Using Machine Learning in Register Automation and Verification

Authors:
Nikita Gulliya
Abhishek Bora
Nitin Chaudhary
Amanjyot Kaur
MACHINE LEARNING IN ASIC DESIGN
“When All You Have Is a Hammer Everything Looks Like a Nail”

• ASIC design requires a lot of manual work
• There is need to automate as much as possible
• Machine Learning is a great technology for automation
FUNCTIONAL FLAWS CASE STUDY

• The functional flaws caused by issues related to specification is higher than issues related to design error

• Specifications are generally very large and error prone

• Functional flaws factors include changes in specification, incorrect or incomplete specification, flaws in internal and external IP Block, design errors

• It has been observed that functional flaws have gone up drastically in 2018 for design projects
ASIC: Root Cause of Functional Flaws

FPGA: Root Cause of Functional Flaws


*Multiple answers possible

© Mentor Graphics Corporation
CHALLENGES FACED IN ASIC DESIGN

• Errors are mainly due to incorrect/changing specification
• Changes must be automated
  – Identifying the type of register based on its specification and functionality
  – Correct RTL code generation based on the type of register
• Generation of SystemVerilog Assertion based on the specification
  – Difficult to understand SystemVerilog Assertion
USING MACHINE LEARNING

• Machine Learning (ML) is a powerful concept
• ML can help users create IP and SoC code
• If the specification is “formal”, one can automatically create design code and verification environment
• Even in case of “informal” English description, Machine Learning algorithms can be used for .
  - Register Automation
  - SystemVerilog Assertions
MACHINE LEARNING ALGORITHM FOR REGISTER AUTOMATION

• Predict the type of register and its functionality
• It can take description provided by the user as input and predicts the type of register
• Result is further processed to generate the relevant RTL code
• Python programming language and Keras Deep Learning Library has been used
  – Keras is a high-level neural networks API, written in Python and run on the top of TensorFlow
  – Concepts of RNN (Recurrent Neural Networks) have been used
CATEGORIZATION OF DIFFERENT TYPES OF REGISTER

Categorization into broader categories:

• Status Registers: counter registers, interrupt registers, FIFO exists.
• Special Registers: paged, virtual registers, TMR, shadow registers etc.
• Control Register: Enumerations, FIFO, counter, lock registers can be categorized under these types of registers.
• Implementation Defined: constant, reserved and registers which depend on external signal.
REGISTER AUTOMATION FLOW

Data Set → ML Algorithm (Adds properties) → IDS → RTL Code
OVERVIEW OF MACHINE LEARNING ALGORITHM

Machine learning Algorithm

Model training with data and compilation

Using model for appropriate output prediction
DATASET CREATION

- Thousands of samples for the dataset has been created for each type of register
- Industry level specifications were studied and analyzed
- Technical specification which defines the functionality of the register
- Considering an example for the lock register
  - Lock
  - Lock_r
  - Lock.set
  - Lock.clear
### LOCK REGISTER DATA SET SAMPLE:

<table>
<thead>
<tr>
<th>Sno</th>
<th>Description</th>
<th>Src Reg</th>
<th>Src Field</th>
<th>output</th>
<th>Key Reg</th>
<th>Key Field</th>
<th>Lock Reg</th>
<th>Lock Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>When KEY FIELD in KEY_REG is set then LOCK_FIELD of LOCK_REG can be written</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
<td>lock</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>2</td>
<td>When KEY_FIELD of KEY_REG is '1' then the field of this register is readable</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>lock_r</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>3</td>
<td>When KEY_REG.KEY_FIELD is active then this bit will be cleared</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
<td>lock.clear</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>4</td>
<td>When KEY_REG.KEY_FIELD is reset, then write access of this field will be disabled</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
<td>lock</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>5</td>
<td>This field will be unlocked when KEY_FIELD of KEY_REG is 0</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
<td>lock</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>6</td>
<td>Read action will only take place on LOCK_FIELD of LOCK_REG when KEY_FIELD</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>lock_r</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>7</td>
<td>At '1' i.e high LOCK_FIELD of LOCK_REG will be able to read</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>lock_r</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>8</td>
<td>LOCK_FIELD of LOCK_REG will be clear when this is low</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>lock.clear</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>9</td>
<td>At '1b0' LOCK_FIELD of LOCK_REG will become readable</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>lock_r</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
<tr>
<td>10</td>
<td>Whenever this bit is zero LOCK_FIELD of LOCK_REG is set to one</td>
<td>KEY_REG</td>
<td>KEY_FIELD</td>
<td>lock.set</td>
<td>KEY_Reg</td>
<td>KEY_FIELD</td>
<td>LOCK_REG</td>
<td>LOCK_FIELD</td>
</tr>
</tbody>
</table>
FEEDING THE TRAINING DATASET

Lock Register Data Set:

```python
import numpy as np
import pandas as pd

seed = 1
np.random.seed(seed)
train = pd.read_csv("impDataT0train1.csv", sep=',', error_bad_lines=False, encoding = "latin1")
train.head()
```

<table>
<thead>
<tr>
<th>Sno</th>
<th>Description</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When KEY_FIELD in KEY_REG is set then LOCK_FIE...</td>
<td>lock</td>
</tr>
<tr>
<td>1</td>
<td>When KEY_FIELD in KEY_REG is active then this ...</td>
<td>lock</td>
</tr>
<tr>
<td>2</td>
<td>When KEY_FIELD of KEY_REG is low, then write a...</td>
<td>lock</td>
</tr>
<tr>
<td>3</td>
<td>This field will be locked when KEY_FIELD of KE...</td>
<td>lock</td>
</tr>
<tr>
<td>4</td>
<td>Freezes pdiff signal and makes it unwriteable</td>
<td>lock</td>
</tr>
</tbody>
</table>
DIFFERENT VARIABLES FOR THE DATA

• Seed value specification and loading of data
• Batch size, vocabulary size, embedding dimensions, maximum sequence length and validation split added

```python
# Batch Size
batch_size=32

# Vocab size
vocabulary_size=20000

# Embedding Dims
embedding_size=EMBEDDING_DIM=300

# Max sequence length
MAX_SEQUENCE_LENGTH=50
VALIDATION_SPLIT=0.8

train=train.drop(['Sno'],axis=1)

train_df=train['Description']

y=train['output']
```
Using TensorFlow backend.

```python
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# Tokenizing
tokenizer = Tokenizer(num_words=vocabulary_size)
tokenizer.fit_on_texts(train_df)
sequences = tokenizer.texts_to_sequences(train_df)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', y.shape)

Found 190 unique tokens.
Shape of data tensor: (118, 50)
Shape of label tensor: (118,)
```
LABEL ENCODING AND OneHotEncode

```python
# For splitting training and testing datarom sklearn.model_selection import train_test_split

# Output : train_df['output']
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to dummy variables (i.e. one hot encoded)
Y = np_utils.to_categorical(encoded_Y)
print(Y)
```

```
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 1.]
```
SAMPLE PLOT OF LOCK AND INTERRUPT REGISTER DATA

```python
train['output'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28e5b66f828>
```

```python
X_train, X_test, y_train, y_test = train_test_split(data, Y, test_size=0.10, random_state=seed)
```
EMBEDDING MATRIX

```python
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

```
[-1.7496e-01  2.2956e-01  2.4924e-01  -2.0512e-01  -1.2294e-01  2.1297e-02
 -2.3815e-01  1.3737e-01  -8.9130e-02  -2.0607e+00  3.5843e-01  -2.0365e-01
 -1.5518e-02  2.5628e-01  2.2963e-01  1.1985e-03  -8.9833e-01  1.3609e-01
  1.8861e-01  -3.3359e-01  1.8397e-02  6.2946e-01  -1.3167e-01  6.4819e-01
  2.3231e-01  4.9195e-01  -4.1841e-01  7.3934e-02  -6.5639e-01  4.8608e-01
 -1.1219e-01  -2.9994e-01  -7.2501e-01  8.5377e-02  -5.0447e-02  2.3105e-01
 -6.4843e-02  3.9056e-03  9.9742e-02  -2.0334e-02  3.8845e-01  2.4464e-01
 -8.6308e-02  -1.1308e-01  1.9281e-02  -1.1205e-01  6.5642e-02  1.8120e-01
 -1.0949e-01  5.5968e-02  -1.9700e-01  4.9184e-01  6.1818e-01  -3.3190e-02
  7.3289e-02  -2.2823e-02  6.6946e-01  1.8233e-01  -4.0082e-01  -3.3717e-01
 -2.8521e-01  -2.8222e-01  -4.4373e-02  1.4881e-01  -4.2135e-01  5.1545e-02
  2.7605e-01  -1.9959e-01  -2.9766e-01  -8.7712e-02  4.6210e-01  1.6891e-01
 -1.9415e-01  2.8327e-01  -2.5327e-01  -6.3275e-02  9.0945e-02  -1.8623e-01
  2.8891e-01  4.5534e-02  -1.0893e-01  3.9545e-01  8.4857e-02  5.4829e-02
 -3.7166e-01  2.8856e-01  -7.2666e-03  -2.3869e-01  1.8698e-01  2.1457e-01
  2.4625e-03  -2.2166e-01  -1.0549e-01  2.6366e-01  6.3795e-01  -2.1856e-01
```
SEQUENTIAL MODEL

Embedding Layer

LSTM Units

Softmax
EMBEDDING LAYER

• Embedding Layer: This layer converts the integers into fixed sized dense vectors
• LSTM: It models time and sequence dependent behavior
• Softmax Layer: This layer is used for the activation of the dense layers

```python
embed_dim = 300  # embedding dimensions
lstm_out = 128  # number of lstm cells

model = Sequential()
model.add(embedding_layer)
model.add(LSTM(lstm_out, dropout_U=0.25, dropout_W=0.25))
model.add(Dense(4, activation='softmax'))
model.add(Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```
RNN (Recurrent Neural Networks)

The Problem of Long-Term Dependencies
LONG SHORT-TERM MEMORY (LSTM)
LONG SHORT-TERM MEMORY (LSTM)

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
TRAINING THE MODEL

```
print("Accuracy: %.2f\%" % (scores[1]*100))
```

Train on 1006 samples, validate on 200 samples
Epoch 1/50 1006/1006
[-----------------------------] - 1s 11ms/step - loss: 1.9463 - acc: 0.1981 - val_loss: 1.8278 - val_acc: 0.3333
Epoch 2/50 1006/1006
[-----------------------------] - 0s 3ms/step - loss: 1.6012 - acc: 0.3679 - val_loss: 1.8820 - val_acc: 0.0833
Epoch 3/50 1006/1006
[-----------------------------] - 0s 2ms/step - loss: 1.3465 - acc: 0.5660 - val_loss: 1.4663 - val_acc: 0.4167
Epoch 4/50 1006/1006
[-----------------------------] - 0s 3ms/step - loss: 1.0703 - acc: 1.0000 - val_loss: 1.1568 - val_acc: 0.7500
Epoch 5/50 1006/1006
[-----------------------------] - 0s 3ms/step - loss: 0.0031 - acc: 1.0000 - val_loss: 1.1459 - val_acc: 0.7500

Accuracy: 75.00\%
EVALUATION AND PREDICTION

```python
# serialize model to JSON
import h5py
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)

# serialize weights to HDF5
model.save_weights("model.h5")

print("Saved model to disk")

# Predicting
pred1 = model.predict(X)

print('OUTPUT : ', encoder.inverse_transform(np.argmax(pred1)))  # Predict output column
```
OUTPUT PREDICTION

- IDS Output (Prediction)

### KEY_REG

<table>
<thead>
<tr>
<th>bits</th>
<th>name</th>
<th>s/w</th>
<th>h/w</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>KEY_FIELD</td>
<td>rw</td>
<td>rw</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### LOCK_REG

<table>
<thead>
<tr>
<th>bits</th>
<th>name</th>
<th>s/w</th>
<th>h/w</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>LOCK_FIELD</td>
<td>rw</td>
<td>rw</td>
<td>0</td>
<td>This field will be locked when KEY_FIELD of KEY_REG is 1</td>
</tr>
</tbody>
</table>
## PROPERTY APPLICATION

### KEY_REG

<table>
<thead>
<tr>
<th>bits</th>
<th>name</th>
<th>s/w</th>
<th>h/w</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>KEY_FIELD</td>
<td>rw</td>
<td>rw</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### LOCK_REG

<table>
<thead>
<tr>
<th>bits</th>
<th>name</th>
<th>s/w</th>
<th>h/w</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>LOCK_FIELD</td>
<td>rw</td>
<td>rw</td>
<td>0</td>
<td>This field will be locked when KEY_FIELD of KEY_REG is 1 {lock=KEY_REG.KEY_FIELD}</td>
</tr>
</tbody>
</table>
RTL CODE GENERATION
MACHINE LEARNING IN VERIFICATION

• The SystemVerilog “Decoder Ring” has been implemented
• Converts the concurrent assertions into plain English text and vice-versa.
• The SVA is first parsed and converted into hierarchal form based on the grammar.
• The SystemVerilog assertion (SVA) grammar has been written in ANTLR form.
ASSER TIONS TO ENGLISH FLOW

• Simple NLP is used to define rule based English output for every SystemVerilog assertion operation.
• Both the inputs, the parsed SVA and simple NLP, are fed into the interpreter to provide the output in plain English text
• This English is not grammatically correct.
• The bad English text is converted into the good English format with the help of machine learning algorithm.
SystemVerilog Assertion “Decoder Ring”
SystemVerilog Assertion to English

- SystemVerilog assertion given as input and the corresponding English output obtained.

**SystemVerilog Assertion Input:**
$\text{rose(a)} \ |-> (\text{a throughout b } [->1]) \ #\#1 !\text{a}\\

**English text Output:**
Whenever a goes high, a must be high until b is asserted and after 1 clock cycle, a must be low.

**SystemVerilog Assertion Input:**
\text{a } \ #\#1 \text{ b } [*1:$] \ #\#1 \text{ c}\\

**English text Output:**
a must be true on the first clock tick, c must be true on the last clock tick, and b must be true at every clock tick strictly in between the first and the last.
Thank You
Any Questions?