2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

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

Darshan Sarode, Pratham Khande and Priyanka Gharat

Silicon Interfaces, Mumbai, India

Silicon Interfaces®

accellera SYSTEMS INITIATIVE

Machine learning in Functional Safety of Design

"Measure twice, cut once"

- Time taken for Simulation of large count of fault list.
- There is need to automate this as much as possible.
- Machine learning can be used to a great extent for automation in this regard.

What is Fault Simulation?

Uncovering design weaknesses and strengthening functional verification for error-free systems

- Functional verification validates designs based on stimulus integrity.
- Fault Simulation analyses potential stimulus failure due to defects/environmental factors.
- It develops a fault free design (minimum faults) at the initial stage of pre-manufacturing of an SOC

Benefits of Fault Simulation

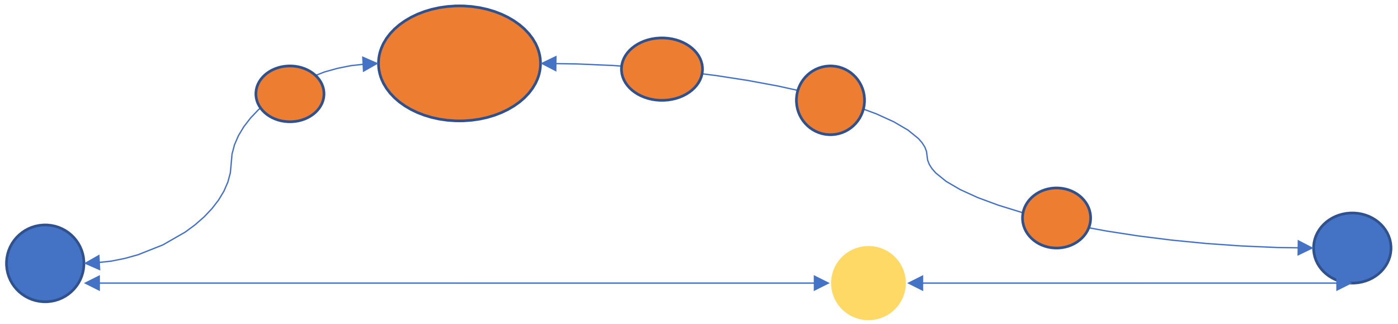
- Comprehensive Fault Analysis
- Efficient Fault Injection
- Accurate Fault Propagation
- Advanced Debugging Capabilities
- Compliance with Safety Standards

Functional verification With AI ML

Taking much time in functional verification of SoC

Simulation of large no.
faults Takes a large amount
of time

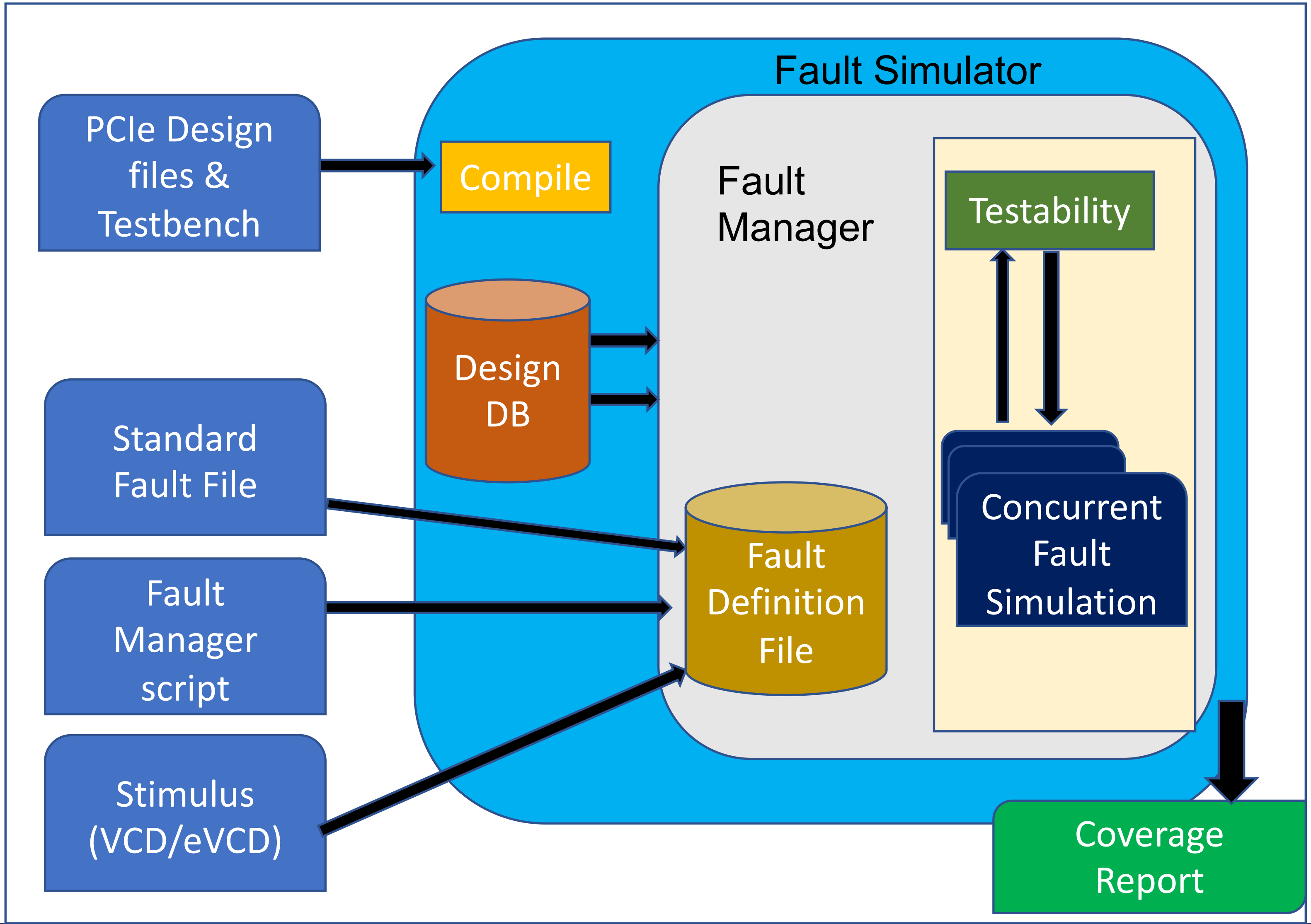
Injection of Faults
within large design



RTL

Simulation of faults list
generated
from AI sampled
Takes less time with same
result fault coverage

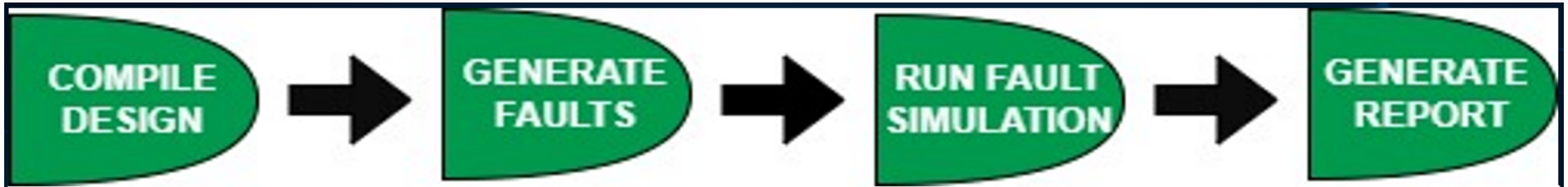
Architecture of Fault Simulation



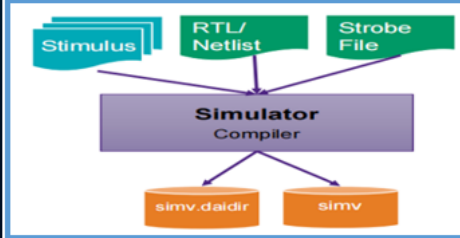
Fault Manager compares Design with Good Machine DB using FSDB on distributed and concurrent simulation engines.

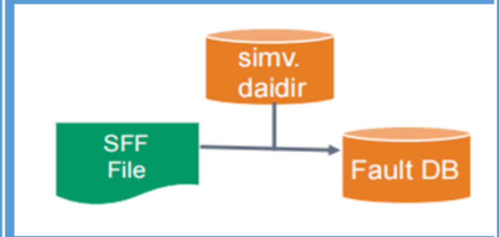
Generated faults are fed back as SF for comparison at Fgen, Coats, and Fsim, ensuring accurate fault simulation.

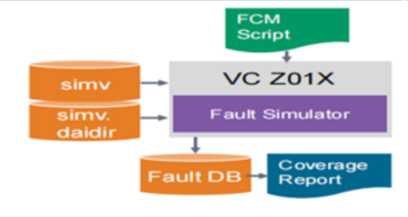
Fault Simulation Flow



STEP 1 : COMPILE THE DESIGN	STEP 2 : GENERATE THE FAULTS	STEP 3 : RUN FAULT SIMULATION	STEP 4: GENERATE REPORT
Syntax : vcs -full64 -fusa=portfaults<options>	Syntax : vc_fcc -full64 -daidir <daidir> -sff <sff_file> \ -campaign <campaign> -dut_path <path>	Syntax : vc_fcm -tcl_script pcie.tcl	Syntax : vc_fdb_report -campaign <campaign> -report <coverage_file>





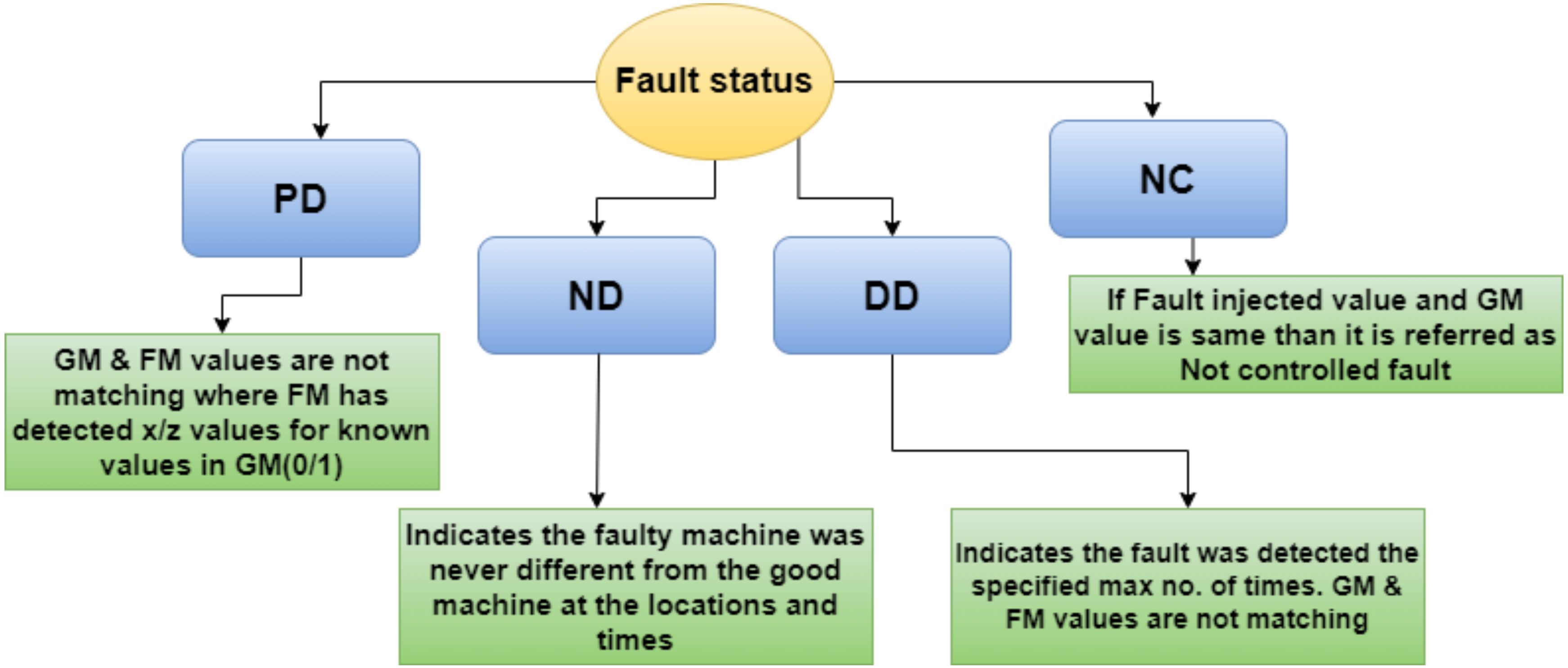


Evidence (PCIe GOOD MACHINE HDL)

```
initial
begin
         t1.phy.cdr.s2p.rcep.reset=1'b1;
         t1.phy.cdr.s2p.rcep.PCIe_Rx_Valid_intr = 1'b0;
         #5 t1.phy.cdr.s2p.rcep.reset = 1'b0;
         t1.phy.cdr.s2p.rcep.PCIe_Rx_Valid_intr = 1'b1;
end

initial
begin
        fork
                #5 data_send_lane0(3190783391);
                #5 data_send_lane1(3190807455);
                #5 data_send_lane2(3190765471);
                #5 data_send_lane3(3190807199);
        join
end

initial
begin
        #10 t1.phy.cdr.rcep.reset=1;
        #10 t1.phy.cdr.rcep.reset=0;
        #5  t1.phy.cdr.rcep.PCIe_Wr_Enable=1'b1;
        #10 t1.phy.cdr.rcep.PCIe_Rd_Enable=1'b1;
end

initial
begin
        $fsdbDumpfile("vcs.fsdb");
        $fsdbDumpvars();
end
```

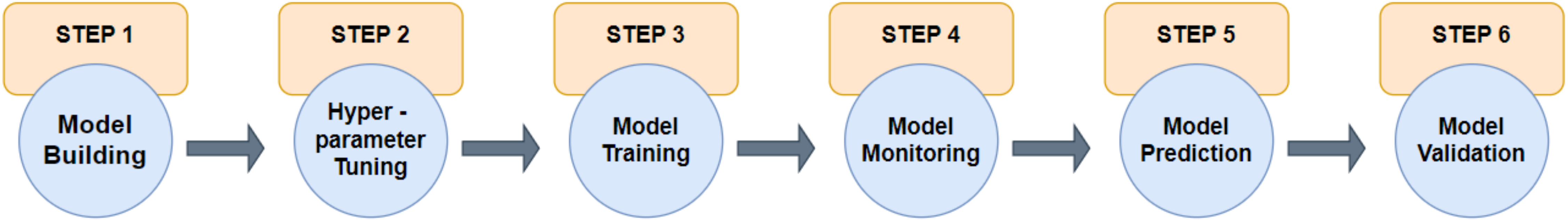
```
$fs_strobe(phy.cdr.ba.ba_out_lane0);
$fs_strobe(phy.cdr.ba.ba_out_lane1);
$fs_strobe(phy.cdr.ba.ba_out_lane2);
$fs_strobe(phy.cdr.ba.ba_out_lane3);
$fs_strobe(phy.cdr.ba.align_done);
$fs_strobe(phy.d0.ba_out_lane0);
$fs_strobe(phy.d0.ba_out_lane1);
$fs_strobe(phy.d0.ba_out_lane2);
$fs_strobe(phy.d0.ba_out_lane3);
$fs_strobe(phy.d0.align_done);
$fs_strobe(phy.d0.dec_out_lane0);
$fs_strobe(phy.d0.dec_out_lane1);
$fs_strobe(phy.d0.dec_out_lane2);
$fs_strobe(phy.d0.dec_out_lane3);
$fs_strobe(phy.d0.dec_done);
$fs_strobe(phy.us.dec_done);
$fs_strobe(phy.us.dec_out_lane0);
$fs_strobe(phy.us.dec_out_lane1);
$fs_strobe(phy.us.dec_out_lane2);
$fs_strobe(phy.us.dec_out_lane3);
$fs_strobe(phy.us.packet_out_lane0);
$fs_strobe(phy.us.packet_out_lane1);
$fs_strobe(phy.us.packet_out_lane2);
$fs_strobe(phy.us.packet_out_lane3);
$fs_strobe(phy.us.pkt_collect);
$fs_strobe(phy.pf.pkt_collect);
$fs_strobe(phy.pf.packet_out_lane0);
$fs_strobe(phy.pf.packet_out_lane1);
$fs_strobe(phy.pf.packet_out_lane2);
$fs_strobe(phy.pf.packet_out_lane3);
```

Fault Status



Different Fault status listed at Fsim Stage

AI ML FLOW



- Model building: Creating a mathematical representation for data analysis.
- Hyper-parameter tuning: Optimizing parameter values to enhance model performance.
- Model training: Teaching a model to make predictions using labeled data.
- Model monitoring: Tracking model performance and detecting anomalies in real-time.
- Model prediction: Generating forecasts or estimations using a trained model.
- Model validation: Assessing the accuracy and reliability of a model's predictions.

Model Building

- This code builds a neural network model using the Keras framework.
- The model has an input layer with a number of parameters equal to the length of x_train.keys().
- It is followed by four dense layers with 256 units each, using the ReLU activation function.
- The final layer is a dense layer with 1 unit and sigmoid activation for binary classification.
- The model is compiled with the Adam optimizer, binary cross-entropy loss, and accuracy as the metric.

```
#--------MODEL BUILDING
num_params = len(x_train.keys())
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',])
```

Code for Model Building

```
_________________________________________________________________
 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
```

Output of Model Building

Hyperparameter Tuning

- The code defines a grid of hyperparameters to tune, including the number of parameters, learning rate, batch size, and number of epochs.
- It uses GridSearchCV to search for the best combination of hyperparameters, evaluates them using cross-validation, and prints the results, including the best score and parameters.

```
# Define the hyperparameters to tune and the search space
param_grid = {'num_params': [len(x_train.keys())],
              'lr': [0.001, 0.01, 0.1],
              'batch_size': [4, 8, 16],
              'epochs': [50, 100, 200]}
# Create the GridSearchCV object
model = tf.keras.wrappers.scikit_learn.KerasClassifier
                  (build_fn=create_model, verbose=0)
grid_search = GridSearchCV(estimator=model,
                           param_grid=param_grid,
                            scoring=scoring, cv=3)
# Fit the GridSearchCV object to the training data
grid_result = grid_search.fit(x_train, y_train)
# Print the results of the grid search
print("Best: %f using %s" % (grid_result.best_score_,
                             grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Code for Hyperparameter Tuning

```
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 6ms/step
2/2 [==============================] - 0s 10ms/step
Best: 1.000000 using {'batch_size': 4, 'epochs': 50, 'lr': 0.001,
```

Output of Model Training

Model Training

- The code trains the model on the training data using parameters such as batch size, epochs, and a validation split.
- It prints the training progress and evaluates the model's performance based on accuracy and loss metrics.

```
#--------MODEL TRAINING
learning_rate = grid_result.best_params_['lr']
batch_size = grid_result.best_params_['batch_size']
epochs = grid_result.best_params_['epochs']
model = create_model(num_params, learning_rate)
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_split=0.1,
                    verbose=1)
```

Code to train model on training data and tuned Hyperparameter

```
Epoch 1/50
41/41 [==============================] - 1s 9ms/step - loss: 0.6528 - acc: 0.8634 - val_loss: 0.4898 - val_acc: 0.9444
Epoch 2/50
41/41 [==============================] - 0s 5ms/step - loss: 0.2500 - acc: 0.9689 - val_loss: 0.0018 - val_acc: 1.0000
Epoch 3/50
41/41 [==============================] - 0s 5ms/step - loss: 0.0026 - acc: 1.0000 - val_loss: 5.8710e-11 - val_acc: 1.0000
Epoch 4/50
41/41 [==============================] - 0s 6ms/step - loss: 2.2432e-04 - acc: 1.0000 - val_loss: 3.9635e-11 - val_acc: 1.0000
Epoch 5/50
41/41 [==============================] - 0s 6ms/step - loss: 1.3192e-05 - acc: 1.0000 - val_loss: 4.0439e-11 - val_acc: 1.0000
```

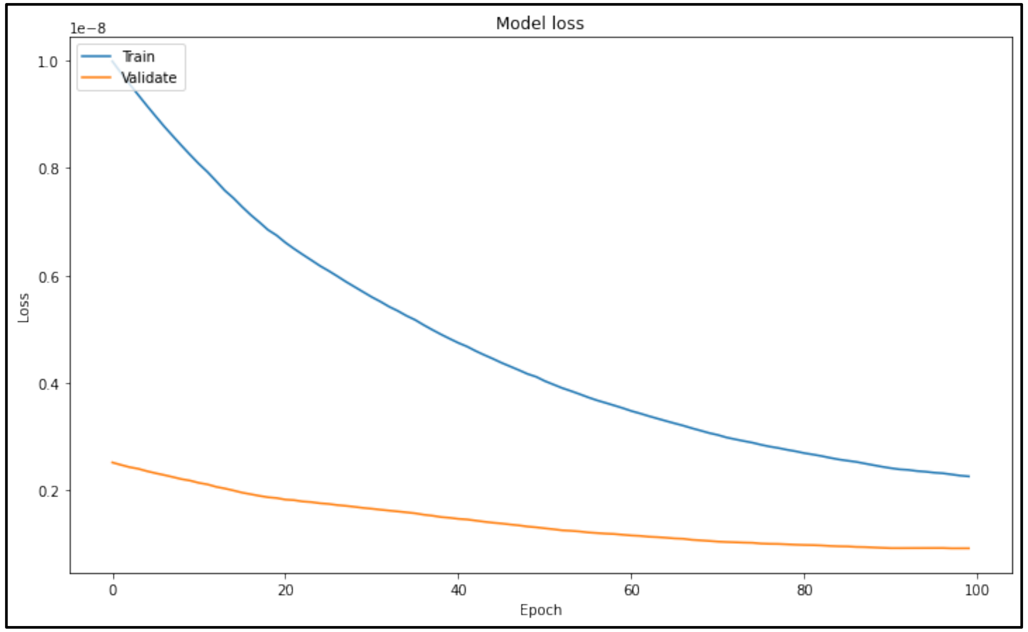
Output of Model Training

Model Monitoring

- The code generates a plot to visualize the training and validation loss values of a model over epochs.
- It creates a figure, plots the training and validation loss curves obtained from the history object, and labels the axes accordingly.
- The plot helps to understand the model's performance during training, highlighting any convergence or divergence of the loss values.

```
#--------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()
```

Code to Monitor the model for any convergence or divergence



Output Plot – Model Loss

Model Prediction

- The code predicts the classes for the test data using the trained model.
- It utilizes the predict function to obtain the predicted probabilities p_test.
- The predicted classes q and the actual classes from the test set are printed, providing a comparison between the predicted and actual values.

```
#--------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))
```

Code for model prediction using predict function

```
Predicted :
[[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
```

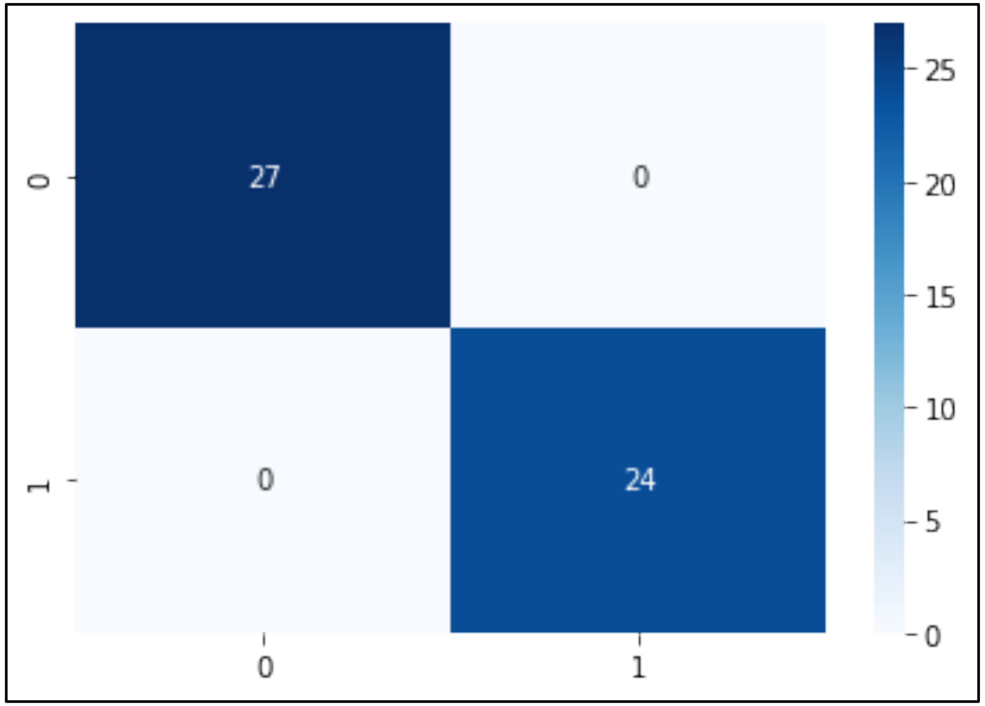
Output Plot – Model Loss

Model Validation

- The code creates a heatmap using seaborn (sns) to visualize the confusion matrix between the true labels (y_test) and the predicted labels (q).
- The confusion matrix is computed using TensorFlow's tf.math.confusion_matrix function. The heatmap uses a blue color scheme (cmap="Blues") and displays the values of the matrix as annotations (annot=True).

```
#model validation
sns.heatmap(tf.math.confusion_matrix(y_test, q),
            cmap="Blues",
            annot=True)
```

Code for model Validation using Heat Map



Output Plot – Heat Map

Results

Without AI Testbench

```
# Fault Coverage Summary
#
#                                       Prime              Total
#----------------------------------------------------------------------
# Total Faults:                          7936              10376
#
# Dropped Detected              DD       2408   30.34%      3311   31.91%
# Dropped Potential             PD          0    0.00%         0    0.00%
# Not Detected                  ND       5528   69.66%      7065   68.09%
#
# Untestable Unused             UU       1208               1208
#
# Detected                      DG       2408   30.34%      3311   31.91%
# Untestable                    UG       1208   15.22%      1208   11.64%
```

Fsim Report without AI Testbench

With AI Testbench

```
# Fault Coverage Summary
#
#                                       Prime              Total
#----------------------------------------------------------------------
# Total Faults:                          7936              10376
#
# Dropped Detected              DD       3552   44.76%      4751   45.79%
# Dropped Potential             PD          0    0.00%         0    0.00%
# Not Detected                  ND       4384   55.24%      5625   54.21%
#
# Untestable Unused             UU       1208               1208
#
# Detected                      DG       3552   44.76%      4751   45.79%
# Untestable                    UG       1208   15.22%      1208   11.64%
```

Fsim Report with AI Testbench

- After applying the AI Testbench, we observed a significant improvement in the fault detection results.
- Out of a total of 10,376 faults, the previous approach detected only 30.34% dropped, while the AI Testbench achieved a higher rate of 44.76%.
- Additionally, previous approach had 69.66% faults that were not detected, whereas AI Testbench lowered it 55.24%, indicating improved fault detection performance.

Conclusion

- The Fault Manager at the Fsim stage used data from an ML model to generate results which were then tested on a 20% sample of test data.
- We observed around 26% drop in Not Detected Faults in the design. The PCIe example identified more detectable faults and improved fault categorization.
- These results are promising and suggest further exploration with a larger dataset to fix bugs/faults earlier in the manufacturing process.
- This would reduce Fault Simulation time and aid in testing larger

Questions Please ?