Use Data Science Techniques in IBIS-AMI Modeling

DesignCon IBIS Summit
Santa Clara, USA
January 31st, 2020

Wei-hsing Huang, SPISim
Wei-hsing.Huang@spisim.com

Agenda:

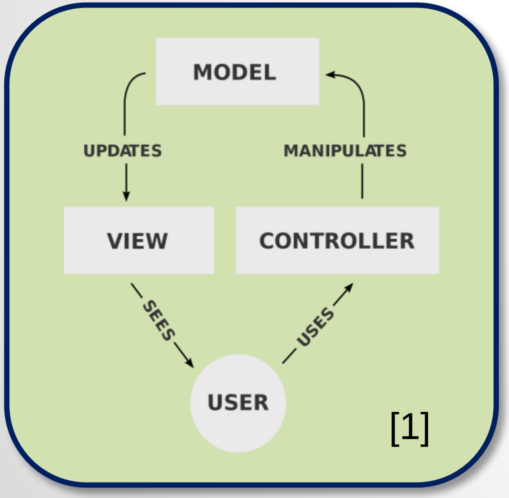
- Motivation
- Background
- TX Example
- RX Example
- Summary
- Q & A

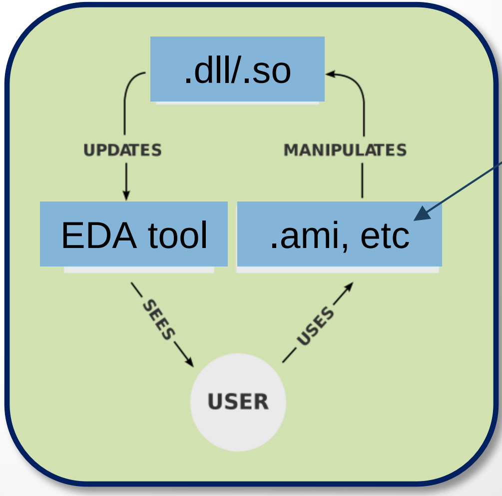
Motivation

- Streamline modeling process
  - Minimize model compilation/testing/revision controls
- Generalize modeling structures
  - An “universal” AMI .dll/.so? (not hard-coded!)
- Explore different modeling techniques
- IBIS-AMI model file (besides .ibs file):
  - .ami: put more info. here (together w/ supporting_files)
  - .dll/.so: minimize changes here

Background:

- Model-View-Control paradigm
  - A software pattern to minimize coupling between components
  - Model(.dll), View(EDA tool) and controller (.ami)
    - E.g. simulator, netlist and waveform viewer in circuit simulation.





“etc” includes “supporting file”, (can be encrypted!)

Parameter:	**Supporting_Files**
Required:	No, and illegal before AMI_Version 6.0
Direction:	Rx, Tx

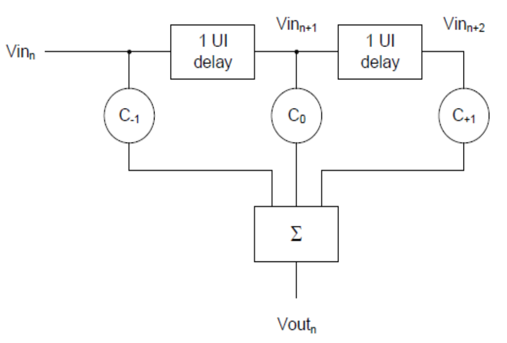
[2]

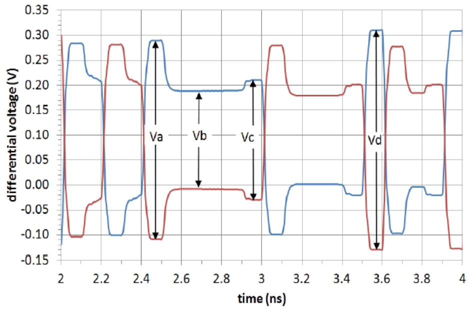
Background:

- AMI/data science modeling process: [3]
  - Define performance measure (inputs and outputs)
  - Get the data (spec., simulation, measurement)
  - Discover/Visualize the data to gain insights
  - Prepare data for modeling (post-processing, data cleaning)
  - Select (and train) model
  - Fine-tune model
  - Deployment (AMI implementation)

TX Example: FIR EQ

- Case 1: spec. has presets (e.g. PCIe/USB-C TX)





Preshoot = 20log(Vc/Vb)
De-emphasis = 20log(Vb/Va)

Table 11. Tx Preset Ratios and Corresponding Coefficient Values

Preset Number	Preshoot (dB)	De-emphasis (dB)	c_{-1}	c_{+1}	Va/Vd	Vb/Vd	Vc/Vd
P4	0.0	0.0	0.000	0.000	1.000	1.000	1.000
P1	0.0	-3.5 ± 1	0.000	-0.167	1.000	0.668	0.668
P0	0.0	-6.0 ± 1.5	0.000	-0.250	1.000	0.500	0.500
P9	3.5 ± 1	0.0	-0.166	0.000	0.688	0.688	1.000
P8	3.5 ± 1	-3.5 ± 1	-0.125	-0.125	0.750	0.500	0.750
P7	3.5 ± 1	-6.0 ± 1.5	-0.100	-0.200	0.800	0.400	0.600
P5	1.9 ± 1	0.0	-0.100	0.000	0.800	0.800	1.000
P6	2.5 ± 1	0.0	-0.125	0.000	0.750	0.750	1.000
P3	0.0	-2.5 ± 1	0.000	-0.125	1.000	0.750	0.750
P2	0.0	-4.4 ± 1.5	0.000	-0.200	1.000	0.600	0.600
P10	0.0	See Note (1)	0.000	See Note (1)	1.000	See Note (1)	See Note (1)

(1) P10 boost limits are not fixed, since its de-emphasis level is a function of the LF level that the Tx advertises during training. See the PCI Express Base Specification 3.0 for more details.

Table 1. Control Pin Settings (Typical Values)

PIN	DESCRIPTION	LOGIC STATE	GAIN	
EQ1/EQ2	Equalization Amount	Low	3 dB	
		Floating	6 dB	
		High	9 dB	
PIN	**DESCRIPTION**	**LOGIC STATE**	**OUTPUT DIFFERENTIAL VOLTAGE FOR THE TRANSISTION BIT**	
OS1/OS2	Output Swing Amplitude	LOW	0.9 Vpp	
		HIGH	1.1 Vpp	
PIN	**DESCRIPTION**	**LOGIC STATE**	**DE-EMPHASIS RATIO**	
			FOR OS = LOW	**FOR OS = HIGH**
DE1/DE2	De-Emphasis Amount	Low	0 dB	0 dB
		Floating	–3.5 dB	–3.5 dB
		High	–6.2 dB	–6.2 dB

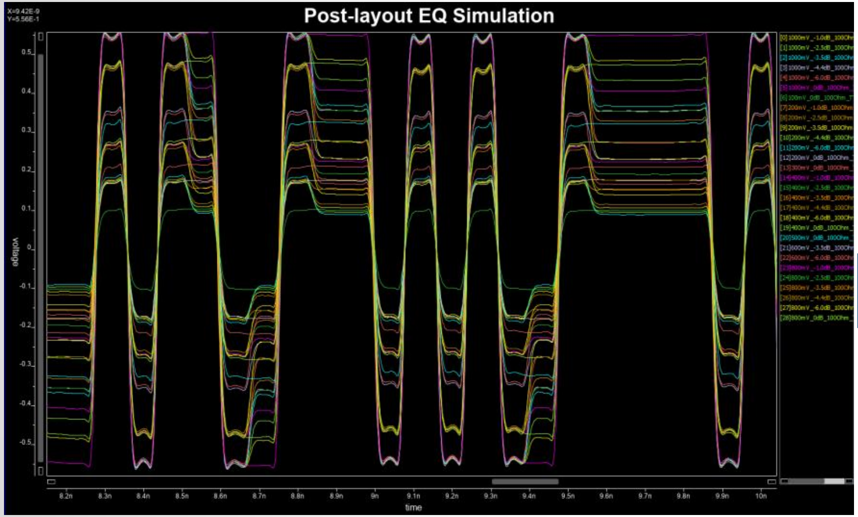
[4]

Modeling = map parameters in data sheet to corresponding numerical values

```
| ------------------ MAIN Settings ------------------
(MDL_SUB_MODS (Usage In) (Type String) (Default "FFE") (Description "Cascaded stages"))
| ------------------- FFE Settings -------------------
(TX_PRESHOOT   (Usage In) (Type String) (Default "0.0") (Description "PreShoot in dB"))
(TX_DEEMPHASIS (Usage In) (Type String) (Default "0.0") (Description "DeEmphasis in dB"))
```

TX Example: FIR EQ

- Case 2: Simulation/Measurement based data:

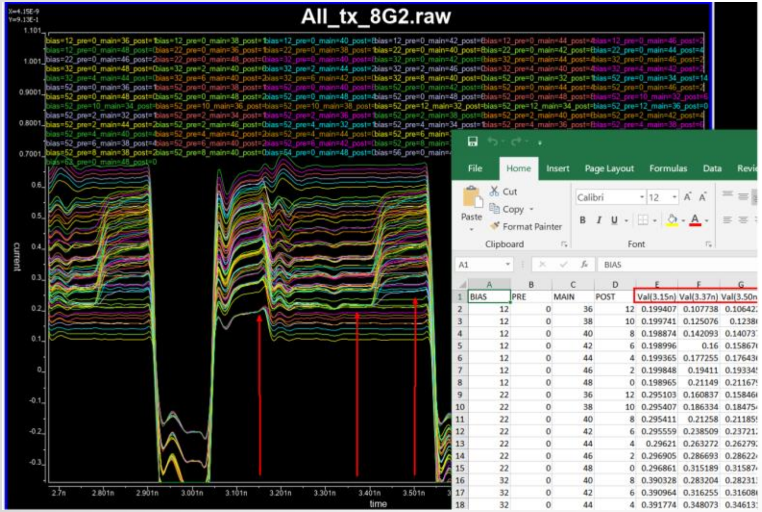


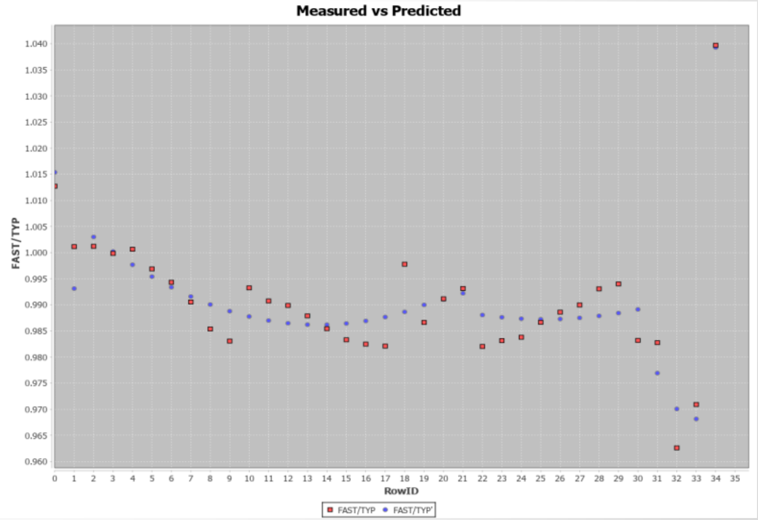
	A	B	C	D	F	G
1	CASEID	TX_RTRIM	TX_PVT	TX_AMPEMPH	PTH_PARM_GAIN	FFE_PARM_POS1
2	1	85	FF	1	0.17793	0
3	2	85	FF	2	0.303125	0
4	3	85	FF	3	0.303125	-0.040625
5	4	85	FF	4	0.303125	-0.160156
6	5	85	FF	5	0.303125	-0.188281
7	6	85	FF	6	0.303125	-0.216406
8	7	85	FF	7	0.303125	-0.2375
9	8	85	FF	8	0.367188	0
10	9	85	FF	9	0.493359	0
11	10	85	FF	10	0.493359	-0.0933594
12	11	85	FF	11	0.493359	-0.149609
13	12	85	FF	12	0.493359	-0.177734
14	13	85	FF	13	0.493359	-0.223438
15	14	85	FF	14	0.493359	-0.248047
16	15	85	FF	15	0.578711	0
17	16	85	FF	16	0.641797	-0.177734
18	17	85	FF	17	0.630664	-0.262109
19	18	85	FF	18	0.860742	0
20	19	85	FF	19	0.860742	-0.0740234

- Data: Use multi-variable look-up table
- Missing row: Linear/bi-linear or Spline/bi-spline interpolation
- Make table external (as one of the supporting files) and encrypt!

TX Example: FIR EQ

- Case 3: Incomplete/insufficient/low quality data
  - Need to create a “prediction” model for missing data
  - E.g. DOE, RSM, regression with regulation to avoid overfit





TX Example: FIR EQ

- Use Python/Sci-kit Learn/Jupyter:
  - Normal equation modeling (e.g. DOE, RSM)
  - Regulation can be used.

```
In [110]:
# Separate input and output attributes
allTars = ['DEEMP', 'PREEMP'']
varList = [e for e in list(eqData) if e not in allTars]
varData = stkData[varList]
```

```
In [111]:
# We have 10,000 cases here, try in-memory normal equation directly first:

# LinearRegression Fit Impedance
from sklearn.linear_model import LinearRegression

tarData = eqData['DEEMP']
lin_reg = LinearRegression()
lin_reg.fit(varData, tarData)

# Fit and check predictions using MSE etc
from sklearn.metrics import mean_squared_error, mean_absolute_error
predict = lin_reg.predict(varData)
resRMSE = np.sqrt(mean_squared_error(tarData, predict))
resRMSE
```

- Support general formula in .dll:

```
**** PREDICTION FORMULA ****
AX+A0*X0+A1*X1+A2*X2+A3*X3+A4*X0*X0+A5*X1*

**** VARIABLE MAPPINGS AND RANGES ****
TAR Y0: FAST/TYP (0.962609 ~ 1.03971)
VAR X0: BIAS (7.00000 ~ 63.0000)
VAR X1: PRE (0.00000 ~ 12.0000)
VAR X2: MAIN (32.0000 ~ 48.0000)
VAR X3: POST (0.00000 ~ 16.0000)

**** CALCULATED COEFFS ****
=== TARGET Y0 ===
AX(CONST.) : 1.29127e-06
A0(For X0) : 1.87086e-05
A1(For X1) : 1.48612e-05
A2(For X2) : 3.19684e-05
A3(For X3) : 1.51513e-05
A4(For X0*X0) : 2.43602e-05
A5(For X1*X1) : 0.000167895
A6(For X2*X2) : 0.000458936
A7(For X3*X3) : 0.000197157
A8(For X0*X1) : 0.000475557
A9(For X0*X2) : -6.23832e-05
A10(For X0*X3) : 0.000484841
A11(For X1*X2) : 0.000545441
A12(For X1*X3) : 0.00000
A13(For X2*X3) : 0.000530104
MSE(Y0) = 1.78553e-05
```

RX Example: CTLE

- Case 1: spec. defined presets

USB-C Gen 1:

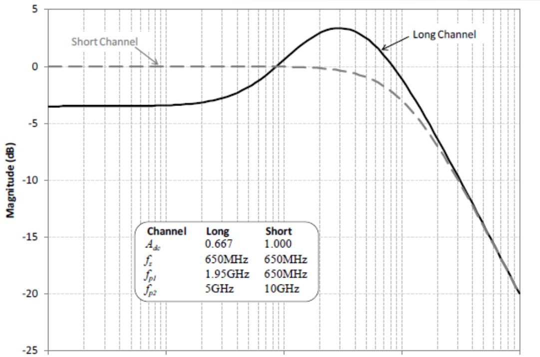
(10) $$H(s) = \frac{A_{dc}\omega_{p1}\omega_{p2}}{\omega_z} \cdot \frac{s+\omega_z}{(s+\omega_{p1})(s+\omega_{p2})}$$

where A_{dc} is the DC gain

$\omega_z = 2\pi f_z$ is the zero frequency

$\omega_{p1} = 2\pi f_{p1}$ is the first pole frequency

$\omega_{p2} = 2\pi f_{p2}$ is the second pole frequency



USB-C Gen 2:

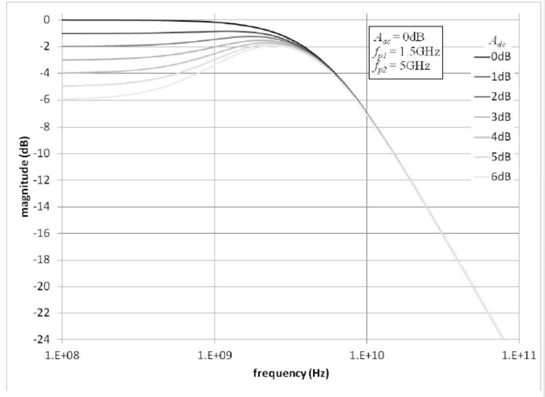
(11) $$H(s) = A_{ac}\omega_{p2}\frac{s+\frac{A_{dc}}{A_{ac}}\omega_{p1}}{(s+\omega_{p1})(s+\omega_{p2})}$$

where A_{ac} is the high frequency peak gain

A_{dc} is the DC gain

$\omega_{p1} = 2\pi f_{p1}$ is the first pole frequency

$\omega_{p2} = 2\pi f_{p2}$ is the second pole frequency



RX Example: CTLE

- CTLE Equation->Bi-Linear Transform
- Poles/Zeros -> FD Response -> TD Response

USB-C Gen 1:

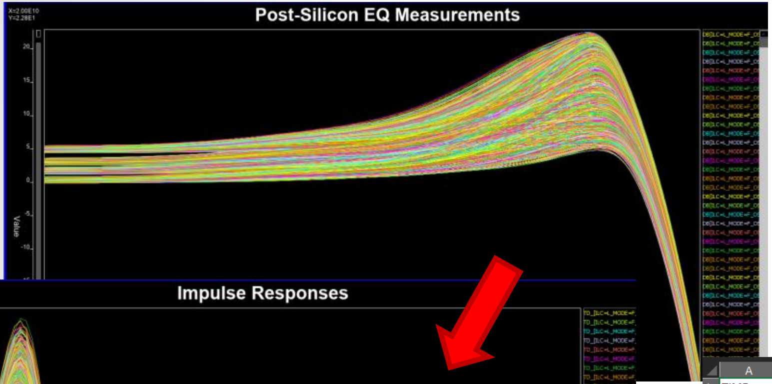
```
| ------------------ MAIN Settings ------------------
(MDL_SUB_MODS (Usage In) (Type String) (Default "CTLE") (Description "Cascaded stages"))
| ------------------ CTLE Settings ------------------
(Adc (Usage In) (Type Float) (Format List 0.0 -3.5) (Default 0.0) (Description "DC gain in dB"))
(fz (Usage In) (Type Float) (Format List 650e6 650e6) (Default 650e6) (Description "zero frequency in Hz"))
(fp1 (Usage In) (Type Float) (Format List 650e6 1.95e9) (Default 650e6) (Description "pole frequency 1 in Hz"))
(fp2 (Usage In) (Type Float) (Format List 10.0e9 5.0e9) (Default 10.0e9) (Description "pole frequency 2 in Hz"))
```

USB-C Gen 2:

```
| ------------------ MAIN Settings ------------------
(MDL_SUB_MODS (Usage In) (Type String) (Default "CTLE,DFECDR") (Description "Cascaded stages"))
| ------------------ CTLE Settings ------------------
(Aac (Usage In) (Type Float) (Format Value 0.0) (Default 0.0) (Description "AC gain in dB"))
(Adc (Usage In) (Type Float) (Format List 0.0 -1.0 -2.0 -3.0 -4.0 -5.0 -6.0) (Default 0.0) (Description "DC gain in dB"))
(fp1 (Usage In) (Type Float) (Format Value 1.5e9) (Default 1.5e9) (Description "pole frequency 1 in Hz"))
(fp2 (Usage In) (Type Float) (Format Value 5.0e9) (Default 5.0e9) (Description "pole frequency 2 in Hz"))
```

RX Example: CTLE

- Case 2: Silicon based measurements:



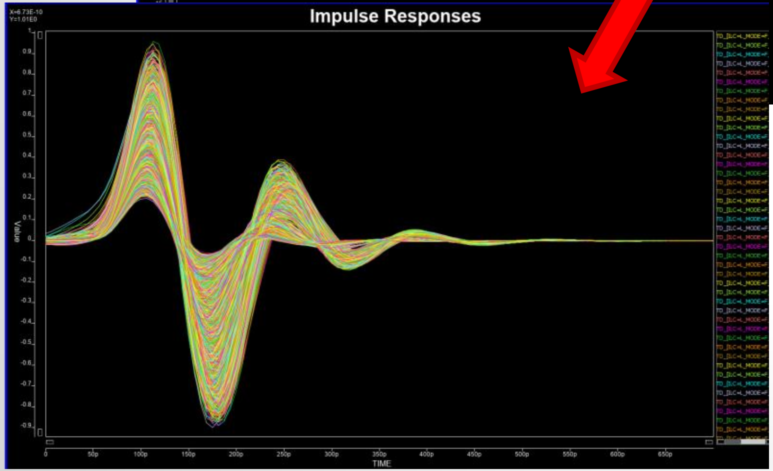


Table 3. EQUALIZATION SETTINGS:

EQA/ EQB	EQ (dB)	
	@ 2.5 GHz	@ 5 GHz
Low "L" (Pin tied to Ground)	5.0	11.5
Rext "R" (68 kΩ tied from pin to Ground)	2.7	7.4
FLOAT "F" (Pin open)	4.0	9.9 (Default)
HIGH "H" (Pin tied to V_{DD})	6.5	13.1

Look-up Table

Table 4. FLAT GAIN SETTING

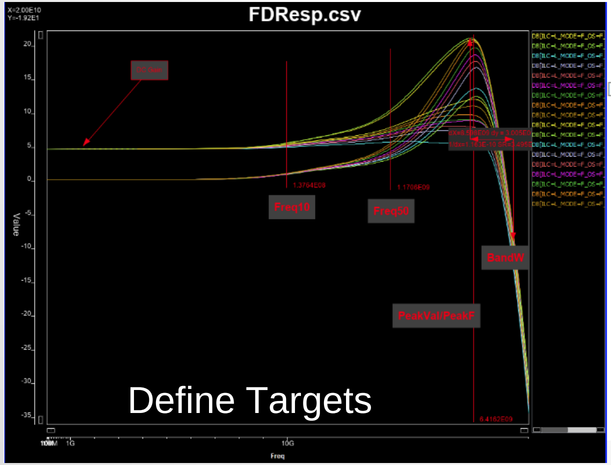
FGA/ FGB	FG (dB)
Low "L" (Pin tied to Ground)	–1.2
Rext "R" (68 kΩ tied from pin to Ground)	0
FLOAT "F" (Pin open)	+1 .0 (Default)
HIGH "H" (Pin tied to V_{DD})	+2.0

	A	B	C	D	E						
1	TIME	TD_[ILC=L	TD_[ILC=L	TD_[ILC=L	TD_[ILC=L						
2	0	-3.21E-04	-3.46E-04	-3.49E-04	-3.24						
3	6.25E-12	-2.89E-04	-3.26E-04	-3.78E-04	-3.67E						
4	1.25E-11	-1.05E-04	-1.39E-04	-2.30E-04	-2.35E-	-2.					
5	1.87E-11	1.41E-04	1.22E-04	4.67E-06	-1.36E-05	-4.					
6	2.50E-11	3.21E-04	3.25E-04	1.88E-04	1.61E-04	1.					
7	3.12E-11	3.45E-04	3.66E-04	2.13E-04	1.82E-04	1.71E-04	1.38E-04	1.52E-04	1.63E-04	2.43E-04	2.69E-04
8	3.75E-11	2.11E-04	2.38E-04	7.01E-05	3.84E-05	3.02E-05	7.24E-06	3.03E-05	4.29E-05	1.11E-04	1.26E-04
9	4.37E-11	1.24E-05	3.19E-05	-1.44E-04	-1.76E-04	-1.87E-04	-2.06E-04	-1.81E-04	-1.69E-04	-3.75E-05	-3.69E-05
10	5.00E-11	-1.21E-04	-1.15E-04	-2.80E-04	-3.13E-04	-3.30E-04	-3.48E-04	-3.27E-04	-3.19E-04	-1.06E-04	-1.16E-04
11	5.63E-11	-1.00E-04	-1.05E-04	-2.33E-04	-2.65E-04	-2.85E-04	-2.99E-04	-2.84E-04	-2.82E-04	-4.99E-05	-6.05E-05
12	6.25E-11	6.01E-05	5.55E-05	-6.38E-06	-3.52E-05	-4.84E-05	-5.16E-05	-4.10E-05	-4.37E-05	9.36E-05	9.36E-05
13	6.87E-11	2.46E-04	2.51E-04	2.74E-04	2.55E-04	2.55E-04	2.72E-04	2.82E-04	2.75E-04	2.23E-04	2.37E-04
14	7.50E-11	3.09E-04	3.26E-04	4.33E-04	4.30E-04	4.47E-04	4.86E-04	4.97E-04	4.87E-04	2.34E-04	2.54E-04
15	8.12E-11	1.65E-04	1.84E-04	3.57E-04	3.73E-04	4.00E-04	4.54E-04	4.63E-04	4.50E-04	8.91E-05	9.99E-05
16	8.75E-11	-1.42E-04	-1.37E-04	7.28E-05	1.06E-04	1.31E-04	1.87E-04	1.85E-04	1.67E-04	-1.51E-04	-1.62E-04
17	9.38E-11	-4.49E-04	-4.70E-04	-2.51E-04	-2.08E-04	-1.93E-04	-1.52E-04	-1.72E-04	-1.96E-04	-3.49E-04	-3.84E-04
18	1.00E-10	-5.61E-04	-6.07E-04	-3.95E-04	-3.51E-04	-3.50E-04	-3.30E-04	-3.70E-04	-4.00E-04	-3.74E-04	-4.20E-04
19	1.06E-10	-3.75E-04	-4.27E-04	-2.28E-04	-1.91E-04	-1.96E-04	-1.90E-04	-2.45E-04	-2.78E-04	-1.81E-04	-2.18E-04
20	1.12E-10	4.45E-05	1.11E-05	1.97E-04	2.25E-04	2.27E-04	2.32E-04	1.78E-04	1.46E-04	1.43E-04	1.32E-04
21	1.19E-10	4.81E-04	4.86E-04	6.51E-04	6.71E-04	6.91E-04	7.10E-04	6.70E-04	6.47E-04	4.23E-04	4.41E-04
22	1.25E-10	6.87E-04	7.31E-04	8.53E-04	8.67E-04	9.05E-04	9.43E-04	9.26E-04	9.15E-04	4.93E-04	5.26E-04

[5]

RX Example: CTLE

- Case 3: Performance based models: [6]
    - Specify CTLE performance at DC, 5GHz, 10GHz, etc.
    - Generate behavioral CTLE model dynamically
        - Map performance matrices to poles/zeros/gain
        - Very useful in architectural/planning stage



Prepare Data:

[42]:

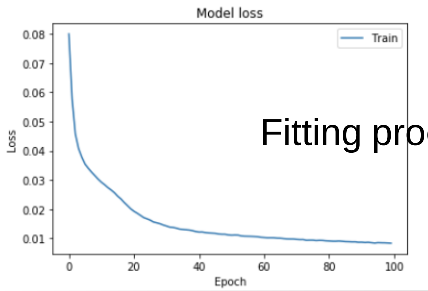
```
## Using COM CTLE as an example below:

# Environment Setup:
import os
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

prjHome = 'C:/Temp/WinProj/CTLEMdl'
workDir = prjHome + '/wsp/'
srcFile = prjHome + '/dat/COM_CTLEData.csv'

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(workDir, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Measured or synthesized data

RX Example: CTLE

- Neural network model fit
  - NOT machine learning (no GPU, not “deep”)
  - Keras is used for Cpp translation for AMI

In [60]:
```python
# plot history
plt.plot(hist.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Fitting process

Create model

In [56]:
```python
tarList = ['Gdc', 'P1', 'P2', 'Z1']
varList = ['Gain', 'PeakF', 'PeakVal', 'Freq10', 'Freq50']

varData = mdlData[varList]
tarData = mdlData[tarList]
```

In [57]:
```python
from keras.models import Sequential
from keras.layers import Dense, Dropout

numVars = len(varList)  # independent variables
numTars = len(tarList)  # output targets
nnetMdl = Sequential()
# input layer
nnetMdl.add(Dense(units=64, activation='relu', input_dim=numVars))

# hidden layers
nnetMdl.add(Dropout(0.3, noise_shape=None, seed=None))
nnetMdl.add(Dense(64, activation = "relu"))
nnetMdl.add(Dropout(0.2, noise_shape=None, seed=None))

# output layer
nnetMdl.add(Dense(units=numTars, activation='sigmoid'))
nnetMdl.compile(loss='mean_squared_error', optimizer='adam')

# Provide some info
#from keras.utils import plot_model
#plot_model(nnetMdl, to_file= workDir + 'model.png')
nnetMdl.summary()
```

Model fit

[58]:
```python
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Prepare Training (tran) and Validation (test) dataset
varTran, varTest, tarTran, tarTest = train_test_split(varData, tarData, test_size=0.2)

# scale the data
from sklearn import preprocessing
varScal = preprocessing.MinMaxScaler()
varTran = varScal.fit_transform(varTran)
varTest = varScal.transform(varTest)

tarScal = preprocessing.MinMaxScaler()
tarTran = tarScal.fit_transform(tarTran)
```

[59]:
```python
# model fit
hist = nnetMdl.fit(varTran, tarTran, epochs=100, batch_size=1000, validation_split=0.1)
tarTemp = nnetMdl.predict(varTest, batch_size=1000)
#predict = tarScal.inverse_transform(tarTemp)
#resRMSE = np.sqrt(mean_squared_error(tarTest, predict))
resRMSE = np.sqrt(mean_squared_error(tarScal.transform(tarTest), tarTemp))
resRMSE
```

RX Example: CTLE

- Export for deployment
  - Cpp for AMI modeling…

Deployment:

Now that we have trained model in Keras' .h5 format, we can translate this model into corresponding cpp codes using Keras2Cpp:

keras2cpp [7]

This is a bunch of code to port Keras neural network model into pure C++. Neural network weights and architecture are stored in plain text file and input is presented as `vector<vector<vector<float> > >` in case of image. The code is prepared to support simple Convolutional network (from MNIST example) but can be easily extended. There are implemented only ReLU and Softmax activations.

It is working with the Theano backend - support for Tensorflow will be added soon.

Usage

1. Save your network weights and architecture.
2. Dump network structure to plain text file with `dump_to_simple_cpp.py` script.
3. Use network with code from `keras_model.h` and `keras_model.cc` files - see example below.

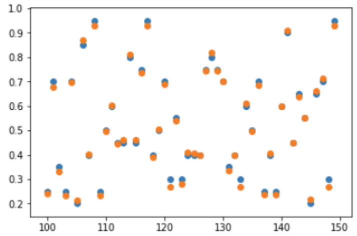
Its github repository is here: Keras2Cpp

Resulting file can be compiled together with keras_model.cc, keras_model.h in our AMI library.

In [62]:
```
# generate prediction
predict = tarScal.inverse_transform(tarTemp)
allData = np.concatenate([varTest, tarTest, predict], axis = 1)
allData.shape
headLst = [varList, tarList, tarList]
headStr = ''.join(str(e) + ',' for e in headLst)
np.savetxt(workDir + 'COMCtleIOP_Rev.csv', allData, delimiter=',', header=headStr)
```

In [63]:
```
# plot Gdc
begIndx = 100
endIndx = 150
indxAry = np.arange(0, len(varTest), 1)
plt.scatter(indxAry[begIndx:endIndx], tarTest.iloc[:,0][begIndx:endIndx])
plt.scatter(indxAry[begIndx:endIndx], predict[:,0][begIndx:endIndx])
```

Out[63]: <matplotlib.collections.PathCollection at 0x242ccefe1d0>



Correlation

```
# Separated Keras' architecture and synopse weight for later Cpp conversion
from keras.models import model_from_json
# serialize model to JSON
nnetMdl_json = nnetMdl.to_json()
with open("COM_nnetMdl_Rev.json", "w") as json_file:
    json_file.write(nnetMdl_json)
# serialize weights to HDF5
nnetMdl.save_weights("COM_nnetMdl_W_Rev.h5")

# save model and architecture to single file
nnetMdl.save(workDir + "COM_nnetMdl_Rev.h5")
print("Saved model to disk")

# also save scaler
from sklearn.externals import joblib
joblib.dump(varScal, workDir + 'Rev_VarScaler.save')
joblib.dump(tarScal, workDir + 'Rev_TarScaler.save')
```

Export…

Summary:

- MVC Paradigm:
  - Decouple model (.dll) with controller (.ami)
  - Minimized unnecessary compilation/testing/revision
    - Avoid “magic numbers” in the .dll/.so
    - Use “supporting_files” (encrypted?) for modeling parameters
    - Leave .ami for matching parameters to datasheets

- Generalized modeling flow:
  - Open source modeling tools: Python, Scikit-Learn, Keras, Jupyter…
  - Very similar flow to data science. NO GPU is involved!
  - Same SI/PI modeling techniques (DOE, RSM, Neural Network etc)
  - Enable model support in the .dll, model parameters in .ami
  - Data science’s techniques can be used for AMI modeling!

Reference:

1. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
2. IBIS Version 7.0 Specification. Subsection 10.5
3. Hands-On Machine Learning with Scikit-Learn and TensorFlow (**ISBN-13:** 978-1491962299)
4. USB3 data sheet (http://www.ti.com/lit/ds/symlink/tusb522p.pdf)
5. Redriver data sheet (https://www.onsemi.com/pub/Collateral/NB7NPQ1102M-D.PDF)
6. Spec-Driven CTLE Model Synthesis through Reinforcement Learning, Daniel Wu, Xilinx, DesignCon 2019
7. Keras2cpp (https://github.com/pplonski/keras2cpp)

SPISIM

EDA Expertise in Signal, Power Integrity & Simulation