might need renaming which is possible to do with a script.

input_data_file

-0.03606276704171762	0.2776679254672653	19.53	41.21
-0.03606276704171762	0.2776679254672653	20.54	42.26
-0.03606276704171762	0.2776679254672653	21.53	43.37
-0.03606276704171762	0.2776679254672653	22.49	44.39
-0.03606276704171762	0.2776679254672653	23.5	45.46
-0.03606276704171762	0.2776679254672653	24.51	46.52
-0.03606276704171762	0.2776679254672653	25.51	47.57
-0.03606276704171762	0.2776679254672653	26.5	48.62
-0.03606276704171762	0.2776679254672653	27.49	49.62
-0.03606276704171762	0.2776679254672653	28.5	50.63

output_data_file.equation

```
y1=(37.53+(8.633559020145748+(-8.117257409537137)*1.0/(1.0+exp(-(7.650565546438886+
3.292074870866821*1.0/(1.0+exp(-(2.038887964476389+0.8729523624619417*(-1.0+2.0*(_v1-
(-0.5772804298962712))/0.8704977715836048)+(-1.133545050700339)*(-1.0+2.0*(_v2-(
-0.4344161637737728))/1.006577675870692)+0.5997250213707039*(-1.0+2.0*(_v3-19.49)/23.3))))
+15.59249850219989*1.0/(1.0+exp(-((-4.188939183329312)+0.1154640251749004*(-1.0+2.0*(_v1-(
-0.5772804298962712))/0.8704977715836048)+(-0.1016316766982854)*(-1.0+2.0*(_v2-(
0.4344161637737728))/1.006577675870692)+1.720290412904268*(-1.0+2.0*(_v3-19.49)/23.3))))+
(-12.26923392138682)*1.0/(1.0+exp(-(1.887295630601667+(-0.4185688203423064)*(-1.0+2.0*(_v1
-(-0.5772804298962712))/0.8704977715836048)+(-0.7342948009810231)*(-1.0+2.0*(_v2-
(-0.4344161637737728))/1.006577675870692)+(-0.2014926858638553)*(-1.0+2.0*(_v3-19.49)/23.3)
```

Figure 7. Input and output files for ANN training on QPD1015 Load Pull data using Keysight's ANN engine.

General Python ANN extraction function

In this section, we introduce a Python function, called extract_ann_model(), to generate an ANN model from a given set of inputs and outputs. The function is general enough to use on diverse types of data beyond the example in this App Note. As of publication, the tool version is ADS 2025, with Python 3.12.2, and pwdatatools 0.10.0. Future versions may have slightly different commands and structures, but the overall approach should not change. This function can run either directly in ADS (from Main ADS Window: Tools->Python Console) or from outside of ADS. If running externally, set the Python interpreter path to the ADS Python install, $HPEESOF_DIR/tools/python/python.exe, where $HPEESOF_DIR points to the ADS installation directory. Figure 8 shows the contents of the extract_ann_model() function. You can paste this code directly into the ADS Python console to load the function.

```python
def extract_ann_model(path: os.PathLike | str, inputs: dict, outputs: dict) ->
list:
    """Function to extract ANN equation(s) from input and output data"""
    from keysight.ads import ann

    # Configure input and output file names
    ann_input_file_path = os.path.join(os.path.dirname(path), "ann_input.csv")
    ann_output_file_path = os.path.join(os.path.dirname(path), "ann_output")

    # Combine input and output into a single dictionary
    combined_data = {**inputs, **outputs}
    pd.DataFrame(combined_data).to_csv(
        ann_input_file_path, sep="\t", header=False, index=False)

    # Configure ANN setup and extract the model
    setup = ann.AnnSetup(len(inputs), len(outputs))
    setup.output_format = setup.output_format.TEXT_FORMULA
    setup.num_hidden_layers = 2
    setup.num_neurons_per_layer = 5
    ann.configure_setup(setup)
    ann.extract_model(ann_input_file_path, ann_output_file_path)

    # Read the model equation(s) and update the variable names
    with open(ann_output_file_path + ".equation", "r") as myfile:
        data = myfile.readlines()
        updated_data = []
        for line in data:
            for index, key in enumerate(inputs, start=1):
                line = line.replace(f"_v{index}", key)
            for index, key in enumerate(outputs, start=1):
                line = line.replace(f"y{index}", key)
            updated_data.append(line.strip())
    return updated_data
```

Figure 8. A useful Python function for extracting an ANN model from measured data.

This paragraph will describe the Python function shown in Figure 8, starting with the inputs. The file path input is necessary since the ANN extraction process generates multiple files. Dictionaries define the input and output data; this enables modeling of arbitrary data types and lengths. The function combines the input and output dictionaries into a single DataFrame used to generate a csv input file in the proper format for ANN training. From there, the function configures the ANN engine– this case uses two hidden layers with five neurons per layer, but that is adjustable. Users can add additional configuration settings or change the current settings as needed. The model file is set to “ann_output.” Since the output format is a text formula, the equation appears by default in a file called ann_output.equation. After the ANN engine creates the file, the script reads in the equation and replaces the generic “y” and “_v” terms with the input

and output names. If there are multiple outputs, there will be multiple equations, therefore, the function returns a list. Despite the convenient nature of producing multiple outputs from the same set of inputs, we highly recommend generating each output individually, with a separate model training (i.e., multiple inputs to one output). This ensures the most accurate model possible.

Using the function is straightforward (Figure 9). To call the ANN library, users must define the environment variable HPEESOF_DIR (this is automatic if invoking from ADS). Then, simply read the measured data file using pwdatatools and pass in a Python dictionary specifying the inputs and outputs. Figure 9 shows an example of how to invoke the “extract_ann_model” function.

```
# Set environment variable (needed for ann module if running outside of ADS)
os.environ["HPEESOF_DIR"] = os.path.normpath("C:\\Program
Files\\Keysight\\ADS2025")

# Read the measured data to a load pull block
meas_file = (
    "LP1_final_QPD1025_E1_0LPR_W1745156_sn030_bot_1743-
2_3_Ohm_50V_750mA_915MHz.lpc"
)
lpblock = pwdt.read_file_as_loadpullblock(meas_file)

# Extract the ANN model
ann_eqn = extract_ann_model(
    meas_file,
    {
        "gamma_real": np.real(lpblock["GammaLoad_F1"]),
        "gamma_imag": np.imag(lpblock["GammaLoad_F1"]),
        "input_power": lpblock["PinDel"],
    },
    {"output_power": lpblock["PLoad"]},
)
print(ann_eqn)
```

Figure 9: Example of how to use the extract_ann_model function to generate an ANN equation for PLoad. Function inputs are gamma_real, gamma_imag, and input_power. Function output is a list containing all ANN equations (in this case, just one).

A Lookup Simulation Model

Let us apply the ANN equation extracted in the prior section to a basic simulation model of the Qorvo GaN device. The term “model” applies lightly here; the operation is more like a lookup table rather than a full emulation. Nonetheless, the simple mechanism described in this section can be extremely useful for circuit design because ANN’s are particularly good at interpolating, and these computations tend to be fast. This technique utilizes the results from a large signal circuit simulation to derive inputs to the ANN equation (gamma load, input power). The ANN equation evaluates after the simulation is complete, and

the computed output (PLoad) saves automatically in the simulation dataset. To an end user, it may appear that an amplifier is in the circuit; in reality, this is just a computation. The “model” produces no voltages or currents; the result appears only in post-process after the circuit analysis is complete.

Lookup model circuit implementation

Engineers can copy and paste the equation output from the script above directly into an ADS Variable Block or Measurement Equation. To exercise the ANN, the simulation needs to sense the stimulus from the circuit and provide those as inputs to the ANN equation. Figure 10 shows an implementation where Measurement Equations compute the input power and load gamma using voltages and currents from the circuit. A matched load provides the input termination while a tiny one tone current source acts as the output sense element. An S1P_Eqn block emulates a load tuner at the fundamental frequency. Variables and functions in the “Tuner Control” component compute the impedance based on user defined settings (set_gamma_real, set_gamma_imag, frequency, Z0).

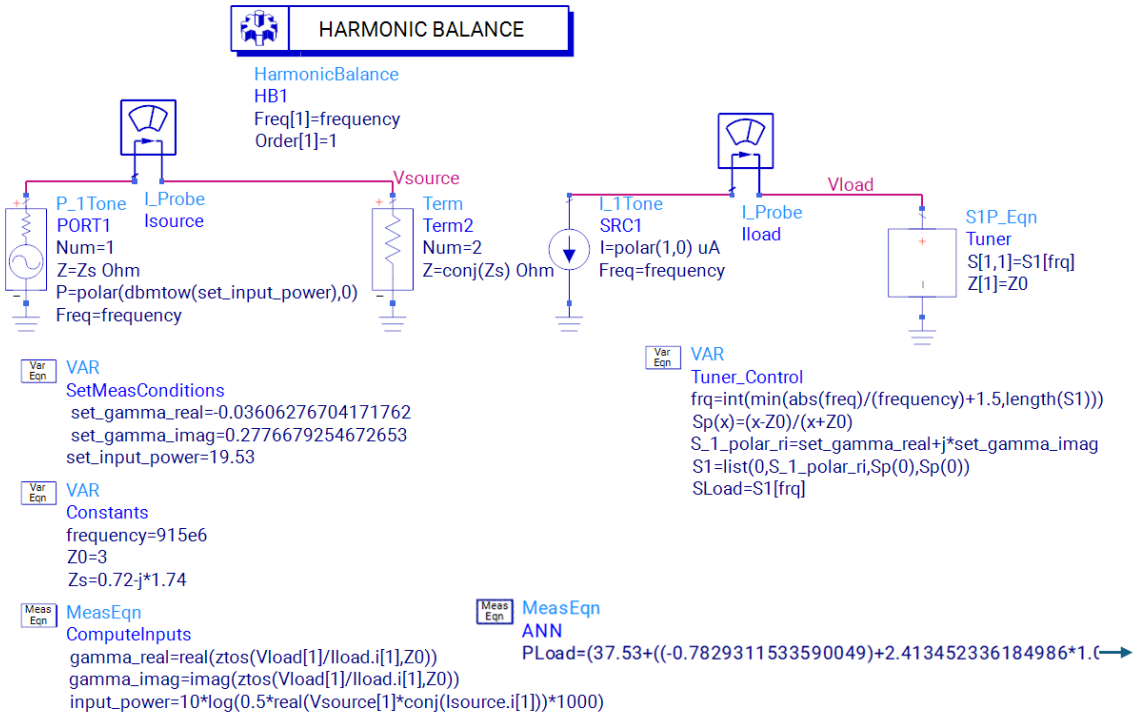


Figure 10. Model implementation in Keysight ADS, with the input as a matched load and the output as a one tone current source connected to an equation-based tunable load. The ANN function for PLoad, implemented as a Measurement Equation, comes directly from the output of the Python script in Figure 9. Additional measurement equations compute the inputs to the ANN function: gamma_real, gamma_imag, and input_power. A set of variables specify the measurement condition to simulate.

Lookup model validation

The configuration above simulates a single point at a time. However, model validation requires comparison at *all* measurement points, which makes the approach in Figure 10 inefficient. One creative solution involves writing a Python function to process the input data into a batch sweep table to drive the circuit simulation. This guarantees that the model simulates the same test conditions as the measurement. The script also writes the measured data sequence to an ADS dataset to facilitate direct comparison of model to measurement. Figure 11 shows the function used to create both the batch and measurement sequence datafiles. Figure 12 shows how to invoke the function from Figure 11 using the same input data as the prior ANN extraction from Figure 9.

Running the function creates two files: batch_sweep.csv and measured.ds. A Batch Simulation controller in ADS can point to the batch file, shown in Figure 13. The batch file sweeps the input variables along the same test conditions as the measurement. The simulation results then compare directly with the generated dataset file to give a good sense of the model's accuracy. Figure 13 shows verification over more than 1800 test conditions, with excellent agreement between model and measurement.

```
def create_ads_validation_files(
    file_path: os.PathLike | str, inputs: dict, outputs: dict
) -> None:
    """Function to create batch and validation files for ADS simulation"""
    batch_file = os.path.join(os.path.dirname(file_path), "batch_sweep.csv")
    data_file = os.path.join(os.path.dirname(file_path), "measured.ds")
    inputs = {f"set_{key}": value for key, value in inputs.items()}
    pd.DataFrame(inputs).to_csv(batch_file, index=False)
    ds_validation_df = pd.DataFrame(outputs).reset_index(drop=True)
    ds_validation_df["batchNumber"] = ds_validation_df.index + 1
    pwdt.Block(ds_validation_df, name="measured").to_file(data_file,
               dst_mode="w")
```

Figure 11. A useful Python function for creating ADS validation files.

```
create_ads_validation_files(
    meas_file,
    {
        "gamma_real": np.real(lpblock["GammaLoad_F1"]),
        "gamma_imag": np.imag(lpblock["GammaLoad_F1"]),
        "input_power": lpblock["PinDel"],
    },
    {"output_power": lpblock["PLoad"]},
)
```

Figure 12. Example of running the Python function to create validation files for the measured data.

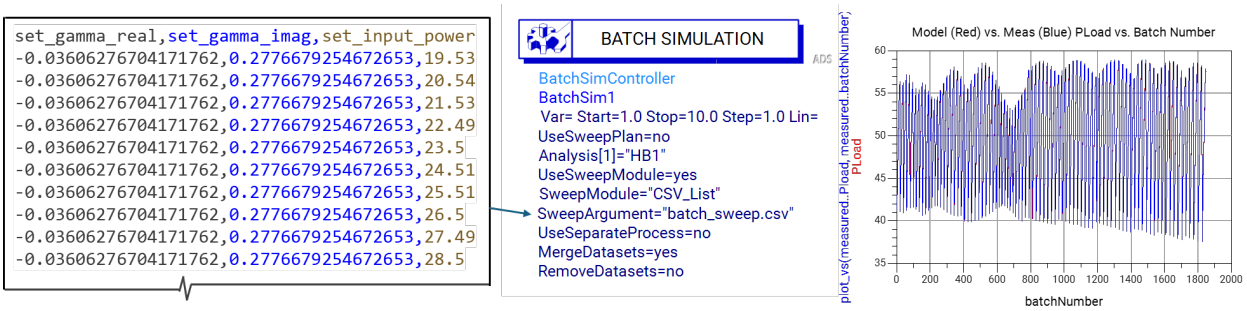


Figure 13. Using the batch file to control an ADS simulation and the data file to assess model vs. measured.

Lookup model summary

The approach described in this section is basic in the sense that it only provides a computed output value based on electrical stimulus from the larger circuit. A real amplifier would provide an electrical response to such a stimulus (more on that in the next section). Nonetheless, this simple model can be quite useful for things like matching network design because “looking up” a value can be faster and more convergence-friendly than crunching a full nonlinear circuit model. One added benefit: the simplicity of the equation allows users to add additional shaping functions on top of the response. For example, users could add a function which sets the response to a low value or zero if an input is outside of the measurement range. Such an approach could ensure that an output matching network optimization stays within the valid range of the data used to generate the model (more on this in the Circuit Design Example section).

A Stimulus - Response Simulation Model

This section describes a more complex model which can both sense the external circuit conditions and provide a nonlinear response to those conditions. There are two components inside ADS which can achieve such behavior: Symbolically Defined Devices (SDD) and Frequency-Domain Defined Devices (FDD). For both types of components, sensed voltage and current on any port, (_v,_i) process through an equation to produce voltage or current on any other port. In this case, an ANN function acts as the equation, taking sensed inputs and computing an output voltage or current at the device port. Since it is easier to define a Power Amplifier in the Frequency Domain, an FDD makes the most sense to use as the modeling component for this application.

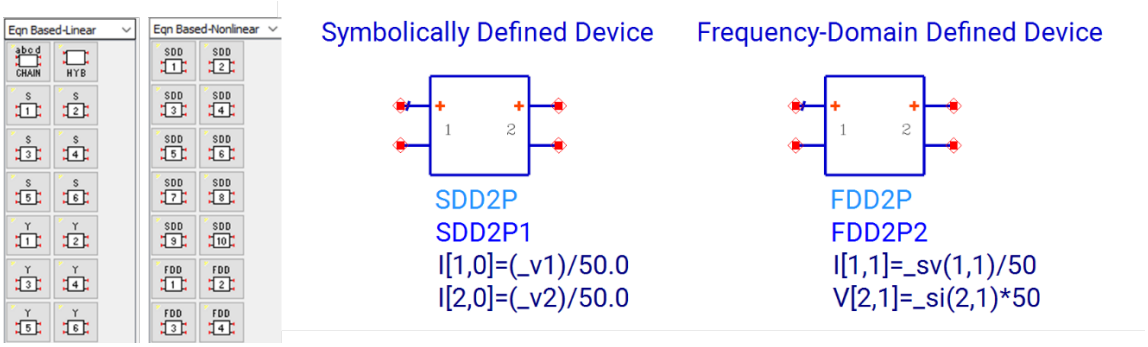


Figure 14. Both Linear and Nonlinear Equation based models exist in ADS. Symbolically Defined Device (SDD) and Frequency-Domain Defined Device (FDD) represent two ways to model nonlinear circuits.

Stimulus-response model circuit implementation

As in the prior case, the ANN equation will take inputs of gamma load and input power, computed from sensed voltage and current, and produce an output which corresponds to PLoad. This is slightly more complex for an FDD because such a component can only produce a voltage on a given port, not power. So, we will need to transform the measured output power to a corresponding voltage with an additional computation. This is straightforward to do using Equation 1 below:

$$P_{watts} = 10^{\frac{P_{dBm}-30}{10}} = \frac{v^2}{2Z} \quad (1)$$

While computing gamma and input power from sensed voltage and current is easy enough, when using an FDD, these equations need to be functionalized because regular ADS variables are not able to access the internal FDD sense terms (_sv(1,1),_si(2,1)). Only a function or variable called directly by the voltage and current output definitions (I[1,1] or V[2,1]) in the FDD may access a sensed voltage or current inside the FDD. To understand why, consider a schematic with multiple FDD components populated: if an arbitrary variable in the schematic accesses a sensed voltage, _sv(1,1), it is not clear which FDD term the variable is referring to. Because of this ambiguity, we will need to create functions to compute gamma and pin and then call these functions from the FDD port definition equations (V[2,1]). The ANN equation will also need to become functionalized because its inputs derive from FDD sense terms.

Figure 15 shows the implementation of the Stimulus-Response Model in ADS. The ANN equation in Figure 15 is the same as used in the prior example, however, it is now an FDD-callable function instead of a Measurement Equation as before. Referring to Figure 15, the variable block called “Functions” defines the functions used to compute gamma, power, and voltage (from Equation 1). A small constant real value ensures that the functions do not error out in a divide by zero situation. Using this approach, the sensed voltage and current inside the FDD pass directly into the predefined functions to produce a voltage tone at the fundamental frequency which corresponds to the measured power at that specific load and input power condition.

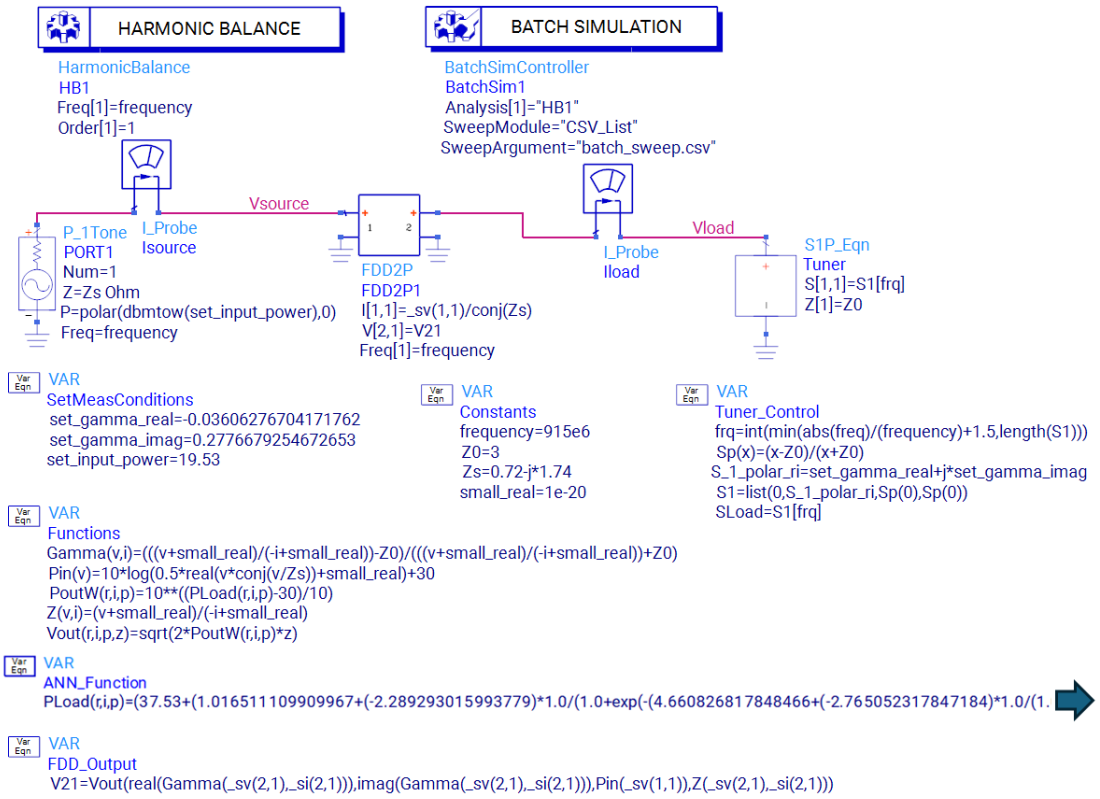


Figure 15. Stimulus-Response Model circuit implementation in Keysight ADS using an FDD component. The ANN equation from the prior example is reformatted as a function in ADS. Inputs to the ANN equation (Gamma, Pin) are also functions. The ANN-computed power converts to a voltage using the Vout() function with inputs real/imaginary Gamma, input power, and load impedance. The sensed voltages and currents in the FDD pass into the voltage function to produce an output tone at the fundamental frequency.

Generating the ANN function is straightforward, it requires a modified Python call shown in Figure 16. The inputs are now one letter variables for simplicity, and the output references those variables.

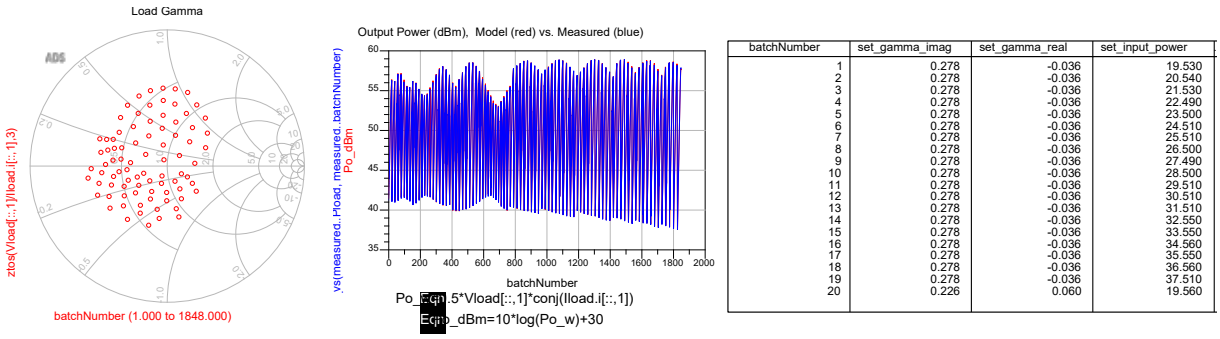
```
ann_eqn = extract_ann_model(

    meas_file,
    {
        "r": np.real(lpblock["GammaLoad_F1"]),
        "i": np.imag(lpblock["GammaLoad_F1"]),
        "p": lpblock["PinDel"],
    },
    {"PLoad(r,i,p)": lpblock["PLoad"]},
)
```

Figure 16. Modifying the input to the model extraction script to produce the ANN function used in ADS.

Stimulus-response model validation

Large signal simulation results (Figure 17) show good agreement between model and measured over all test conditions. In this case, the modeled output power derives from large signal voltage and current in the simulation rather than post process computation through an ANN equation.



batchNumber	set_gamma_imag	set_gamma_real	set_input_power
1	0.278	-0.036	19.530
2	0.278	-0.036	20.540
3	0.278	-0.036	21.530
4	0.278	-0.036	22.490
5	0.278	-0.036	23.500
6	0.278	-0.036	24.510
7	0.278	-0.036	25.510
8	0.278	-0.036	26.500
9	0.278	-0.036	27.490
10	0.278	-0.036	28.500
11	0.278	-0.036	29.510
12	0.278	-0.036	30.510
13	0.278	-0.036	31.510
14	0.278	-0.036	32.550
15	0.278	-0.036	33.550
16	0.278	-0.036	34.560
17	0.278	-0.036	35.550
18	0.278	-0.036	36.560
19	0.278	-0.036	37.510
20	0.226	0.060	19.560

Figure 17. Large Signal simulation results validating the Stimulus-Response model over all measurement conditions.

Stimulus-response model summary

This one tone stimulus-response model is best thought of as an emulation of the device rather than a full, design capable nonlinear model. This model can be useful for matching network design and system level analysis; however, it does not capture the full complexity of the waveforms inside of the device itself. The model simply senses the load and provides a fundamental voltage tone which matches the power delivered by the real device. Nonetheless, the jump made from a simple lookup approach to a full stimulus-response element which interacts dynamically with the surrounding circuit is significant and can serve as the basis for more advanced models.

It is not too difficult to expand the model described here, for example, by adding additional equations and ports to capture other effects such as DC voltage and current variation, Source Gamma variation (this device was assumed to have a simple conjugate input match), and phase shift through the device (there was no phase data provided in the file). For more advanced measurement setups, ANNs could even train on measured AB waves at multiple harmonics with the resulting equations implemented inside an FDD device. Designers could use such a model for more in-depth circuit level design, assuming the fitted measurements are accurate and contain enough data to cover a sufficient portion of the design space. While such a model is beyond the scope of this App Note, there are similar techniques described in the literature[7] and model extraction tools for this approach already exist.[8]

7 J. Louro, L. Nunes, F. Barradas and J. C. Pedro, "A Frequency-Domain Neural-Network Model for High-Power RF Transistors," *2023 19th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, Funchal, Portugal, 2023, pp. 1-4.

8 https://docs.keysight.com/pages/viewpage.action?spaceKey=eesofkcads&title=Load+Pull+Data+GUI+for+ADS

Circuit Design Example

Simulation and optimization setup

The last section shows how to use the stimulus-response model to design an output matching network in ADS. The design starts with a transmission line and surface mount topology implemented using Rogers 4360G2, with 0.032" thickness and 1 oz Copper cladding. Initial values were set arbitrarily (i.e., 1 pF for matching caps), with optimization constraints configured to keep lines and components within reasonable physical limits. Optimization ran with twenty-five iterations using a random minimax approach.

To ensure that the results are inside the measurement data range, limiter equations attenuate the value of Pout by 90% if the impedance presented by the matching network is outside the measurement limits. Such an approach heavily influences the optimizer to achieve a solution within the measurement range. An optimization goal for power was set higher than the maximum output power at this condition so that the optimizer would drive the circuit to the highest achievable value.

Simulation results

Figure 18 shows the optimization results and the final schematic. Figure 19 shows the initial and final matching network impedances presented to the device. Note that the initial value (blue point) was outside of the measurement range, but the final value (red point) ended up in the center of the measurement range because of the limiter approach. Overall, the optimization completed in under one second, achieving an output power of over 45 dBm while ensuring the circuit values are physically realizable. This illustrates the power of the stimulus-response modeling technique.

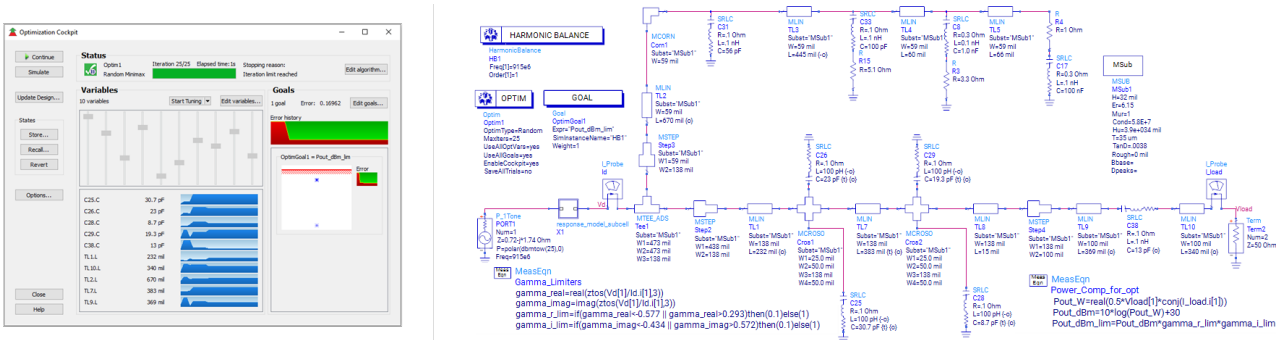


Figure 18. ADS Optimization Cockpit (left) and optimized output matching network schematic (right).

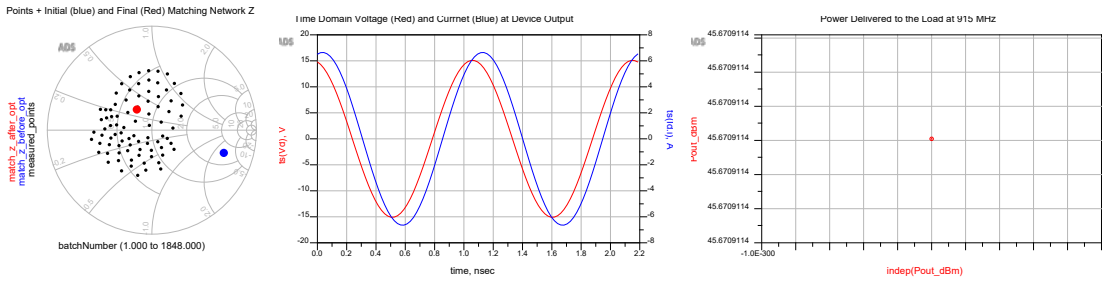


Figure 19. Simulation results for the optimized schematic, showing load values before and after optimization (left).

For more information

For more insights on the concepts described in this Application Note, check out these links:

To access the ADS workspace and Python scripts from this Application Note, plus additional Jupyter Notebooks and schematic creation scripts: Measurement-based Artificial Neural Network Simulation Models for RF Power Amplifiers (Keysight Knowledge Center)

To download a Python based add-on for ADS which reads measured load pull data and generates models similar to the ones described in this work: Load Pull Data GUI for ADS (Keysight Knowledge Center)

To watch a video series on extracting ANN's for Power Amplifiers: Using AI to Create a Simulation Model (YouTube)

To learn more about PathWave Data Tools Python Library: Welcome to the PathWave Data Tools Docs - PathWave Data Tools (keysight.com)

To learn more about Python automation in Keysight Advanced Design System: Python APIs for EDA Workflows (keysight.com)

Conclusion

This Application Note described two types of Artificial Neural Network (ANN) models derived from measured load pull data that are potentially useful for high frequency Power Amplifier design. We demonstrated these approaches using measured data from a real Qorvo GaN device characterized on a Focus measurement system. It is worth noting that the functions and approach described in this App Note are broadly applicable to different types of measurement systems and device technologies because the Keysight PathWave Data Tools (pwdt) Python library is capable of reading data in many formats and fitting this data to a well-known structure that is particularly useful for model extraction.

Python scripts processed the measured data and generated the ANN models. The functions and techniques are also applicable to different scenarios and measured data types. For example, users can simply change the inputs to the functions to point to the proper variable names. This allows function reuse for different data, measurement, and modeling scenarios. Users can copy and paste these scripts directly into the Python console in Keysight Advanced Design System (ADS) 2025 (from main ADS window, Tools->Python Console). The scripts can also run outside of ADS if the interpreter path points to the ADS Python installation ($HPEESOF_DIR/tools/python/python.exe).

The first modeling approach, the “lookup” model, post processes the results from a circuit simulation through an ANN to derive key metrics using a MeasEqn component in ADS. This technique acts as a fast and accurate interpolation function across a multi-dimensional parameter space. The second approach, the “stimulus-response” model utilizes a nonlinear Frequency-Domain Defined Device (FDD) component in ADS combined with ANN equations to build a model which both senses and reacts to the surrounding circuit environment. The example “stimulus-response” model described in this App Note emulates the measured amplifier by outputting a tone at the fundamental frequency which corresponds to the measured output power at a given load and input power value. This model is useful for matching network design and system-level analysis. While the example model depicted here is basic, the concept is extendable to more complex modeling methods.

The Application Note also described how to validate both types of models using test circuits inside Keysight ADS. The validation approach utilized a csv-based batch simulation file which swept the simulation over all the measurement conditions and compared the result with a measured dataset. A simple Python function generated both the batch csv file and the measurement dataset.

Finally, this App Note highlighted a real design example using the “stimulus-response” model with an output network consisting of transmission line and surface mount devices. The ADS optimizer adjusted circuit parameters to maximize power delivered to the load at a single frequency / input power. Limiting functions ensured that the load value presented by the matching network stayed within the valid measurement range. The optimization completed twenty-five iterations in under one second and presented a load which resulted in over 45 dBm of power delivered at 915 MHz.