DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Reducing the simulation life cycle time 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)
Gopi Srinivas Deepala, VLSI Design Engineer, Silicon Interfaces, Mumbai, India(gopi@siliconinterfaces.com)
Priyanka Gharat, VLSI Design Engineer, Silicon Interfaces, Mumbai, India
(priyanka@siliconinterfaces.com)
Avnita Pal, VLSI Design Engineer, Silicon Interfaces, Mumbai, India
(avnita@siliconinterfaces.com)

***Abstract*— This paper applies AI/ML algorithms to reduce fault simulation cycles, replacing standard techniques where each node is switched with Stuck@0/Stuck@1 by generation, testability, and simulation. AI/ML techniques partition datasets, build/compile ML models, and perform model fitting with error reduction techniques such as optimization and activation/loss functions. Predicted match results run through different sized datasets and hyper-parameters resulted in a 20% reduction in simulation cycles. The paper also analyzes the SafeSPI protocol and its "safing" features, such as monitoring sensors, adding CRC checks for error detection and fault tolerance. The protocol is analyzed using ISO26262 Safety Verification simulation, ensuring accurate data transmission by detecting potential bugs with Stuck@0/@1 injection.**

Keywords*— Machine Learning, ML, Artificial Intelligence, AI, Functional Verification; Fault Simulation, Testability, Controllability, Observability, GM, FM, Fault Status, Safe SPI, Functional Safety, Fault Simulation,CRC.

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. The paper also aims to analyze the SafeSPI protocol and its "safing" features, including sensor monitoring, CRC checks for error detection, and fault tolerance. The analysis is conducted using ISO26262 Safety Verification simulation to evaluate the protocol's effectiveness in real-world scenarios across various industries such as Automotive, Augmented/Virtual Reality, Drones, Aerospace, Military, and Medical.

Safety is critical for safe SPI communication used in aircraft systems to transmit data between different electronic components, including the control system, navigation system and communication system. Error detection through the SPI is crucial in ensuring the safety and reliability of airplanes. These techniques are relevant to Automotive, Augmented/Virtual Reality, Drones, Aerospace, Military and Medical. To this end, SafeSPI™ protocol addresses key safety features, like adding monitoring of devices and packet formats and CRC checks. However, that only goes as far as Protocol features is concerned, it does not consider environment issues, like temperature, pressure and radiation, which may bring signals getting Stuck@0 and Stuck@1 either due to electrical or transient factors. To this extent, Fault Simulation is an answer to ensure functional safety is ensured when the IC is in the field.

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.

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 simulationcycles can be reduced to achieve better results in less time.

II. Overview Of SPI And SafeSPI

SPI (Serial Peripheral Interface) is a synchronous serial communication interface used for short-range communication between devices on a PCB. Multiple slave devices can be connected using multiple chip select lines or one common CS with logical addressing. Microcontrollers act as the master device, generating SCLK and selecting the slave using the CS line. Data is transmitted via MOSI and MISO wires. SPI facilitates reliable communication between microcontrollers and peripherals, making it popular in many applications. SafeSPI protocol enhances SPI with safety features.

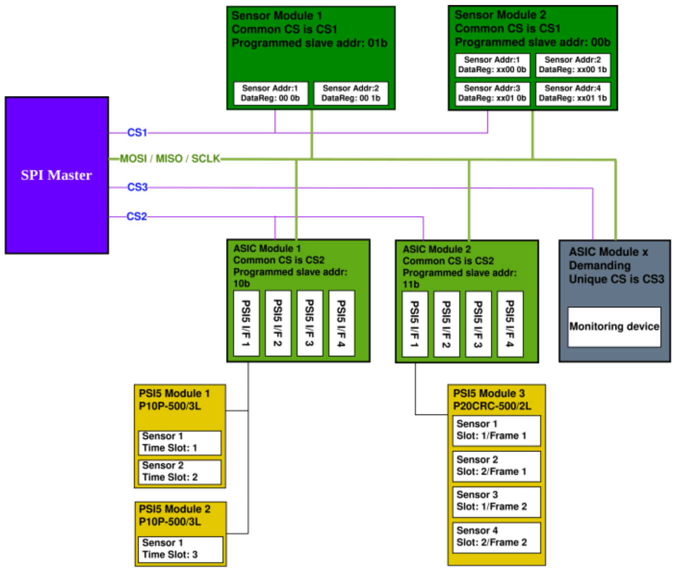


Figure II (a). SafeSPI Architecture

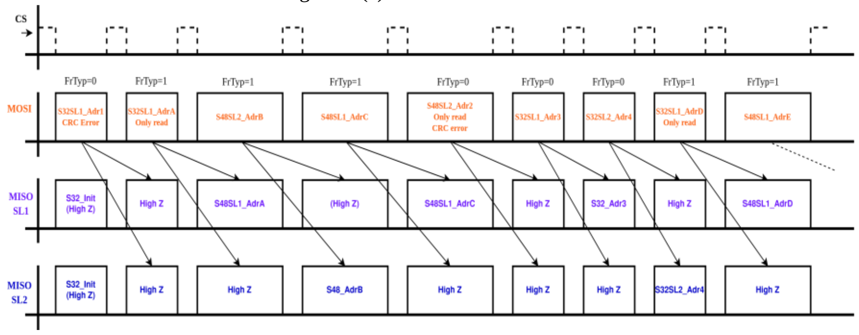


Figure III (b). Timing Diagram

III. Architecture And Building Block Of SPI

SafeSPI is a safety-enhanced protocol used in mission-critical safety applications, such as automotive. It involves an independent monitoring device that listens to sensor data on the SPI bus. The device is implemented in hardware and requires a dedicated SPI format. All signals on the monitor device are inputs, and the MSB is transferred first within a frame. To ensure data integrity, CRC is used to calculate a checksum over the data and appended to the message. There are two types of CRC frames used: in-frame and out-of-frame. The use of CRC is vital for reliable operation in SPI communication for accurate and dependable data transmission.

Advanced Features of SafeSPI:

SafeSPI is a secure communication protocol used in electronics to transfer data between a master device and multiple slave devices. It provides encryption, message authentication, and error detection and correction to maintain data integrity during transmission. CRC is crucial for data accuracy, especially in critical systems such as automotive safety.

Our system employs SafeSPI with one master and four slave devices: two sensors, two ASIC slaves containing sensor data and addresses, and one monitoring slave. This configuration facilitates efficient and streamlined communication between the devices, providing a better understanding of the system's operation. The monitoring slave is useful for real-time detection of issues and quick resolution. SafeSPI is a reliable and secure communication protocol, making it a popular choice for critical systems where data accuracy and integrity are essential.

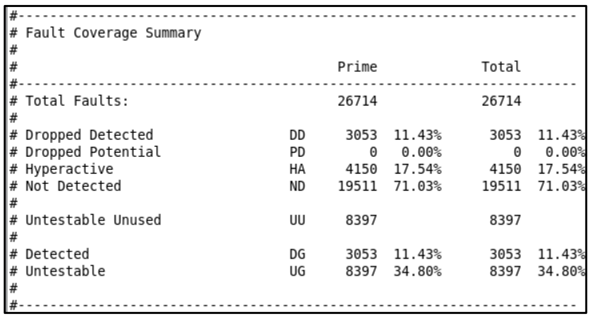
Overall, the SafeSPI system with a common chip select configuration offers a flexible and reliable solution for communicating with multiple slave devices in a variety of electronic applications.

IV. FAULT SIMULATION

Fault simulation is a valuable technique for device development that offers many benefits throughout the device lifecycle. It can identify systematic errors in the verification environment, enabling informed decisions and improving the device development and verification process. It is also useful in manufacturing to assess coverage and identify potential issues early on, ensuring high-quality products. Additionally, fault simulation can evaluate a mechanism's ability to handle random failures during the operational phase, making devices more resilient and reliable. EDA tools can replicate and analyze common error sources, allowing for a reliable and efficient method of detecting potential issues.

In this paper, we have used SafeSPI . 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:                       26714            26714
#
# Dropped Detected              DD     3053  11.43%     3053  11.43%
# Dropped Potential             PD        0   0.00%        0   0.00%
# Hyperactive                   HA     4150  17.54%     4150  17.54%
# Not Detected                  ND    19511  71.03%    19511  71.03%
#
# Untestable Unused             UU     8397             8397
#
# Detected                      DG     3053  11.43%     3053  11.43%
# Untestable                    UG     8397  34.80%     8397  34.80%
#
#--------------------------------------------------------------------
```

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

V. MACHINE LEARNING APPLICATION ON BIG DATA DATASET

Using a Dataset for SafeSPI 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
```

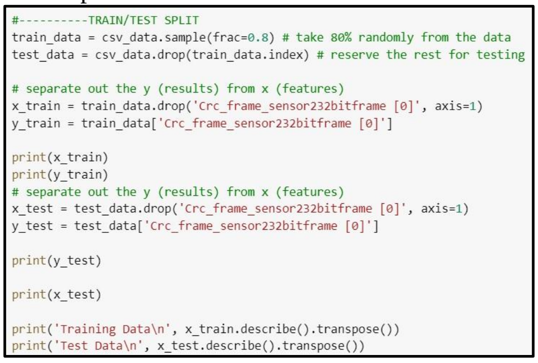
Along with the libraries, the file including the test vector data in csv format of the Safe SPI is passed as all the operation of sampling the 100% data to 20% data is being carried out on it.

```
filename = '/content/SafeSPI Dataset.csv
```

```
csv_data = pd.read_csv(filename)
```

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

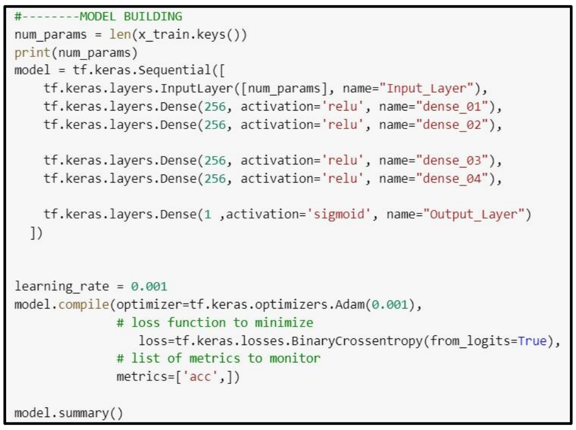
Further, the data in the test vector is sampled in 80%as train data and 20% as test data ratio.



```
#----------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('Crc_frame_sensor232bitframe [0]', axis=1)
y_train = train_data['Crc_frame_sensor232bitframe [0]']

print(x_train)
print(y_train)
# separate out the y (results) from x (features)
x_test = test_data.drop('Crc_frame_sensor232bitframe [0]', axis=1)
y_test = test_data['Crc_frame_sensor232bitframe [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. Regression ML model was built using TensorFlowML libraries



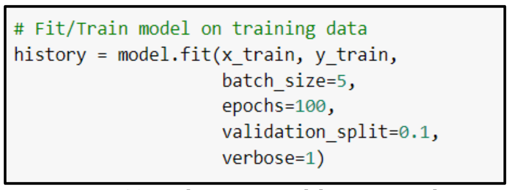
```
#--------MODEL BUILDING
num_params = len(x_train.keys())
print(num_params)
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer([num_params], name="Input_Layer"),
    tf.keras.layers.Dense(256, activation='relu', name="dense_01"),
    tf.keras.layers.Dense(256, activation='relu', name="dense_02"),

    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 MLLibraries.

VI. 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 SafeSPI^{TM} Bus.

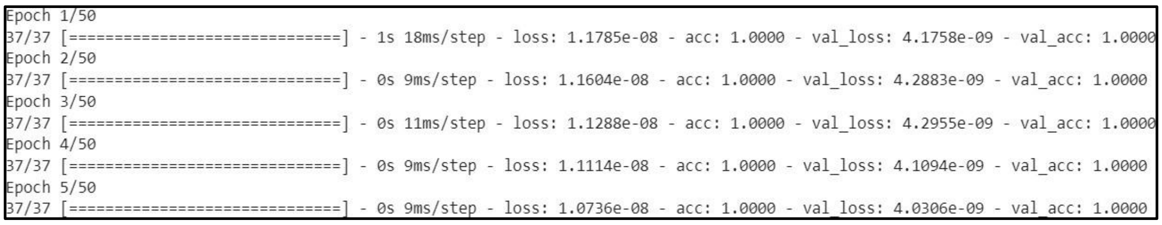
A. At the first pass, we fit the model on train data over several epoch runs.



```
# 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.



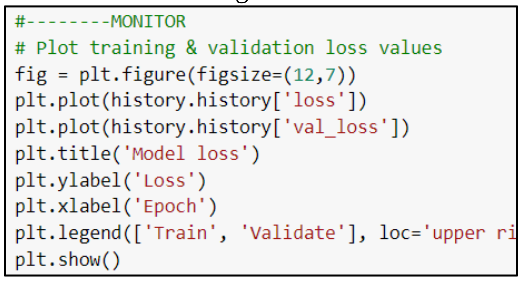
```
Epoch 1/50
37/37 [==============================] - 1s 18ms/step - loss: 1.1785e-08 - acc: 1.0000 - val_loss: 4.1758e-09 - val_acc: 1.0000
Epoch 2/50
37/37 [==============================] - 0s 9ms/step - loss: 1.1604e-08 - acc: 1.0000 - val_loss: 4.2883e-09 - val_acc: 1.0000
Epoch 3/50
37/37 [==============================] - 0s 11ms/step - loss: 1.1288e-08 - acc: 1.0000 - val_loss: 4.2955e-09 - val_acc: 1.0000
Epoch 4/50
37/37 [==============================] - 0s 9ms/step - loss: 1.1114e-08 - acc: 1.0000 - val_loss: 4.1094e-09 - val_acc: 1.0000
Epoch 5/50
37/37 [==============================] - 0s 9ms/step - loss: 1.0736e-08 - acc: 1.0000 - val_loss: 4.0306e-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 ri
plt.show()
```

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

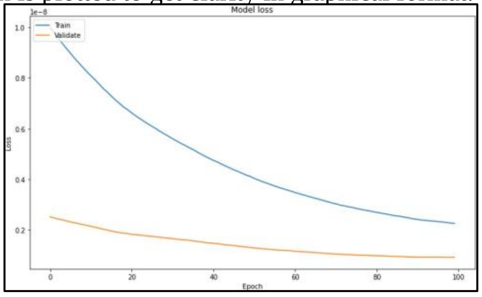


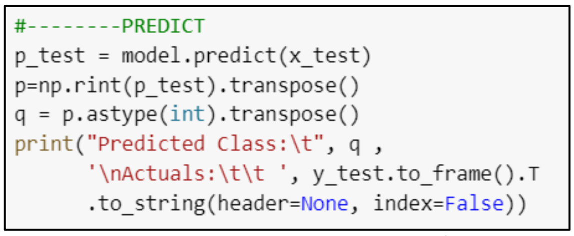
Figure IV(d). Model loss.

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)

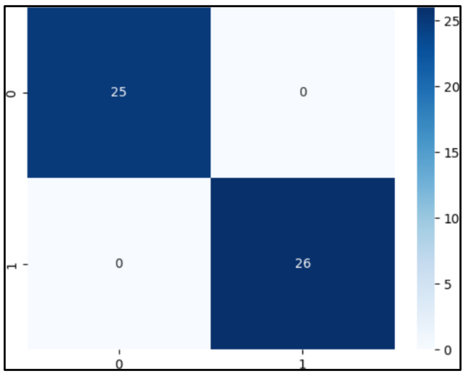
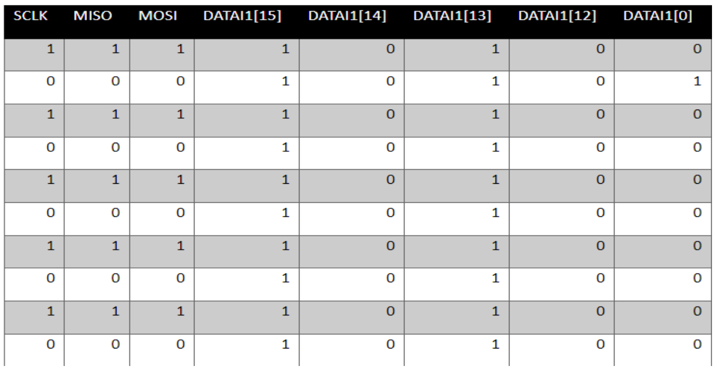


Figure IV(f). Heat Map graph generated from the Colab Tool

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)

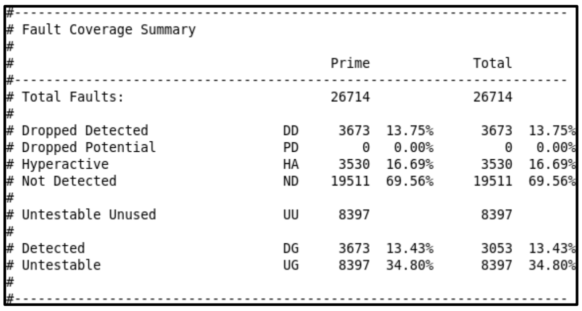
Below is the sampled data.



SCLK	MISO	MOSI	DATAI1[15]	DATAI1[14]	DATAI1[13]	DATAI1[12]	DATAI1[0]
1	1	1	1	0	1	0	0
0	0	0	1	0	1	0	1
1	1	1	1	0	1	0	0
0	0	0	1	0	1	0	0
1	1	1	1	0	1	0	0
0	0	0	1	0	1	0	0
1	1	1	1	0	1	0	0
0	0	0	1	0	1	0	0
1	1	1	1	0	1	0	0
0	0	0	1	0	1	0	0

Table 1. 20% Sampled Data.

And the same vector list (input) is being injected as a fault in SafeSPI^{TM} 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:                        26714            26714
#
# Dropped Detected              DD      3673  13.75%     3673  13.75%
# Dropped Potential             PD         0   0.00%        0   0.00%
# Hyperactive                   HA      3530  16.69%     3530  16.69%
# Not Detected                  ND     19511  69.56%    19511  69.56%
#
# Untestable Unused             UU      8397             8397
#
# Detected                      DG      3673  13.43%     3053  13.43%
# Untestable                    UG      8397  34.80%     8397  34.80%
#
#-----------------------------------------------------------------------
```

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

VII. Conclusion

Our paper describes the successful Safety verification of the SafeSPI using fault simulation and Fault list status Analysis. Through a Fault Campaign, we were able to identify stuck@0 and stuck@1 faults, which are essential for developing more robust communication within controllers and sensors. With the help of the process such as analyzing the fault status and techniques of ISO 26262 used in EDA we are able to achieve 100% of diagnostic coverage.

In the automotive industry, this technology can improve the safety of vehicles by enhancing communication between different components. In AR/VR applications, SafeSPI can help achieve better synchronization between devices, leading to improved user experience. In the aerospace industry, SafeSPI can enhance communication between different systems, leading to greater reliability and safety.

In the medical industry, SafeSPI can be used to improve communication between various medical devices, ensuring accurate and reliable data transfer. In military applications, it can be used to enhance communication and coordination between different systems, increasing mission success rates.
Overall, our research provides a valuable contribution to the development of more robust communication systems that can improve safety, reliability, and efficiency across various industries.

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.

[7] SafeSPI - Serial Peripheral Interface for Automotive Safety. SafeSPI_specification_v1.0.doc. 3rd june 2016

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

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

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