IZyneT Toolchain Guide - Revision 2023.1

Michele Pio Fragasso

April 2023

Contents

1 Introduction

This is a guide to IZyneT (Intermittent Zynet Toolchain) usage which is a set of python scripts for reconfiguration and evaluation of a NORM VHDL compatible intermittent feed forward deep neural network called I-DNN. Zynet is a set of scripts for reconfiguration of a Verilog fully connected feed forward DNN, developed by Vipin Kizhepatt.

2 IZyneT filesystem

To let the user familiarize with the toolchain functioning the filesystem is described in this section.

```
I-DNN/vivado
├── scripts
│   ├── files
│   ├── functions
│   └── mains
├── src
│   ├── DNN
│   └── NORM
├── tb_files
│   ├── DNN
│   │   ├── single_images
│   │   └── multiple_images
│   └── layer_hazard_scenarios
├── test
└── vivado
```

2.1 scripts

The scripts folder contains IZyneT, that is the set of scripts and files for reconfiguration and evalutation of the I-DNN.
The script folder contains `files`, `functions` and `mains` folders.

2.1.1 files

- files
  - input_file
    - training_file.txt
    - old-training-files.txt
  - MNIST_package
  - voltage_traces
    - 1.txt
    - ...
    - 10.txt
  - weights_n_biases
    - training_<mm-dd-yy>_<hh>-<mm>-<ss>
    - training_04-20-23_15-25-10
    - training_04-20-23_14-50-41

The `input_file` folder contains the input training files used by `trainNN.py` script to train the DNN network. The name must necessarily be `training_file.txt`. Every line is of the type `<field>`,`<field_value>` separated with a comma with no space in between.

```
num_hidden_layers,<number of hidden layers>
act_fun_type,<activation function type>
epochs,<number of training epochs>
eta,<learning rate>
batch_size,<mini batch size>
lmbda,<regularization parameter>
sigmoid_inputSize,<sigmoid input size>
sigmoid_inputIntSize,<sigmoid input Integer Part size>
sizes,<sizes in size format>
```

- The activation function type is a string containing either Sig or ReLU. Pay attention that the strings are case sensitive.
- The sigmoid sizes refers to the sigmoid input size, which determines the number of sigmoid ROM content. Refers to the Appendix A for more information about how to set up these values.
- The sizes are numbers enclosed in parenthesis and split by semicolumns: `(30;25;15;10)` is an example for a 4-layer DNN.

2.1.2 mains

This folder contains all the python scripts for reconfiguration, analysis and evaluation of I-DNN.

- mains
  - DNN_architectures
  - min_hazard_res
  - plots
  - results
  - tcl
  - trainNN.py
  - DNN_tester.py
  - vts_plots.pt
  - VoltageTrace_file_generation.py
  - preanalysis_single_vt.py
  - gen_CB_results_fixed_time.py
  - gen_DB_results_fixed_time.py
  - pwr_cmpt_validation.py
  - components_pwr_cmpt.py
  - analysis_DNN.py
  - analysis_vt.py
  - DB_results_plots.py

The script description is below:

- The script `trainNN.py` is the main file of IZyneT, used for reconfiguring I-DNN.
- The script `VoltageTrace_file_generation.py` is used for generating the voltage trace, i.e. the harvesting scenario according to the the need of the user.
- The script `preanalysis_single_vt.py` is used for analyzing the harvesting scenario, and determine the minimum hazard threshold to ensure correct backup during hardware simulation.
- The script `vts_plots.py` is used to visually inspect the different voltage traces.
- The script `gen_DB_results_fixed_time.py` is the python script for simulating the intermittent architecture with a dynamic backup policy.
- The script `gen_CB_results_fixed_time.py` does the same for a constant backup policy.
- The script `pwr_cmpt_validation.py` is used for validating the power consumption model of the single components.
- The script `components_pwr_cmpt.py` is used to plot the power consumption from the results files
- `vts_plots.py` produces the plots for visually inspecting the voltage traces
- `analysis_DNN.py` is used to collect and plot the results of the dynamic backup policy applied to 4 differrent DNN architectures.
- `analysis_vt.py` is used to collect and plot the results relative to 10 different voltage traces for a 4-layer DNN.
- `DB_results_plots.py` is used for plotting the raw end-of-simulation values corresponding to a dynamic backup policy.

The folder description is below:

- The folder `DNN_architectures` contains an informal description of the 4 DNNs analyzed to produce the performance-based voltage trace characterization.
- The folder `min_hazard_res` contains the minimum hazard results containing the minimum hazard threshold for different DNN architectures, for different voltage traces and different `NV_REG_DELAY_FACTOR`.
- The folder `plots` contains the plots generated by the python scripts.
- The folder `results` contains the simulation results produced by the simulation scripts (`gen_CB_results_fixed_time.py` and `gen_DB_results_fixed_time.py`)
- The folder `tcl` contains the tcl scripts executed in batch mode by vivado.

For every python script there is extensive script description inside the file, that the user can read.

2.2 `src`

The src folder contains the VHDL I-DNN entities and packages both from the NORM framework and DNN (respectively inside NORM and DNN folders). IZyneT puts here the VHDL generated files during reconfiguration for vivado project to point at.

2.3 `tb_files`

`tb_files` folder contains the IZyneT generated files for testbenching the I-DNN. Inside this folder they are located:

- `layer_hazard_scenario` is a static folder used for testbenching of the layer and test its capabilities to perform backup correctly under different hazard scenarios, that is under different states of the I-layer.
- `single_images` is a folder generated by IZyneT to testbench the DNN with a single MNIST image.
- `multiple_images` contains files to testbench the DNN with a set of 8 consecutive MNIST samples.

2.4 `test`

The test folder contains the testbenches for the DNN and of the layer.

```
test
├─ I_DNN_tb.vhd
├─ I_DNN_multiple_images.vhd
└─ I_layer_tb_hazard_scenarios.vhd
```

- `I_DNN_tb.vhd` is used to testbench I-DNN with a single MNIST sample.
- `I_DNN_multiple_images.vhd` testbenches I-DNN with a set of 8 MNIST samples.
- `I_layer_tb_hazard_scenarios.vhd` testbenches the I_layer and the capability to perform backup in all conditions.

3 I-DNN reconfiguration and evaluation

This section shows how to reconfigure the I-DNN. Testing and evaluation are also covered here. The path to the files are relative to the parent directory I-DNN/vivado/

3.1 Reconfiguration of DNN and energy environment

To reconfigure the I-DNN two approaches are possible.

- Train a new I-DNN.
- Load a previously trained I-DNN.

3.1.1 Training a new I-DNN

It starts with the editing of the training_file.txt containing the DNN PARAMETERS inside ./scripts/files/input_file - as described in 2.1.1 - so that the reconfiguration process can start (Figure 1).

After that you can run the script trainNN.py. It will generate the I-DNN model and place it in different locations:

- I-DNN.vhd, I-DNN_package.vhd and MI_DNN_package.vhd are placed in ./src/DNN while NVME_package.vhd is placed in ./src/NORM. The new architecture is embedded inside the Vivado project.
- A training_<mm-dd-yy>_<hh>-<mm>-<ss> folder containing the DNN model files is placed inside tb_files inside the folders single_images and multiple_images.
- The backup of the DNN model files are placed in /files/weights_n_biases inside the generated training folder training_<mm-dd-yy>_<hh>-<mm>-<ss>.

3.1.2 Load a previously generated I-DNN

To load a previously generated I-DNN it is necessary to copy and paste the NORM compatible VHDL backup files (located at ./files/weights_n_biases/VHDL_output and paste them in the src folder overwriting the previously file). I-DNN_package.vhd, I-DNN_package and I-DNN.vhd must be pasted inside ./src/DNN while the NVME_package.vhd inside ./src/NORM. Refreshing the vivado hierarchy will update the new files.

3.2 Setting up energy environment: generate the Voltage Trace

After I-DNN is generated the VHDL compatible voltage trace can be generated. Run the script VoltageTrace_file_generation.py and select the desired characteristics inside the python script. Here is a brief description of the inputs:

```
shtdw_value = 2300  #System Shutdown Value
SC_P = 40           #System clock period
sim_time = 3_000    #Simulation time in us
vt_ts = 160         #Voltage Trace Timescale in ns
```

3.3 Testbenching I-DNN functionality

To testbench I-DNN open Vivado and load the vivado project at ./vivado/vivado/I-DNN/I-DNN.xpr. Select the top level testbench as I_DNN_multiple_images_tb.vhd and run the simulation for the simulation time set up at the previous step. Remember to set the backup policy by uncommenting the instantiation code inside I-DNN.vhd.

Load the configuration waveform file I_DNN_multiple_images_tb_behav.wcfg located at vivado/vivado/I-DNN/ to inspect the waveforms. The output of the I-DNN is active when the I-DNN data_v bit goes high. Select data_v and point to the next rising edge transition. It's now possible to inspect the signal digit_out which contains the classified digit performed by the VHDL I-DNN for the corresponding input image. The input image is located inside the constant digit_filenames. The signal image_no contains the pointer to the collection of paths to the VHDL compatible MNIST samples. This collection is inside the package I_DNN_MI_package.vhd.
Below the content of this collection:

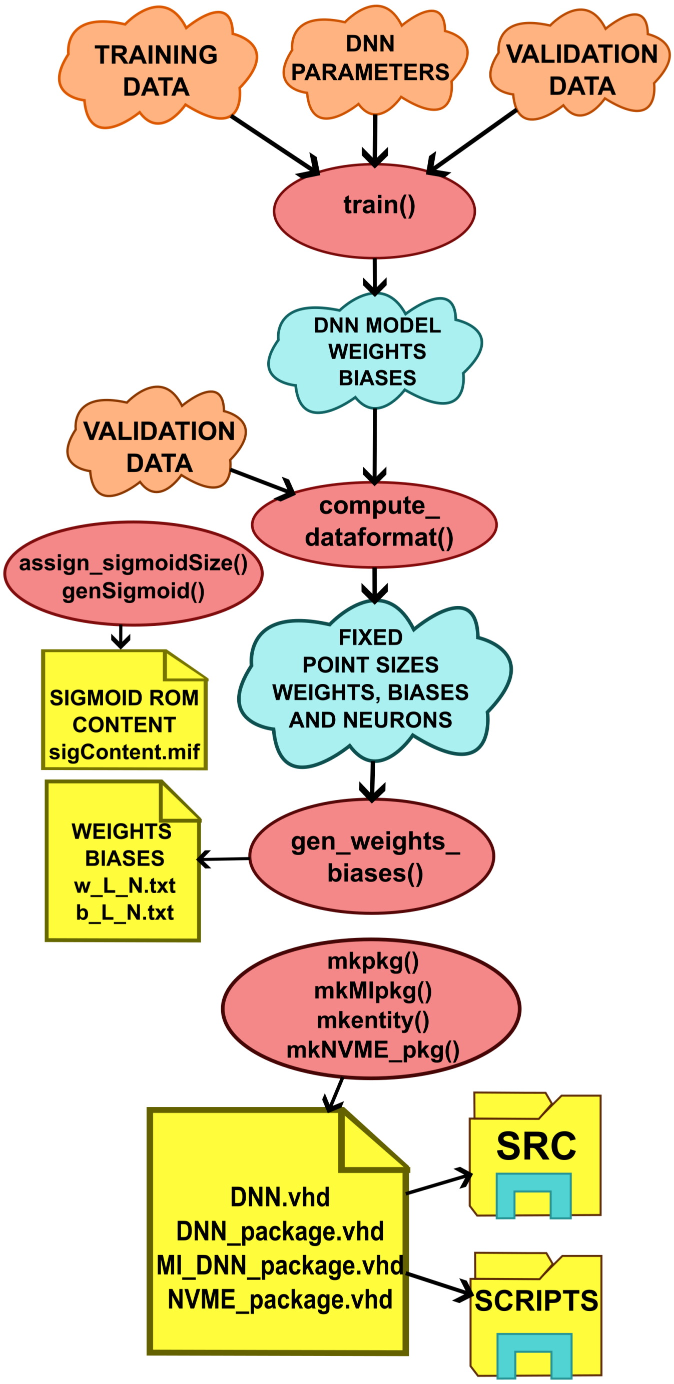

Figure 1: Reconfiguration flow I-DNN

```
    constant digit_filenames: digit_filenames_type :=
    ("test_dataset_6542/dataset_6542_classdigit.txt",
    "test_dataset_0910/dataset_0910_classdigit.txt",
    "test_dataset_1000/dataset_1000_classdigit.txt",
    "test_dataset_1160/dataset_1160_classdigit.txt",
    "test_dataset_1549/dataset_1549_classdigit.txt",
    "test_dataset_6542/dataset_6542_classdigit.txt",
    "test_dataset_1570/dataset_1570_classdigit.txt",
    "test_dataset_1290/dataset_1290_classdigit.txt");
```

So if the signal `image_no` is 0 before the rising edge of `out_v` the MNIST being tested is 6545 from the test data set. If it is 1 the sample is the one with index 910, etc...
To inspect the neurons' output for the last layer, expand the `data_out_vect` inside the *last_layer* group. This output is to be compared with the python DNN model. If the last_layer group is not logged into the waveform you may need to log it yourself. (Figure 2)

To compute the "true" I-DNN output, open the python script `DNN_tester.py`. The DNN model is loaded with the method `network.load("../files/weights_n_biases/training_mm-dd-yy_ss-mm-hh/WeightsAndBiases.txt")`. The date `mm-dd-yy_ss-mm-hh` is extracted from the VHDL DNN model. It is located inside the constant `DNN_prms_path` in `./src/DNN/I_DNN_package.vhd`.
For example if `DNN_prms_path` content is:

```
constant DNN_prms_path: string :=
    "./tb_files/DNN/single_image/tb\_training\_04-20-23\_14-27-16/";
```

You only need to copy and paste the string `04-20-23_14-27-16` replacing the string `mm-dd-yy_ss-mm-hh`. After that you need to set up the MNIST sample to test inside the python script, which must be the same as the sample being inferenced by the VHDL DNN at a particularly time instant.
It is now possible to execute `DNN_tester.py` which will print:

1. The label the sample belongs to
2. The classified digit

Label and DNN output classified digit do not necessarily corresponds since the accuracy of the DNN is not 100%. The last layer neurons' output is stored inside the variable `out_0`. You can access that variable content inside the python interpreter and compare the two vectors. (Figure 2 shows an example of comparison).

3.4 Evaluation

The general description is not covered here but only in the specific example. See section 4.3.

4 Example 1 - 4 layer I-DNN ReLU activation function

This section describes how to reconfigure a 4-layer DNN with sizes (30;25;15;10) using a ReLU activation function.

4.1 Reconfiguration

The first step is to set the `training_file.txt` located at `./scripts/files/input_file/` in the following way:

```
num_hidden_layers,4
act_fun_type,ReLU
epochs,10
eta,10
batch_size,100
lmbda,1
sigmoid_inputSize,10
sigmoid_inputIntSize,4
sizes,(30;25;15;10)
```

Run the training python script `train.py` and wait for it to finish. You can test the layer according to section ??.

The energy environment characteristics are selected by modifying the input values of `VoltageTrace_file_generation.py` script in the following way:

```
shtdw_value = 2300  #System Shutdown Value in mV
SC_P = 40           #System clock period
sim_time = 3_000    #Simulation time in us
vt_ts = 160         #Voltage Trace Timescale in ns
```

You can now run the script which will generate the voltage traces to be simulated and automatically upload them into the vivado project.
To set up a safe hazard threshold, you set up the script `preanalysis_single_vt.py` input variables as follows:

```
#INPUTS
#Voltage trace parameters
vt_ts = 160            #Voltage Trace Timescale
trace_number=0         #They range from 0 to 9
sim_time = 3_000       #Simulation Time
shtdw_value = 2300     #Shutdown Value
#Analsysis parameters
wrng_start = 2_300     #Start Hazard Value
wrng_final = 4_500     #End Hazard Value
wrng_step = 50         #Hazard Value step
DNN_max_size = 30      #DNN maximum layer size
#Plot Options
enable_plots_save = False
plt_show = False
```

The `trace_number` value need to be ranged between 0 and 9 (required for later evaluation). The minimum hazard thresholds for every `NV_REG_DELAY_FACTOR` logs will end up in the folder `./scripts/files/input_file`.

4.2 Testbenching

In this case the `NV_REG_DELAY_FACTOR` = 2 will be simulated for the voltage trace number 2. Therefore open the file `/vt-analysis-report_vt-2_sim-time-3000_vt-ts-160_size-30.txt` inside `./scripts/mains/min_hazard_res/vt-analysis-report_sim-time-3000_vt-ts-160_size-30` and take note of the minimum hazard threshold that will be set up later (it should be 2500).
Open Vivado and open the I-DNN project at `./vivado/vivado/I-DNN/I-DNN.xpr` and set the top level testbench as `I_DNN_multiple_images_tb.vhd`. Also, before running the simulation uncomment the DB policy inside `I-DNN.vhd` entity.

```
--#UNCOMMENT from Start to End
--##DB##Start
--FMS_NV_REG_DB_COMP
--fsm_nv_reg_db_comp: fsm_nv_reg_db
--    port map(
--        clk             => clk,
--        resetN          => resetN_emulator,
--        thresh_stats    => thresh_stats,
--        task_status     => task_status,
--        fsm_state       => fsm_nv_reg_state,
--        fsm_state_sig   => fsm_state_sig
--    );
--##DB##End
```

Finally, set `hazard_threshold` constant value to 2500 inside `I_DNN_multiple_images_tb.vhd` and set the voltage trace 2 by modifying the constant `voltage_trace_path` inside the very same VHDL file.

```
    constant voltage_trace_path: string(1 to 33) :=
        "voltage_traces/voltage_trace2.txt";
```

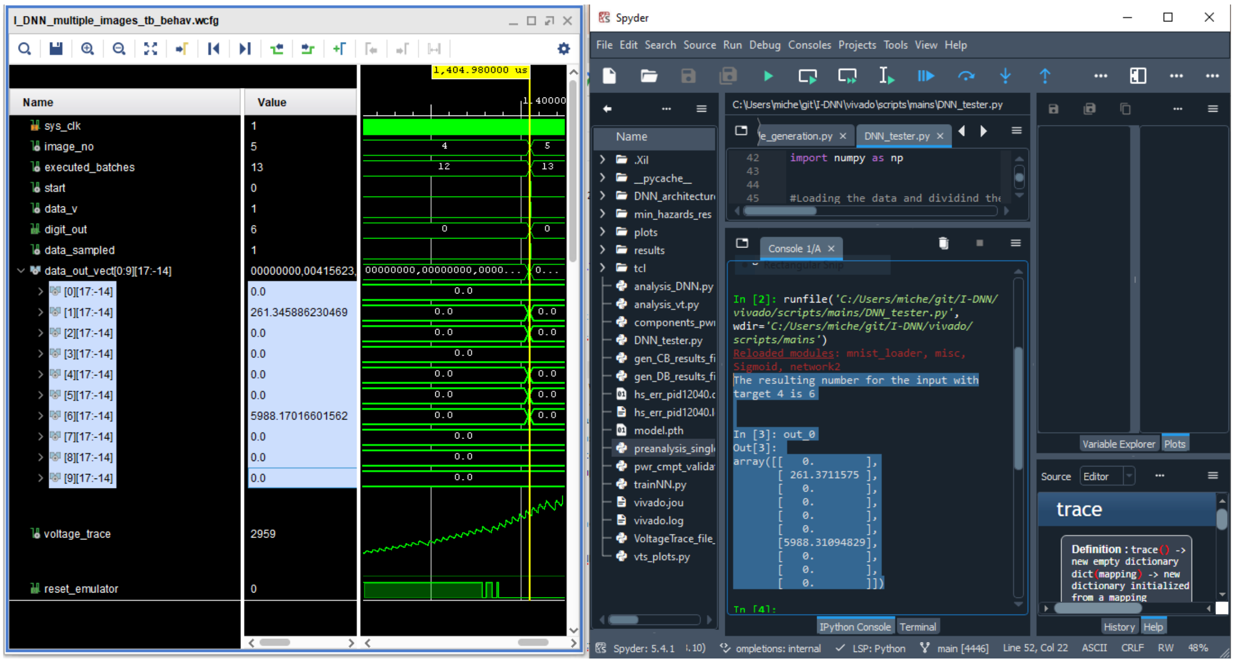

Figure 2: Comparison `image_no = 4` (MNIST testdata sample 1549) inference after power off

Run the simulation for 3ms and open the waveform configuration file `I_DNN_multiple_images_tb_behav.wcfg`. Inspect the DNN output for `image_no` = 4, the image number corresponds to the MNIST sample with index 1549 within the test dataset. To inspect the output go to the next rising edge of the signal `out_v` of the DNN and inspect the collection of outputs `data_out_vect` of layer4 at time `1404.98 us`. Set up the correct data format by right-clicking on the signal -> Radix -> Real Settings -> Fixed Point and set the size of the fractional size inside binary point to 14. It's also explicitly showed on the waveform signal `data_out_vect`.
Open the python script `DNN_tester.py` and set up the path to the newly generated net. To find the training ID read the constant value `DNN_prms_path` in `./src/DNN/I_DNN_package.vhd`, as described in section 3.3.
Run the python script. The variable `out_0` will contain the neurons' output of the DNN model. Compare this value with the `data_out_vect` value.

For this specific DNN model the image recognition is not accurate since the label is 4 but the classified digit is 6.

4.3 Evaluation

In this example the evaluation of a DB backup policy is carried out.

4.3.1 Simulation Data generation

The simulation data for the dynamic backup policy is generated with the python script
`gen_DB_results_fixed_time.py`.
The python scripts inputs are set as follows:

```
values = [0,2500,0,0,0,0,0,0,0,0,0]           #minimum hazard threshold for every trace.
DNN_architecture["num_hidden_layers"] = 4    #DNN number of layers
NV_REG_FACTOR = 2                             #Non-Volatile register delay factor
indexes = [1]             #Voltage trace indexes to be simulated. Voltage trace number 2
overall_final_value = 2950                   #End Hazard Threshold simulated
step = 30                                     #Hazard threshold step increment
num_sim = 5                                   #Number of consecutive simulations performed
```

This will produce chunk of results collecting 5 different hazard threshold simulations. Within the specified range [2500, 2950] there are a total of 450/30 = 15 different hazard threshold, so it will perform a total of 3 runs each with 5 consecutive simulations. It will produce 3 different files placed inside `./mains/results/DB_results/`:[1]

[1]It takes about 2 minutes and a half to perform a single simulation, so it will take approximately 37 minutes to perform all of them.

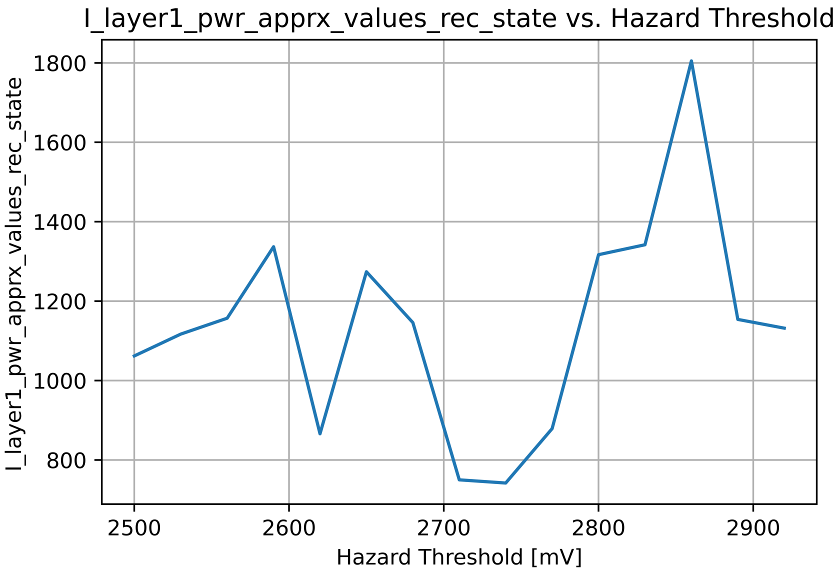

Figure 3: Power Approximation number of clock cycle values for I-layer 1 for the recovery power state.

```
DB_results_fixedtime_NVREG_DELAY_FACTOR2_voltage_trace2_4_2500_2650.txt
DB_results_fixedtime_NVREG_DELAY_FACTOR2_voltage_trace2_4_2650_2800.txt
DB_results_fixedtime_NVREG_DELAY_FACTOR2_voltage_trace2_4_2800_2950.txt
```

Before simulating, make sure you recomment the backup policy uncommented during testbenching. Also close the simulation open in Vivado if there are any.
Once you get the simulation results, you can merge all of them and make another output file. In this demo it is renamed `DB_results_fixedtime_NVREG_DELAY_FACTOR2_demo.txt`.
The data produced consists in end-of-simulation values and includes the number of executed batches, power approximation values for the I-DNN components.

4.3.2 Plotting Data

To plot this end-of-simulation data run the python script `DB_results_plot.py` by changing the python input script to:

```
results_path = "./results/DB_results/DB_results_fixedtme_NVREG_DELAY_FACTOR2_demo.txt"
plt_show=True
plt_save=True
```

Figure 3 is one of the plot produced. It shows the number of clock cycles the I-layer number 1 spent in the recovery power state.

5 Performance-based characterization scripts

To plot the figure to produce the performance-base characterization results of the thesis run the python scripts `analysis_DNN.py` and `analysis_vt.py`. For the power validation model run `pwr_cmpt_validation.py` and for the analysis of the single components power consumption `pwr_cmpt_validation.py` is the one to refer to.

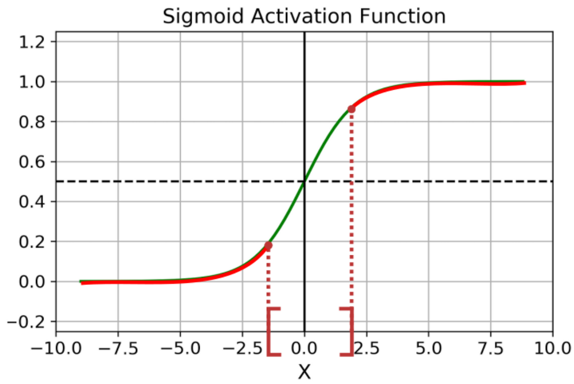

Figure 4: Sigmoid Activation Function - The red tail values are lost due to input quantization.

INPUT	**ADDRESS**	**OUTPUT**
10.0000_2 = -2.0_{10}	000000_2	0.1192
...	...	...
00.0000_2 = 0_{10}	100000_2	0.5
...	...	...
01.1111_2 = 1.9375_{10}	111111_2	0.8740

Table 1: Sigmoid LUT for 6 bit size sigmoid input

A Sigmoid Implementation

The sigmoid is implemented using a LUT storing the sigmoid value for a given input. The sigmoid function is:

$$y = \sigma(x) = \frac{1}{1+e^{-x}} \tag{1}$$

The domain is $D = \mathbb{R}$, while the codomain is $C = [0, 1]$

The domain and codomain are both quantized using fixed-point representation, obtaining respectively input and output sizes of the sigmoid. The dimension of the sigmoid LUT depends on the size of the input. In fact if `sigmoid_input_size` is the size of the sigmoid, the total number of sigmoid LUT output values will be $2^{\sigma_{SIZE}}$. The greater the `sigmoid_input_size`, the bigger the LUT, the more resources the FPGA needs to allocate.

`sigmoid_input_size` is split into the integer and the fractional part of the sigmoid (respectively `sigmoid_input_int_size` and `sigmoid_input_frac_size`).

`sigmoid_input_int_size` determines the input range of the sigmoid, which is $[-2^{N_{int}}, 2^{N_{int}} - 2^{-N_{frac}}$, while `sigmoid_input_frac_size` determines the sampling step within such interval.

The `training`$_{f}ile.txtallowstosetonly$`sigmoid_input_Int_size`$and$`sigmoid_input_int_size`$.The fractional partissetbydoing$