2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Reducing the simulation life cycle time of Fault Simulations using Artificial Intelligence and Machine Learning techniques on Big Data dataset

Darshan Sarode, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (darshan@siliconinterfaces.com)
Pratham Khande, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (pratham@siliconinterfaces.com)
Priyanka Gharat, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (priyanka@siliconinterfaces.com)

***Abstract*— This paper presents an application of AI/ML algorithms to reduce fault simulation cycles from standard fault simulation techniques where each node is switched with Stuck@0/Stuck@1 by generation, testability and simulation to reduce simulation cycles. AI/ML techniques are used to partition sample datasets into training/test datasets, build/compile a ML model, and perform model fitting using standard AI packages, optimization techniques, and activation/loss functions for error reduction to predict outputs. The hypothesis is based on the predicted match results run through different sized datasets and hyper-parameters, resulting in a ML model with 20% of the predicted test data results, similar to error simulation.**

Keywords*— *Machine Intelligence, Machine Learning, ML, Artificial Intelligence, AI, TensorFlow, Keras, Hidden Layers, Functional Verification; Fault Simulation, Testability, Controllability, Observability, GM, FM, Fault Status, EDA tool, Concurrent, Distributed Computing Introduction.

I. INTRODUCTION

The objective of this paper is to improve fault simulation by utilizing intelligent and learning algorithms, which will lead to better testing quality and fault coverage, as well as shorter simulation cycles. While incorporating the standard fault simulation methods that are widely used in the industry, the paper aims to enhance the effectiveness of these techniques with the help of intelligent algorithms.

During the fault generation phase, the SFF fault list is created through fault simulation, which covers various nodes of the design like PORTS, FLOP, WIRE, VARI, and ARRAY. The generated fault list is then analyzed through machine learning to classify the faults as potentially detected or not detected. Stuck-At faults are utilized in different parts of the circuitry, such as input/output terminals, module connections, nets, arrays, and continuous assignments. To detect faults, the good machine is compared to the faulty machine using divergence and convergence techniques in concurrent, serial, and parallel modes. The fault coverage is increased by applying strobes to the generated fault list, including transient faults that occur once and then disappear. Intelligent algorithms and machine learning techniques are used to enhance fault detection by analyzing the fault status of the generated fault list. This allows the identification of nodes that can be dropped and detected, as well as divergence between good and faulty machines. Machine learning also shortens simulation cycles, increasing efficiency. To summarize, the application of AI/ML techniques in the fault simulation process improves the accuracy and efficiency of fault detection. This can identify faults that may be missed by traditional methods, and simulation cycles can be reduced to achieve better results in less time.

II. FAULT SIMULATION

Fault simulation is a powerful technique that offers numerous benefits to the device development process. It can be utilized at different stages of the device lifecycle, providing valuable insights and helping to ensure high-quality devices. One of its most significant advantages is its ability to identify systematic errors in the verification environment. By detecting such errors, developers can make more informed decisions and improve the overall effectiveness of the device development and verification process. In addition to development, fault simulation is also essential in manufacturing. It enables the assessment of the necessary coverage needed to produce SoCs, ASICs, or IPs of high quality. By using this technique, manufacturers can identify potential issues early on, ensuring that their products meet the highest standards. Furthermore, fault simulation can evaluate how well a mechanism handles random failures during the operational phase. With the use of an EDA tool, common error sources can be replicated and analyzed, allowing for a reliable and efficient method of detecting

potential issues. This is particularly important as it helps ensure that devices can effectively handle a range of potential failures, making them more resilient and reliable.

```
pcie_pll pll(.rcep(rcep));

pcie_s2p s2p(.rcep(rcep),.par_reg(par_reg),.par_read_done(par_read_done));

pcie_elastic_buffer eb(.rcep(rcep),.par_reg(par_reg),.par_read_done(par_read_done),.mem_elastic_fifo_lane0(mem_elastic_fifo_lane0),.mem_elastic_fifo_lane1(mem_elastic_f
ifo_lane1),.mem_elastic_fifo_lane2(mem_elastic_fifo_lane2),.mem_elastic_fifo_lane3(mem_elastic_fifo_lane3),.buf_full(buf_full));

pcie_phy_cdr_ba ba(.rcep(rcep),.mem_elastic_fifo_lane0(mem_elastic_fifo_lane0),.mem_elastic_fifo_lane1(mem_elastic_fifo_lane1),.mem_elastic_fifo_lane2(mem_elastic_fifo_
lane2),.mem_elastic_fifo_lane3(mem_elastic_fifo_lane3),.buf_full(buf_full),.ba_out_lane0(ba_out_lane0),.ba_out_lane1(ba_out_lane1),.ba_out_lane2(ba_out_lane2),.ba_out_l
ane3(ba_out_lane3));

pcie_phy_cdr_skew skew();

decoder_10x8 d0(.rcep(rcep),.ba_out_lane0(ba_out_lane0),.ba_out_lane1(ba_out_lane1),.ba_out_lane2(ba_out_lane2),.ba_out_lane3(ba_out_lane3),.dec_out_lane0(dec_out_lane0
),.dec_out_lane1(dec_out_lane1),.dec_out_lane2(dec_out_lane2),.dec_out_lane3(dec_out_lane3),.dec_done_lane(dec_done_lane));

pcie_byte_unstrip us(.rcep(rcep),.dec_out_lane0(dec_out_lane0),.dec_done_lane(dec_done_lane),.dec_out_lane1(dec_out_lane1),.dec_out_lane2(dec_out_lane2),.dec_out_lane3(
dec_out_lane3),.packet_out_lane0(packet_out_lane0),.packet_out_lane1(packet_out_lane1),.packet_out_lane2(packet_out_lane2),.packet_out_lane3(packet_out_lane3),.pkt_coll
ect_lane0(pkt_collect_lane0),.pkt_collect_lane1(pkt_collect_lane1),.pkt_collect_lane2(pkt_collect_lane2),.pkt_collect_lane3(pkt_collect_lane3));

endmodule
```

Figure II(a). CDR block of PCIe bus.

In this paper, we have used PCIe bus. The fault has been injected at the port level (i.e. input and output locations) with all the different vector lists. Below is a snippet of the DUT design in which the faults have been injected,

Below is the fault coverage report generated for a Faulty/Safe machine at the Fsim stage.

```
#------------------------------------------------------------------------
# Fault Coverage Summary
#
#                                      Prime             Total
#------------------------------------------------------------------------
# Total Faults:                         2646              3424
#
# Dropped Detected                DD       0   0.00%         0   0.00%
# Dropped Potential               PD       0   0.00%         0   0.00%
# Hyperactive                     HA       4   0.15%         4   0.12%
# Not Detected                    ND    2642  99.85%      3420  99.88%
#
# Untestable Unused               UU    1114              1114
#
# Hyper                           HG       4   0.15%         4   0.12%
# Untestable                      UG    1114  42.10%      1114  32.54%
```

Figure II(b). Coverage Report generated for Faulty/Safe Machine

III. Machine Learning Application on Big Data Dataset

Using a Dataset for PCIe bus in the current paper it can also be tried on more complex designs, AI/ML Model has been developed with 4 hidden neural network hidden layers using statistical and probabilistic functions from pandas and ML libraries, like TensorFlow function Keras and studying the effect of hyperparameters, optimizers, and activation functions. For this data set, we have activation function “relu” and optimizer selected “Adam” with the learning rate of 0.001 further the loss function is “MSE” on train data, this model is fitted to train data, and the model is applied to test data for actual versus predicted values which are plotted on heat maps showing almost 95% accuracy.

In the first pass, we are importing standard and routine python libraries mentioned below to carry out for mathematical, statistical, graph plotting, and machine learning

```
%tensorflow_version 2.x
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
```

Along with the libraries, the file including the test vector data in csv format of the PCIe is passed as all the operation of sampling the 100% data to 20% data is being carried out on it.

```
filename = '/content/PCIe_ML_Output.csv'
```

csv_data = pd.read_csv(filename)

A. *The dataset was split into 80:20 for train data and test data (this may be in other ratios)*

Further, the data in the test vector is sampled in 80% as train data and 20% as test data ratio.

```
#----------TRAIN/TEST SPLIT
train_data = csv_data.sample(frac=0.8) # take 80% randomly from the data
test_data = csv_data.drop(train_data.index) # reserve the rest for testing

# separate out the y (results) from x (features)
x_train = train_data.drop('packet_out_lane0[0]', axis=1)
y_train = train_data['packet_out_lane0[0]']

print(x_train)
print(y_train)
# separate out the y (results) from x (features)
x_test = test_data.drop('packet_out_lane0[0]', axis=1)
y_test = test_data['packet_out_lane0[0]']

print(y_test)

print(x_test)

print('Training Data\n', x_train.describe().transpose())
print('Test Data\n', x_test.describe().transpose())
```

Figure III(a). Code to extract 20% data randomly.

B. *A regression ML model was built using TensorFlow ML libraries*

```
#--------MODEL BUILDING
num_params = len(x_train.keys())
print(num_params)
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer([num_params], name="Input_Layer"),
    tf.keras.layers.Dense(256, activation='relu', name="dense_01"),
    tf.keras.layers.Dense(256, activation='relu', name="dense_02"),

    tf.keras.layers.Dense(256, activation='relu', name="dense_03"),
    tf.keras.layers.Dense(256, activation='relu', name="dense_04"),

    tf.keras.layers.Dense(1 ,activation='sigmoid', name="Output_Layer")
  ])


learning_rate = 0.001
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              # loss function to minimize
                 loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              # list of metrics to monitor
              metrics=['acc',])

model.summary()
```

Figure III(b). Code to build a ML model using TensorFlow ML Libraries.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_01 (Dense)             (None, 256)               142336

 dense_02 (Dense)             (None, 256)               65792

 dense_03 (Dense)             (None, 256)               65792

 dense_04 (Dense)             (None, 256)               65792

 Output_Layer (Dense)         (None, 1)                 257

=================================================================
Total params: 339,969
Trainable params: 339,969
Non-trainable params: 0
```

Figure III(c). Dense Layer Result.

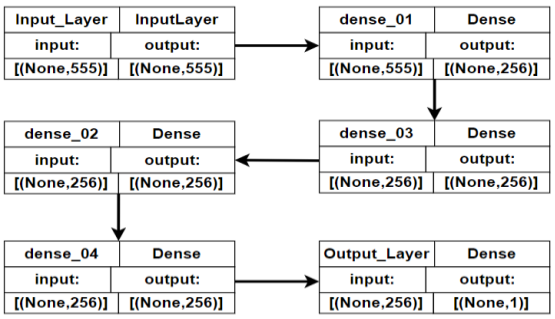

Figure III(d). Plot Model

C. The model was fitted to the training data

In step three the generated model is being mapped/fitted with/to training data using transpose functions

```
#--------PREDICT
p_test = model.predict(x_test)
p=np.rint(p_test).transpose()
q = p.astype(int).transpose()
print("Predicted Class:\t", q ,
      '\nActuals:\t\t ', y_test.to_frame().T
      .to_string(header=None, index=False))
```

Figure III(e). Code to Predict using transpose functions.

IV. PRELIMINARY RESULT

Using Python, NumPy, pandas, TensorFlow, Etc, and Colab from Google® the above stochastic behaviours were implemented by using a Data Set from Simulation Outputs for PCIe Bus.

A. At the first pass, we fit the model on train data over several epoch runs.

```
# Fit/Train model on training data
history = model.fit(x_train, y_train,
                    batch_size=5,
                    epochs=100,
                    validation_split=0.1,
                    verbose=1)
```

Figure IV(a). Code to train model on training data.

Using the model.fit function the train data ie. 80% sampled data is being mapped over train data. In this basically, the data is being evaluated on basis of the accuracy, val_loss and val_acc.

```
Epoch 1/100
37/37 [==============================] - 1s 14ms/step - loss: 9.9895e-09 - acc: 1.0000 - val_loss: 2.5215e-09 - val_acc: 1.0000
Epoch 2/100
37/37 [==============================] - 0s 12ms/step - loss: 9.7727e-09 - acc: 1.0000 - val_loss: 2.4764e-09 - val_acc: 1.0000
Epoch 3/100
37/37 [==============================] - 0s 12ms/step - loss: 9.5642e-09 - acc: 1.0000 - val_loss: 2.4343e-09 - val_acc: 1.0000
Epoch 4/100
37/37 [==============================] - 0s 11ms/step - loss: 9.3610e-09 - acc: 1.0000 - val_loss: 2.4044e-09 - val_acc: 1.0000
Epoch 5/100
37/37 [==============================] - 0s 9ms/step - loss: 9.1613e-09 - acc: 1.0000 - val_loss: 2.3599e-09 - val_acc: 1.0000
```

Figure IV(b). Epoch Results.

From the Epoch results, it is seen that with help of sampled 20% Train data that both loss and validation loss functions are generally reduced.

B. Data is being plotted against loss and epoch

To monitor the Train data against the loss of information A graph is plotted to get clarity in graphical format.

```
#--------MONITOR
# Plot training & validation loss values
fig = plt.figure(figsize=(12,7))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()
```

Figure IV(c). Code to monitor data using matplotlib library.

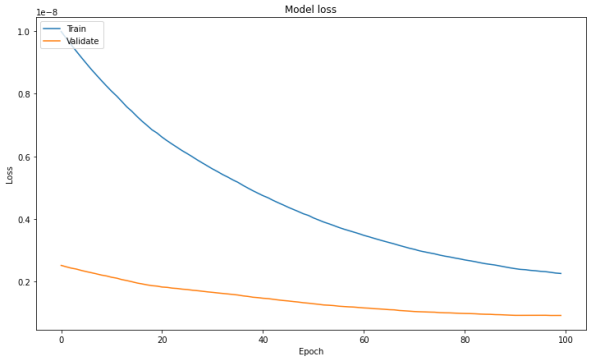

Figure IV(d). Model loss.

C. Further data is being evaluated and listed in form of loss and accuracy on test data.

#--------EVALUATE
loss, acc = model.evaluate(x_test, y_test, verbose=1)
print('Loss:', loss, 'Accuracy:', acc)

2/2 [==============================] - 0s 8ms/step - loss: 8.6777e-10 - acc: 1.0000
Loss: 8.677686902380799e-10 Accuracy: 1.0

We observed loss is very low Loss and accuracy is 100%.

D. Finally, The predicted data is being compared with actual data.

```
#--------PREDICT
p_test = model.predict(x_test)
p=np.rint(p_test).transpose()
q = p.astype(int).transpose()
print("Predicted Class:\t", q ,
      '\nActuals:\t\t ', y_test.to_frame().T
      .to_string(header=None, index=False))
```

Figure IV(e). Code to Predict using transpose functions.

```
[[1][1][0][1][0][0][0][0][0][0][0][1][0][0][1][1][0][1][1][0][0][0][1][1][1][1][0
][1][0][0][0][1][0][1][1][1][0][1][0][0][0][1][1][1][0][1][0][0][1][0]]
Actuals:     1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1
1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0
```

Below it seems that the predicted data and actual data are quite similar. Which is being further validated with the heatmap.
This model has given high matched results; further study will yield better results.
sns.heatmap(tf.math.confusion_matrix(y_test,q),cmap="Blues", annot=True)

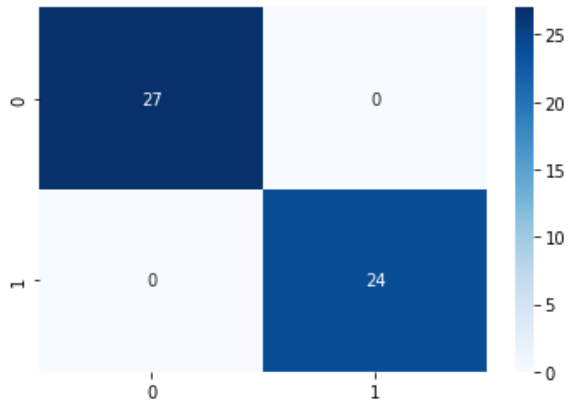

Figure IV(f). Heat Map graph generated from the Colab Tool

Further, the 20% sample data is being generated from the tensor flow AI ML platform using below statement,

train_data = csv_data.sample(frac=0.8)

Below is the sampled data.

Column 1	par_reg[7]	data_size_lane0[0]	pattern_check_reg[1]	mem_elastic_fifo_lane0[2]	ba_out_lane0[0]	dec_out_lane0[1]	packet_out_lane0[3]
9	1	0	1	1	0	1	0
19	1	0	0	0	0	0	0
23	1	0	0	1	0	0	0
28	0	0	1	0	0	1	0
29	1	1	1	0	1	1	0
40	0	0	1	1	0	1	0
...	...	...	...	...	...	...	...
224	0	0	1	1	0	1	1
236	0	0	1	0	0	1	1
237	1	1	1	0	1	1	1
246	0	0	0	1	0	0	1

Table 1. 20% Sampled Data.

And the same vector list (input) is being injected as a fault in PCIe design of Fault Simulation. This 20% sampled input vectors are simulated using the fault campaign manager which resulted in more count of DD i.e., Dropped Detected faults. Below is the fault simulation coverage report generated for 20% of sampled input data.

```
#------------------------------------------------------------------------
# Fault Coverage Summary
#
#                                        Prime             Total
#------------------------------------------------------------------------
# Total Faults:                           2646              3424
#
# Dropped Detected                DD         0   0.00%         0   0.00%
# Dropped Potential               PD         0   0.00%         0   0.00%
# Hyperactive                     HA        40   1.51%        40   1.17%
# Not Detected                    ND      2606  98.49%      3384  98.83%
#
# Untestable Unused               UU      1114              1114
#
# Hyper                           HG        40   1.51%        40   1.17%
# Untestable                      UG      1114  42.10%      1114  32.54%
#
```

Figure IV(g). Coverage Report from the 20% Generated Data.

V. CONCLUSION

The results generated by the Fault Manager in the Fsim phase from data provided via the ML model were tested on 20% of the test data from a test file after applying the test vectors of the AI ML engine, which can lead to predicted results similar to the fault simulation. The PCIe bus example used here resulted in more detectable faults (2→3) being identified from the 20% of test vector data. In addition, an error that was labelled as ND (not detected) is now detected as DD (dropped detected), which is a better categorization. These results are encouraging and warrant further investigation, perhaps with a larger data set using the ML engine, which can help address defects that arise early in the manufacturing process. Fault simulation time would be reduced by a large time factor and help in testing large ASICS and SOCs.

REFERENCES

[1] Siggraph Machine Learning & Neural Networks with Rajesh Sharma (SIGGRAPH Now | Hands-on Workshop: Machine Learning and Neural Networks – Lecture 1 - YouTube))

[2] https://spdocs.synopsys.com/dow_retrieve/latest/home_public/vc_z01x.html.

[3] https://spdocs.synopsys.com/dow_retrieve/latest/home_public/Z01X.html

[4] https://www.accellera.org/downloads/ieee IEEE 1800-2017: SystemVerilog (SV)

[5] SystemVerilog 3.1a Language Reference Manual Accellera's Extensions to Verilog

[6] Functional Safety Verification Challenges for Automotive ICs | Verification Horizons - July 2022 | Verification Academy.