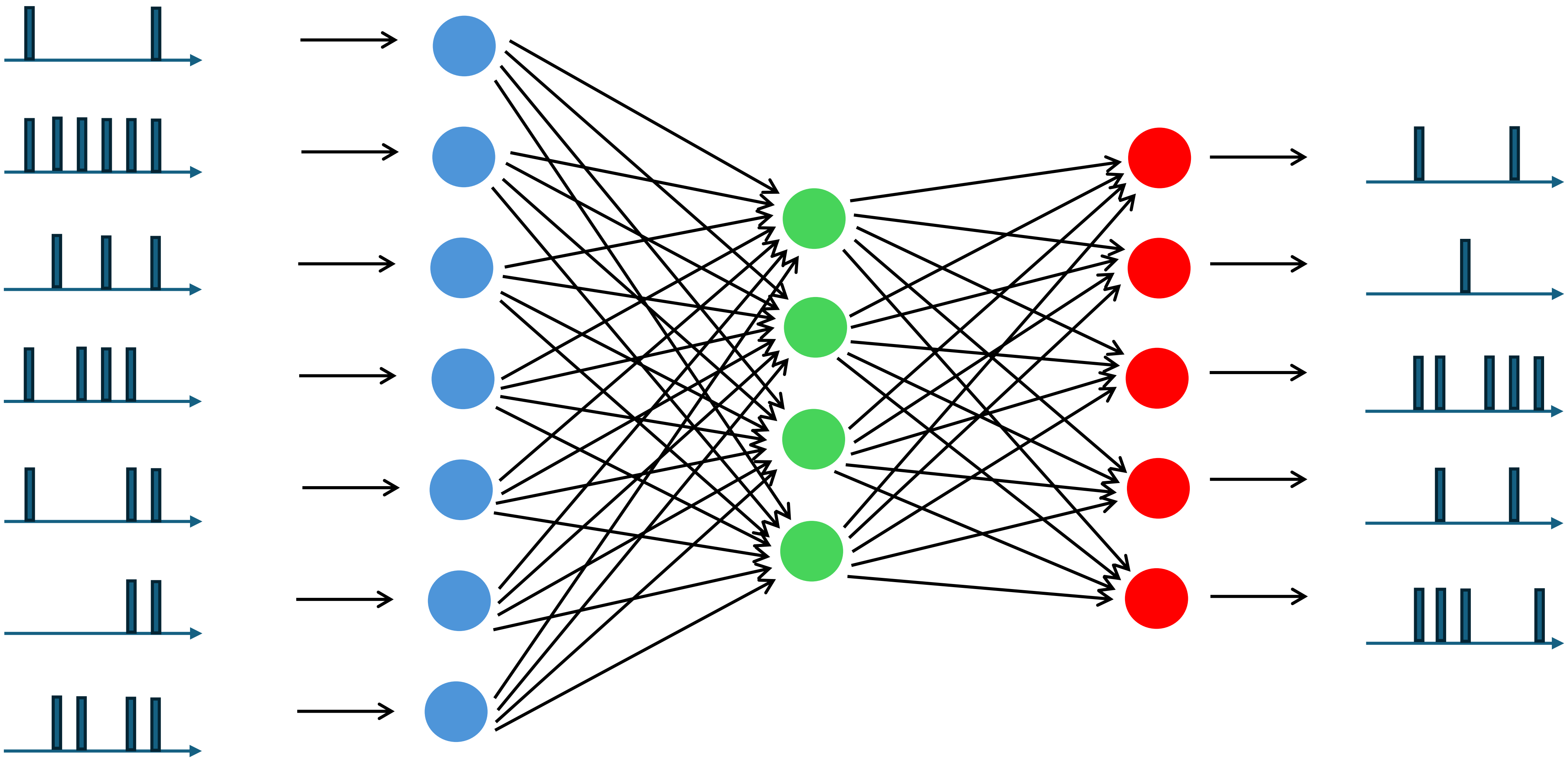
# Tutorial on Spiking Neural Networks



# Mentored Work in Didactics

## A) Preface

This tutorial was developed as part of a teaching training course. Some introductory pages are therefore dedicated to the didactic discussion and explain the motivation and structure of the tutorial.

The tutorial is suitable for self-study. Reading the text and completing the exercises takes approximately 4 hours.

Readers of this tutorial require the following skills and previous knowledge:
- Basic knowledge of machine learning, i.e., knowledge of how an artificial neural network is structured.
- Python programming knowledge in TensorFlow.
- Knowledge of vhdl programming including compilation, elaboration and simulation.

## B) Motivation

The motivation to create this tutorial is to give students writing their bachelor’s or master’s thesis at IBM Research Europe on the design of spiking neural networks, a way to quickly familiarize with the subject through self-directed learning. This means that students must set their own learning goals, engage in the learning process and evaluate their learning. The tutorial supports the latter two goals

by explicitly applying a simplified version of Bloom’s learning taxonomy. The didactic concept applied is to introduce a specific topic in order to provide a first insight into the subject and then to deepen certain aspects by asking questions of different levels of complexity according to Bloom's learning taxonomy. The questions are then answered on separate pages, often with further insight into the theory or topic covered.
The idea behind this self-directed learning tutorial is to give supervisors some time back by relieving them from training students dealing with the topic of hardware implementation of neural works.

## C) Knowledge Modules

**INTRODUCTION** – The tutorial starts with a brief historical overview on ANNs and SNNs. ANNs are the main workhorse of todays machine learning. SNNs are biologically inspired by how our brain works in terms of spike transmissions between neurons.
The motivation to investigate SNNs is related to the sparsity of spikes, which may lead to the implementation of energy efficient neural networks.

**ANN** – This knowledge module provides a brief summary of how an ANN works so that all readers can start with the same basic knowledge.  Key terms used in this  a priori knowledge module are neuron, layer, input feature, weights, biases, activation function, training, inference, predicted and true labels, loss function, gradient descent algorithm, learning rate, hyper-parameters, batch size and epoch.

**Spike Transmission** – This knowledge module gives an introduction into the basics of spike transmission used in SNNs. A simplified version of Bloom's taxonomy is used to review the material presented by answering questions with different levels of difficulty and complexity. The contents taught in this knowledge module are related to different encoding schemes such as rate encoding, inter-spike interval encoding, time-to-spike encoding and time-to-first-spike encoding. In addition, the data processing of spikes within an SNN neuron is presented and compared with the data processing of an ANN neuron in order to relate to the previously presented knowledge module about the a priori knowledge of ANNs and to provide an outlook on the hardware implementation of SNNs, which will be presented in one of the next knowledge modules.

**Modeling of SNN** – The content of this knowledge model is related to the application of the spike transmission theory to the dataset of the Iris flower classification. The Iris flower dataset was chosen because it allows the implementation of the entire neural network as an ASIC, embedded in a padcage with the dimensions 1 mm x 0.5 mm, which is a commonly used size for ASIC experiments on MPWs. This knowledge module first starts with an introduction to the Iris dataset, followed by the presentation of network topologies used for the Iris flower classification.

**Neuron Models** – In this knowledge module three block diagrams of neurons used for the hardware implementation of ANNs and SNNs are presented. The ANN and TTS neuron models are used below for the hardware implementation based on synthesized logic using vhdl in a 5nm finFET CMOS technology.

**Framework Modeling with TensorFlow** – The Iris flower classification is programmed using Python in TensorFlow for the training of the weights and biases. This knowledge module first discusses the floating-point training and inference. In a second step, quantization-aware training is introduced to prepare for the VHDL implementation, which cannot use floating-point values.

**VHDL Implementation** – This knowledge module presents the vhdl implementation of the Iris flower SNN with TTS encoding.

## D) Didactic Concept

A key aspect from a didactical point of view is the use of a simplified version of Bloom's learning taxonomy. In this tutorial, the original taxonomy pyramid is reduced to just three levels. The basic idea is to first present the background and description of a specific aspect of the topic, for instance, the encoding of the time-to-spike (TTS) signaling scheme. Each page of the tutorial is dedicated to a specific topic – indicated by the title of the page.

Bloom's learning taxonomy is then applied by asking questions labeled L1, L2 and L3 according to their level of complexity. For example, an L1-question applied to the example above asks to summarize the explanations given in the text about how TTS signaling works. A question marked L2 asks to assume a network consisting of two neurons that are transmitting to a third neuron using the TTS encoding scheme and to draw the associated waveforms using your own example. Where L1-questions target Bloom's taxonomy levels of remembering and understanding the concept, L2-questions are about the application and analysis of the concept.

The L3-questions refer to the highest level of complexity and require the evaluation of a problem and the creation of solutions that go beyond the original work or description of the problem. For the example given above, an L3-question might require developing a TTS encoding receiver using a digital integrator and drawing the appropriate block diagrams.

Each of the questions is answered on a separate page, accessible in both directions via hyperlinks in the text. This allows a self-directed learning, which is the main goal of the tutorial.

## E) Didactic Considerations

The concept of using a simplified version of Bloom’s learning taxonomy for this tutorial is motivated by shaping the learning process using the proposed L1, L2 and L3 questions.

Here are some points about the didactic design of the tutorial:

Because the tutorial can be reused by different students during self-study, the availability of a tutorial is a time advantage for the supervisors so that they do not have to explain the basics all over again every time a new student comes in and starts a project work on the development of SNN hardware.

Text slides with questions can be kept simple. The complexity can be put into the answer slides. If someone just goes through the text slides without studying the corresponding answer slides, a rough understanding of the topic can still be gained.

Bloom's classification of questions makes it possible to use L1 questions to underline a certain topic without providing any new information.  An example of this is the L1 question on the topic of integrate-and-fire dynamics, where the name already says it all and is intended to encourage students to think about what the connections are between the technical implementation (e.g. integrator, counter) and the biological model (e.g. membrane potential, sending out a spike when a threshold value is reached.)

L2 questions can also serve to direct the reader to a new topic by asking the question in such a way that the answer inevitably leads one into the new topic. An example of this method is the question about the advantages and disadvantages of rate encoding. When answering this, one encounters the concept of using a counter for demodulation. The further train of thought is then to view a counter as a digital integrator, which brings us to the next topic that introduces the temporal encoding of spikes, in which integrators are of fundamental importance for demodulating time-encoded spikes.

One additional aspect that makes the self-direct learning tutorial attractive is that the question-answer structure with having interlinked pages for each top can easily be adapted to new topics or reshaped to a new focus of the SNN design in the context of applying it in a research environment.

## F) Conclusions

The idea of writing a self-directed learning tutorial to engage students in SNN hardware implementation projects is a didactic experiment. It is not possible to evaluate the students' feedback within the available time frame of this teacher training course. New students to be introduced to the topic could not begin until spring 2025, after this work has been submitted.

Nevertheless, it might also be useful for people who have experience with hardware implementation of SNNs, as the tutorial covers all aspects (except the actual physical design of the ASIC), from spike transfer theory to framework simulations to determining the weights and bias to the VHDL description of the simple Iris flower classification task.

# Contents (1)

# Contents (2)

# Contents (3)

# Objectives

This tutorial aims to provide an overview of spiking neural networks (SNN). It consists of short texts that explain certain aspects of SNNs. Each text block ends with a few questions.

The questions are intended to query what has been learned or to encourage the application of what has been learned. According to Bloom's learning taxonomy (see Fig. 1 and reference [1]), the cognitive skill levels required to answer the questions are denoted by L1, L2 and L3.

The questions marked L1 ask for knowledge from the text. The questions marked L2 go further and require the application and analysis of what has been learned. The correct answers to L1 and L2 questions can be obtained by clicking on the pertinent cross-references belonging to these questions.

The questions marked L3 involve the development of program code. A suggested solution to a L3 question can be obtained in the form of a program listing by selecting the provided cross-reference.

The tutorial requires prior knowledge of machine learning principles and basic knowledge of the Python programming language as well as vhdl skills.

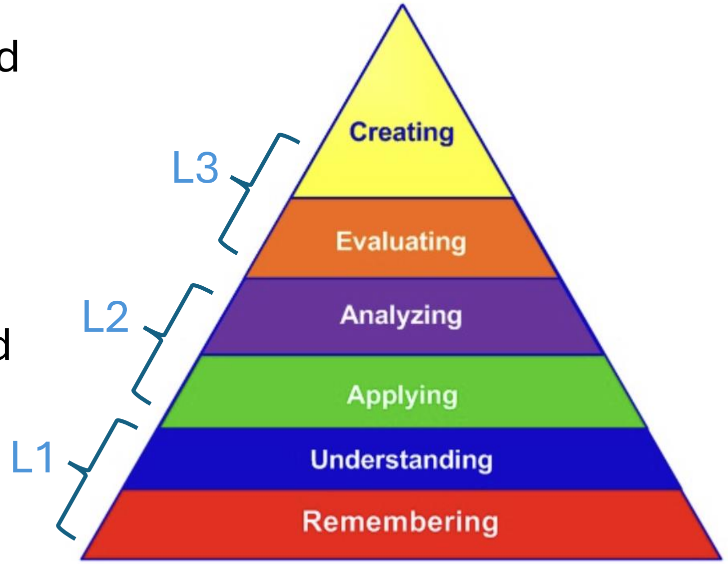



**Fig. 1**: Bloom's taxonomy [1].

# Introduction

The tutorial is organized as follows. It begins with a brief overview on the history of artificial neural networks (ANN) and spiking neural networks (SNN). ANNs are the workhorse of todays artificial intelligence (AI), mostly based on large language models (LLMs) or foundation models running on graphics processing unit (GPU) clusters. SNNs are biologically inspired by the integrate-and-fire dynamics of biological neuron cells and aim at reducing the power consumption to become more energy efficient than ANNs.

It is assumed that the reader of the tutorial already has some background of neural networks – in particularly of ANNs. To get everyone on the same page, a brief summary is given on how ANNs  are structured and operated.

After this introductory part, the tutorial is focused on the operation of SNNs. First the different encoding schemes are presented. Emphasis is given to the temporal encoding of spikes.

Next the Iris flower dataset is introduced as a simple dataset to perform the program coding exercises with Python and TenserFlow given towards the end of the tutorial.

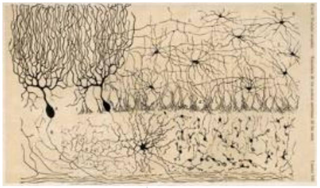
A further part of the  tutorial is devoted to hardware implementation concepts of SNNs suitable for hdl description. The tutorial ends with the presentation of the vhdl code of the TTS-encoded implementation of the Iris flower classification task. The RLM construction flow is not part of the tutorial.

# History and Fundamentals of Neural Networks

# 1 History of Artificial and Spiking Neural Networks

Simulation of the structure

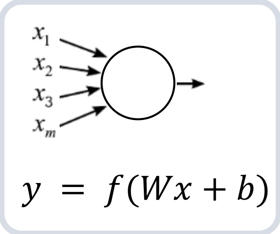
**1906**
Ramon y Cajal - the structure of the nervous system – Nobel Prize

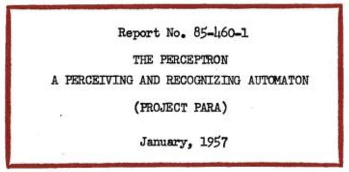


Bio-realistic emulation of the dynamics

## Artificial Neural Networks (ANNs)

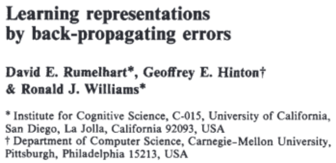
**1943**
McCulloch and Pitts present an Artificial Neuron





Report No. 85-460-1
THE PERCEPTRON
A PERCEIVING AND RECOGNIZING AUTOMATON
(PROJECT PARA)
January, 1957

**1957**
Rosenblatt's perceptron (1L ANN)

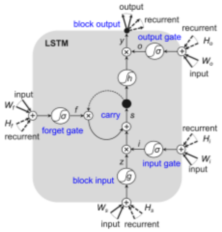
**1986**
Hinton – learning representations with BP (multi-layer ANN)


**Learning representations by back-propagating errors**

**David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams***

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

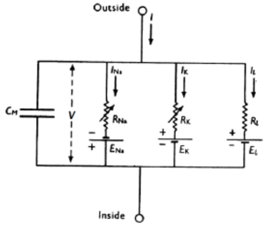
**1997**
Schmidhuber LSTM

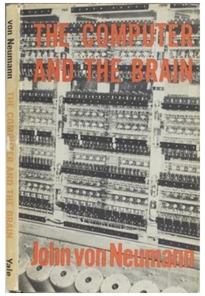




**2012 - today**
GPUs for ANNs, large-scale processing systems deep learning "revolution

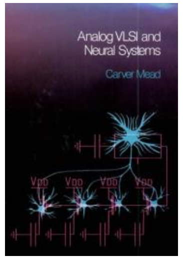
**1952**
Hodgkin-Huxley model of spiking neural dynamics – Nobel Prize

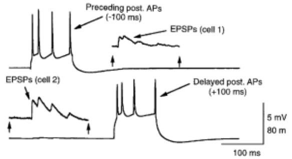



**1956**
von Neumann postulates SNN-based architectures

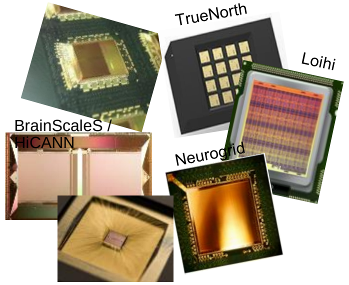
**1980s**
C.Mead @ Caltech – Neuromorphic Engineering




**1997**
Markram STDP learning

**2014 - today**
Neuromorphic accelerators




## Spiking Neural Networks (SNNs)

# 1.1 Basics of Neural Networks (Recap 1)

The single-layer perceptron (SLP) is the simplest form of an ANN. As illustrated in Fig. 2 it consists of an input layer with the input features $x_1, x_2, \ldots, x_n$. Each input $x_i$ has an associated weight $w_i$ that is a parameter to be trained. The partial products $w_i x_i$ are summed up and an additional fitting parameter called bias b is added that helps fit the data to the activation function. This function mimics the crossing of a membrane potential in a biological neuron and determines whether the neuron should be activated or not. Figure 3 shows a few activation functions used in this tutorial. Another commonly used function is SoftMax that performs the classification at the output node.

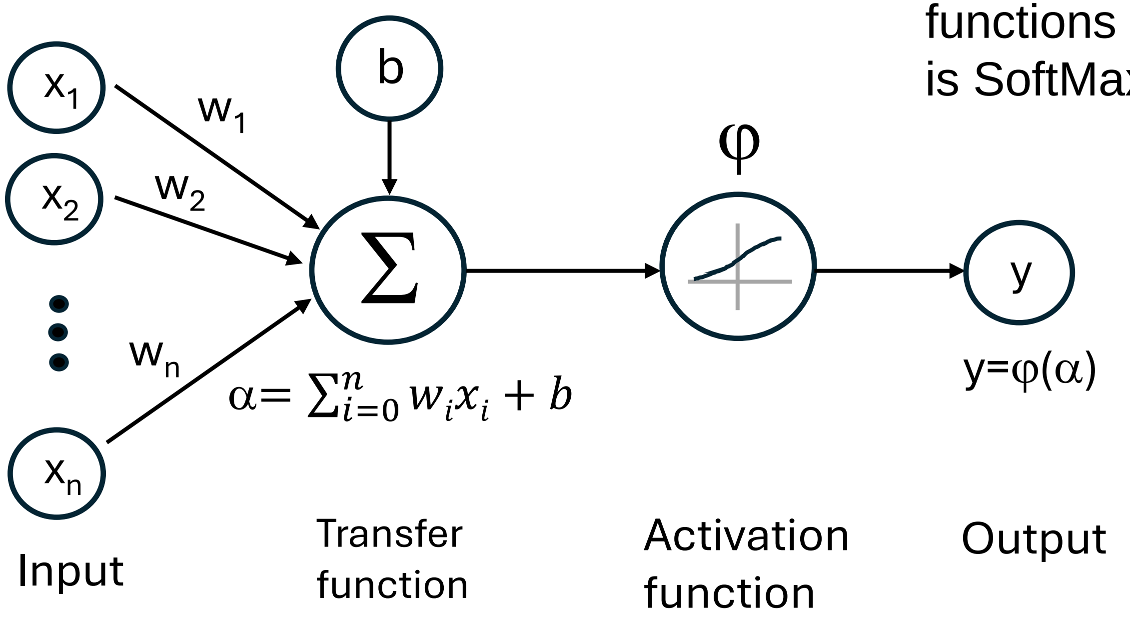



**Fig. 2**: Single-layer perceptron (introduced by Franck Rosenblatt in 1957).

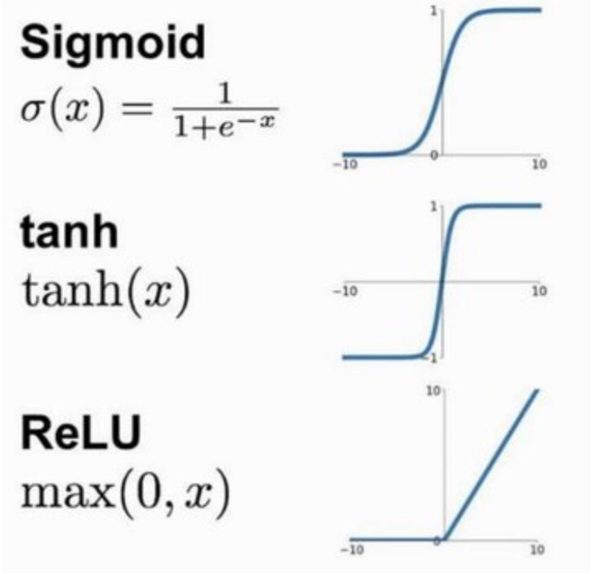



**Fig. 3**: Activation functions.

# 1.2 Training and Inference (Recap 2)

Figure 4 shows a multi-layer perceptron (MLP) with more than one hidden layer. In the learning or training phase the weights and biases associated with the individual neurons need to be determined by an iterative process of minimizing a loss function that describes the difference between the predicted outputs and the true labels. The adjustment of the weights and biases is based on using a gradient descent algorithm. The learning rate η can be used as a hyperparameter to determine the step size during the weight update. As an example, the Delta-rule applied to the training of a SLP is given below. Adaptive methods such as e.g. Adam can used to adjust the learning rate. Apart from the learning rate there are other hyperparameters including the number of hidden layers, the number of neurons per layer, the activation function or the batch size. The term epoch describes one complete pass through the entire training dataset. Several epochs are required to achieve a decent accuracy level (e.g. >96%). Once the training phase is completed, the inference can be performed with user data.

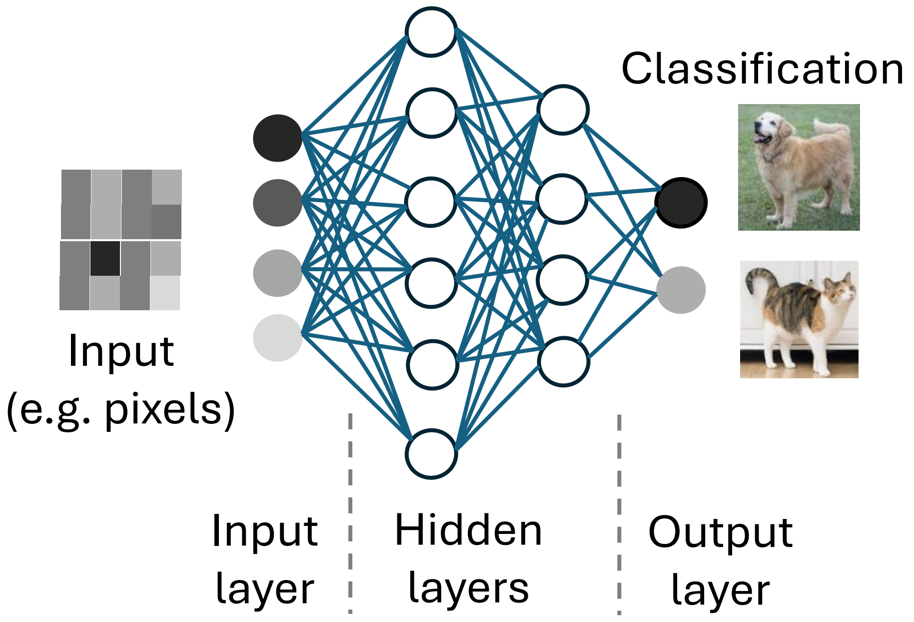


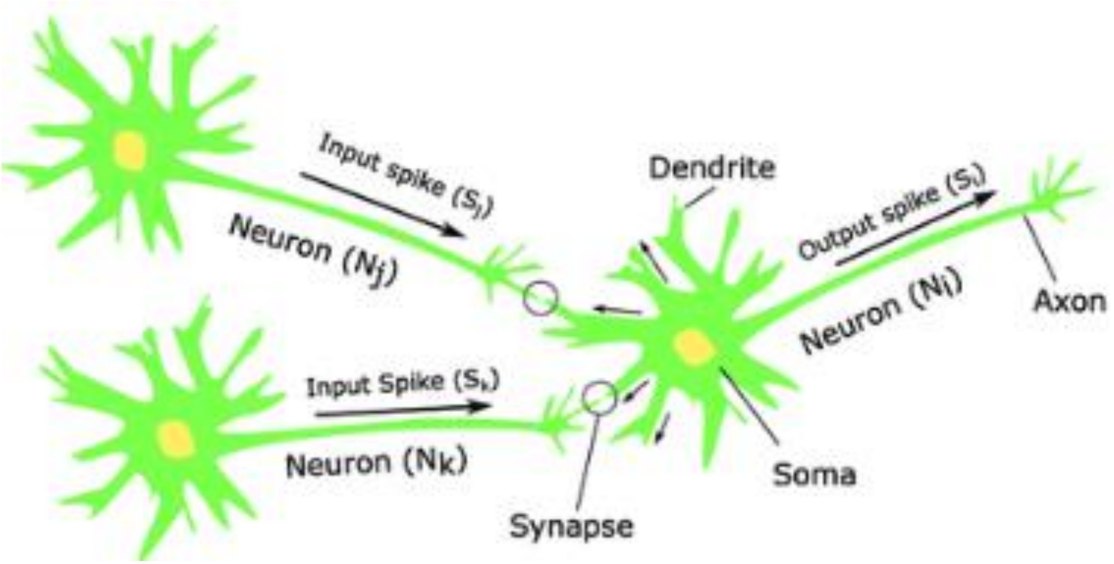

**Fig. 4**: Multi-layer perceptron.

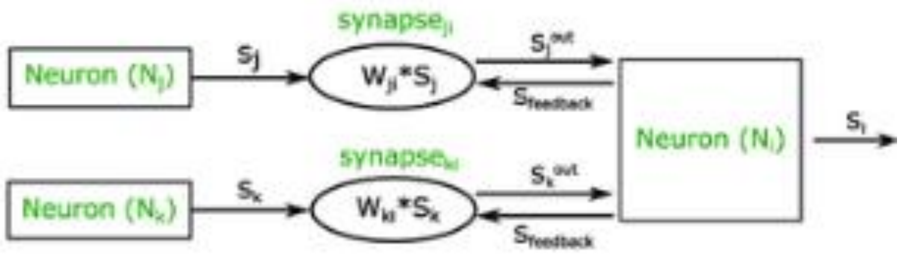
Delta-rule for SLP training:

$$w_{i_{new}} = w_{i_{old}} + \eta(y_i - \hat{y}_i) \cdot x_i$$

where w: weights, y: true label, $\hat{y}$: predicted output, $\eta$: learning rate

# Basics of Spike Transmission




Source: https://www.researchgate.net/figure/Spiking-neural-network-illustration_fig1_347207968

# 2 Spiking Neural Networks

The objective of spiking neural networks (SNNs) is to reproduce a behavioral model of the sophisticated chemical processes occurring in our brain [2]. This is achieved by means of electrical signal processing and neuronal transmission based on encoding information as sequences of short pulses, called spikes, and by applying neuronal dynamics, here modeled as integrate-and-fire dynamics.

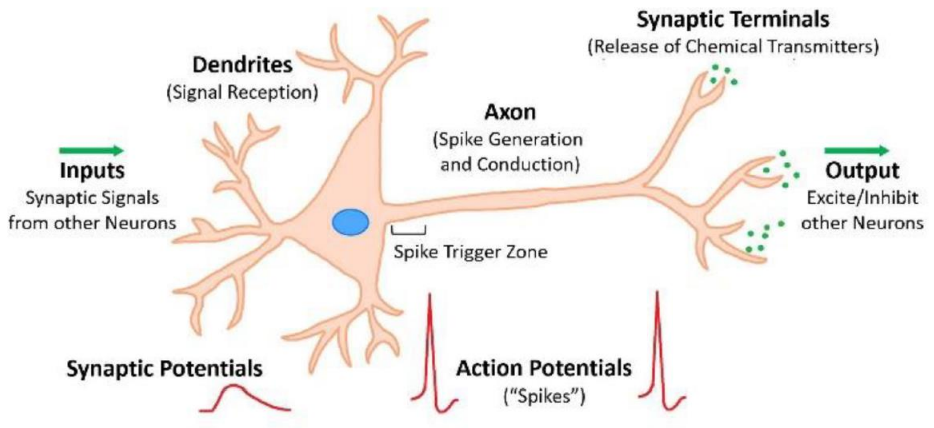



**Fig. 5**: Biological neuron cell applying spike transmission [2].

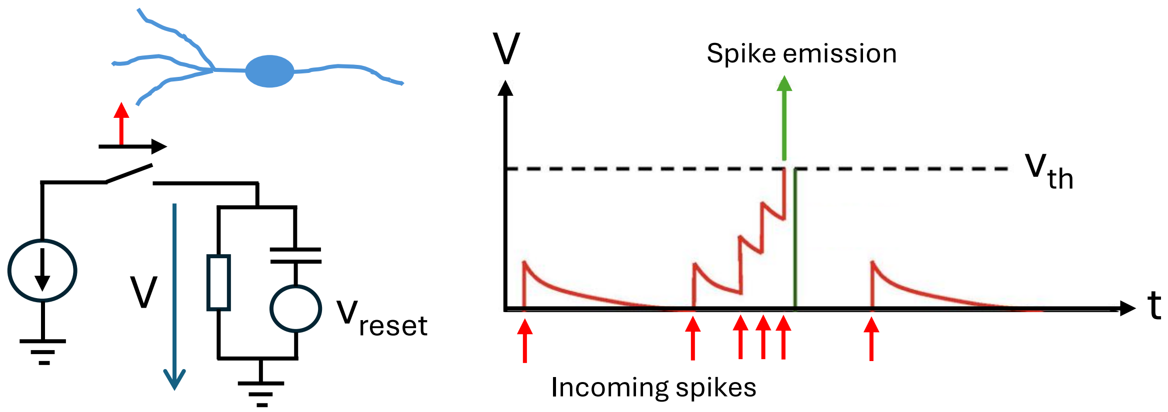



**Fig. 6**: Electrical modelling of the leaky integrate-and-fire dynamics.

Questions:

L1: What is the name of the process used to deal with spikes in neurons?

L2: Study Fig. 5 and Fig. 6 and explain how the two figures are related.

# 2.1 Rate Encoding of Spikes in SNNs

In ANNs the data transmission between different neurons occurs using floating-point values. In SNNs, however, the data to be transmitted is mapped to spikes that either perform rate encoding or temporal encoding as described in more detail below.
The most common method for transmitting information in SNNs is rate encoding. The information to be transmitted is encoded in the rate at which the spikes – also known as spike trains – are being fired or transmitted. Rate encoding originated as a method to encode neuronal signals after it had been first observed in a stretch receptor of a muscle in 1926 [3].

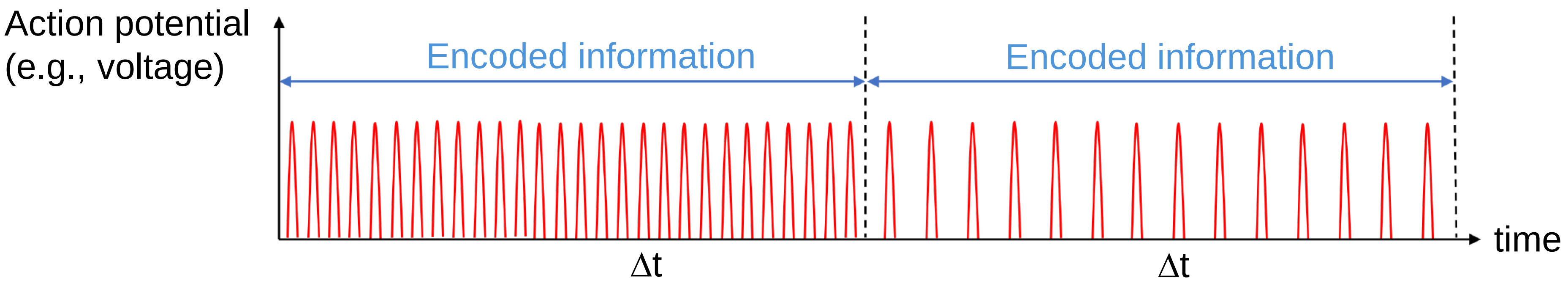



**Fig. 9**: Rate encoding illustrated with two spike trains [2].

L2: Describe the advantages and disadvantages of rate encoding when implemented with an electrical circuit, e.g. a resettable counter measuring the rate of the encoded data.

# 2.2 Time Encoding of Spikes in SNNs

The time encoding of spikes assumes that the information is contained in the precise spike timing. This contrasts with rate encoding where a single spike just contains parts of the information. An approach to model time encoding can be implemented by defining a logic 1 as a spike whereas the absence of any spike can be seen as a logic 0. Hence, the information conveyed is contained in the time interval between adjacent spikes whereas in rate encoding the information is encoded in the spike density.
An initial discovery that not all neuron cells might work with rate encoding is based on a study from 1996 [4]. In this study the time had been measured for a human brain to recognize animals on photographs. This can be done in less than 150 milliseconds. The study concluded that the processing must be based on feedforward mechanism when considering the number of processing stages involved with the brain. Moreover, the determination of a rate - which represents the information being transmitted - is not possible in such a short period of time. Therefore, rate encoding to accelerate the transmission of spikes can be ruled out; instead, time encoding of spikes becomes more promising.

L2: Explain what the reasoning could be why time encoding of spikes can increase the data rate compared to rate conceding?

# 2.2.1 Inter-Spike Interval Encoding

The inter-spike interval encoding represents the information to be transmitted through the time interval set by a leading spike and closed by the next spike. Hence the information is transmitted by measuring the time elapsed between two adjacent spikes. Except for the first spike in the spike train all other spikes have a dual role to play by first acting as an information spike, followed by acting as a reference spike for the next information. An advantage of this approach is that there is no need for a collective time synchronization over all neurons which communicate with each other. What remains is the problem of defining a system clock.

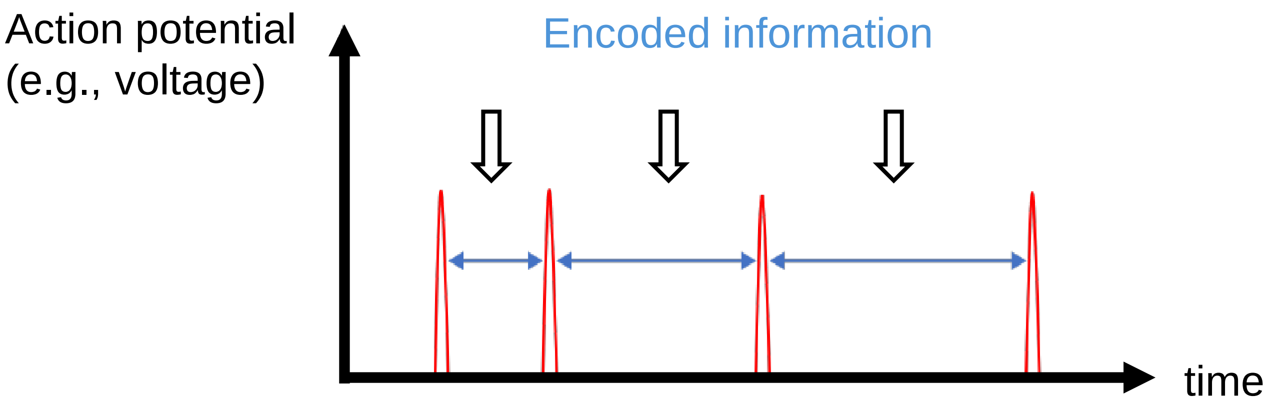



**Fig. 10**: Illustration of inter-spike interval encoding: The time difference between two adjacent spikes contains the information.

L1: What is the advantage of this encoding scheme? What is the drawback?

# 2.2.2 Time-to-Spike Encoding (TTS)

The time-to-spike (TTS) encoding is a synchronous spiking scheme. It is well suited to mimic the behavior of ANNs. In ANNs the data to be transmitted is a code, i.e., a binary or decimal number. In TTS encoding the data to be transmitted is just a single spike that occurs at the time position within the observation interval $\Delta t$ belonging to that code. It is important to note that the code value corresponds to the elapsed time between the spike and the end of the observation interval. This definition of time encoding is motivated by the fact that the transmitted spike triggers the integration of the membrane potential in the next neuron receiver.

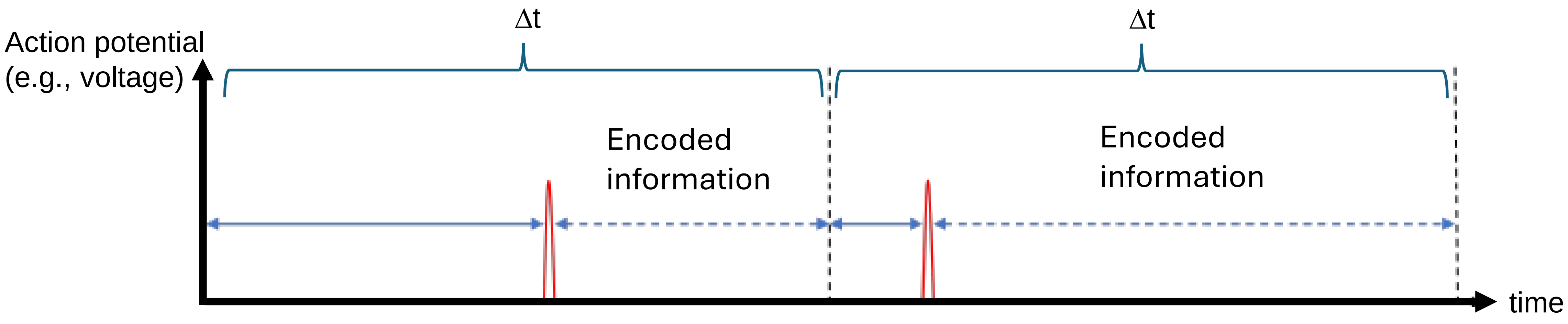



**Fig. 12**: Illustration of time-to-spike encoding where the information is encoded by the time interval between the spike and the end of the observation interval.

L1: What is the purpose of the observation interval? L2: Explain how the TTS encoding works at the neuron receiver.

# 2.2.3 Time-to-First-Spike Encoding (TTFS)

The time-to-first-spike (TTFS) encoding is an asynchronous spiking scheme. It comes closest to how our human brain works.
In TTFS encoding a stimulated neuron spikes once its membrane potential reaches a given threshold. The stronger the stimuli, the earlier the threshold is reached, and the next spike is sent out. As opposed to the synchronous TTS encoding scheme there is no prefixed observation interval. Hence, TTFS encoding is optimal for neural networks requiring low latency.

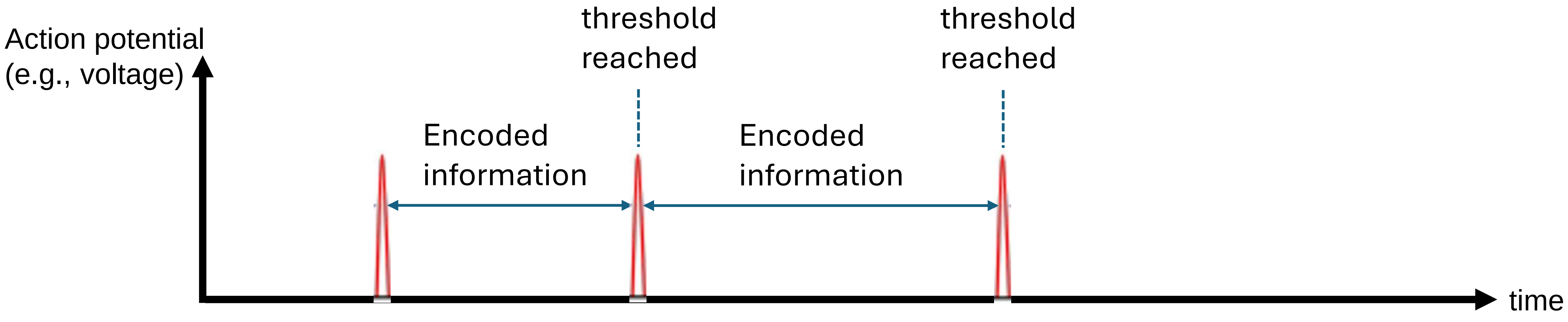



**Fig. 13**: Illustration of time-to-first-spike encoding where the information is encoded in the spiking time with respect to when the membrane potential crosses the threshold.

L2: Explain the working principle of TTFS with a neuron receiving spikes from two preceding neurons.

# 2.3 Data Processing within Neuron

Figure 14 shows an exemplary neural network consisting two neurons (N1, N2) that send information to a third neuron (N3). The information $x_1$ is weighted by the weight $w_1$ in neuron N3 and $x_2$ is weighted by $w_2$. Depending on whether the data is processed by an ANN or SNN, the hardware required to perform the data processing looks differently as illustrated below.

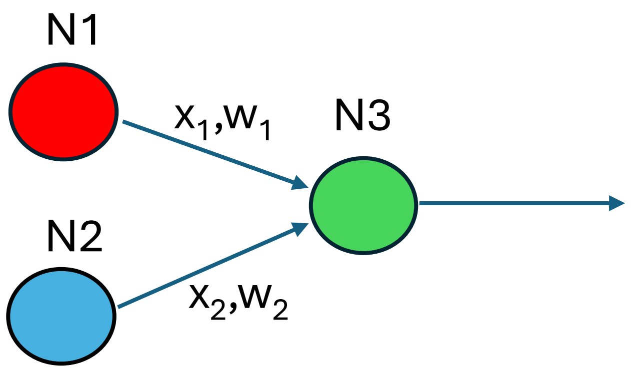
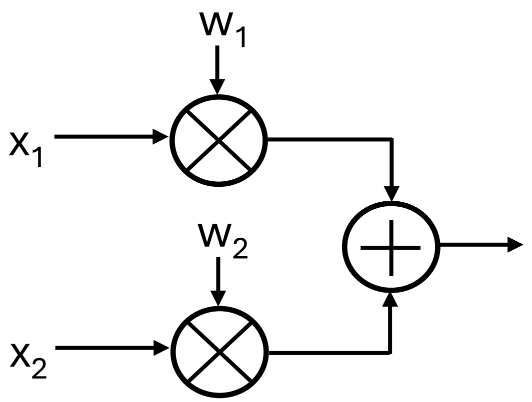
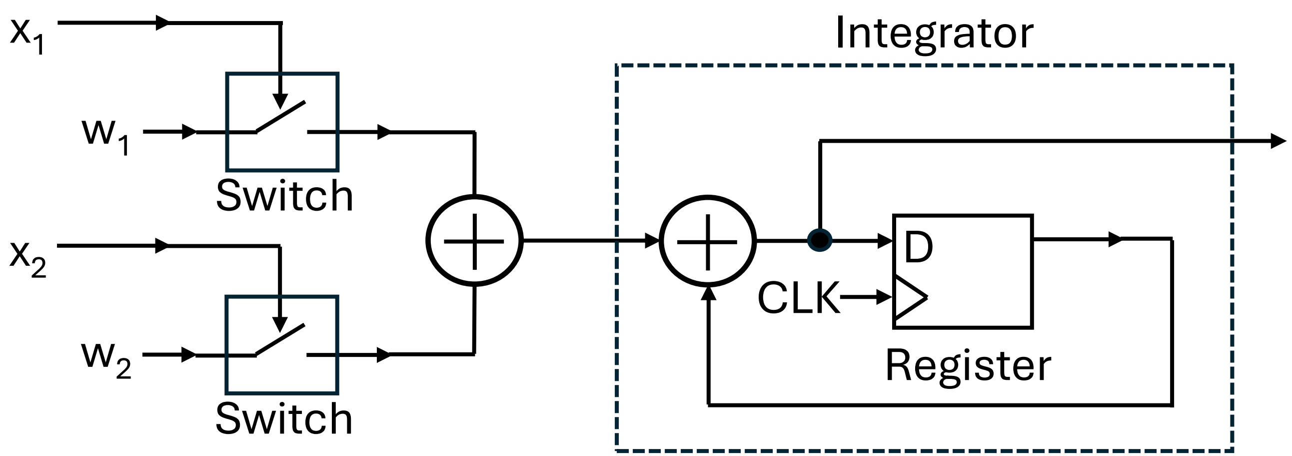



**Fig. 14**: Exemplary neural network.

**ANN**: The data processing is performed with a Multiply-and-Accumulate (MAC) operation.



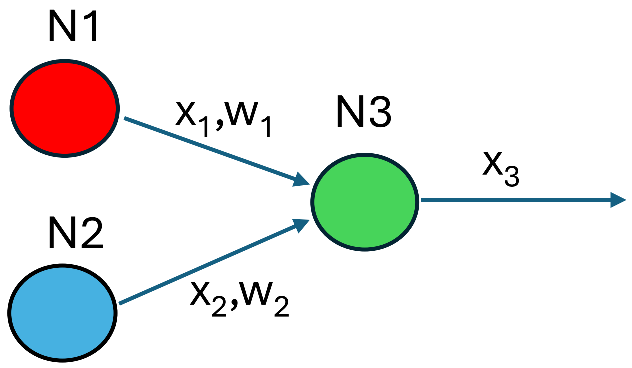

**SNN**: The data processing is performed by an integrator.

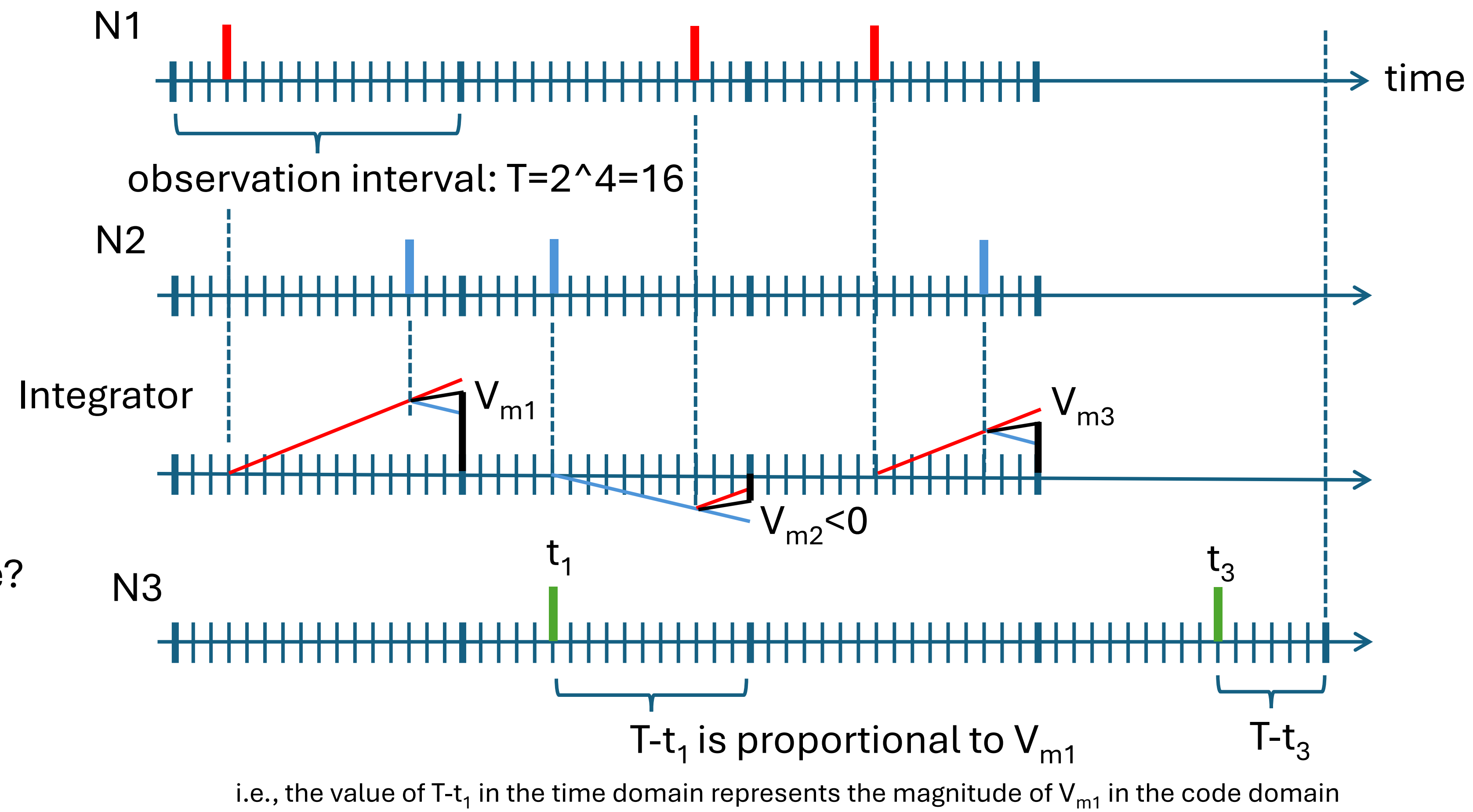



L1: Demonstrate with a numerical example that the MAC operation of the ANN and the SNN-integration are the same.

# 2.4 TTS Transmission Example

Later in this tutorial, the vhdl programming of a TTS-based SNN will be presented. To that end a TTS transmission example is given here to consolidate the material learned so far.

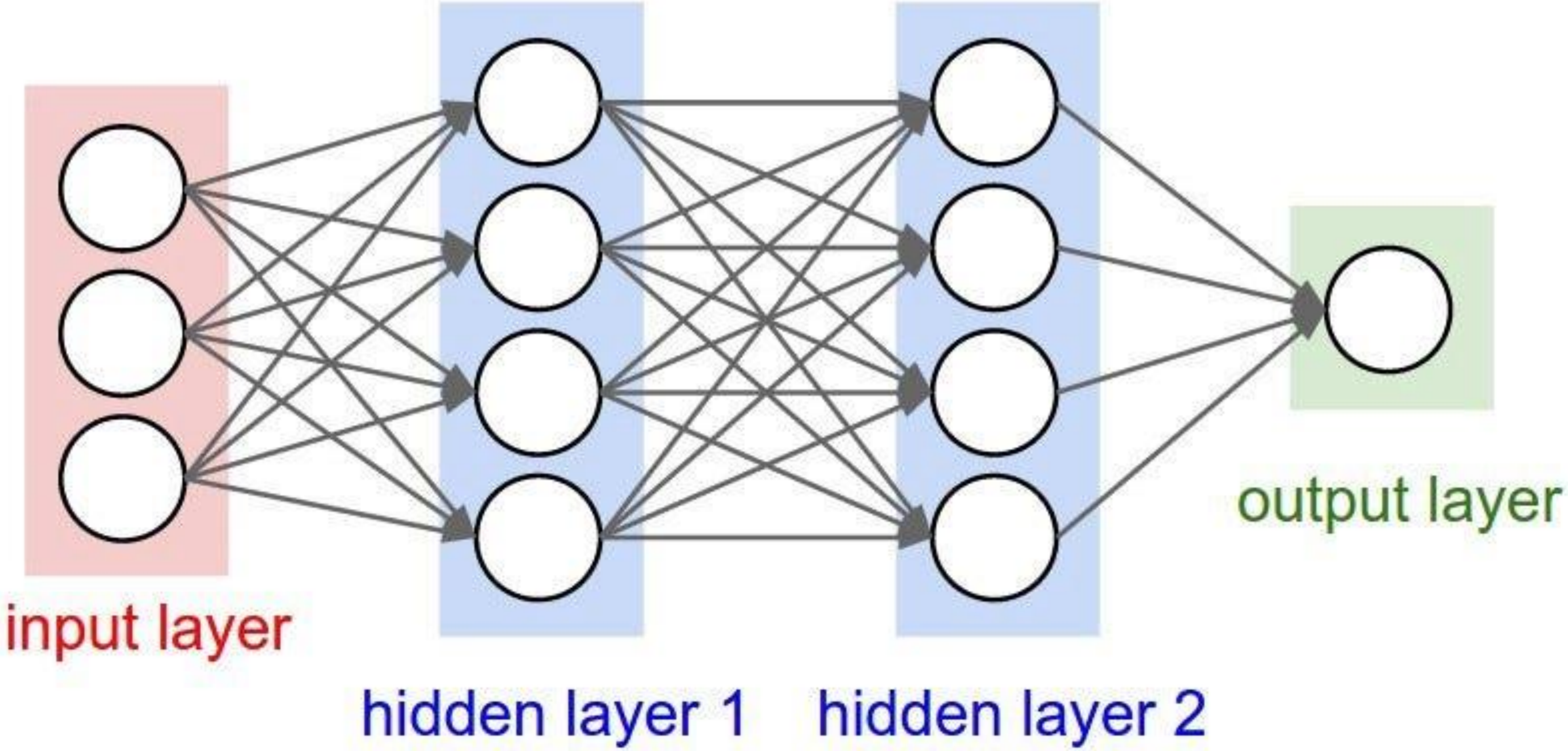



L1: Why does N3 generate no $t_2$-spike?

L2: How can $(T-t_i)$ be implemented?

# Modeling of Spiking Neural Networks



# 3 Iris Dataset (1)

The Iris flower dataset is used in this tutorial for training and inference. It was introduced in 1936 by the British statistician and biologist Roland Fisher who classified three related species of Iris flowers based on the data collected by the American botanist Edgar Anderson [5], [6].

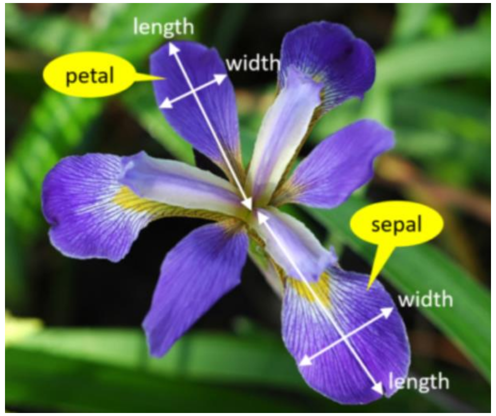
The dataset consists of 150 samples, i.e., 50 samples from each of the three species of the Iris flower (*Iris setosa*, *Iris versicolor*, *Iris virginica*). Four features are measured from each sample: the petal length, the petal width, the sepal length and the sepal width. All measurements are in centimeters. An example of each Iris flower type is given in the figures below as wells as in the figure at the bottom right with the petal and sepal leaf definition. The whole dataset used in VHDL simulations is depicted here.

Iris setosa

Iris versicolor

Iris virginica

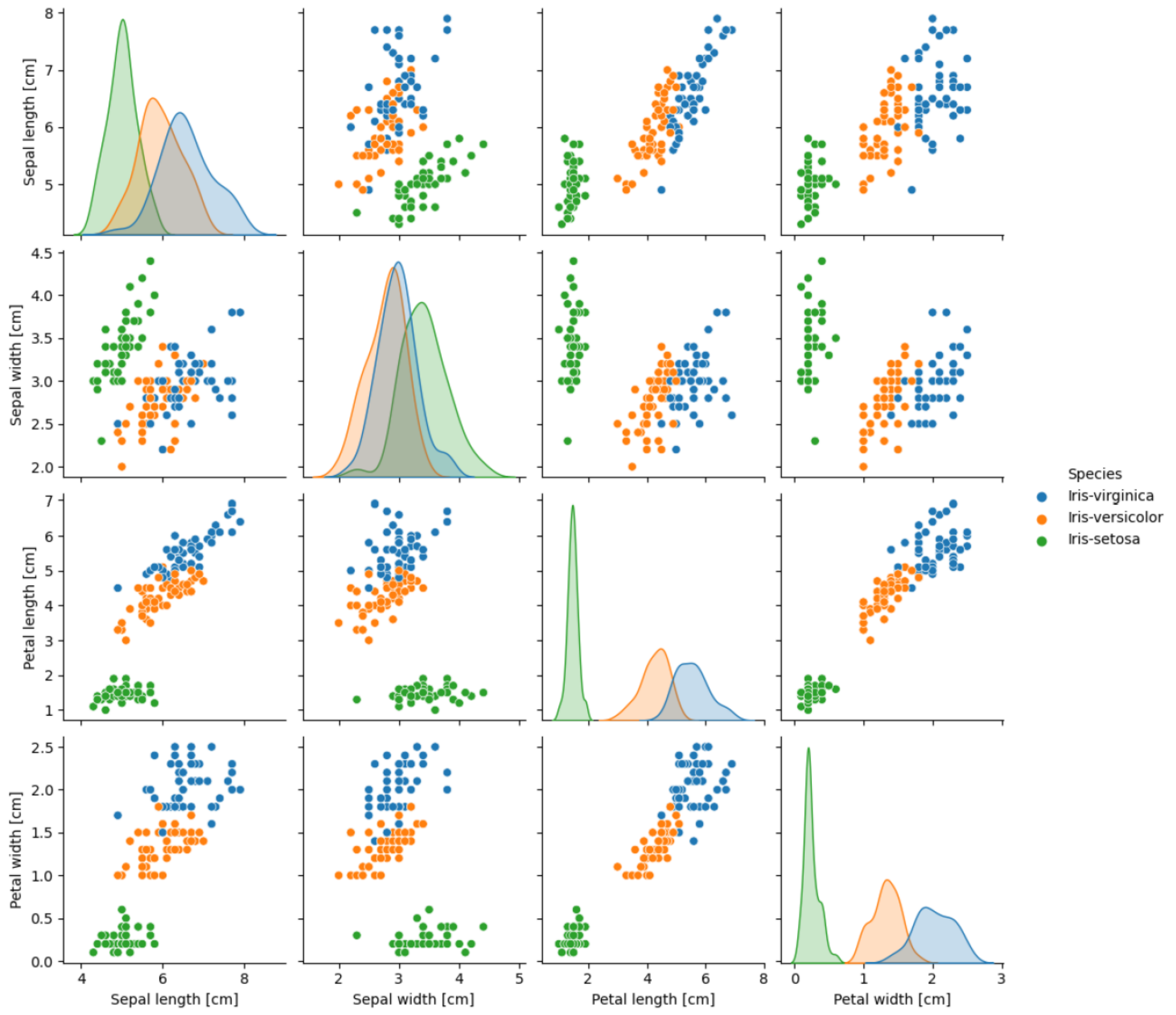
Definition of width and length of petal and sepal leaves



# 3 Iris Dataset (2)

The scatter matrix visualizes the Iris dataset with a density plot for each attribute (diagonal from upper left to lower right) and by plotting the attributes against each other in scatter plots.

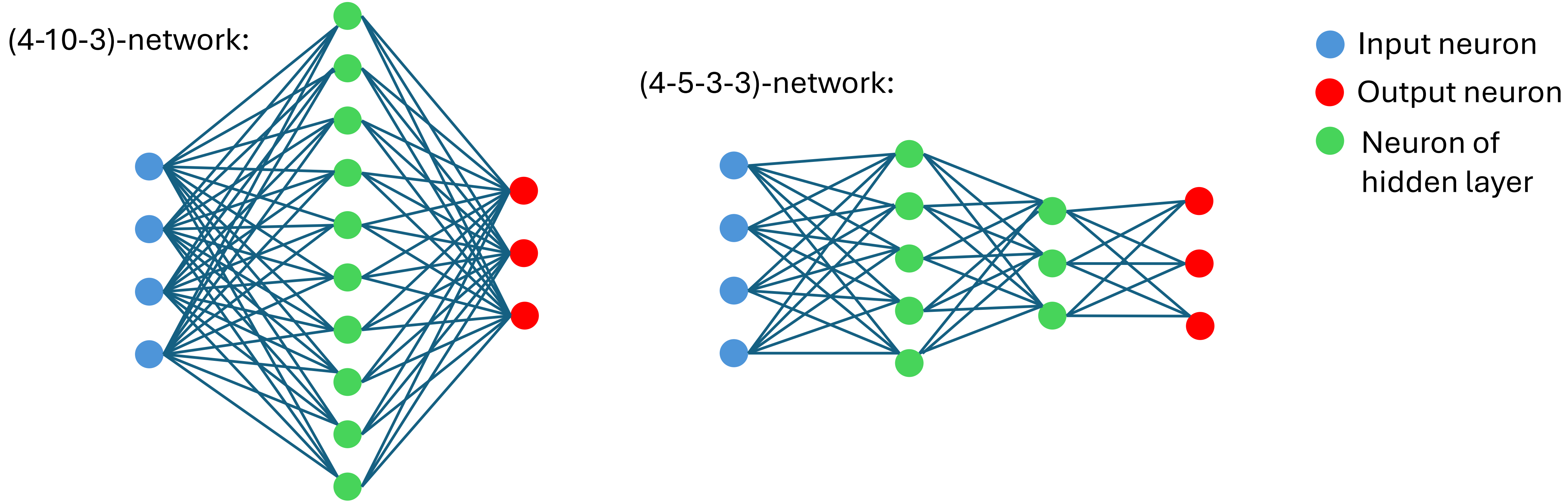
In this tutorial 80% of the 150 samples are used for training, 20% for inference.




L1: Which of the three species is easiest to classify when looking at the scatter matrix?

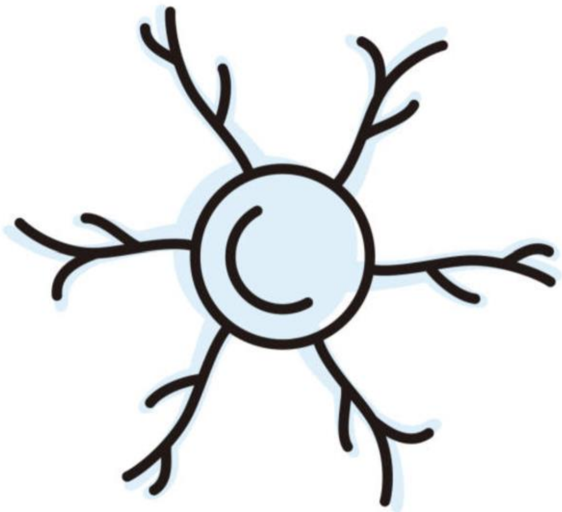
# 3.1 Network topologies for Iris dataset classification

The choice of the data set (input) and the task (output) defines the input and output layer size. For the Iris flower classification there is an input layer for 4 features (petal/sepal width/length) and an output layer for 3 labels (classification of 3 Iris species). The number of hidden layers and the number of neurons in each hidden layer can be chosen arbitrarily. The networks below show a 1-hidden (4-10-3) and a 2-hidden layer (4-5-3-3) version.

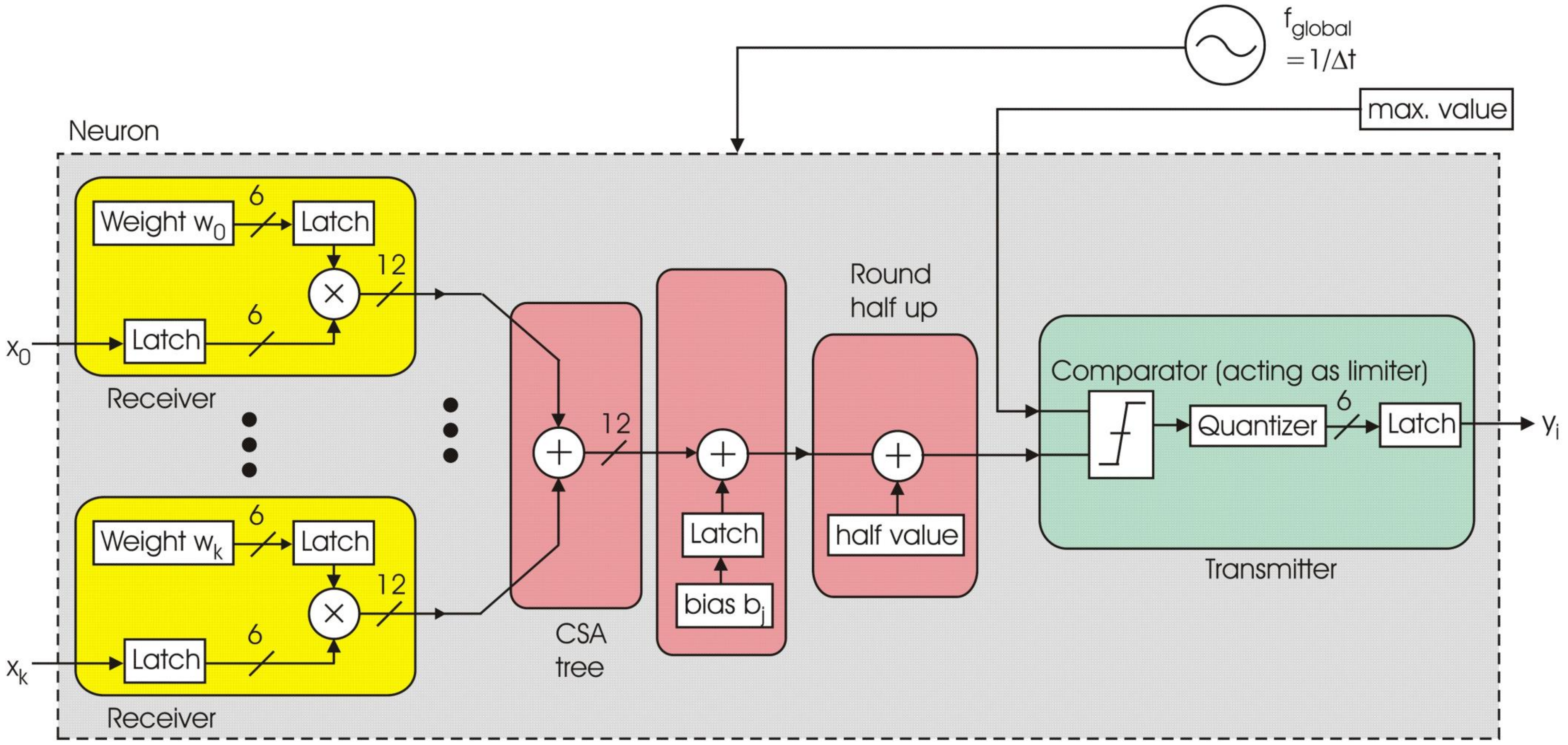



L2: Evaluate the two networks qualitatively in terms of hardware implementation complexity.

# Neuron Models

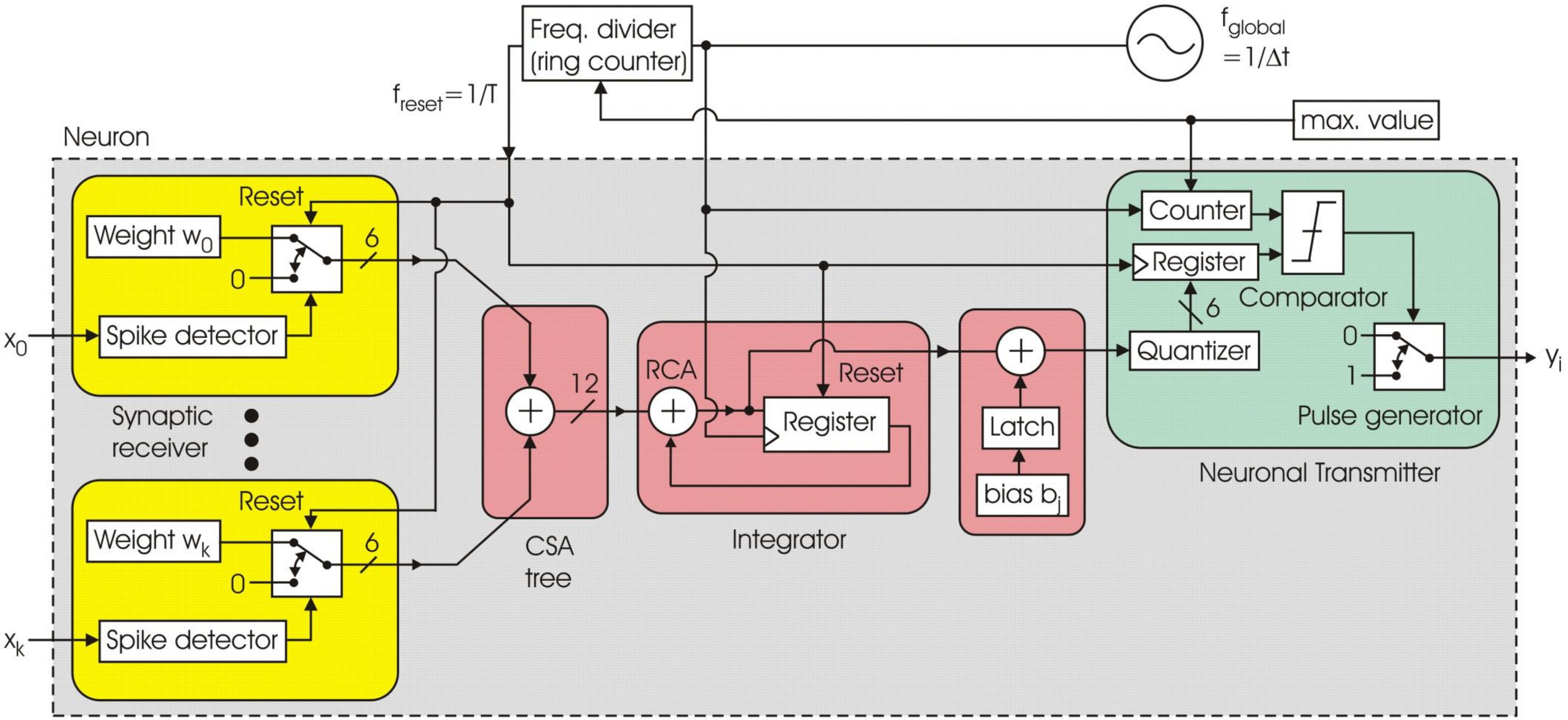


# 3.2 ANN Neuron Model for Hardware Implementation




L1: Describe the operation of the block diagram that shows the hardware implementation of an ANN neuron.

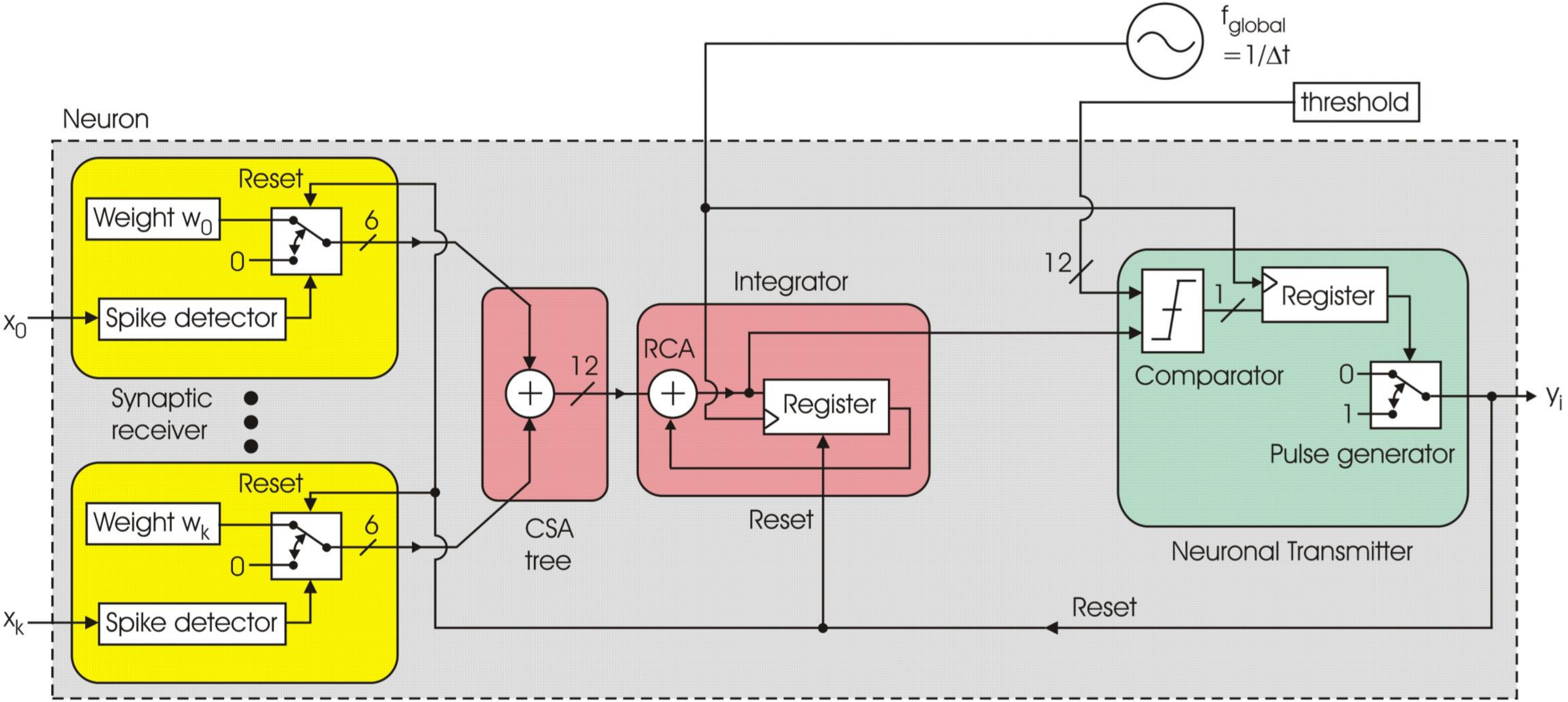
# 3.3 TTS Neuron Model (synchronous spiking)




CSA: carry save adder RCA: ripple carry adder

L1: Describe the differences of this SNN TTS neuron implementation compared to the previous ANN neuron.

# 3.4 TTFS Neuron Model (asynchronous spiking)




L1: What are the differences between the TTS and TTFS neuron model?

# Floating-Point Training of Iris Flower Network in TensorFlow

# 4.1 Floating-Point Training and Inference of ANN

1) Install Anaconda to perform Framework Simulations:
   https://docs.anaconda.com/free/anaconda/install/windows/

2) Open Spyder (Python 3.10) and run the 'Floating-point – ANN model' code. If the link does not work copy the code from here into the Editor in Spyder.

L1: Explain from a high-level perspective what the code performs.

L2: Which of the three classes yields 100% accuracy? Is this to be expected?

L2: Draw the topology of the neural network that is represented by the code in terms of input, hidden and output layers.

L3: What needs to be changed in the code to simulate a network with 2 fully-connected hidden layers consisting of 5 and 3 neurons, respectively?
Which of the two networks does obtain higher accuracy: (4,10,3) or (4,5,3,3)?

# 4.2 Listing of Floating-Point ANN Training (1)

ANN_models.py

```
import keras #Keras acts as an interface for the TensorFlow library
import time #import the time module in Python
import tensorflow as tf
from tensorflow.keras.layers import LeakyReLU, Dense, Dropout, BatchNormalization, LayerNormalization
from tensorflow.keras.utils import to_categorical
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.utils import shuffle

# Download Iris dataset:
zip_file = tf.keras.utils.get_file('iris.csv','https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data')
data = pd.read_csv(zip_file,names=['Sepal length (cm)','Sepal width (cm)','Petal length (cm)','Petal width (cm)','Class'])
data = data.sample(frac=1, random_state=0).reset_index(drop=True) #reshuffling of rows and reset of row index

data_labels = np.array(data.pop('Class')) # extraction of the column labeled 'Class'
data_values = np.array(data) #conversion of DataFrame to NumPy array
```

# Listing of Floating-point ANN Training (2)

```
# TensorFlow requires labels in form of integers rather than text, so we create a mapping:
mapping = np.sort(np.unique(data_labels)) #find unique labels
data_y = np.searchsorted(mapping, data_labels) #replace labels by numbers


data_labels, data_values, data_y = shuffle(data_labels, data_values, data_y, random_state=2) #ensures reproducibilty of shuffling

# We assume the following hyperparameters:
# epochs = 300
# validation_split=0.2 in fit() functions that excludes the last 20% of examples before
# shuffling (deterministic between calls) and uses them for reporting the validation accuracy
epochs = 400
v_s = 0.2

#%% Floating point values - ANN model
ann_model = tf.keras.Sequential() #creates a feed-forward neural network
ann_model.add(tf.keras.layers.InputLayer(input_shape=[4])) #this defines the shape of the input data, i.e. petal/sepal width/length
ann_model.add(Dense(10, activation='relu')) #, use_bias=False, fully connected hidden layer of 10 neurons
ann_model.add(Dense(3, activation='softmax')) #output layer consisting of 3 neurons
```

# Listing of Floating-Point ANN Training (3)

```
# The standard way of training would be as follows:
#ann_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#ann_model.fit(data_values, data_y, validation_split=v_s, epochs=epochs, batch_size=10)
#
# However, the epochs are very short, so there would be way too much output.
# Below is a 'hack' that reports only each 20th epoch during the training:

time_start = time.time()
ann_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history_ann = ann_model.fit(data_values, data_y, validation_split=v_s, epochs=epochs, batch_size=10, verbose=0)
print('Finished. Total time: {0:.1f} [s]'.format(time.time() - time_start))

# => output weights and biases (manually create the directory Weights_Biases if running for the first time)
ann_weights = ann_model.get_weights()
np.savetxt("Weights_Biases/ANN_weights_hiddenLayer.csv", ann_weights[0], delimiter=",")
np.savetxt("Weights_Biases/ANN_biases_hiddenLayer.csv", ann_weights[1], delimiter=",")
np.savetxt("Weights_Biases/ANN_weights_outputLayer.csv", ann_weights[2], delimiter=",")
np.savetxt("Weights_Biases/ANN_biases_outputLayer.csv", ann_weights[3], delimiter=",")
# weights_hiddenLayer = np.loadtxt("weights_hiddenLayer.csv", delimiter=",") #to reload the weights from the saved file
```

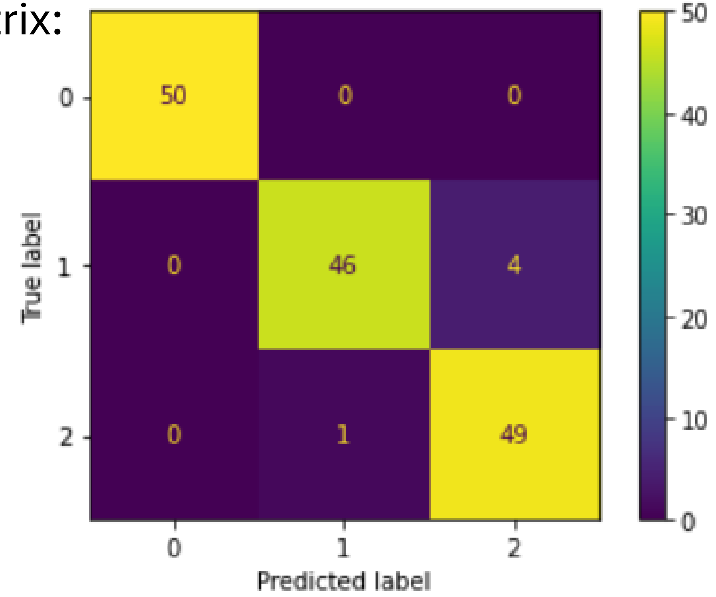
# Listing of Floating-Point ANN Training (4)

```
# summarize history for accuracy
plt.plot(history_ann.history['accuracy'])
plt.plot(history_ann.history['val_accuracy'])
plt.legend(['Training', 'Validation'],loc=4)
plt.xlabel("Epoch")
plt.ylabel("Accuracy [%]")
plt.ylim([0.0,1.0])
plt.xlim([0,400])
plt.grid()
plt.show()
# summarize history for loss
plt.plot(history_ann.history['loss'])
plt.plot(history_ann.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.xlim([0,400])
plt.gca().set_ylim(bottom=0)
plt.grid()
plt.show()
```

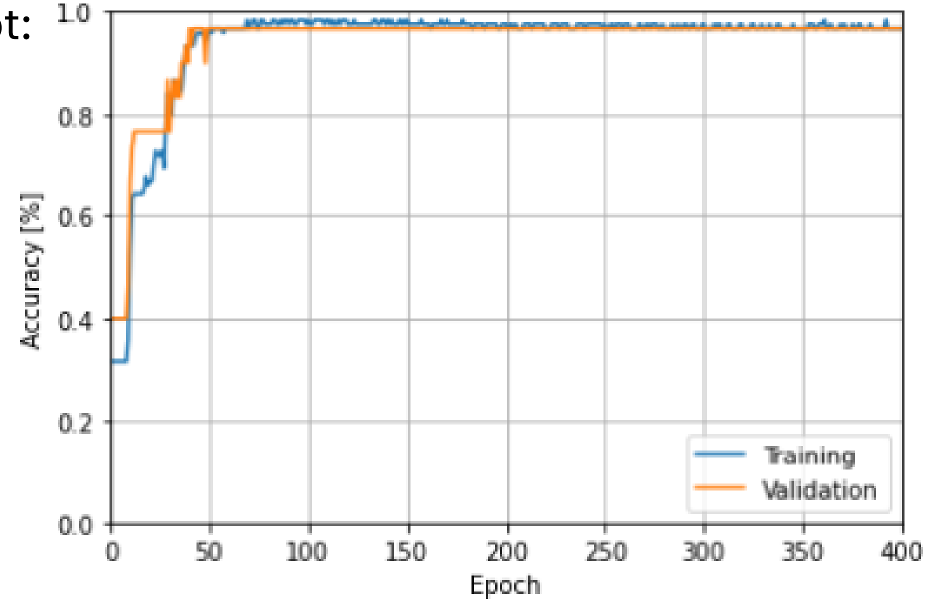
# Listing of Floating-point ANN Training (5)

```
data_predicted = np.argmax(ann_model.predict(data_values), axis=-1) #vector of predicted labels
ann_acc = np.sum((np.equal(data_predicted,data_y)*1))/data_y.shape[0] #convert boolean to integers, sum them up, divide by 150
print("Accuracy: ", ann_acc)

cm = confusion_matrix(data_y, data_predicted, labels=[0, 1, 2])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1, 2])
disp.plot()
plt.show()
```

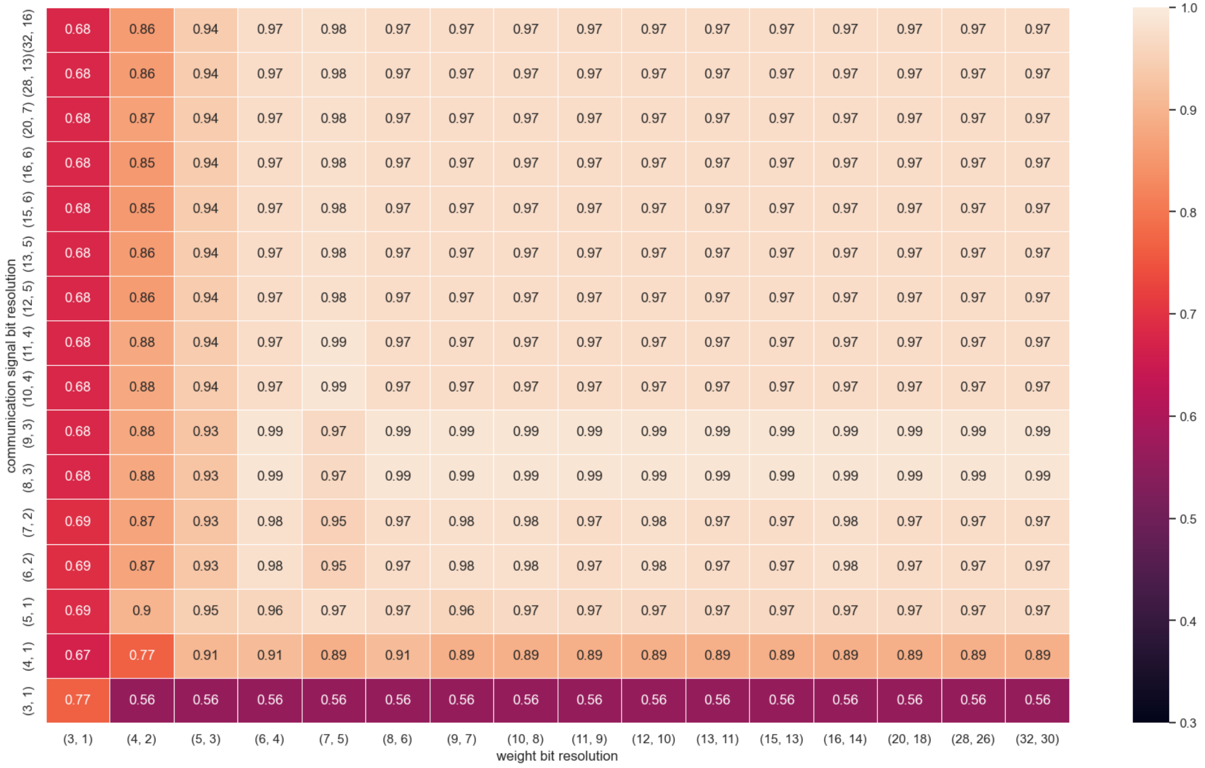
Convolution matrix:




Accuracy plot:



# Quantization-Aware Training of Iris Flower Network in TensorFlow



# 4.3 Quantization-Aware Training of ANN

The previous TensorFlow code performed the training and inference using floating point values. However, for a hardware implementation quantized values are required. The hardware complexity increases with increasing bit width. The aim is therefore to derive a hardware model with the coarsest possible resolution, but still with sufficient accuracy.
The TensorFlow code outputs a heatmap of the accuracy versus quantization of the trained weights and biases. The quantization calculations are performed for the ANN implementation of the Iris flower classification. The vhdl code presented in the next chapter implements the Iris flower network as a TTS-encoded SNN. The basic idea is to use the quantized training values from the ANN and apply them to the TTS SNN since TensorFlow does not provide support for a time-encoded training of the weights and biases. The ANN-trained values need to be mapped from the code domain to the time domain.

L3: Draw a flowchart of the TensorFlow code listed below that performs a quantization-aware training of the ANN implementing the (4-10-3)-network of the Iris flower classification task.

# 4.4 Listing of Quantization-Aware ANN Training (a1)

ANN_precision_calc.py

```
import pandas as pd #Pandas is a library for data manipulation, providing structures like Series (1-dimensional) and
                    #DataFrame (2-dimensional).
import numpy as np #NumPy is a fundamental package for scientific computing in Python
import matplotlib #Mathplotlib is a library used for creating static, animated, and interactive visualizations
from matplotlib import cm #cm is Matplotlib's color mapping module, which contains colormaps
import matplotlib.pyplot as plt #pyplot is a collection of functions that make Matplotlib work like MATLAB
import seaborn as sns  #Seaborn is a powerful data visualization library based on Matplotlib
from fxpmath import Fxp #fxpmath is a library for fixed-point arithmetic
import tensorflow as tf #TensorFlow is an open-source library developed by Google for machin learning
from tensorflow.keras.layers import LeakyReLU, Dense, Dropout #Keras is a high-level API for building and training
                    #deep learning models:
                    #The 'layers' module contains various types of layers used to construct neural networks
                    #Leaky ReLU is a variant of the Rectified Linear Unit (ReLU) activation function, which allows a small,
                    #non-zero gradient when the unit is not active. This helps prevent issues with neurons "dying" during training.
                    #Dense is a fully connected layer, which means each neuron is connected to every neuron in the previous layer.
                    #Dropout is a regularization layer that helps prevent overfitting by randomly setting a fraction of input units
                    #to 0 at each update during training time. This helps the model to generalize better to unseen data.
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from ANNModel import ANNModel, ANNCell, ANNInput, quantize_weights
```

# Listing of Quantization-Aware ANN Training (a2)

```
# Load the iris data and load the weights from the ANN file
matplotlib.rc_file_defaults() #reset runtime configuration (rc) parameters; changes to font size, etc. will be undone
# Download Iris dataset:
zip_file = tf.keras.utils.get_file('iris.csv','https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data')
data = pd.read_csv(zip_file,names=['Sepal length (cm)','Sepal width (cm)','Petal length (cm)','Petal width (cm)','Class'])
data = data.sample(frac=1, random_state=0).reset_index(drop=True) #frac=1: 100% row shuffling, random_state=0: seed 0 will
# always produce the same random shuffling of the DatarFrame, drop=True: old index is not added as a column in the DataFrame

data_labels = np.array(data.pop('Class’)) #removes the ‘Class’ column from the DataFrame
data_values = np.array(data) #convert the Pandas DataFrame into a NumPy array

# TensorFlow requires labels in form of integers rather than text, so we create a mapping:
mapping = np.sort(np.unique(data_labels)) #find unique labels, sort them and store them in the new array called ‘mapping’
data_y = np.searchsorted(mapping, data_labels) #convert labels into numerical values based on their position in the array

#use old weights for the model
weights_HL = np.loadtxt('Weights_Biases/ANN_weights_hiddenLayer.csv', delimiter=',')
weights_OL = np.loadtxt('Weights_Biases/ANN_weights_outputLayer.csv', delimiter=',’)
biases_HL = np.loadtxt('Weights_Biases/ANN_biases_hiddenLayer.csv', delimiter=',')
biases_OL = np.loadtxt('Weights_Biases/ANN_biases_outputLayer.csv', delimiter=',’)
```

# Listing of Quantization-Aware ANN Training (a3)

```
#The first entry in the bracket describes the number of bits used
and the second entry describes how many of them are fraction bits.
data_resBits = [(3,1), (4,1), (5,1), (6,2), (7,2), (8,3), (9,3), (10,4), (11,4), (12,5),
                (13,5), (15,6), (16,6), (20,7), (28,13), (32,16)]
weights_resBits = [(3,1), (4,2), (5,3), (6,4), (7,5), (8,6), (9,7), (10,8), (11,9), (12,10),
                   (13,11), (15,13), (16,14), (20,18), (28,26), (32,30)]

accuracy = np.zeros((len(data_resBits),len(weights_resBits))) #initializes a 2D NumPy array filled with zeros
```

# Listing of Quantization-Aware ANN Training (a4)

```
#%% precision table !! Attention this code part runs longer than an hour !!

idx_d = 0
len_loop = len(data_resBits)*len(weights_resBits) #16x16=256
idx_l = 1
print("Start precision development loop (length ", len_loop, ")")
for d_bits in data_resBits:
   idx_w = 0
   for w_bits in weights_resBits:
      weights_HL_q = quantize_weights(weights_HL, w_bits) #sub-routine call
      weights_OL_q = quantize_weights(weights_OL, w_bits) #sub-routine call
      biases_HL_q = quantize_weights(biases_HL, w_bits) #sub-routine call
      biases_OL_q = quantize_weights(biases_OL, w_bits) #sub-routine call

      model = ANNModel(signal_bits=d_bits, weight_bits=w_bits)
      model.addLayer(ANNInput(units=4))
      model.addLayer(ANNCell(units=10, name='HL'))
      model.addLayer(ANNCell(units=3, name='OL'))
      model.build()
```

# Listing of Quantization-Aware ANN Training (a5)

```
layer_list = model.getModel() #getModel() returns a list of layers
for layer in layer_list: #iterate through each layer in ‘layer_list’
   if layer.getName() == 'IL':
      layer.raw_out = False #attribute ‘raw_out’ is set to ‘False’ such that the output undergoes some form of post-processing
   elif layer.getName() == 'HL':
      layer.setWeights(weights_HL_q, biases_HL_q) #set weights and biases in hidden layer
      layer.raw_out = True
      layer.reLu_out = True
   elif layer.getName() == 'OL':
      layer.setWeights(weights_OL_q, biases_OL_q) #set weights and biases in output layer
      layer.softmax_out = False
      layer.raw_out = False

input_data = data_values[:,:] #extracts all rows and columns from the ‘data_values’-array
y_pred = model.predict(input_data) #uses the trained model to predict the output labels for the input data
train_acc = np.sum((np.equal(y_pred,data_y)*1))/y_pred.shape[0]
#np.equal(y_pred, data_y) compares the predicted labels (y_pred) with the true labels (data_y). This returns a boolean array where each element is True if the corresponding prediction is
#correct and False otherwise.
# (np.equal(y_pred, data_y) * 1) converts the boolean array to an integer array, where True becomes 1 and False becomes 0.
# np.sum(...) sums up the elements of the integer array, effectively counting the number of correct predictions.
# The sum of correct predictions is divided by the total number of predictions to calculate the accuracy of the model on the training data (train_acc).
```

# Listing of Quantization-Aware ANN Training (a6)

```
        accuracy[idx_d, idx_w] = train_acc #The value of train_acc is being assigned to the position in the accuracy matrix at the row idx_d and column idx_w.

        print("Standing: ", idx_l, "/", len_loop)
        idx_l += 1
        idx_w += 1
    idx_d += 1

np.savetxt('ANN_resolutionTable.csv', accuracy, delimiter=',')

#%% resolution of quantization
matplotlib.rc_file_defaults()
accuracy = np.loadtxt('ANN_resolutionTable.csv', delimiter=',')

fig = plt.figure()
fig.set_size_inches(20, 11.25)
ax = fig.add_subplot(111, projection='3d')

xx, yy = np.meshgrid([x[0] for x in data_resBits], [x[0] for x in weights_resBits]) #[x[0] for x in data_resBits]: This creates a list containing the first
                                                                                    #element of each sublist (or tuple) in data_resBits.
Z = np.transpose(accuracy)
ax.plot_surface(xx, yy, Z, cmap=cm.coolwarm)
```

# Listing of Quantization-Aware ANN Training (a7)

```
ax.set_xlabel('communication signal bit resolution')
ax.set_ylabel('weight bit resolution')
ax.set_zlabel('accuracy')
plt.gca().invert_yaxis()
plt.show()

#%%
sns.set_theme()
df3 = pd.DataFrame(accuracy, columns=weights_resBits, index=data_resBits)

fig = plt.figure()
fig.set_size_inches(20, 11.25)

ax = sns.heatmap(df3, annot=True, vmin=0.3, vmax=1, linewidths=.5)
ax.invert_yaxis()
ax.set_xlabel('weight bit resolution')
ax.set_ylabel('communication signal bit resolution')

plt.show()
```

# Listing of ANN Model (b1)

ANNModel.py

```
import numpy as np
from fxpmath import Fxp
import matplotlib.pyplot as plt

''' Class '''
class ANNModel:
    def __init__(self, signal_bits=(10, 5), weight_bits=(10, 5), visualize=False, softmax_out=False, raw_out=False):
        self.signal_bits = signal_bits
        self.weight_bits = weight_bits
        self.layers = []
        self.built = False
        self.visualize = visualize
        self.softmax_out = softmax_out
        self.raw_out = raw_out
```

# Listing of ANN Model (b2)

```
 def build(self):
  prev_units = 1
  for layer in self.layers:
    layer.addArguments(prev_units, self.signal_bits, self.weight_bits, self.visualize)
    prev_units = layer.getUnits()
  self.built = True


def addLayer(self, layer):
  self.layers.append(layer)

def __looping(self, data):
  y_pred = []
  if len(data.shape) == 1:
    data = data.reshape((1,data.shape[0]))
```

# Listing of ANN Model (b3)

```
for idx_data in range(data.shape[0]):
    X = data[idx_data,:]
    softmax = False
    raw = False
    for layer in self.layers:
        y = layer.call(X)
        X = y
        softmax = layer.softmax_out
        raw = layer.raw_out
    try:
        if softmax:
            y_class = self.softmax(y)
        elif raw:
            y_class = y
        else:
            y_class = np.argmax(y)
    except ValueError:
        y_class = 3
    except IndexError:
        print("IndexError")
        y_class = 4
    y_pred.append(y_class)
return np.array(y_pred)
```

# Listing of ANN Model (b4)

```
    def predict(self, X_data):
        y_pred = self.__looping(X_data)
        return y_pred

    def calculate_Accuracy(y_pred, y):
        (y_pred == y)*1

    def getModel(self):
        if self.built:
            return self.layers
        else:
            return "Please build first"

    def softmax(self, vector):
         e = np.exp(vector)
         return e / np.sum(e)


''' Class '''
class ANNBasicCell(object):
    nr = 0
```

# Listing of ANN Model (b5)

```
def __init__(self, units, name):
  self.units = units
  self.signal_bits = (10, 5)
  self.weight_bits = (10, 5)
  self.weights = np.zeros((1,1))
  self.biases = np.zeros((1,1))
  self.prev_units = 1
  self.visualize = False
  self.softmax_out = False
  self.raw_out = False
  self.reLu_out = False
  ANNBasicCell.nr += 1
  self.name = name

def addArguments(self, prev_units, signal_bits, weight_bits, visualize):
  self.prev_units = prev_units
  self.signal_bits = signal_bits
  self.weight_bits = weight_bits
  self.visualize = visualize
```

# Listing of ANN Model (b6)

```
def _initWeights(self):
    self.weights =  Fxp(np.random.normal(loc=0.0, scale=1.0, size=(self.prev_units, self.units)),
                    signed=True, n_word=self.weight_bits[0], n_frac=self.weight_bits[1], rounding='around')
    self.biases = Fxp(np.random.normal(loc=0.0, scale=1.0, size=(self.units,)),
                    signed=True, n_word=self.weight_bits[0], n_frac=self.weight_bits[1], rounding='around')

def call(self, data):
    ...

def __doTimestep(self, data):
    ...

def getName(self):
    return self.name
```

# Listing of ANN Model (b7)

```
    def setWeights(self, weights, biases):
        print("Weights set for", self.name)
        if weights.shape == (self.prev_units, self.units):
            self.weights = weights
        else:
            print('Attention random weights')
        if biases.shape == (self.units,):
            self.biases = biases
        else:
            print('Attention random biases')

    def getUnits(self):
        return self.units


''' Class '''
class ANNInput(ANNBasicCell):
    def __init__(self, units, name='IL'):
        super().__init__(units, name)
```

# Listing of ANN Model (b8)

```
    def call(self, data):
        y = self.__doTimestep(data)
        return y

    def __doTimestep(self, data):
        signal = Fxp(data, signed=True, n_word=self.signal_bits[0],
n_frac=self.signal_bits[1], rounding='around')
        return signal


''' Class '''
class ANNCell(ANNBasicCell):
    def __init__(self, units, name='ANNLayer'):
        super().__init__(units, name)

    def call(self, data):
        y = self.__doTimestep(data, self.weights, self.biases)
        return y
```

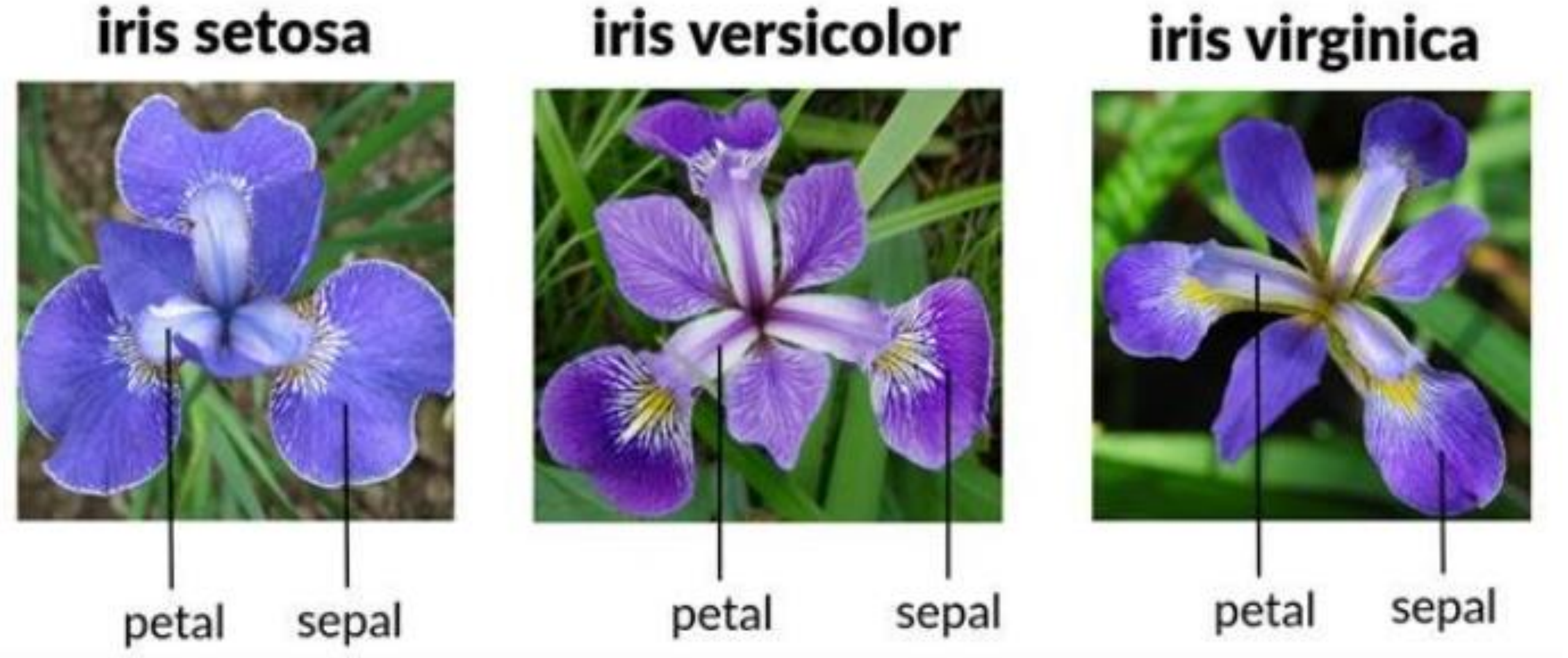
# Listing of ANN Model (b9)

```
    def __doTimestep(self, signal, weights, biases):
        output = Fxp(np.zeros(self.units), True, self.signal_bits[0] + self.weight_bits[0], self.signal_bits[1] + self.weight_bits[1],
rounding='around')
        for idx in range(self.units):
            product = Fxp(weights[:,idx] * signal, True, self.signal_bits[0] + self.weight_bits[0], self.signal_bits[1] + self.weight_bits[1],
rounding='around')
            bias = Fxp(biases[idx], True, self.signal_bits[0] + self.weight_bits[0], self.signal_bits[1] + self.weight_bits[1],
rounding='around')
            sum_potential = Fxp(sum(product)+bias, True, self.signal_bits[0] + self.weight_bits[0], self.signal_bits[1] +
self.weight_bits[1], rounding='around')
            out = output.get_val().tolist()
            out[idx] = sum_potential
            output = Fxp(out, True, self.signal_bits[0] + self.weight_bits[0], self.signal_bits[1] + self.weight_bits[1], rounding='around')

        if self.reLu_out:
            output = Fxp(np.where(output < 0.0, 0.0, output), True, self.signal_bits[0] + self.weight_bits[0], self.signal_bits[1] +
self.weight_bits[1], rounding='around')
```

# Listing of ANN Model (b10)

```
        rounding = Fxp(0, True, self.signal_bits[0] + self.weight_bits[0], self.signal_bits[1] + 1, rounding='trunc')
        output = Fxp(output + rounding.precision, signed=True, n_word=self.signal_bits[0], n_frac=self.signal_bits[1], rounding='trunc')
        return output

''' Functions for quantization '''
def quantize_weights(weights, bits):
    TEMPLATE = Fxp(None, True, bits[0], bits[1]) #FxP: This is the class used to create a fixed-point object.
                                                 #None: The object is being created without an initial value.
                                                 #True: signed, False: unsigned.
                                                 #bit[0]: Total number of bits.
                                                 #bit[1]: Number of fractional bits.
    TEMPLATE.rounding = 'around' #Rounds to the nearest even number
    weights_q = Fxp(weights, signed=True, n_word=bits[0], n_frac=bits[1], rounding='around').like(TEMPLATE)
    #The .like() method modifies the newly created fixed-point object to have the same configuration as TEMPLATE.
    return weights_q
```

# Iris Flower Dataset in VHDL



# 5.1 VHDL-Implementation of TTS SNN

The vhdl code listed below implements the time-to-first spike (TTS) spiking neural network (SNN) of the Iris flower dataset. It uses the neuron model depicted on page 33, which is applied to the (3-10-4)-network illustrated on page 30. The spiking scheme applied is shown on page 26 with an observation interval length of 2^6=64.
The quantization of the weights and biases trained with the TensorFlow modeling described on the pages 36 and 43 are (6,4) for the weights and biases and (6,2) for the features (petal/sepal length/width) or communication signals between the neurons, where the format is (#bits, #fractional bits).
The RLM of the ANN SNN is synthesized in a 5 nm finFET CMOS technology for a clock speed of 2.667 GHz, derived from DDR4 clocking specifications. The layout of the synthesized design measures 40 um x 40 um and is shown on page 149.

L1 Examples of quantized features, weights and biases are shown on the pages 64 and 77 together with the pertinent floating-point numbers. Select a few examples and familiarize yourself with the quantized data format.
L2 Run the TensorFlow code of the quantization-aware training depicted in the previous chapter on page 42 and identify the accuracy in the heatmap diagram that is selected for this design.
L3 Illustrate the structure of the vhdl code with block diagrams.

# 5.1.1 Compilation and Simulation of VHDL Code

[redacted] : technology related data are blanked

**VHDL, 5nm finFET technology, 5HPP**

cd [redacted]/ncsim

**./myncshell.proxy**

cd [redacted]/ncsim/mko_stuff

Run the contents of **snn_tts_compile_dz** in ncsim directory

```
#Compilation of packages:
xmvhdl -v200x ../[redacted]/snn_tts_support_pkg.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_dataset_pkg.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_weights_biases_pkg.vhdl

#Compilation of design data:
xmvhdl -v200x ../[redacted]/snn_tts_inputlayer_neuron.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_layer1_neuron.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_receiver.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_outputlayer_neuron.vhdl
xmvhdl -v200x ../[redacted]/snn_tts.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_wrapper.vhdl

#Compilation of testbench:
xmvhdl -v200x ../[redacted]/snn_tts_tb.vhdl

# elaboration
xmelab -access +rc snn_tts_tb

# launch SimVision
xmsim snn_tts_tb -gui
```

Files without IBM technology information

```
#Compilation of packages:
xmvhdl -v200x ../[redacted]/snn_tts_support_pkg.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_dataset_pkg.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_weights_biases_pkg.vhdl

#Compilation of design data:
xmvhdl -v200x ../[redacted]/behavioral_dff.vhdl
xmvhdl -v200x ../[redacted]/behavioral_dff_1bit.vhdl
xmvhdl -v200x ../[redacted]/behavioral_lcb.vhdl

xmvhdl -v200x ../[redacted]/snn_tts_inputlayer_neuron.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_layer1_neuron.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_receiver.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_outputlayer_neuron.vhdl
xmvhdl -v200x ../[redacted]/snn_tts.vhdl
xmvhdl -v200x ../[redacted]/snn_tts_wrapper.vhdl

#Compilation of testbench:
xmvhdl -v200x ../[redacted]/snn_tts_tb.vhdl

# elaboration
xmelab -access +rc snn_tts_tb

# launch SimVision
xmsim snn_tts_tb -gui
```

# 5.1.2 Quantized Iris Flower Dataset (1)

**snn_tts_dataset_pkg.vhdl**

Petal width [cm]
Petal length [cm]
Sepal width [cm]
Sepal length [cm]

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package snn_tts_dataset_pkg is

  -- feature data from the Iris dataset --
  type feature_vector is array (0 to 3) of std_ulogic_vector(0 to 5);
  type feature_array is array (0 to 149) of feature_vector;
  constant feature_data : feature_array := (
     ( "010111", "001011", "010100", "001010"),  -- (0):  [5.75 2.75 5.   2.5 ]  label: 2   real values: [5.8 2.8 5.1 2.4]
     ( "011000", "001001", "010000", "000100"),  -- (1):  [6.   2.25 4.   1.  ]  label: 1   real values: [6.  2.2 4.  1. ]
     ( "010110", "010001", "000110", "000001"),  -- (2):  [5.5  4.25 1.5  0.25]  label: 0   real values: [5.5 4.2 1.4 0.2]
     ( "011101", "001100", "011001", "000111"),  -- (3):  [7.25 3.   6.25 1.75]  label: 2   real values: [7.3 2.9 6.3 1.8]
     ( "010100", "001110", "000110", "000001"),  -- (4):  [5.   3.5  1.5  0.25]  label: 0   real values: [5.  3.4 1.5 0.2]
     ( "011001", "001101", "011000", "001010"),  -- (5):  [6.25 3.25 6.   2.5 ]  label: 2   real values: [6.3 3.3 6.  2.5]
     ( "010100", "001110", "000101", "000001"),  -- (6):  [5.   3.5  1.25 0.25]  label: 0   real values: [5.  3.5 1.3 0.3]
     ( "011011", "001100", "010011", "000110"),  -- (7):  [6.75 3.   4.75 1.5 ]  label: 1   real values: [6.7 3.1 4.7 1.5]
```

# 5.1.2 Quantized Iris Flower Dataset (2)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "011011", "001011", "010011", "000110"),  -- (8):   [6.75 2.75 4.75 1.5 ]  label: 1   real values: [6.8 2.8 4.8 1.4]
( "011000", "001011", "010000", "000101"),  -- (9):   [6.   2.75 4.   1.25]  label: 1   real values: [6.1 2.8 4.  1.3]
( "011000", "001010", "010110", "000110"),  -- (10):  [6.   2.5  5.5  1.5 ]  label: 2   real values: [6.1 2.6 5.6 1.4]
( "011010", "001101", "010010", "000110"),  -- (11):  [6.5  3.25 4.5  1.5 ]  label: 1   real values: [6.4 3.2 4.5 1.5]
( "011000", "001011", "010011", "000101"),  -- (12):  [6.   2.75 4.75 1.25]  label: 1   real values: [6.1 2.8 4.7 1.2]
( "011010", "001011", "010010", "000110"),  -- (13):  [6.5  2.75 4.5  1.5 ]  label: 1   real values: [6.5 2.8 4.6 1.5]
( "011000", "001100", "010011", "000110"),  -- (14):  [6.   3.   4.75 1.5 ]  label: 1   real values: [6.1 2.9 4.7 1.4]
( "010100", "001100", "000110", "000000"),  -- (15):  [5.   3.   1.5  0.  ]  label: 0   real values: [4.9 3.1 1.5 0.1]
( "011000", "001100", "010010", "000110"),  -- (16):  [6.   3.   4.5  1.5 ]  label: 1   real values: [6.  2.9 4.5 1.5]
( "010110", "001010", "010010", "000101"),  -- (17):  [5.5  2.5  4.5  1.25]  label: 1   real values: [5.5 2.6 4.4 1.2]
( "010011", "001100", "000110", "000001"),  -- (18):  [4.75 3.   1.5  0.25]  label: 0   real values: [4.8 3.  1.4 0.3]
( "010110", "010000", "000101", "000010"),  -- (19):  [5.5  4.   1.25 0.5 ]  label: 0   real values: [5.4 3.9 1.3 0.4]
( "010110", "001011", "010100", "001000"),  -- (20):  [5.5  2.75 5.   2.  ]  label: 2   real values: [5.6 2.8 4.9 2. ]
( "010110", "001100", "010010", "000110"),  -- (21):  [5.5  3.   4.5  1.5 ]  label: 1   real values: [5.6 3.  4.5 1.5]
( "010011", "001110", "001000", "000001"),  -- (22):  [4.75 3.5  2.   0.25]  label: 0   real values: [4.8 3.4 1.9 0.2]
( "010010", "001100", "000110", "000001"),  -- (23):  [4.5  3.   1.5  0.25]  label: 0   real values: [4.4 2.9 1.4 0.2]
( "011001", "001011", "010011", "000111"),  -- (24):  [6.25 2.75 4.75 1.75]  label: 2   real values: [6.2 2.8 4.8 1.8]
( "010010", "001110", "000100", "000001"),  -- (25):  [4.5  3.5  1.   0.25]  label: 0   real values: [4.6 3.6 1.  0.2]
```

# 5.1.2 Quantized Iris Flower Dataset (3)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "010100", "001111", "001000", "000010"),  -- (26):  [5.   3.75 2.   0.5 ]  label: 0   real values: [5.1 3.8 1.9 0.4]
( "011001", "001100", "010001", "000101"),  -- (27):  [6.25 3.   4.25 1.25]  label: 1   real values: [6.2 2.9 4.3 1.3]
( "010100", "001001", "001101", "000100"),  -- (28):  [5.   2.25 3.25 1.  ]  label: 1   real values: [5.  2.3 3.3 1. ]
( "010100", "001110", "000110", "000010"),  -- (29):  [5.   3.5  1.5  0.5 ]  label: 0   real values: [5.  3.4 1.6 0.4]
( "011010", "001100", "010110", "000111"),  -- (30):  [6.5  3.   5.5  1.75]  label: 2   real values: [6.4 3.1 5.5 1.8]
( "010110", "001100", "010010", "000110"),  -- (31):  [5.5  3.   4.5  1.5 ]  label: 1   real values: [5.4 3.  4.5 1.5]
( "010101", "001110", "000110", "000001"),  -- (32):  [5.25 3.5  1.5  0.25]  label: 0   real values: [5.2 3.5 1.5 0.2]
( "011000", "001100", "010100", "000111"),  -- (33):  [6.   3.   5.   1.75]  label: 2   real values: [6.1 3.  4.9 1.8]
( "011010", "001011", "010110", "001001"),  -- (34):  [6.5  2.75 5.5  2.25]  label: 2   real values: [6.4 2.8 5.6 2.2]
( "010101", "001011", "010000", "000110"),  -- (35):  [5.25 2.75 4.   1.5 ]  label: 1   real values: [5.2 2.7 3.9 1.4]
( "010111", "001111", "000111", "000001"),  -- (36):  [5.75 3.75 1.75 0.25]  label: 0   real values: [5.7 3.8 1.7 0.3]
( "011000", "001011", "010100", "000110"),  -- (37):  [6.   2.75 5.   1.5 ]  label: 1   real values: [6.  2.7 5.1 1.6]
( "011000", "001100", "010001", "000110"),  -- (38):  [6.   3.   4.25 1.5 ]  label: 1   real values: [5.9 3.  4.2 1.5]
( "010111", "001010", "010000", "000101"),  -- (39):  [5.75 2.5  4.   1.25]  label: 1   real values: [5.8 2.6 4.  1.2]
( "011011", "001100", "010110", "001000"),  -- (40):  [6.75 3.   5.5  2.  ]  label: 2   real values: [6.8 3.  5.5 2.1]
( "010011", "001101", "000101", "000001"),  -- (41):  [4.75 3.25 1.25 0.25]  label: 0   real values: [4.7 3.2 1.3 0.2]
( "011100", "001100", "010100", "001001"),  -- (42):  [7.   3.   5.   2.25]  label: 2   real values: [6.9 3.1 5.1 2.3]
( "010100", "001110", "000110", "000010"),  -- (43):  [5.   3.5  1.5  0.5 ]  label: 0   real values: [5.  3.5 1.6 0.6]
```

# 5.1.2 Quantized Iris Flower Dataset (4)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "010110", "001111", "000110", "000001"),  -- (44):  [5.5  3.75 1.5  0.25]  label: 0   real values: [5.4 3.7 1.5 0.2]
( "010100", "001000", "001110", "000100"),  -- (45):  [5.   2.   3.5  1.  ]  label: 1   real values: [5.  2.  3.5 1. 0]
( "011010", "001100", "010110", "000111"),  -- (46):  [6.5  3.   5.5  1.75]  label: 2   real values: [6.5 3.  5.5 1.8]
( "011011", "001101", "010111", "001010"),  -- (47):  [6.75 3.25 5.75 2.5 ]  label: 2   real values: [6.7 3.3 5.7 2.5]
( "011000", "001001", "010100", "000110"),  -- (48):  [6.   2.25 5.   1.5 ]  label: 2   real values: [6.  2.2 5.  1.5]
( "011011", "001010", "010111", "000111"),  -- (49):  [6.75 2.5  5.75 1.75]  label: 2   real values: [6.7 2.5 5.8 1.8]
( "010110", "001010", "010000", "000100"),  -- (50):  [5.5  2.5  4.   1.  ]  label: 1   real values: [5.6 2.5 3.9 1.1]
( "011111", "001100", "011000", "001001"),  -- (51):  [7.75 3.   6.   2.25]  label: 2   real values: [7.7 3.  6.1 2.3]
( "011001", "001101", "010011", "000110"),  -- (52):  [6.25 3.25 4.75 1.5 ]  label: 1   real values: [6.3 3.3 4.7 1.6]
( "010110", "001010", "001111", "000100"),  -- (53):  [5.5  2.5  3.75 1.  ]  label: 1   real values: [5.5 2.4 3.8 1.1]
( "011001", "001011", "010100", "000111"),  -- (54):  [6.25 2.75 5.   1.75]  label: 2   real values: [6.3 2.7 4.9 1.8]
( "011001", "001011", "010100", "000110"),  -- (55):  [6.25 2.75 5.   1.5 ]  label: 2   real values: [6.3 2.8 5.1 1.5]
( "010100", "001010", "010010", "000111"),  -- (56):  [5.   2.5  4.5  1.75]  label: 2   real values: [4.9 2.5 4.5 1.7]
( "011001", "001010", "010100", "001000"),  -- (57):  [6.25 2.5  5.   2.  ]  label: 2   real values: [6.3 2.5 5.  1.9]
( "011100", "001101", "010011", "000110"),  -- (58):  [7.   3.25 4.75 1.5 ]  label: 1   real values: [7.  3.2 4.7 1.4]
( "011010", "001100", "010101", "001000"),  -- (59):  [6.5  3.   5.25 2.  ]  label: 2   real values: [6.5 3.  5.2 2.0 ]
( "011000", "001110", "010010", "000110"),  -- (60):  [6.   3.5  4.5  1.5 ]  label: 1   real values: [6.  3.4 4.5 1.6]
( "010011", "001100", "000110", "000001"),  -- (61):  [4.75 3.   1.5  0.25]  label: 0   real values: [4.8 3.1 1.6 0.2]
```

# 5.1.2 Quantized Iris Flower Dataset (5)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "010111", "001011", "010100", "001000"), -- (62): [5.75 2.75 5.   2.  ] label: 2  real values: [5.8 2.7 5.1 1.9]
( "010110", "001011", "010001", "000101"), -- (63): [5.5  2.75 4.25 1.25] label: 1  real values: [5.6 2.7 4.2 1.3]
( "010110", "001100", "001110", "000101"), -- (64): [5.5  3.   3.5  1.25] label: 1  real values: [5.6 2.9 3.6 1.3]
( "010110", "001010", "010000", "000101"), -- (65): [5.5  2.5  4.   1.25] label: 1  real values: [5.5 2.5 4.  1.3]
( "011000", "001100", "010010", "000110"), -- (66): [6.   3.   4.5  1.5 ] label: 1  real values: [6.1 3.  4.6 1.4]
( "011101", "001101", "011000", "000111"), -- (67): [7.25 3.25 6.   1.75] label: 2  real values: [7.2 3.2 6.  1.8]
( "010101", "001111", "000110", "000001"), -- (68): [5.25 3.75 1.5  0.25] label: 0  real values: [5.3 3.7 1.5 0.2]
( "010001", "001100", "000100", "000000"), -- (69): [4.25 3.   1.   0.  ] label: 0  real values: [4.3 3.  1.1 0.1]
( "011010", "001011", "010101", "001000"), -- (70): [6.5  2.75 5.25 2.  ] label: 2  real values: [6.4 2.7 5.3 1.9]
( "010111", "001100", "010001", "000101"), -- (71): [5.75 3.   4.25 1.25] label: 1  real values: [5.7 3.  4.2 1.2]
( "010110", "001110", "000111", "000001"), -- (72): [5.5  3.5  1.75 0.25] label: 0  real values: [5.4 3.4 1.7 0.2]
( "010111", "010010", "000110", "000010"), -- (73): [5.75 4.5  1.5  0.5 ] label: 0  real values: [5.7 4.4 1.5 0.4]
( "011100", "001100", "010100", "000110"), -- (74): [7.   3.   5.   1.5 ] label: 1  real values: [6.9 3.1 4.9 1.5]
( "010010", "001100", "000110", "000001"), -- (75): [4.5  3.   1.5  0.25] label: 0  real values: [4.6 3.1 1.5 0.2]
( "011000", "001100", "010100", "000111"), -- (76): [6.   3.   5.   1.75] label: 2  real values: [5.9 3.  5.1 1.8]
( "010100", "001010", "001100", "000100"), -- (77): [5.   2.5  3.   1.  ] label: 1  real values: [5.1 2.5 3.  1.1]
( "010010", "001110", "000110", "000001"), -- (78): [4.5  3.5  1.5  0.25] label: 0  real values: [4.6 3.4 1.4 0.3]
( "011001", "001001", "010010", "000110"), -- (79): [6.25 2.25 4.5  1.5 ] label: 1  real values: [6.2 2.2 4.5 1.5]
```

# 5.1.2 Quantized Iris Flower Dataset (6)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "011101", "001110", "011000", "001010"),  -- (80):  [7.25 3.5  6.   2.5 ]  label: 2   real values: [7.2 3.6 6.1 2.5]
( "010111", "001100", "010001", "000101"),  -- (81):  [5.75 3.   4.25 1.25]  label: 1   real values: [5.7 2.9 4.2 1.3]
( "010011", "001100", "000110", "000000"),  -- (82):  [4.75 3.   1.5  0.  ]  label: 0   real values: [4.8 3.  1.4 0.1]
( "011100", "001100", "011000", "001000"),  -- (83):  [7.   3.   6.   2.  ]  label: 2   real values: [7.1 3.  5.9 2.1]
( "011100", "001101", "010111", "001001"),  -- (84):  [7.   3.25 5.75 2.25]  label: 2   real values: [6.9 3.2 5.7 2.3]
( "011010", "001100", "010111", "001001"),  -- (85):  [6.5  3.   5.75 2.25]  label: 2   real values: [6.5 3.  5.8 2.2]
( "011010", "001011", "010110", "001000"),  -- (86):  [6.5  2.75 5.5  2.  ]  label: 2   real values: [6.4 2.8 5.6 2.1]
( "010100", "001111", "000110", "000001"),  -- (87):  [5.   3.75 1.5  0.25]  label: 0   real values: [5.1 3.8 1.6 0.2]
( "010011", "001110", "000110", "000001"),  -- (88):  [4.75 3.5  1.5  0.25]  label: 0   real values: [4.8 3.4 1.6 0.2]
( "011010", "001101", "010100", "001000"),  -- (89):  [6.5  3.25 5.   2.  ]  label: 2   real values: [6.5 3.2 5.1 2. ]
( "011011", "001101", "010111", "001000"),  -- (90):  [6.75 3.25 5.75 2.  ]  label: 2   real values: [6.7 3.3 5.7 2.1]
( "010010", "001001", "000101", "000001"),  -- (91):  [4.5  2.25 1.25 0.25]  label: 0   real values: [4.5 2.3 1.3 0.3]
( "011001", "001110", "010110", "001001"),  -- (92):  [6.25 3.5  5.5  2.25]  label: 2   real values: [6.2 3.4 5.4 2.3]
( "010100", "001100", "000110", "000001"),  -- (93):  [5.   3.   1.5  0.25]  label: 0   real values: [4.9 3.  1.4 0.2]
( "010111", "001010", "010100", "001000"),  -- (94):  [5.75 2.5  5.   2.  ]  label: 2   real values: [5.7 2.5 5.  2. ]
( "011100", "001100", "010110", "001000"),  -- (95):  [7.   3.   5.5  2.  ]  label: 2   real values: [6.9 3.1 5.4 2.1]
( "010010", "001101", "000101", "000001"),  -- (96):  [4.5  3.25 1.25 0.25]  label: 0   real values: [4.4 3.2 1.3 0.2]
( "010100", "001110", "000110", "000001"),  -- (97):  [5.   3.5  1.5  0.25]  label: 0   real values: [5.  3.6 1.4 0.2]
```

# 5.1.2 Quantized Iris Flower Dataset (7)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "011101", "001100", "010111", "000110"),  -- (98):  [7.25 3.   5.75 1.5 ]  label: 2   real values: [7.2 3.  5.8 1.6]
( "010100", "001110", "000110", "000001"),  -- (99):  [5.   3.5  1.5  0.25]  label: 0   real values: [5.1 3.5 1.4 0.3]
( "010010", "001100", "000101", "000001"),  -- (100): [4.5  3.   1.25 0.25]  label: 0   real values: [4.4 3.  1.3 0.2]
( "010110", "010000", "000111", "000010"),  -- (101): [5.5  4.   1.75 0.5 ]  label: 0   real values: [5.4 3.9 1.7 0.4]
( "010110", "001001", "010000", "000101"),  -- (102): [5.5  2.25 4.   1.25]  label: 1   real values: [5.5 2.3 4.  1.3]
( "011011", "001101", "011000", "001001"),  -- (103): [6.75 3.25 6.   2.25]  label: 2   real values: [6.8 3.2 5.9 2.3]
( "011110", "001100", "011010", "001000"),  -- (104): [7.5  3.   6.5  2.  ]  label: 2   real values: [7.6 3.  6.6 2.1]
( "010100", "001110", "000110", "000001"),  -- (105): [5.   3.5  1.5  0.25]  label: 0   real values: [5.1 3.5 1.4 0.2]
( "010100", "001100", "000110", "000000"),  -- (106): [5.   3.   1.5  0.  ]  label: 0   real values: [4.9 3.1 1.5 0.1]
( "010101", "001110", "000110", "000001"),  -- (107): [5.25 3.5  1.5  0.25]  label: 0   real values: [5.2 3.4 1.4 0.2]
( "010111", "001011", "010010", "000101"),  -- (108): [5.75 2.75 4.5  1.25]  label: 1   real values: [5.7 2.8 4.5 1.3]
( "011010", "001100", "010010", "000110"),  -- (109): [6.5  3.   4.5  1.5 ]  label: 1   real values: [6.6 3.  4.4 1.4]
( "010100", "001101", "000101", "000001"),  -- (110): [5.   3.25 1.25 0.25]  label: 0   real values: [5.  3.2 1.2 0.2]
( "010100", "001101", "000111", "000010"),  -- (111): [5.   3.25 1.75 0.5 ]  label: 0   real values: [5.1 3.3 1.7 0.5]
( "011010", "001100", "010001", "000101"),  -- (112): [6.5  3.   4.25 1.25]  label: 1   real values: [6.4 2.9 4.3 1.3]
( "010110", "001110", "000110", "000010"),  -- (113): [5.5  3.5  1.5  0.5 ]  label: 0   real values: [5.4 3.4 1.5 0.4]
( "011111", "001010", "011100", "001001"),  -- (114): [7.75 2.5  7.   2.25]  label: 2   real values: [7.7 2.6 6.9 2.3]
( "010100", "001010", "001101", "000100"),  -- (115): [5.   2.5  3.25 1.  ]  label: 1   real values: [4.9 2.4 3.3 1. ]
```

# 5.1.2 Quantized Iris Flower Dataset (8)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "011111", "001111", "011010", "001000"), -- (116): [7.75 3.75 6.5  2.  ] label: 2   real values: [7.9 3.8 6.4 2. ]
( "011011", "001100", "010010", "000110"), -- (117): [6.75 3.   4.5  1.5 ] label: 1   real values: [6.7 3.1 4.4 1.4]
( "010101", "010000", "000110", "000000"), -- (118): [5.25 4.   1.5  0.  ] label: 0   real values: [5.2 4.1 1.5 0.1]
( "011000", "001100", "010011", "000111"), -- (119): [6.   3.   4.75 1.75] label: 2   real values: [6.  3.  4.8 1.8]
( "010111", "010000", "000101", "000001"), -- (120): [5.75 4.   1.25 0.25] label: 0   real values: [5.8 4.  1.2 0.2]
( "011111", "001011", "011011", "001000"), -- (121): [7.75 2.75 6.75 2.  ] label: 2   real values: [7.7 2.8 6.7 2. ]
( "010100", "001111", "000110", "000001"), -- (122): [5.   3.75 1.5  0.25] label: 0   real values: [5.1 3.8 1.5 0.3]
( "010011", "001101", "000110", "000001"), -- (123): [4.75 3.25 1.5  0.25] label: 0   real values: [4.7 3.2 1.6 0.2]
( "011110", "001011", "011000", "001000"), -- (124): [7.5  2.75 6.   2.  ] label: 2   real values: [7.4 2.8 6.1 1.9]
( "010100", "001101", "000110", "000001"), -- (125): [5.   3.25 1.5  0.25] label: 0   real values: [5.  3.3 1.4 0.2]
( "011001", "001110", "010110", "001010"), -- (126): [6.25 3.5  5.5  2.5 ] label: 2   real values: [6.3 3.4 5.6 2.4]
( "010111", "001011", "010000", "000101"), -- (127): [5.75 2.75 4.   1.25] label: 1   real values: [5.7 2.8 4.1 1.3]
( "010111", "001011", "010000", "000101"), -- (128): [5.75 2.75 4.   1.25] label: 1   real values: [5.8 2.7 3.9 1.2]
( "010111", "001010", "001110", "000100"), -- (129): [5.75 2.5  3.5  1.  ] label: 1   real values: [5.7 2.6 3.5 1. ]
( "011010", "001101", "010101", "001001"), -- (130): [6.5  3.25 5.25 2.25] label: 2   real values: [6.4 3.2 5.3 2.3]
( "011011", "001100", "010101", "001001"), -- (131): [6.75 3.   5.25 2.25] label: 2   real values: [6.7 3.  5.2 2.3]
( "011001", "001010", "010100", "000110"), -- (132): [6.25 2.5  5.   1.5 ] label: 1   real values: [6.3 2.5 4.9 1.5]
( "011011", "001100", "010100", "000111"), -- (133): [6.75 3.   5.   1.75] label: 1   real values: [6.7 3.  5.  1.7]
```

# 5.1.2 Quantized Iris Flower Dataset (9)

**snn_tts_dataset_pkg.vhdl (continued)**

```
( "010100", "001100", "000110", "000001"), -- (134): [5.   3.   1.5  0.25] label: 0  real values: [5.  3.  1.6 0.2]
( "010110", "001010", "001111", "000100"), -- (135): [5.5  2.5  3.75 1.  ] label: 1  real values: [5.5 2.4 3.7 1. ]
( "011011", "001100", "010110", "001010"), -- (136): [6.75 3.   5.5  2.5 ] label: 2  real values: [6.7 3.1 5.6 2.4]
( "010111", "001011", "010100", "001000"), -- (137): [5.75 2.75 5.   2.  ] label: 2  real values: [5.8 2.7 5.1 1.9]
( "010100", "001110", "000110", "000001"), -- (138): [5.   3.5  1.5  0.25] label: 0  real values: [5.1 3.4 1.5 0.2]
( "011010", "001100", "010010", "000101"), -- (139): [6.5  3.   4.5  1.25] label: 1  real values: [6.6 2.9 4.6 1.3]
( "010110", "001100", "010000", "000101"), -- (140): [5.5  3.   4.   1.25] label: 1  real values: [5.6 3.  4.1 1.3]
( "011000", "001101", "010011", "000111"), -- (141): [6.   3.25 4.75 1.75] label: 1  real values: [5.9 3.2 4.8 1.8]
( "011001", "001001", "010010", "000101"), -- (142): [6.25 2.25 4.5  1.25] label: 1  real values: [6.3 2.3 4.4 1.3]
( "010110", "001110", "000101", "000001"), -- (143): [5.5  3.5  1.25 0.25] label: 0  real values: [5.5 3.5 1.3 0.2]
( "010100", "001111", "000110", "000010"), -- (144): [5.   3.75 1.5  0.5 ] label: 0  real values: [5.1 3.7 1.5 0.4]
( "010100", "001100", "000110", "000000"), -- (145): [5.   3.   1.5  0.  ] label: 0  real values: [4.9 3.1 1.5 0.1]
( "011001", "001100", "010110", "000111"), -- (146): [6.25 3.   5.5  1.75] label: 2  real values: [6.3 2.9 5.6 1.8]
( "010111", "001011", "010000", "000100"), -- (147): [5.75 2.75 4.   1.  ] label: 1  real values: [5.8 2.7 4.1 1. ]
( "011111", "001111", "011011", "001001"), -- (148): [7.75 3.75 6.75 2.25] label: 2  real values: [7.7 3.8 6.7 2.2]
( "010010", "001101", "000110", "000001")  -- (149): [4.5  3.25 1.5  0.25] label: 0  real values: [4.6 3.2 1.4 0.2]
);
```

# Global Declaration of Variables in VHDL of Iris SNN Model

# 5.1.3 Declaration of Parametrizable Variables (1)

**snn_tts_support_pkg.vhdl**

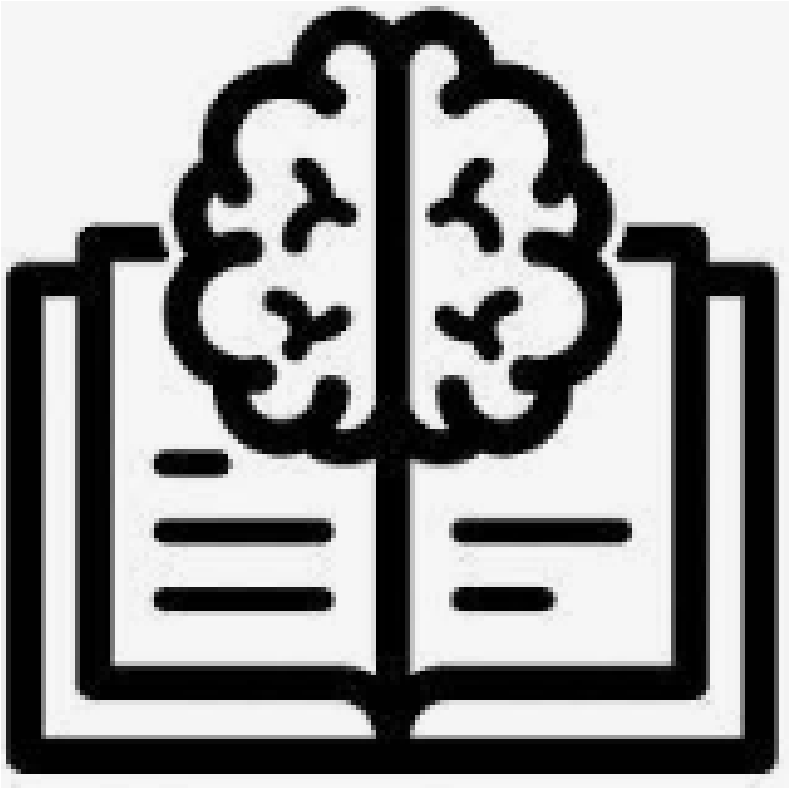
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package snn_ttfs_support_pkg is

 constant NF    : integer :=  4;   -- number of incoming features as signals
 constant NHL   : integer := 10;   -- number of hidden layer neurons
 constant NOL   : integer :=  3;   -- number of output layer neurons
 constant NBS   : integer :=  6;   -- number of bits of signed signal vector
 constant FBS   : integer :=  6;   -- number of fraction bits of the signal vector
 constant NBW   : integer :=  6;   -- number of bits of signed weight vector
 constant FBW   : integer :=  3;   -- number of fraction bits of the weight vector

 -- Input layer arrays
 type f_array    is array(0 to NF-1)  of std_ulogic_vector(0 to NBS-1); -- feature array
 type spikes_IL  is array(0 to NF-1)  of std_ulogic; -- spikes to encode the features
```

# 5.1.3 Declaration of Parametrizable Variables (2)

**snn_tts_support_pkg.vhdl (continued)**

```
-- Hidden layer arrays
type spikes_HL   is array(0 to NHL-1) of std_ulogic; -- spikes to encode the output of the hidden layer
type receiver_HL is array(0 to NF-1)   of std_ulogic_vector(0 to NBW-1);
type w_array_HL_neuron is array(0 to NF-1) of std_ulogic_vector(0 to NBW-1); -- all weights for one hidden layer neuron
type w_array_HL is array(0 to NHL-1) of w_array_HL_neuron; -- weight array for the hidden layer
type b_array_HL is array(0 to NHL-1) of std_ulogic_vector(0 to NBW-1); -- bias array for the hidden layer

-- Output layer arrays
type spikes_OL is array(0 to NOL-1) of std_ulogic; -- spikes to encode the output of the output layer
type receiver_OL is array(0 to NHL-1) of std_ulogic_vector(0 to NBW-1);
type w_array_OL_neuron is array(0 to NHL-1) of std_ulogic_vector(0 to NBW-1); -- all weights for one output layer neuron
type w_array_OL is array(0 to NOL-1) of w_array_OL_neuron; -- weight array for the output layer
type b_array_HL is array(0 to NHL-1) of std_ulogic_vector(0 to NBW-1); -- bias array for the hidden layer

end package snn_ttfs_support_pkg;


package body snn_ttfs_support_pkg is

end package body snn_ttfs_support_pkg;
```

# Quantized Neuron Control in VHDL



# 5.1.4 Quantized Weights and Biases (1)

**snn_tts_weights_biases_pkg.vhdl**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package snn_tts_weights_biases_pkg is

  -- weights hidden layer --
  type weights_HL_vector is array (0 to 3) of std_ulogic_vector(0 to 5);
  type weights_HL_array is array (0 to 9) of weights_HL_vector;
  constant weights_HL : weights_HL_array := (
     ( "001001", "001010", "111100", "110110"),
     -- (0): [ 0.5625  0.625  -0.25   -0.625 ]   real values: [ 0.56443214  0.62897128 -0.2581394  -0.63886768]
     ( "001011", "111101", "111100", "111101"),
     -- (1): [ 0.6875 -0.1875 -0.25   -0.1875]   real values: [ 0.70403385 -0.19055735 -0.24839784 -0.20883079]
     ( "110110", "111101", "000010", "001001"),
     -- (2): [-0.625  -0.1875  0.125   0.5625]   real values: [-0.64027494 -0.19385475  0.12458247  0.58392215]
     ( "111000", "001000", "111000", "000010"),
     -- (3): [-0.5    0.5    -0.5     0.125 ]   real values: [-0.47367612  0.51902795 -0.47925103  0.10448104]
```

# 5.1.4 Quantized Weights and Biases (2)

**snn_tts_weights_biases_pkg.vhdl (continued)**

```
( "110111", "110111", "000011", "110111"),
 -- (4): [-0.5625 -0.5625  0.1875 -0.5625]   real values: [-0.53283244 -0.56416273  0.18218595 -0.53248036]
( "000111", "110110", "001100", "010000"),
-- (5): [ 0.4375 -0.625   0.75    1.    ]   real values: [ 0.41168469 -0.61702091  0.76997805  0.974567  ]
( "111111", "111110", "001011", "000000"),
-- (6): [-0.0625 -0.125   0.6875  0.     ]   real values: [-0.04759748 -0.15084279  0.69958144 -0.0196865 ]
( "000100", "110010", "001111", "011010"),
-- (7): [ 0.25   -0.875   0.9375  1.625 ]   real values: [ 0.27743301 -0.84648818  0.91938794  1.62858176]
( "001101", "001011", "110001", "101110"),
-- (8): [ 0.8125  0.6875 -0.9375 -1.125 ]   real values: [ 0.82760614  0.68355834 -0.91967201 -1.12798941]
( "111100", "000000", "111110", "000010")
-- (9): [-0.25    0.     -0.125   0.125 ]   real values: [-0.27342811  0.02397382 -0.14947277  0.1172204 ]
);
```

# 5.1.4 Quantized Weights and Biases (3)

**snn_tts_weights_biases_pkg.vhdl (continued)**

```
-- biases hidden layer --
type biases_HL_vector is array (0 to 9) of std_ulogic_vector(0 to 5);
constant biases_HL : biases_HL_vector := ( "000111", "000100", "000000", "000000", "000000", "110110",
"111010", "111010", "001010", "000000");
-- [ 0.4375  0.25    0.      0.      0.     -0.625  -0.375  -0.375   0.625  0.    ]
   real values: [ 0.4435366094112396 0.2311095893383026 0.0 0.0 0.0 -0.6203048825263977
                  -0.3804432153701782 -0.372809499502182 0.613123893737793 0.0  ]

-- weights output layer --
type weights_OL_vector is array (0 to 9) of std_ulogic_vector(0 to 5);
type weights_OL_array is array (0 to 2) of weights_OL_vector;
constant weights_OL : weights_OL_array := (
    ( "001000", "000111", "111100", "111010", "110111", "110000", "111010", "110000", "011000", "000101"),
    -- (0): [ 0.5    0.4375 -0.25   -0.375  -0.5625 -1.     -0.375  -1.      1.5    0.3125]
       real values: [ 0.50706267  0.41606125 -0.2590642  -0.3526026  -0.58153099 -1.02304268 -0.36179411
       -0.98597175  1.52065063  0.33047605]
```

# 5.1.4 Quantized Weights and Biases (4)

**snn_tts_weights_biases_pkg.vhdl (continued)**

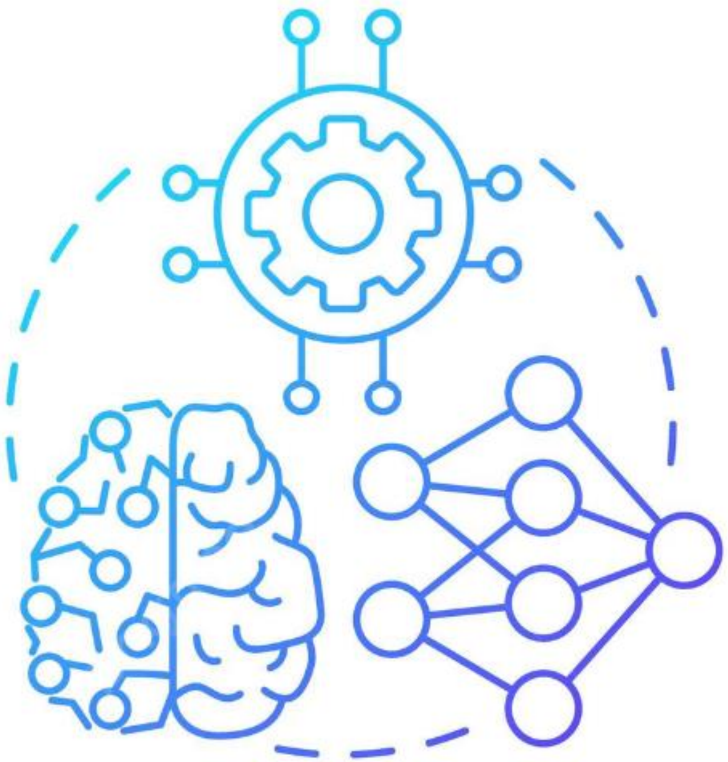
```
( "001011", "001010", "111011", "001010", "000001", "111001", "110111", "000010", "000010", "111011"),
-- (1): [ 0.6875  0.625  -0.3125  0.625   0.0625 -0.4375 -0.5625  0.125   0.125 -0.3125]
   real values: [ 0.67761981  0.63552946 -0.32347474  0.6209265   0.05692106 -0.43432039 -0.55743742
   0.12044821  0.15219755 -0.28212503]
( "110000", "000100", "000010", "000111", "110111", "000100", "001100", "001010", "101000", "111110")
-- (2): [-1.     0.25    0.125   0.4375 -0.5625  0.25    0.75    0.625  -1.5   -0.125 ]
   real values: [-0.989797    0.2248435   0.13220942  0.42212081 -0.53845721  0.28033954  0.75175864
   0.60383815 -1.48336387 -0.09707433]
);
```

# 5.1.4 Quantized Weights and Biases (5)

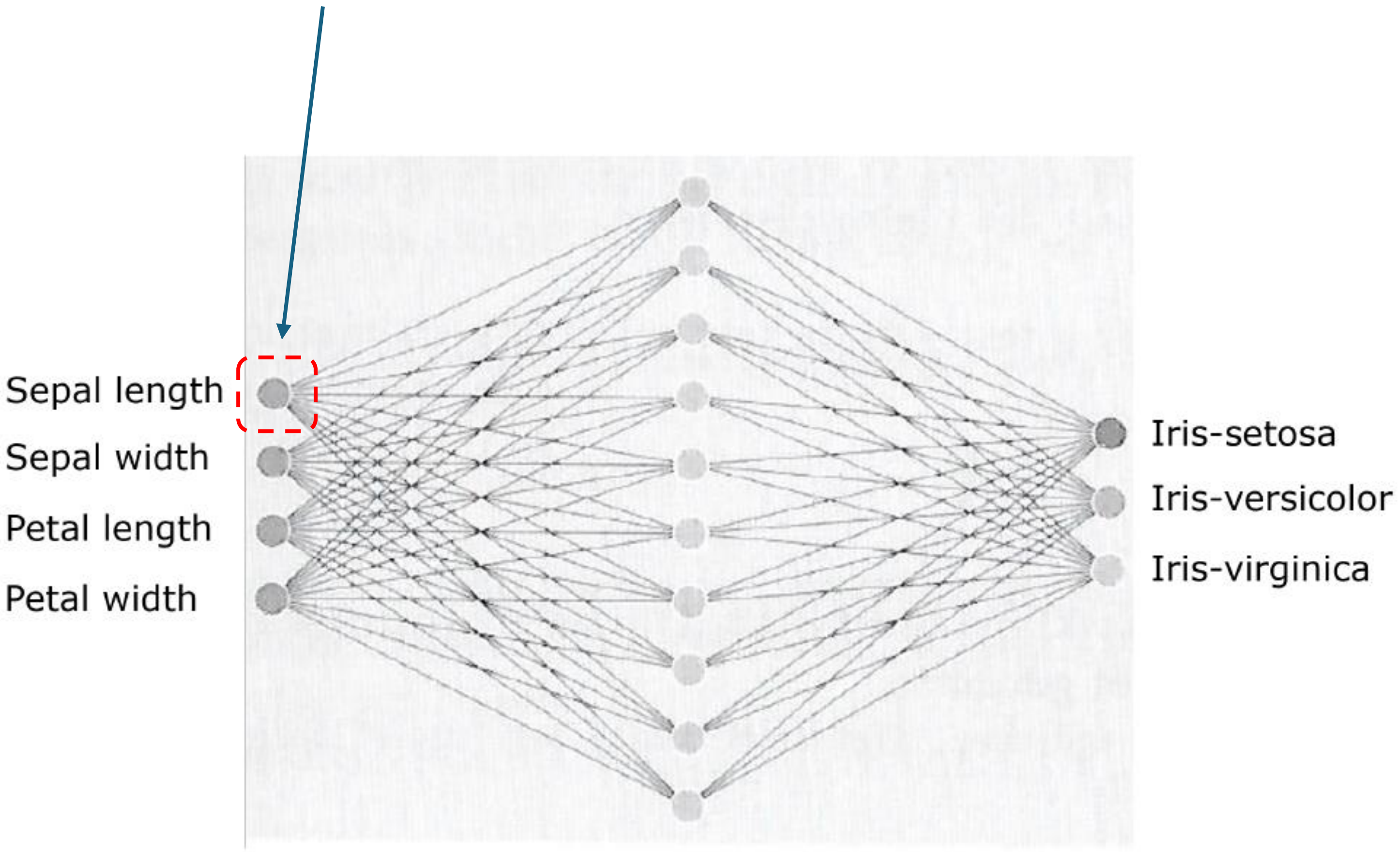
**snn_tts_weights_biases_pkg.vhdl (continued)**

```
  -- biases output layer --
  type biases_OL_vector is array (0 to 2) of std_ulogic_vector(0 to 5);
  constant biases_OL : biases_OL_vector := ( "000101", "000100", "111001");
  -- [ 0.3125  0.25   -0.4375]   real values: [ 0.2986317574977875  0.2707380950450897 -0.4682842195034027]

end package snn_tts_weights_biases_pkg;

package body snn_tts_weights_biases_pkg is


end package body snn_tts_weights_biases_pkg;
```

# VHDL-Modeling of TTS-Encoded Neurons



# Neuron of Input Layer



# 5.1.5 Input Layer Neuron (1)

**snn_tts_inputlayer_neuron.vhdl**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.snn_tts_support_pkg.all;

library [redacted]
use [redacted]
use [redacted]
use [redacted]
use [redacted]

library [redacted]
use [redacted]
use [redacted]

library [redacted]
```

# 5.1.5 Input Layer Neuron (2)

**snn_tts_inputlayer_neuron.vhdl**

```
entity snn_tts_inputlayer_neuron is

 port (
  -------------------------------------------------------------
  -- clock and test IOs, supply
  -------------------------------------------------------------
  gckn   :  in std_ulogic;  -- toggles: global clock (N)

  ckoffn :  in std_ulogic;  -- dc, 1: lck off (N)
  hld    :  in std_ulogic;  -- ac, 0: test hold
  se     :  in std_ulogic;  -- ac, 0: scan enable
  edis   :  in std_ulogic;  -- dc, 0: force enable lck

  e      :  in std_ulogic;  -- ac, 1: enable lck
  dlylck :  in std_ulogic;  -- dc, 0: delay lck
  mpw1n  :  in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw2n  :  in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw3n  :  in std_ulogic;  -- dc, 1: modify pulse width (N)
```

# 5.1.5 Input Layer Neuron (3)

**snn_tts_inputlayer_neuron.vhdl**

[redacted]

```
---------------------------------------------------------------
-- functional IOs
---------------------------------------------------------------
valid_signal : in  std_ulogic;

reset_spike   :    in  std_ulogic;
cycle_counter : in  std_ulogic_vector(0 to NBS-1);
feature_in    :     in  std_ulogic_vector(0 to NBS-1);

transmitter_out : out std_ulogic   -- outgoing spike signal
);

attribute [redacted]
attribute
```

# 5.1.5 Input Layer Neuron (4)

**snn_tts_inputlayer_neuron.vhdl**

```
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]

attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]

attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
```

# 5.1.5 Input Layer Neuron (5)

**snn_tts_inputlayer_neuron.vhdl**

```
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]

attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]

attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
```

# 5.1.5 Input Layer Neuron (6)

**snn_tts_inputlayer_neuron.vhdl**

```
end snn_tts_inputlayer_neuron;

architecture snn_tts_inputlayer_neuron of snn_tts_inputlayer_neuron is

  signal n_feature    : std_ulogic_vector(0 to NBS-1); -- input of data register for feature data
  signal c_feature    : std_ulogic_vector(0 to NBS-1); -- output of data register for feature data

  signal fce  : std_ulogic;
  signal hldn : std_ulogic;

  signal lck_reset : std_ulogic;
  signal e_reset   : std_ulogic;

begin

  e_reset <= reset_spike and e;
```

# 5.1.5 Input Layer Neuron (7)

**snn_tts_inputlayer_neuron.vhdl**

```
-------- Transmitter ---------
-- Register
n_feature <= feature_in;

-- Comparator
transmitter_out <= '1' when valid_signal = '1'
                       and reset_spike = '0'
                       and  c_feature = cycle_counter
                     else
                     '0';

---------------------------------------------------------------
```

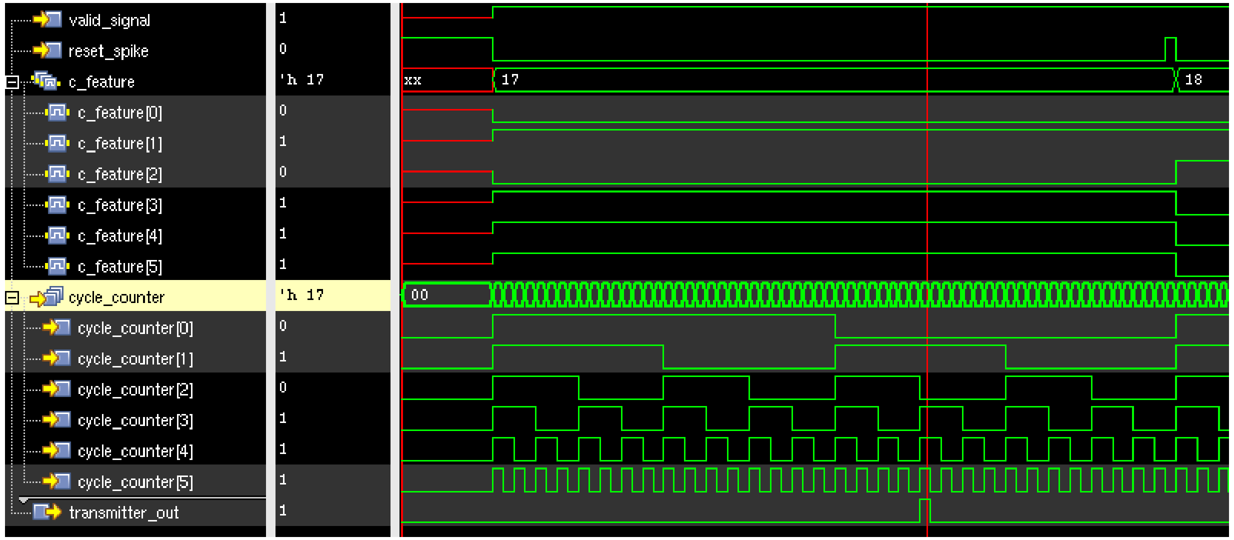



Fig. XX: Illustration of spike generation. Note that a down-counter is used to implement ($T$-$t_{spike}$), i.e., the higher the membrane potential is the earlier the outgoing spike occurs. Here the counter counts down from decimal 64 (hex11111) to hex17 until the spike is generated that then gets integrated in the successive neuron during 23 (hex17) time ticks.

# 5.1.5 Input Layer Neuron (8)

**snn_tts_inputlayer_neuron.vhdl**

```
feature_reg : entity latches.c_elat
   generic map (width => NBS, [redacted])
   port map (
   [redacted]
    lck  => lck_reset,
    d    => n_feature,
    q    => c_feature
    );

--------------------------------------------------------------------------------------------------------

  bidi_lcb_reset : entity latches.c_lcble
   port map (
   [redacted]
    e      => e_reset,   -- in, from PI
    gckn   => gckn,      -- in, from PI
```

# 5.1.5 Input Layer Neuron (9)

**snn_tts_inputlayer_neuron.vhdl**

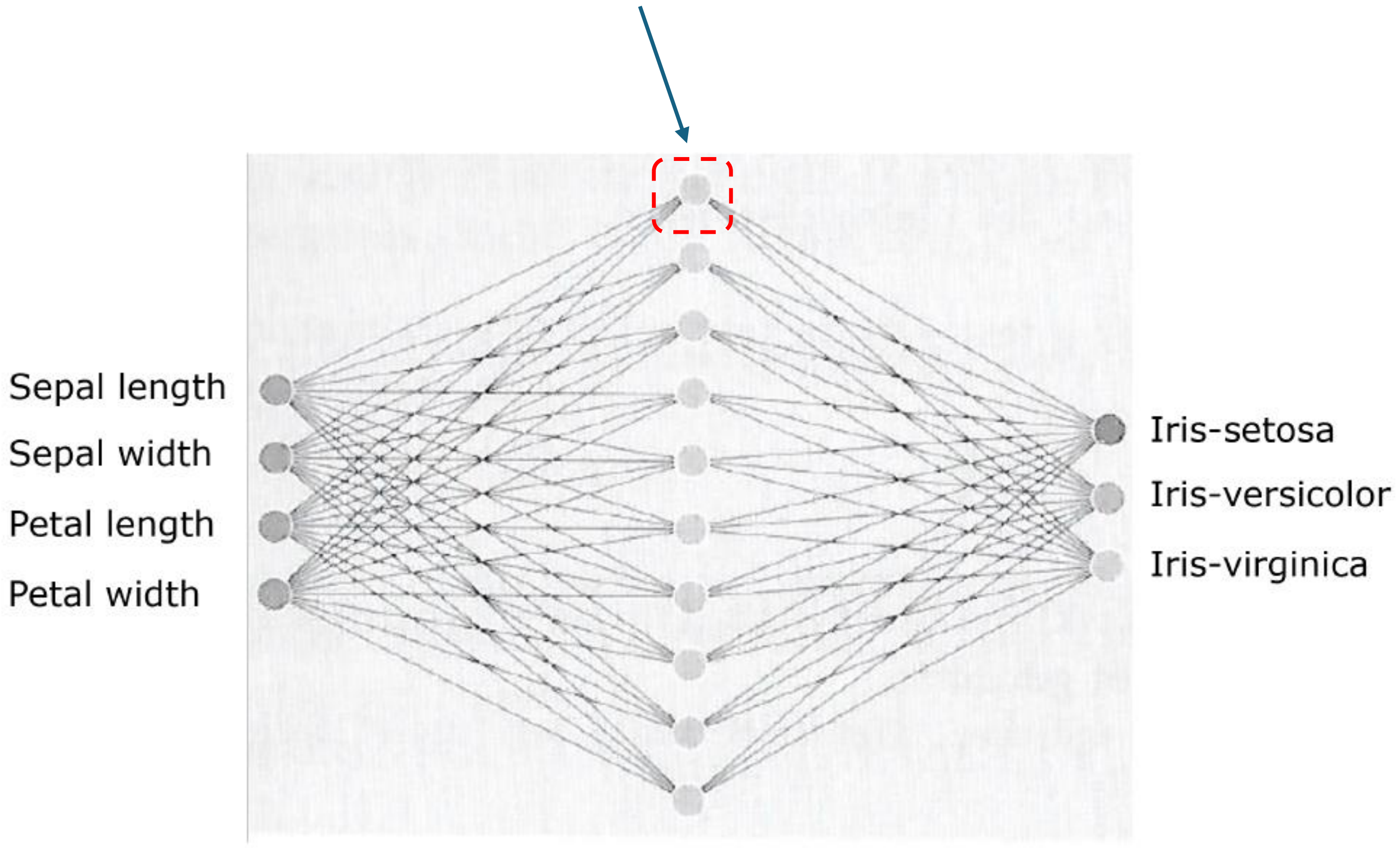
```
    fce    => fce,               -- in, from lcbor
                                 -- (force lck to run, overrides e)
    hldn   => hldn,              -- in, from lcbor
                                 -- (no new data launched, priority over e/fce)
    dlylck => dlylck,            -- in, from PI
    mpw1n  => mpw1n,             -- in, from PI
    mpw2n  => mpw2n,             -- in, from PI
    mpw3n  => mpw3n,             -- in, from PI
    lck    => lck_reset);        -- out, to latches

  bidi_lcbor : entity latches.c_lcbor
   port map (
    [redacted]
    ckoffn => ckoffn,            -- in, from PI
    hld    => hld,               -- in, from PI
    se     => se,                -- in, from PI
```

# 5.1.5 Input Layer Neuron (10)

**snn_tts_inputlayer_neuron.vhdl**

```
    edis   => edis,     -- in, from PI
    fce    => fce,      -- out, to lcb
    hldn   => hldn);    -- out, to lcb

end snn_tts_inputlayer_neuron;
```

# Neuron of Hidden Layer



# 5.1.6 Hidden Layer Neuron (1)

**snn_tts_layer1_neuron.vhdl**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.snn_tts_support_pkg.all;

library [redacted]
use [redacted]
use [redacted]
use [redacted]
use [redacted]

library [redacted]
use [redacted]
use [redacted]


library [redacted]
```

# 5.1.6 Hidden Layer Neuron (2)

**snn_tts_layer1_neuron.vhdl (continued)**

```
---------------------------------------------------------------------------
entity snn_tts_layer1_neuron is

 port (
  ----------------------------------------------------------
  -- clock and test IOs, supply
  ----------------------------------------------------------
  gckn   : in std_ulogic;  -- toggles: global clock (N)

  ckoffn :  in std_ulogic;  -- dc, 1: lck off (N)
  hld    :  in std_ulogic;  -- ac, 0: test hold
  se     :  in std_ulogic;  -- ac, 0: scan enable
  edis   :  in std_ulogic;  -- dc, 0: force enable lck

  e         : in std_ulogic;  -- ac, 1: enable lck
  e_weights : in std_ulogic;  -- 1: enables weights to load
  dlylck    : in std_ulogic;  -- dc, 0: delay lck
```

# 5.1.6 Hidden Layer Neuron (3)

**snn_tts_layer1_neuron.vhdl (continued)**

```
mpw1n    : in std_ulogic;  -- dc, 1: modify pulse width (N)
mpw2n    : in std_ulogic;  -- dc, 1: modify pulse width (N)
mpw3n    : in std_ulogic;  -- dc, 1: modify pulse width (N)

[redacted]

------------------------------------------------------------
-- functional IOs
------------------------------------------------------------
valid_signal : in  std_ulogic;

reset_spike   : in  std_ulogic;
cycle_counter : in  std_ulogic_vector(0 to NBS-1);

spikes_in    : in  spikes_IL; -- incoming spikes of the TTS encoded signal
weights_in    : in  w_array_HL_neuron;
bias_in  : in  std_ulogic_vector(0 to NBW-1);
```

# 5.1.6 Hidden Layer Neuron (4)

**snn_tts_layer1_neuron.vhdl (continued)**

```
  transmitter_out : out std_ulogic   -- outgoing spike signal
    );

    attribute [redacted]
    attribute [redacted]

    attribute [redacted]
    attribute [redacted]
    attribute [redacted]
    attribute [redacted]
    attribute [redacted]

    attribute [redacted]
    attribute [redacted]
    attribute [redacted]
    attribute [redacted]
```

# 5.1.6 Hidden Layer Neuron (5)

**snn_tts_layer1_neuron.vhdl (continued)**

```
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]

end snn_tts_layer1_neuron;

architecture snn_tts_layer1_neuron of snn_tts_layer1_neuron is

  constant roundUp    : std_ulogic_vector(0 to NBS+NBW-1) := (NBW+NBS-FBW => '1', others => '0');
  constant slope_init : std_ulogic_vector(0 to NBW+NBS-1) := (0 to NBW+NBS-1 => '0');

  signal receivers_out : receiver_HL;

  signal slope      : std_ulogic_vector(0 to NBW+NBS-1);
```

# 5.1.6 Hidden Layer Neuron (6)

**snn_tts_layer1_neuron.vhdl (continued)**

```
signal n_membranPot : std_ulogic_vector(0 to NBW+NBS-1);
signal c_membranPot : std_ulogic_vector(0 to NBW+NBS-1);

signal n_bias     : std_ulogic_vector(0 to NBW+NBS-1);
signal c_bias     : std_ulogic_vector(0 to NBW+NBS-1);
signal membranPot : std_ulogic_vector(0 to NBW+NBS-1);

signal n_quantPot : std_ulogic_vector(0 to NBS-1);
signal c_quantPot : std_ulogic_vector(0 to NBS-1);

signal fce  : std_ulogic;
signal hldn : std_ulogic;
signal lck  : std_ulogic;

signal lck_reset   : std_ulogic;
signal e_reset     : std_ulogic;
signal lck_weights : std_ulogic;
```

# 5.1.6 Hidden Layer Neuron (7)

**snn_tts_layer1_neuron.vhdl (continued)**

[redacted]

```
component snn_tts_receiver is
 port (
  gckn : in std_ulogic;  -- toggles: global clock (N)

  ckoffn  : in std_ulogic;  -- dc, 1: lck off (N)
  hld     : in std_ulogic;  -- ac, 0: test hold
  se      : in std_ulogic;  -- ac, 0: scan enable
  edis    : in std_ulogic;  -- dc, 0: force enable lck

  e               : in std_ulogic;  -- ac, 1: enable lck
  e_weights       : in std_ulogic;  -- 1: enables weights to load
  dlylck          : in std_ulogic;  -- dc, 0: delay lck
  mpw1n           : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw2n           : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw3n           : in std_ulogic;  -- dc, 1: modify pulse width (N)
```

# 5.1.6 Hidden Layer Neuron (8)

**snn_tts_layer1_neuron.vhdl (continued)**

[redacted]

```
    reset_spike : in std_ulogic;  -- reset spike to set receiver back

    spike_in        : in       std_ulogic;  -- incoming spikes of the TTS encoded signal
    weight_in       : in       std_ulogic_vector(0 to NBW-1);
    receiver_out    : out      std_ulogic_vector(0 to NBW-1)
  );
 end component;

begin

 e_reset <= reset_spike and e;
```

# 5.1.6 Hidden Layer Neuron (9)

## snn_tts_layer1_neuron.vhdl (continued)

```
-------- Receivers ---------
receiver_0 : snn_tts_receiver
 port map(
  gckn       => gckn,
  ckoffn     => ckoffn,
  hld        => hld,
  se         => se,
  edis       => edis,

  e          => e,
  e_weights => e_weights,
  dlylck   => dlylck,
  mpw1n    => mpw1n,
  mpw2n    => mpw2n,
  mpw3n    => mpw3n,

  [redacted]

  reset_spike  => reset_spike,

  spike_in       => spikes_in(0),
  weight_in      => weights_in(0),
  receiver_out => receivers_out(0)
  );
```

# 5.1.6 Hidden Layer Neuron (10)

## snn_tts_layer1_neuron.vhdl (continued)

```
receiver_1 : snn_tts_receiver
 port map(
  gckn        => gckn,
  ckoffn      => ckoffn,
  hld         => hld,
  se          => se,
  edis        => edis,

  e      => e,
  e_weights => e_weights,
  dlylck    => dlylck,
  mpw1n     => mpw1n,
  mpw2n     => mpw2n,
  mpw3n     => mpw3n,

[redacted]

  reset_spike  => reset_spike,

  spike_in     => spikes_in(1),
  weight_in    => weights_in(1),
  receiver_out => receivers_out(1)
  );
```

# 5.1.6 Hidden Layer Neuron (11)

## snn_tts_layer1_neuron.vhdl (continued)

```
receiver_2 : snn_tts_receiver
 port map(
  gckn        => gckn,
  ckoffn      => ckoffn,
  hld         => hld,
  se          => se,
  edis        => edis,

  e      => e,
  e_weights => e_weights,
  dlylck    => dlylck,
  mpw1n     => mpw1n,
  mpw2n     => mpw2n,
  mpw3n     => mpw3n,

[redacted]

  reset_spike  => reset_spike,

  spike_in       => spikes_in(2),
  weight_in      => weights_in(2),
  receiver_out => receivers_out(2)
  );
```

# 5.1.6 Hidden Layer Neuron (12)

## snn_tts_layer1_neuron.vhdl (continued)

```
receiver_3 : snn_tts_receiver
  port map(
    gckn        => gckn,
    ckoffn      => ckoffn,
    hld         => hld,
    se          => se,
    edis        => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

  [redacted]

    reset_spike  => reset_spike,

    spike_in       => spikes_in(3),
    weight_in      => weights_in(3),
    receiver_out => receivers_out(3)
    );
```

# 5.1.6 Hidden Layer Neuron (13)

**snn_tts_layer1_neuron.vhdl (continued)**

```
-------- adder tree ---------
slope <= std_ulogic_vector(signed(slope_init) + signed(receivers_out(0)) +
signed(receivers_out(1)) + signed(receivers_out(2)) + signed(receivers_out(3)));


-------- integrator ---------
n_membranPot <= (others => '0') when reset_spike = '1' else
                std_ulogic_vector(signed(slope) + signed(c_membranPot));


-------- adder for bias ---------
n_bias <= (0 to NBS-FBS-1 => '1') & bias_in & (0 to FBS-1 => '0') when bias_in(0) = '1' else
      (0 to NBS-FBS-1 => '0') & bias_in & (0 to FBS-1 => '0');
```

# 5.1.6 Hidden Layer Neuron (14)

**snn_tts_layer1_neuron.vhdl (continued)**

```
-- add bias to potential and round half up
membranPot <= std_ulogic_vector(signed(c_bias) + signed(c_membranPot) + signed(roundUp));

-------- Transmitter ---------
-- Quantizer
n_quantPot <= (others => '0') when membranPot(0) = '1' else -- reLU
              (others => '1') when membranPot(1) = '1' else -- clipping
               membranPot(2 to 7);

-- Comparator
transmitter_out <= '1' when valid_signal = '1'
                   and  reset_spike = '0'
                   and  c_quantPot = cycle_counter
                   else '0';
```

# 5.1.6 Hidden Layer Neuron (15)

**snn_tts_layer1_neuron.vhdl (continued)**

```
----------------------------------------------------------------------------------------
  membranPot_reg : entity latches.c_elat
   generic map (width => NBW+NBS, [redacted])
   port map (
    [redacted]
     lck  => lck,
     d    => n_membranPot,
     q    => c_membranPot
     );

  bias_reg : entity latches.c_elat
   generic map (width => NBW+NBS, [redacted])
   port map (
    [redacted]
     lck  => lck_weights,
     d    => n_bias,
     q    => c_bias
     );
```

# 5.1.6 Hidden Layer Neuron (16)

**snn_tts_layer1_neuron.vhdl (continued)**

```
quantPot_reg : entity latches.c_elat
   generic map (width => NBS, [redacted])
   port map (
    [redacted]
    lck  => lck_reset,
    d    => n_quantPot,
    q    => c_quantPot
    );
------------------------------------------------------------------------------------------------
  bidi_lcb : entity latches.c_lcble
   port map (
    [redacted]              -- inout, from PI
                            -- inout, from PI
    e      => e,            -- in, from PI
    gckn   => gckn,         -- in, from PI
    fce    => fce,          -- in, from lcbor
                            -- (force lck to run, overrides e)
    hldn   => hldn,         -- in, from lcbor
                            -- (no new data launched, priority over e/fce)
    dlylck => dlylck,       -- in, from PI
    mpw1n  => mpw1n,        -- in, from PI
    mpw2n  => mpw2n,        -- in, from PI
    mpw3n  => mpw3n,        -- in, from PI
    lck    => lck);         -- out, to latches
```

# 5.1.6 Hidden Layer Neuron (17)

**snn_tts_layer1_neuron.vhdl (continued)**

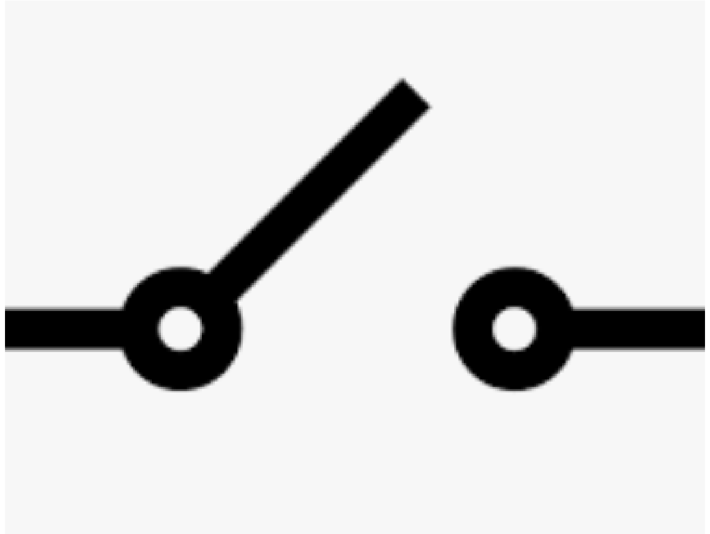
```
bidi_lcb_reset : entity latches.c_lcble
 port map (
  [redacted]
  e      => e_reset,              -- in, from PI
  gckn   => gckn,                 -- in, from PI
  fce    => fce,                  -- in, from lcbor
                                  -- (force lck to run, overrides e)
  hldn   => hldn,                 -- in, from lcbor
                                  -- (no new data launched, priority over e/fce)
  dlylck => dlylck,               -- in, from PI
  mpw1n  => mpw1n,                -- in, from PI
  mpw2n  => mpw2n,                -- in, from PI
  mpw3n  => mpw3n,                -- in, from PI
  lck    => lck_reset);           -- out, to latches
```

# 5.1.6 Hidden Layer Neuron (18)

## snn_tts_layer1_neuron.vhdl (continued)

```
bidi_lcb_weights : entity latches.c_lcble
 port map (
  [redacted]
  e      => e_weights,        -- in, from PI
  gckn   => gckn,             -- in, from PI
  fce    => fce,              -- in, from lcbor
                              -- (force lck to run, overrides e)
  hldn   => hldn,             -- in, from lcbor
                              -- (no new data launched, priority over e/fce)
  dlylck => dlylck,           -- in, from PI
  mpw1n  => mpw1n,            -- in, from PI
  mpw2n  => mpw2n,            -- in, from PI
  mpw3n  => mpw3n,            -- in, from PI
  lck    => lck_weights);     -- out, to latches

bidi_lcbor : entity latches.c_lcbor
 port map (
  [redacted]
  ckoffn => ckoffn,           -- in, from PI
  hld    => hld,              -- in, from PI
  se     => se,               -- in, from PI
  edis   => edis,             -- in, from PI
  fce    => fce,              -- out, to lcb
  hldn   => hldn);            -- out, to lcb

end snn_tts_layer1_neuron;
```

# Receiver within Neuron



# 5.1.7 Neuron Receiver (1)

## snn_tts_receiver.vhdl

**library** ieee;
**use** ieee.std_logic_1164.**all**;
**use** ieee.numeric_std.**all**;
**use** work.zrlswi_snn_tts_support_pkg.**all**;

**library** [redacted]
**use** [redacted]
**use**
**use**
**use**

**library** [redacted]
**use** [redacted]
**use**

**library** [redacted]

------------------------------------------------------------------------------

**entity** snn_tts_receiver **is**

**port (**
-------------------------------------------------------------
-- clock and test IOs, supply
-------------------------------------------------------------
gckn : **in** std_ulogic; -- toggles: global clock (N)

# 5.1.7 Neuron Receiver (2)

**snn_tts_receiver.vhdl (continued)**

```
ckoffn :     in std_ulogic;  -- dc, 1: lck off (N)
hld   :      in std_ulogic;  -- ac, 0: test hold
se    :      in std_ulogic;  -- ac, 0: scan enable
edis  :      in std_ulogic;  -- dc, 0: force enable lck

e      :     in std_ulogic;  -- ac, 1: enable lck
e_weights :  in std_ulogic;  -- 1: enables weights to load
dlylck   :   in std_ulogic;  -- dc, 0: delay lck
mpw1n   :    in std_ulogic;  -- dc, 1: modify pulse width (N)
mpw2n   :    in std_ulogic;  -- dc, 1: modify pulse width (N)
mpw3n   :    in std_ulogic;  -- dc, 1: modify pulse width (N)

[redacted]

-----------------------------------------------------------
-- functional IOs
-----------------------------------------------------------

reset_spike : in std_ulogic; -- resets the receiver

spike_in    : in  std_ulogic; -- incoming spike of the TTS encoded signal
weight_in   : in  std_ulogic_vector(0 to NBW-1);
receiver_out : out std_ulogic_vector(0 to NBW-1)  -- outgoing weight
);

attribute [redacted]
attribute [redacted]
```

# 5.1.7 Neuron Receiver (3)

## snn_tts_receiver.vhdl (continued)

```
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]

  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]

  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]

end snn_tts_receiver;

architecture snn_tts_receiver of snn_tts_receiver is

  signal n_weight : std_ulogic_vector(0 to NBW-1);
  signal c_weight : std_ulogic_vector(0 to NBW-1);
  signal n_switch : std_ulogic; -- works as switch when spike arrives
  signal c_switch : std_ulogic;
```

# 5.1.7 Neuron Receiver (4)

**snn_tts_receiver.vhdl (continued)**

```
  signal fce  : std_ulogic;
  signal hldn : std_ulogic;
  signal lck  : std_ulogic;

  signal lck_weights : std_ulogic;

[redacted]

begin

  -- weight register
  n_weight <= weight_in;

  -- signal changes to 1  when spike comes in. Works as a switch.
  n_switch <= '0' when reset_spike = '1' else
              '1' when spike_in = '1' else
              c_switch;

  -- sends out weight as long as switch is closed
  receiver_out <= c_weight when n_switch = '1' else
                  (others => '0');
```

# 5.1.7 Neuron Receiver (5)

## snn_tts_receiver.vhdl (continued)

```
-----------------------------------------------------------------------------------------------------

weight_reg : entity latches.c_elat
 generic map (width => NBW, [redacted])
 port map (
 [redacted]
  lck => lck_weights,
  d   => n_weight,
  q   => c_weight
  );

switch_reg : entity latches.c_elat
 generic map (width => 1, [redacted])
 port map (
 [redacted]
  lck  => lck,
  d(0) => n_switch,
  q(0) => c_switch
  );
```

# 5.1.7 Neuron Receiver (6)

**snn_tts_receiver.vhdl (continued)**

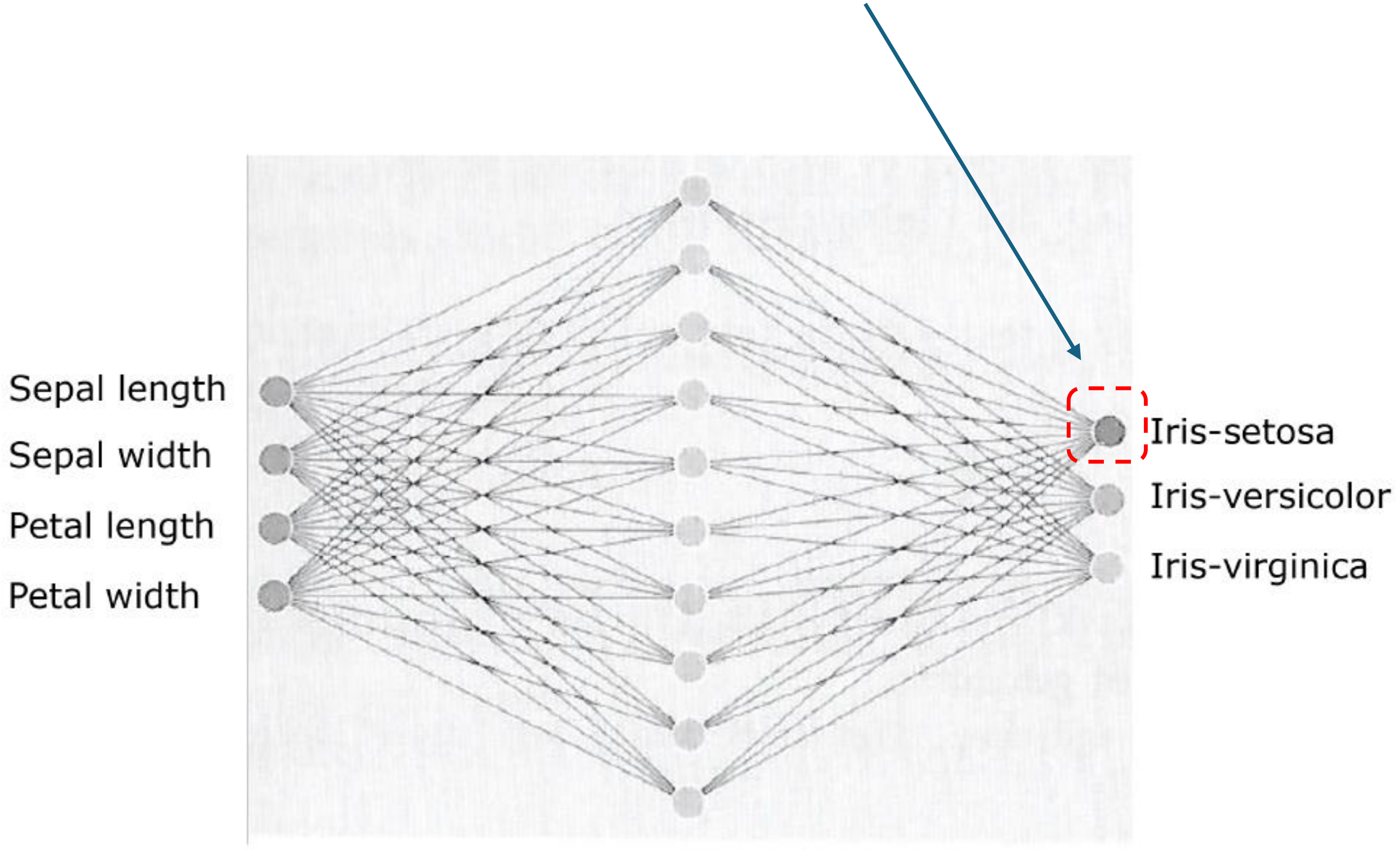
```
bidi_lcb_weights : entity latches.c_lcble
  port map (
    [redacted]
    e      => e_weights,  -- in, from PI
    gckn   => gckn,       -- in, from PI
    fce    => fce,        -- in, from lcbor
                          -- (force lck to run, overrides e)
    hldn   => hldn,       -- in, from lcbor
                          -- (no new data launched, priority over e/fce)
    dlylck => dlylck,     -- in, from PI
    mpw1n  => mpw1n,                -- in, from PI
    mpw2n  => mpw2n,                -- in, from PI
    mpw3n  => mpw3n,                -- in, from PI
    lck    => lck_weights);         -- out, to latches
```

# 5.1.7 Neuron Receiver (7)

## snn_tts_receiver.vhdl (continued)

```
bidi_lcb : entity latches.c_lcble
  port map (
   [redacted]
   e     => e,                          -- in, from PI
   gckn  => gckn,                       -- in, from PI
   fce   => fce,                        -- in, from lcbor
                                        -- (force lck to run, overrides e)
   hldn  => hldn,                       -- in, from lcbor
                                        -- (no new data launched, priority over e/fce)
   dlylck => dlylck,                    -- in, from PI
   mpw1n  => mpw1n,                     -- in, from PI
   mpw2n  => mpw2n,                     -- in, from PI
   mpw3n  => mpw3n,                     -- in, from PI
   lck   => lck);                       -- out, to latches

 bidi_lcbor : entity latches.c_lcbor
  port map (
   [redacted]
   ckoffn => ckoffn,                    -- in, from PI
   hld   => hld,                        -- in, from PI
   se    => se,                         -- in, from PI
   edis  => edis,                       -- in, from PI
   fce   => fce,                        -- out, to lcb
   hldn  => hldn);                      -- out, to lcb

end snn_tts_receiver;
```

# Neuron of Output Layer



# 5.1.8 Output Layer Neuron (1)

## snn_tts_outputlayer_neuron.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.snn_tts_support_pkg.all;

library [redacted]
use
use
use
use [redacted]

library [redacted]
use
use [redacted]

library [redacted]

--------------------------------------------------------------------------------

entity snn_tts_outputlayer_neuron is

 port (
  -------------------------------------------------------------
  -- clock and test IOs, supply
  -------------------------------------------------------------
  gckn   : in std_ulogic;          -- toggles: global clock (N)

  ckoffn : in std_ulogic;          -- dc, 1: lck off (N)
  hld    : in std_ulogic;          -- ac, 0: test hold
  se     : in std_ulogic;          -- ac, 0: scan enable
  edis   : in std_ulogic;          -- dc, 0: force enable lck
```

# 5.1.8 Output Layer Neuron (2)

## snn_tts_outputlayer_neuron.vhdl (continued)

```
    e               : in std_ulogic;  -- ac, 1: enable lck
    e_weights       : in std_ulogic;  -- 1: enables weights to load
    dlylck          : in std_ulogic;  -- dc, 0: delay lck
    mpw1n           : in std_ulogic;  -- dc, 1: modify pulse width (N)
    mpw2n           : in std_ulogic;  -- dc, 1: modify pulse width (N)
    mpw3n           : in std_ulogic;  -- dc, 1: modify pulse width (N)

[redacted]

    --------------------------------------------------------------
    -- functional IOs
    --------------------------------------------------------------
    valid_signal : in  std_ulogic;

    reset_spike     : in  std_ulogic;
    cycle_counter : in  std_ulogic_vector(0 to NBS-1);

    spikes_in        : in  spikes_HL;  -- incoming spikes of the TTS encoded signal
    weights_in       : in  w_array_OL_neuron;
    bias_in          : in  std_ulogic_vector(0 to NBW-1);

    transmitter_out : out std_ulogic   -- outgoing spike signal
    );

attribute [redacted]
attribute [redacted]

attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
```

# 5.1.8 Output Layer Neuron (3)

## snn_tts_outputlayer_neuron.vhdl (continued)

```
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]

attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]

end snn_tts_outputlayer_neuron;

architecture snn_tts_outputlayer_neuron of snn_tts_outputlayer_neuron is

 constant roundUp   : std_ulogic_vector(0 to NBS+NBW-1) := (NBW+NBS-FBW => '1', others => '0');
 constant slope_init : std_ulogic_vector(0 to NBW+NBS-1) := (0 to NBW+NBS-1 => '0');

 signal receivers_out : receiver_OL;

 signal slope     : std_ulogic_vector(0 to NBW+NBS-1);

 signal n_membranPot  : std_ulogic_vector(0 to NBW+NBS-1);
 signal c_membranPot  : std_ulogic_vector(0 to NBW+NBS-1);

 signal n_bias     : std_ulogic_vector(0 to NBW+NBS-1);
 signal c_bias     : std_ulogic_vector(0 to NBW+NBS-1);
 signal membranPot : std_ulogic_vector(0 to NBW+NBS-1);

 signal n_quantPot : std_ulogic_vector(0 to NBS-1);
 signal c_quantPot : std_ulogic_vector(0 to NBS-1);
```

# 5.1.8 Output Layer Neuron (4)

## snn_tts_outputlayer_neuron.vhdl (continued)

```
signal fce   : std_ulogic;
signal hldn : std_ulogic;
signal lck   : std_ulogic;

signal lck_reset     : std_ulogic;
signal e_reset        : std_ulogic;
signal lck_weights : std_ulogic;

attribute [redacted]

component snn_tts_receiver is
 port (
  gckn : in std_ulogic;    -- toggles: global clock (N)

  ckoffn : in std_ulogic;    -- dc, 1: lck off (N)
  hld      : in std_ulogic;    -- ac, 0: test hold
  se        : in std_ulogic;    -- ac, 0: scan enable
  edis    : in std_ulogic;    -- dc, 0: force enable lck

  e               : in std_ulogic;  -- ac, 1: enable lck
  e_weights : in std_ulogic;  -- 1: enables weights to load
  dlylck       : in std_ulogic;  -- dc, 0: delay lck
  mpw1n     : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw2n     : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw3n     : in std_ulogic;  -- dc, 1: modify pulse width (N)

 [redacted]

  reset_spike : in std_ulogic;  -- reset spike to set receiver back

  spike_in     : in  std_ulogic;  -- incoming spikes of the TTS encoded signal
  weight_in   : in  std_ulogic_vector(0 to NBW-1);
```

# 5.1.8 Output Layer Neuron (5)

## snn_tts_outputlayer_neuron.vhdl (continued)

```
      receiver_out : out std_ulogic_vector(0 to NBW-1)
    );
  end component;

 begin

  e_reset <= reset_spike and e;

  -------- Receivers ---------
  receiver_0 : snn_tts_receiver
   port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e        => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

   [redacted]

    reset_spike  => reset_spike,

    spike_in     => spikes_in(0),
    weight_in    => weights_in(0),
    receiver_out => receivers_out(0)
    );
```

# 5.1.8 Output Layer Neuron (6)

## snn_tts_outputlayer_neuron.vhdl (continued)

```
receiver_1 : snn_tts_receiver
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

    [redacted]

    reset_spike  => reset_spike,

    spike_in     => spikes_in(1),
    weight_in    => weights_in(1),
    receiver_out => receivers_out(1)
    );
```

```
receiver_2 : snn_tts_receiver
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

    [redacted]

    reset_spike  => reset_spike,

    spike_in     => spikes_in(2),
    weight_in    => weights_in(2),
    receiver_out => receivers_out(2)
    );
```

```
receiver_3 : snn_tts_receiver
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

    [redacted]

    reset_spike  => reset_spike,

    spike_in     => spikes_in(3),
    weight_in    => weights_in(3),
    receiver_out => receivers_out(3)
    );
```

```
receiver_4 : snn_tts_receiver
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

    [redacted]

    reset_spike  => reset_spike,

    spike_in     => spikes_in(4),
    weight_in    => weights_in(4),
    receiver_out => receivers_out(4)
    );
```

```
receiver_5 : snn_tts_receiver
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

    [redacted]

    reset_spike  => reset_spike,

    spike_in     => spikes_in(5),
    weight_in    => weights_in(5),
    receiver_out => receivers_out(5)
    );
```

# 5.1.8 Output Layer Neuron (7)

## snn_tts_outputlayer_neuron.vhdl (continued)

```
receiver_6 : snn_tts_receiver
  port map(
   gckn  => gckn,
   ckoffn => ckoffn,
   hld   => hld,
   se    => se,
   edis  => edis,

   e      => e,
   e_weights => e_weights,
   dlylck   => dlylck,
   mpw1n    => mpw1n,
   mpw2n    => mpw2n,
   mpw3n    => mpw3n,
```

[redacted]

```
   reset_spike  => reset_spike,

   spike_in    => spikes_in(6),
   weight_in    => weights_in(6),
   receiver_out => receivers_out(6)
   );
```

```
receiver_7 : snn_tts_receiver
  port map(
   gckn  => gckn,
   ckoffn => ckoffn,
   hld   => hld,
   se    => se,
   edis  => edis,

   e      => e,
   e_weights => e_weights,
   dlylck   => dlylck,
   mpw1n    => mpw1n,
   mpw2n    => mpw2n,
   mpw3n    => mpw3n,
```

[redacted]

```
   reset_spike  => reset_spike,

   spike_in    => spikes_in(7),
   weight_in    => weights_in(7),
   receiver_out => receivers_out(7)
   );
```

```
receiver_8 : snn_tts_receiver
  port map(
   gckn  => gckn,
   ckoffn => ckoffn,
   hld   => hld,
   se    => se,
   edis  => edis,

   e      => e,
   e_weights => e_weights,
   dlylck   => dlylck,
   mpw1n    => mpw1n,
   mpw2n    => mpw2n,
   mpw3n    => mpw3n,
```

[redacted]

```
   reset_spike  => reset_spike,

   spike_in    => spikes_in(8),
   weight_in    => weights_in(8),
   receiver_out => receivers_out(8)
   );
```

```
receiver_9 : snn_tts_receiver
  port map(
   gckn  => gckn,
   ckoffn => ckoffn,
   hld   => hld,
   se    => se,
   edis  => edis,

   e      => e,
   e_weights => e_weights,
   dlylck   => dlylck,
   mpw1n    => mpw1n,
   mpw2n    => mpw2n,
   mpw3n    => mpw3n,
```

[redacted]

```
   reset_spike  => reset_spike,

   spike_in    => spikes_in(9),
   weight_in    => weights_in(9),
   receiver_out => receivers_out(9)
   );
```

# 5.1.8 Output Layer Neuron (8)

## snn_tts_outputlayer_neuron.vhdl (continued)

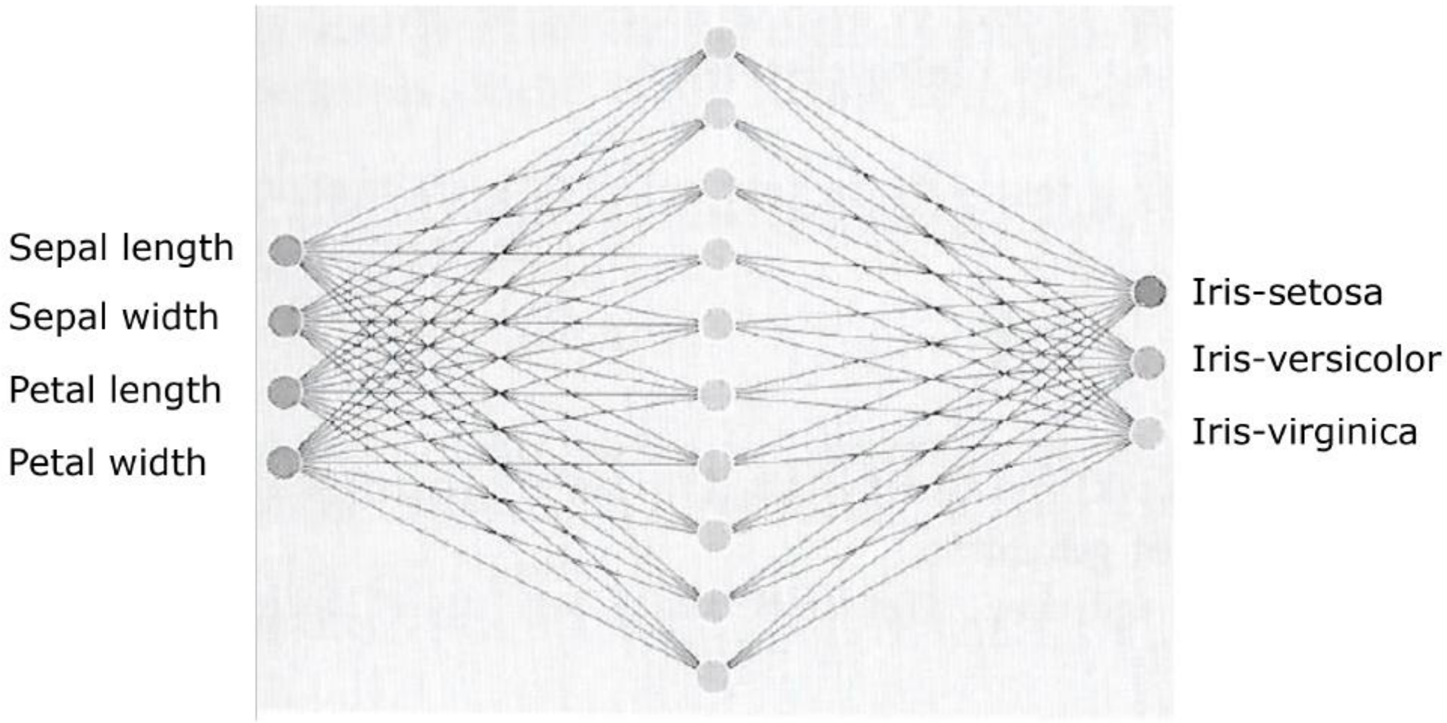
```
-------- adder tree ---------
slope <= std_ulogic_vector(signed(slope_init) + signed(receivers_out(0)) + signed(receivers_out(1)) + signed(receivers_out(2)) + signed(receivers_out(3)) + signed(receivers_out(4)) +
signed(receivers_out(5)) + signed(receivers_out(6)) + signed(receivers_out(7)) + signed(receivers_out(8)) + signed(receivers_out(9)));


-------- adder with register  ---------
n_membranPot <= (others => '0') when reset_spike = '1' else
                std_ulogic_vector(signed(slope) + signed(c_membranPot));

-------- adder for bias ---------
-- bias register
n_bias <= (0 to NBS-FBS-1 => '1') & bias_in & (0 to FBS-1 => '0') when bias_in(0) = '1' else
          (0 to NBS-FBS-1 => '0') & bias_in & (0 to FBS-1 => '0');


-- add bias to potential and round half up
membranPot <= std_ulogic_vector(signed(c_bias) + signed(c_membranPot) + signed(roundUp));

-------- Transmitter ---------
-- Quantizer
n_quantPot <= (others => '0') when membranPot(0) = '1' else -- reLU
              (others => '1') when membranPot(1) = '1' else -- clipping
              membranPot(2 to 7);

-- Comparator
transmitter_out <= '1' when valid_signal = '1' and  reset_spike = '0' and  c_quantPot = cycle_counter
                   else '0';

------------------------------------------------------------------------------------------------------------------
```

# 5.1.8 Output Layer Neuron (9)

## snn_tts_outputlayer_neuron.vhdl (continued)

```
membranPot_reg : entity latches.c_elat
  generic map (width => NBW+NBS, [redacted])
  port map (
    [redacted]
    lck  => lck,
    d    => n_membranPot,
    q    => c_membranPot
  );

bias_reg : entity latches.c_elat
  generic map (width => NBW+NBS, [redacted])
  port map (
    [redacted]
    lck  => lck_weights,
    d    => n_bias,
    q    => c_bias
  );

quantPot_reg : entity latches.c_elat
  generic map (width => NBS, [redacted])
  port map (
    [redacted]
    lck  => lck_reset,
    d    => n_quantPot,
    q    => c_quantPot
  );

--------------------------------------------------------------

bidi_lcb : entity latches.c_lcble
  port map (
    [redacted]
    e      => e,                -- in, from PI
    gckn   => gckn,             -- in, from PI
    fce    => fce,              -- in, from lcbor
                                -- (force lck to run, overrides e)
    hldn   => hldn,             -- in, from lcbor
                                -- (no new data launched, priority over e/fce)
    dlylck => dlylck,           -- in, from PI
    mpw1n  => mpw1n,            -- in, from PI
    mpw2n  => mpw2n,            -- in, from PI
    mpw3n  => mpw3n,            -- in, from PI
    lck    => lck);             -- out, to latches

bidi_lcb_reset : entity latches.c_lcble
  port map (
    [redacted]
    e      => e_reset,          -- in, from PI
    gckn   => gckn,             -- in, from PI
    fce    => fce,              -- in, from lcbor
                                -- (force lck to run, overrides e)
    hldn   => hldn,             -- in, from lcbor
                            -- (no new data launched, priority over e/fce)
    dlylck => dlylck,           -- in, from PI
    mpw1n  => mpw1n,            -- in, from PI
    mpw2n  => mpw2n,            -- in, from PI
    mpw3n  => mpw3n,            -- in, from PI
    lck    => lck_reset);       -- out, to latches

bidi_lcb_weights : entity latches.c_lcble
  port map (
    [redacted]
    e      => e_weights,        -- in, from PI
    gckn   => gckn,             -- in, from PI
    fce    => fce,              -- in, from lcbor
                                -- (force lck to run, overrides e)
    hldn   => hldn,             -- in, from lcbor
                            -- (no new data launched, priority over e/fce)
    dlylck => dlylck,           -- in, from PI
    mpw1n  => mpw1n,            -- in, from PI
    mpw2n  => mpw2n,            -- in, from PI
    mpw3n  => mpw3n,            -- in, from PI
    lck    => lck_weights);     -- out, to latches

bidi_lcbor : entity latches.c_lcbor
  port map (
    [redacted]
    ckoffn => ckoffn,           -- in, from PI
    hld    => hld,              -- in, from PI
    se     => se,               -- in, from PI
    edis   => edis,             -- in, from PI
    fce    => fce,              -- out, to lcb
    hldn   => hldn);            -- out, to lcb

end snn_tts_outputlayer_neuron;
```

# (4,10,3)-Neural Network Model



# 5.1.9 (4,10,3)-Neural Network Model (1)

## snn_tts.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.snn_tts_support_pkg.all;

library [redacted]
use [redacted]
use [redacted]
use [redacted]
use [redacted]

library [redacted]
use [redacted]
use [redacted]

library latches;

--------------------------------------------------------------------------------

entity snn_tts is

 port (
  --------------------------------------------------------------
  -- clock and test IOs, supply
  --------------------------------------------------------------
  gckn    : in std_ulogic;  -- toggles: global clock (N)

  ckoffn : in std_ulogic;  -- dc, 1: lck off (N)
  hld     : in std_ulogic;  -- ac, 0: test hold
  se      : in std_ulogic;  -- ac, 0: scan enable
  edis    : in std_ulogic;  -- dc, 0: force enable lck

  e            : in std_ulogic;  -- ac, 1: enable lck
  e_weights : in std_ulogic;  -- 1: enables weights to load
  dlylck     : in std_ulogic;  -- dc, 0: delay lck
  mpw1n     : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw2n     : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw3n     : in std_ulogic;  -- dc, 1: modify pulse width (N)

  [redacted]

  --------------------------------------------------------------
  -- functional IOs
  --------------------------------------------------------------
  reset_network    : in std_ulogic;
  reset_spike_out  : out std_ulogic;

  f_in          : in  f_array;     -- incoming signal, features of the data
  w_HL_in    : in  w_array_HL;  -- weights for hidden layer
  w_OL_in    : in  w_array_OL;  -- weights for output layer
  bias_HL_in : in  b_array_HL;  -- bias for hidden layer
  bias_OL_in : in  b_array_OL;  -- bias for output layer

  label_out  : out std_ulogic_vector(0 to NOL-1)  -- output predicted label
  );

attribute [redacted]
attribute [redacted]

attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
attribute [redacted]
```

# 5.1.9 (4,10,3)-Neural Network Model (2)

## snn_tts.vhdl (continued)

```
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]

  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]
  attribute [redacted]

end snn_tts;

architecture snn_tts of snn_tts is

  signal n_counter     : std_ulogic_vector(0 to NBS-1);
  signal c_counter     : std_ulogic_vector(0 to NBS-1);
  signal cycle_counter : std_ulogic_vector(0 to NBS-1);
  signal reset_spike   : std_ulogic;

  signal n_active_layers : std_ulogic_vector(0 to 3);
  signal c_active_layers : std_ulogic_vector(0 to 3);

  signal spikes_IL_out : spikes_IL; -- spike signal between input layer and hidden layer
  signal spikes_HL_out : spikes_HL; -- spike signal between hidden layer and output layer
  signal spikes_OL_out : spikes_OL; -- spike signal output of output layer

  signal n_label  : std_ulogic_vector(0 to NOL-1);
  signal c_label  : std_ulogic_vector(0 to NOL-1);

  signal n_label_out  : std_ulogic_vector(0 to NOL-1);
  signal c_label_out  : std_ulogic_vector(0 to NOL-1);

  signal fce  : std_ulogic;
  signal hldn : std_ulogic;
  signal lck  : std_ulogic;

  signal lck_reset : std_ulogic;
  signal e_reset   : std_ulogic;

  signal lck_label : std_ulogic;
  signal e_label   : std_ulogic;

  component snn_tts_inputlayer_neuron is
   port (
    gckn          : in std_ulogic;  -- toggles: global clock (N)

    ckoffn        : in std_ulogic;  -- dc, 1: lck off (N)
    hld           : in std_ulogic;  -- ac, 0: test hold
    se            : in std_ulogic;  -- ac, 0: scan enable
    edis          : in std_ulogic;  -- dc, 0: force enable lck

    e     : in std_ulogic;       -- ac, 1: enable lck
    dlylck : in std_ulogic;      -- dc, 0: delay lck
    mpw1n  : in std_ulogic;  -- dc, 1: modify pulse width (N)
    mpw2n  : in std_ulogic;  -- dc, 1: modify pulse width (N)
    mpw3n  : in std_ulogic;  -- dc, 1: modify pulse width (N)

    [redacted]

    valid_signal : in  std_ulogic;

    reset_spike   : in  std_ulogic;
    cycle_counter : in  std_ulogic_vector(0 to NBS-1);
```

# 5.1.9 (4,10,3)-Neural Network Model (3)

## snn_tts.vhdl (continued)

```
  feature_in     : in  std_ulogic_vector(0 to NBS-1);
  transmitter_out : out std_ulogic   -- outgoing spike signal
 );
end component;

component snn_tts_layer1_neuron is
 port (
  gckn   : in std_ulogic;  -- toggles: global clock (N)

  ckoffn  : in std_ulogic;     -- dc, 1: lck off (N)
  hld      : in std_ulogic;     -- ac, 0: test hold
  se       : in std_ulogic;     -- ac, 0: scan enable
  edis     : in std_ulogic;     -- dc, 0: force enable lck

  e            : in std_ulogic;  -- ac, 1: enable lck
  e_weights : in std_ulogic;  -- 1: enables weights to load
  dlylck       : in std_ulogic;  -- dc, 0: delay lck
  mpw1n      : in std_ulogic;   -- dc, 1: modify pulse width (N)
  mpw2n      : in std_ulogic;   -- dc, 1: modify pulse width (N)
  mpw3n      : in std_ulogic;   -- dc, 1: modify pulse width (N)

  [redacted]

  valid_signal : in std_ulogic;

  reset_spike     : in std_ulogic;
  cycle_counter : in std_ulogic_vector(0 to NBS-1);

  spikes_in      : in spikes_IL;  -- incoming spikes of the TTS encoded signal
  weights_in    : in w_array_HL_neuron;
  bias_in         : in std_ulogic_vector(0 to NBW-1);
  transmitter_out : out std_ulogic   -- outgoing spike signal
 );
end component;

component snn_tts_outputlayer_neuron is
 port (
  gckn : in std_ulogic;    -- toggles: global clock (N)

  ckoffn  : in std_ulogic;  -- dc, 1: lck off (N)
  hld       : in std_ulogic;   -- ac, 0: test hold
  se        : in std_ulogic;  -- ac, 0: scan enable
  edis      : in std_ulogic;  -- dc, 0: force enable lck

  e       : in std_ulogic;  -- ac, 1: enable lck
  e_weights : in std_ulogic;  -- 1: enables weights to load
  dlylck    : in std_ulogic;  -- dc, 0: delay lck
  mpw1n    : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw2n    : in std_ulogic;  -- dc, 1: modify pulse width (N)
  mpw3n    : in std_ulogic;  -- dc, 1: modify pulse width (N)

  [redacted]

  valid_signal : in std_ulogic;

  reset_spike      : in std_ulogic;
  cycle_counter : in std_ulogic_vector(0 to NBS-1);

  spikes_in     : in spikes_HL;  -- incoming spikes of the TTS encoded signal
  weights_in   : in w_array_OL_neuron;
  bias_in        : in std_ulogic_vector(0 to NBW-1);

  transmitter_out : out std_ulogic   -- outgoing spike signal
 );
end component;
```

# 5.1.9 (4,10,3)-Neural Network Model (4)

**snn_tts.vhdl (continued)**

```
begin

  e_reset <= reset_spike and e;

  -------- Validation Check ---------
  n_active_layers(0) <= '0' when reset_network = '1' else
               '1' when f_in(0) /= 0 and reset_spike = '1' else
               '1' when f_in(1) /= 0 and reset_spike = '1' else
               '1' when f_in(2) /= 0 and reset_spike = '1' else
               '1' when f_in(3) /= 0 and reset_spike = '1' else
               '0';

   n_active_layers(1 to 3) <= (others => '0') when reset_network = '1' else
                  c_active_layers(0 to 2);

  -------- Reset clock & Counter ---------
  n_counter <= std_ulogic_vector(unsigned(c_counter) - 1) when c_active_layers(0) = '1' else
                              std_ulogic_vector(unsigned(c_counter) - 1) when c_active_layers(1) = '1' else
                              std_ulogic_vector(unsigned(c_counter) - 1) when c_active_layers(2) = '1' else
                              std_ulogic_vector(unsigned(c_counter) - 1) when c_active_layers(3) = '1' else
                              (others => '0');

  cycle_counter <= n_counter;

  reset_spike <= '1' when reset_network = '1' else
           '1' when n_counter = 0 else
           '0';

  reset_spike_out <= reset_spike;
```

# 5.1.9 (4,10,3)-Neural Network Model (5)

## snn_tts.vhdl (continued)

```
-------- Input layer ---------
  -- Neuron 0 input layer
  neuron_0_IL : snn_tts_inputlayer_neuron
   port map(
     gckn   => gckn,
     ckoffn => ckoffn,
     hld    => hld,
     se     => se,
     edis   => edis,

     e      => e,
     dlylck => dlylck,
     mpw1n  => mpw1n,
     mpw2n  => mpw2n,
     mpw3n  => mpw3n,
```

[redacted]

```
     valid_signal => c_active_layers(0),

     reset_spike   => reset_spike,
     cycle_counter => cycle_counter,

     feature_in     => f_in(0), -- sending feature signal to the neuron
     transmitter_out => spikes_IL_out(0)  -- receiving output signal from the neuron
     );
```

```
  -- Neuron 1 input layer
  neuron_1_IL : snn_tts_inputlayer_neuron
   port map(
     gckn   => gckn,
     ckoffn => ckoffn,
     hld    => hld,
     se     => se,
     edis   => edis,

     e      => e,
     dlylck => dlylck,
     mpw1n  => mpw1n,
     mpw2n  => mpw2n,
     mpw3n  => mpw3n,
```

[redacted]

```
     valid_signal => c_active_layers(0),

     reset_spike   => reset_spike,
     cycle_counter => cycle_counter,

     feature_in     => f_in(1), -- sending feature signal to the neuron
     transmitter_out => spikes_IL_out(1)     -- receiving output signal from the neuron
     );
```

# 5.1.9 (4,10,3)-Neural Network Model (6)

## snn_tts.vhdl (continued)

```
-- Neuron 2 input layer
 neuron_2_IL : snn_tts_inputlayer_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e      => e,
   dlylck => dlylck,
   mpw1n  => mpw1n,
   mpw2n  => mpw2n,
   mpw3n  => mpw3n,
```

[redacted]

```
   valid_signal => c_active_layers(0),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   feature_in     => f_in(2), -- sending feature signal to the neuro
   transmitter_out => spikes_IL_out(2)     -- receiving output signal from the neuron
   );
```

```
-- Neuron 3 input layer
 neuron_3_IL : snn_tts_inputlayer_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e      => e,
   dlylck => dlylck,
   mpw1n  => mpw1n,
   mpw2n  => mpw2n,
   mpw3n  => mpw3n,
```

[redacted]

```
   valid_signal => c_active_layers(0),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   feature_in     => f_in(3), -- sending feature signal to the neuron
   transmitter_out => spikes_IL_out(3)     -- receiving output signal from the neuron
   );
```

# 5.1.9 (4,10,3)-Neural Network Model (7)

## snn_tts.vhdl (continued)

```
-------- Hidden layer ---------
 -- Neuron 0 hidden layer
 neuron_0_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck    => dlylck,
   mpw1n     => mpw1n,
   mpw2n     => mpw2n,
   mpw3n     => mpw3n,

 [redacted]

   valid_signal => c_active_layers(1),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(0),      -- sending weights to the neuron
   bias_in    => bias_HL_in(0),   -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(0)     -- receiving output signal from the neuron
   );
```

```
 -- Neuron 1 hidden layer
 neuron_1_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck    => dlylck,
   mpw1n     => mpw1n,
   mpw2n     => mpw2n,
   mpw3n     => mpw3n,

 [redacted]

   valid_signal => c_active_layers(1),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(1),      -- sending weights to the neuron
   bias_in    => bias_HL_in(1),   -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(1)     -- receiving output signal from the neuron
   );
```

# 5.1.9 (4,10,3)-Neural Network Model (8)

## snn_tts.vhdl (continued)

```
-- Neuron 2 hidden layer
 neuron_2_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck   => dlylck,
   mpw1n    => mpw1n,
   mpw2n    => mpw2n,
   mpw3n    => mpw3n,

   valid_signal => c_active_layers(1),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(2),      -- sending weights to the neuron
   bias_in    => bias_HL_in(2),    -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(2)     -- receiving output signal from the neuron
   );
```

```
-- Neuron 3 hidden layer
 neuron_3_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck    => dlylck,
   mpw1n     => mpw1n,
   mpw2n     => mpw2n,
   mpw3n     => mpw3n,

   valid_signal => c_active_layers(1),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(3), -- sending weights to the neuron
   bias_in    => bias_HL_in(3), -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(3)     -- receiving output signal from the neuron
   );
```

# 5.1.9 (4,10,3)-Neural Network Model (9)

## snn_tts.vhdl (continued)

```
-- Neuron 4 hidden layer
 neuron_4_HL : snn_tts_layer1_neuron
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

    valid_signal => c_active_layers(1),

    reset_spike   => reset_spike,
    cycle_counter => cycle_counter,

    spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
    weights_in => w_HL_in(4),      -- sending weights to the neuron
    bias_in    => bias_HL_in(4),   -- sending bias to the neuron

    transmitter_out  => spikes_HL_out(4)     -- receiving output signal from the neuron
    );
```

```
-- Neuron 5 hidden layer
 neuron_5_HL : snn_tts_layer1_neuron
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,

    valid_signal => c_active_layers(1),

    reset_spike   => reset_spike,
    cycle_counter => cycle_counter,

    spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
    weights_in => w_HL_in(5),      -- sending weights to the neuron
    bias_in    => bias_HL_in(5),   -- sending bias to the neuron

    transmitter_out  => spikes_HL_out(5)     -- receiving output signal from the neuron
    );
```

# 5.1.9 (4,10,3)-Neural Network Model (10)

## snn_tts.vhdl (continued)

```
-- Neuron 6 hidden layer
 neuron_6_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck    => dlylck,
   mpw1n     => mpw1n,
   mpw2n     => mpw2n,
   mpw3n     => mpw3n,

[redacted]

   valid_signal => c_active_layers(1),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(6),      -- sending weights to the neuron
   bias_in    => bias_HL_in(6),   -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(6)     -- receiving output signal from the neuron
   );
```

```
neuron_7_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck    => dlylck,
   mpw1n     => mpw1n,
   mpw2n     => mpw2n,
   mpw3n     => mpw3n,

[redacted]

   valid_signal => c_active_layers(1),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(7),      -- sending weights to the neuron
   bias_in    => bias_HL_in(7),   -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(7)     -- receiving output signal from the neuron
   );
```

# 5.1.9 (4,10,3)-Neural Network Model (11)

## snn_tts.vhdl (continued)

```
-- Neuron 8 hidden layer
 neuron_8_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck    => dlylck,
   mpw1n     => mpw1n,
   mpw2n     => mpw2n,
   mpw3n     => mpw3n,

[redacted]

   valid_signal => c_active_layers(1),

   reset_spike   => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(8),      -- sending weights to the neuron
   bias_in    => bias_HL_in(8),    -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(8)     -- receiving output signal from the neuron
   );
```

```
-- Neuron 9 hidden layer
 neuron_9_HL : snn_tts_layer1_neuron
  port map(
   gckn   => gckn,
   ckoffn => ckoffn,
   hld    => hld,
   se     => se,
   edis   => edis,

   e       => e,
   e_weights => e_weights,
   dlylck    => dlylck,
   mpw1n     => mpw1n,
   mpw2n     => mpw2n,
   mpw3n     => mpw3n,

[redacted]

   valid_signal => c_active_layers(1),

   reset_spike  => reset_spike,
   cycle_counter => cycle_counter,

   spikes_in  => spikes_IL_out, -- sending spike signal to the neuron
   weights_in => w_HL_in(9),      -- sending weights to the neuron
   bias_in    => bias_HL_in(9),    -- sending bias to the neuron

   transmitter_out  => spikes_HL_out(9)     -- receiving output signal from the neuron
   );
```

# 5.1.9 (4,10,3)-Neural Network Model (12)

## snn_tts.vhdl (continued)

```
-------- Output layer ---------
  -- Neuron 0 output layer
  neuron_0_OL : snn_tts_outputlayer_neuron
   port map(
     gckn   => gckn,
     ckoffn => ckoffn,
     hld    => hld,
     se     => se,
     edis   => edis,

     e       => e,
     e_weights => e_weights,
     dlylck    => dlylck,
     mpw1n     => mpw1n,
     mpw2n     => mpw2n,
     mpw3n     => mpw3n,
```

[redacted]

```
     valid_signal => c_active_layers(2),

     reset_spike  => reset_spike,
     cycle_counter => cycle_counter,

     spikes_in  => spikes_HL_out, -- sending spike signal to the neuron
     weights_in => w_OL_in(0),    -- sending weights to the neuron
     bias_in    => bias_OL_in(0), -- sending bias to the neuron

     transmitter_out  => spikes_OL_out(0)  -- receiving output signal from the neuron
     );
```

```
-- Neuron 1 output layer
 neuron_1_OL : snn_tts_outputlayer_neuron
  port map(
    gckn   => gckn,
    ckoffn => ckoffn,
    hld    => hld,
    se     => se,
    edis   => edis,

    e       => e,
    e_weights => e_weights,
    dlylck    => dlylck,
    mpw1n     => mpw1n,
    mpw2n     => mpw2n,
    mpw3n     => mpw3n,
```

[redacted]

```
    valid_signal => c_active_layers(2),

    reset_spike  => reset_spike,
    cycle_counter => cycle_counter,

    spikes_in  => spikes_HL_out, -- sending spike signal to the neuron
    weights_in => w_OL_in(1),      -- sending weights to the neuron
    bias_in    => bias_OL_in(1),    -- sending bias to the neuron

    transmitter_out  => spikes_OL_out(1)    -- receiving output signal from the neuron
    );
```

# 5.1.9 (4,10,3)-Neural Network Model (13)

## snn_tts.vhdl (continued)

```
-------- Output layer ---------
  -- Neuron 0 output layer
  neuron_0_OL : snn_tts_outputlayer_neuron
   port map(
     gckn   => gckn,
     ckoffn => ckoffn,
     hld    => hld,
     se     => se,
     edis   => edis,

     e       => e,
     e_weights => e_weights,
     dlylck    => dlylck,
     mpw1n     => mpw1n,
     mpw2n     => mpw2n,
     mpw3n     => mpw3n,

[redacted]

     valid_signal => c_active_layers(2),

     reset_spike  => reset_spike,
     cycle_counter => cycle_counter,

     spikes_in  => spikes_HL_out, -- sending spike signal to the neuron
     weights_in => w_OL_in(0),    -- sending weights to the neuron
     bias_in    => bias_OL_in(0), -- sending bias to the neuron

     transmitter_out  => spikes_OL_out(0)  -- receiving output signal from the neuron
     );
```

```
  -- Neuron 1 output layer
  neuron_1_OL : snn_tts_outputlayer_neuron
   port map(
     gckn   => gckn,
     ckoffn => ckoffn,
     hld    => hld,
     se     => se,
     edis   => edis,

     e       => e,
     e_weights => e_weights,
     dlylck    => dlylck,
     mpw1n     => mpw1n,
     mpw2n     => mpw2n,
     mpw3n     => mpw3n,

[redacted]

     valid_signal => c_active_layers(2),

     reset_spike  => reset_spike,
     cycle_counter => cycle_counter,

     spikes_in  => spikes_HL_out, -- sending spike signal to the neuron
     weights_in => w_OL_in(1),      -- sending weights to the neuron
     bias_in    => bias_OL_in(1),    -- sending bias to the neuron

     transmitter_out  => spikes_OL_out(1)     -- receiving output signal from the neuron
     );
```

# 5.1.9 (4,10,3)-Neural Network Model (14)

## snn_tts.vhdl (continued)

```
-- Neuron 2 output layer
  neuron_2_OL : snn_tts_outputlayer_neuron
   port map(
     gckn   => gckn,
     ckoffn => ckoffn,
     hld    => hld,
     se     => se,
     edis   => edis,

     e       => e,
     e_weights => e_weights,
     dlylck    => dlylck,
     mpw1n     => mpw1n,
     mpw2n     => mpw2n,
     mpw3n     => mpw3n,

[redacted]

     valid_signal => c_active_layers(2),

     reset_spike  => reset_spike,
      cycle_counter => cycle_counter,

     spikes_in  => spikes_HL_out, -- sending spike signal to the neuron
     weights_in => w_OL_in(2),    -- sending weights to the neuron
     bias_in    => bias_OL_in(2), -- sending bias to the neuron

     transmitter_out => spikes_OL_out(2)  -- receiving output signal from the neuron
     );
```

# 5.1.9 (4,10,3)-Neural Network Model (15)

**snn_tts.vhdl (continued)**

```
-------- Class label ---------
  n_label <= (others => '0') when reset_network = '1' else
              "100" when spikes_OL_out(0) = '1' else
              "010" when spikes_OL_out(1) = '1' else
              "001" when spikes_OL_out(2) = '1' else
              (others => '0');

  e_label <= '1' and e when reset_spike = '1' else
        '1' and e when c_label = "000" and c_active_layers(2) = '1'  else
        '0';

  n_label_out <= c_label;

  label_out <= c_label_out;
```

# 5.1.9 (4,10,3)-Neural Network Model (14)

## snn_tts.vhdl (continued)

```
reset_counter_reg : entity latches.c_elat
  generic map (width => NBS, offset => 0)
  port map (
  [redacted]
   lck  => lck,
   d    => n_counter,
   q    => c_counter
   );

 active_layers_reg : entity latches.c_elat
  generic map (width => 4, offset => 0)
  port map (
  [redacted]
   lck  => lck_reset,
   d    => n_active_layers(0 to 3),
   q    => c_active_layers(0 to 3)
   );

 label_reg : entity latches.c_elat
  generic map (width => NOL, offset => 0)
  port map (
  [redacted]
   lck  => lck_label,
   d    => n_label,
   q    => c_label
   );
```

```
label_out_reg : entity latches.c_elat
  generic map (width => NOL, offset => 0)
  port map (
  [redacted]
   lck  => lck_reset,
   d    => n_label_out,
   q    => c_label_out
   );

---------------------------------------------------------------------------

 bidi_lcb : entity latches.c_lcble
  port map (
  [redacted]
   e      => e,       -- in, from PI
   gckn   => gckn,    -- in, from PI
   fce    => fce,     -- in, from lcbor
                 -- (force lck to run, overrides e)
   hldn   => hldn,    -- in, from lcbor
                 -- (no new data launched, priority over e/fce)
   dlylck => dlylck,  -- in, from PI
   mpw1n  => mpw1n,   -- in, from PI
   mpw2n  => mpw2n,   -- in, from PI
   mpw3n  => mpw3n,   -- in, from PI
   lck    => lck);    -- out, to latches
```

# 5.1.9 (4,10,3)-Neural Network Model (15)

## snn_tts.vhdl (continued)

```
bidi_lcb_reset : entity latches.c_lcble
  port map (
   [redacted]
   e     => e_reset, -- in, from PI
   gckn   => gckn,    -- in, from PI
   fce    => fce,     -- in, from lcbor
                -- (force lck to run, overrides e)
   hldn   => hldn,    -- in, from lcbor
                -- (no new data launched, priority over e/fce)
   dlylck => dlylck,  -- in, from PI
   mpw1n  => mpw1n,   -- in, from PI
   mpw2n  => mpw2n,   -- in, from PI
   mpw3n  => mpw3n,   -- in, from PI
   lck    => lck_reset);    -- out, to latches

 bidi_lcb_label : entity latches.c_lcble
  port map (
   [redacted]
   e     => e_label, -- in, from PI
   gckn   => gckn,    -- in, from PI
   fce    => fce,     -- in, from lcbor
                -- (force lck to run, overrides e)
   hldn   => hldn,    -- in, from lcbor
                -- (no new data launched, priority over e/fce)
   dlylck => dlylck,  -- in, from PI
   mpw1n  => mpw1n,   -- in, from PI
   mpw2n  => mpw2n,   -- in, from PI
   mpw3n  => mpw3n,   -- in, from PI
   lck    => lck_label);    -- out, to latches

bidi_lcbor : entity latches.c_lcbor
  port map (
   [redacted]
   ckoffn => ckoffn,  -- in, from PI
   hld    => hld,     -- in, from PI
   se     => se,      -- in, from PI
   edis   => edis,    -- in, from PI
   fce    => fce,     -- out, to lcb
   hldn   => hldn);   -- out, to lcb

end snn_tts;
```

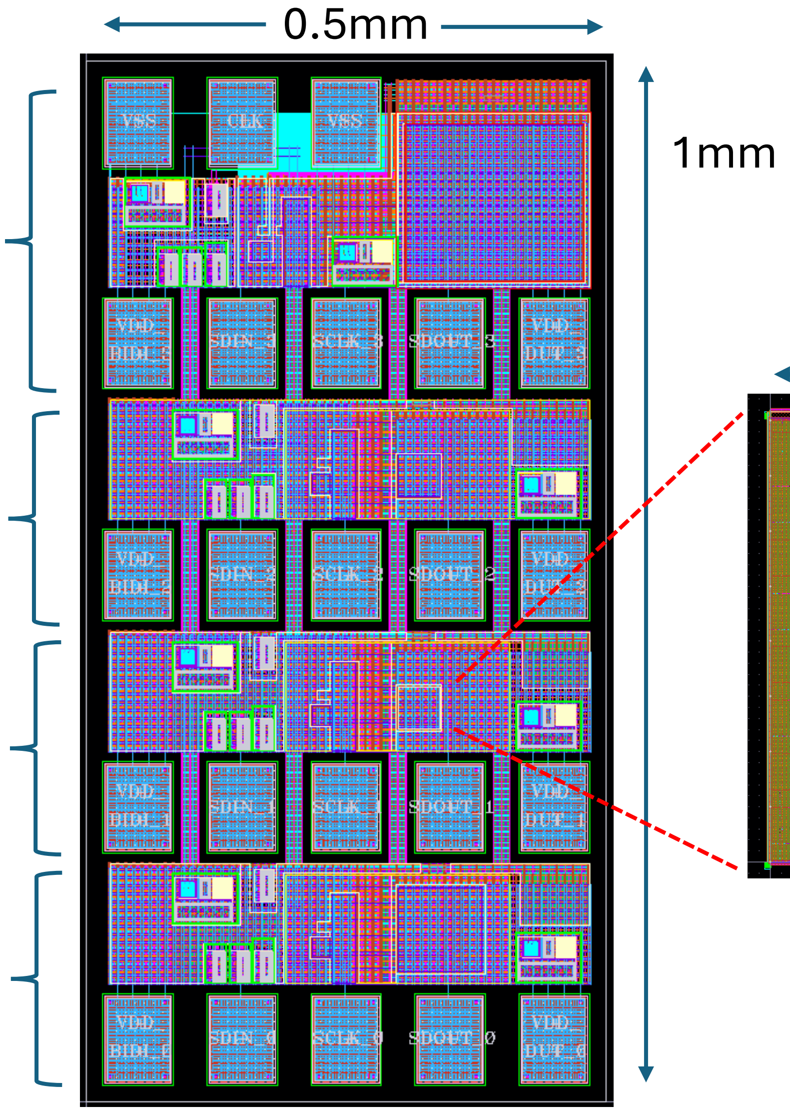
# 5.2 Layout of Synthesized ANN and SNN Designs

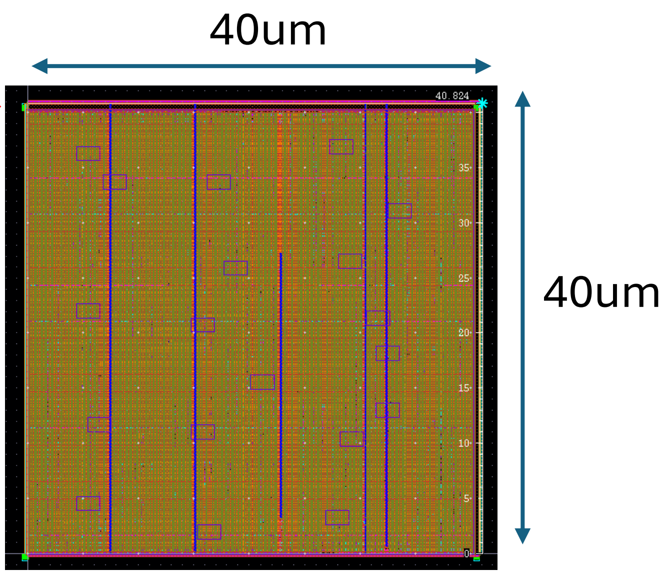
The layout shows the padcage experiments of ANN and SNN implementations performing the Iris flower classification task.

The padcage labeled TTS contains the TTS design whose vhdl is shown in this tutorial. The synthesized SNN core has a size of 40 um x 40 um and is depicted on the right.

The synthesis and physical design flow to get from the vhdl to the layout is not contained in this tutorial for confidentiality and license cost reasons.

The technology is 5nm finFET CMOS.




TTS SNN based on the vhdl code presented in this tutorial

# Answers to the Questions Related to Bloom's Learning Taxonomy

# Q&A 2 Spiking Neural Networks

Answer to L1: The processing of spikes in neurons is called integrate-and-fire dynamics.

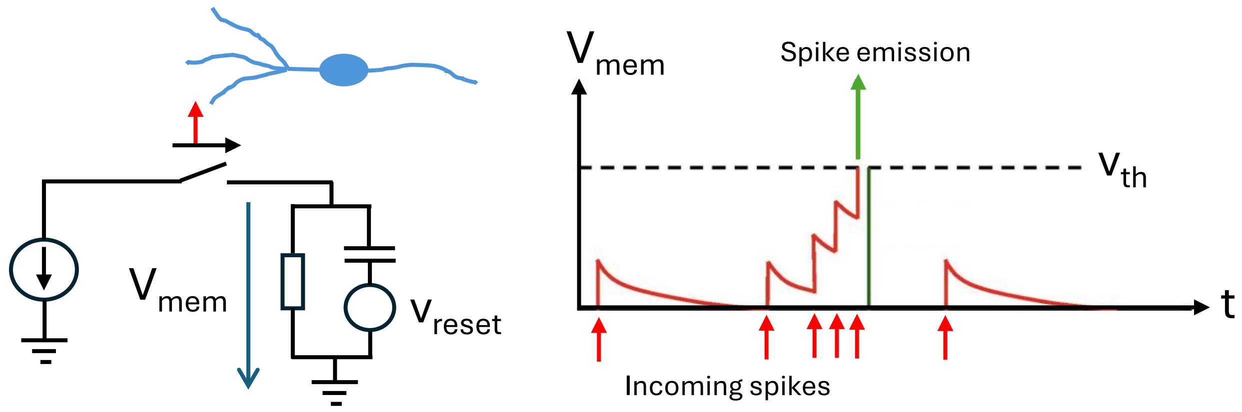



**Fig. 7**: Electrical modelling of the leaky integrate-and-fire dynamics.

**Working principle:**

A neuron is represented by an RC circuit with a threshold voltage $v_{th}$. Each input pulse (e.g., a spike from a different neuron) causes a short current pulse that is **integrated** onto a capacitor, which builds up the **membrane potential** $V_{mem}$.
There is a leakage resistor that decays the voltage.
If $V_{mem}$ reaches $v_{th}$, an **output spike** is generated and the voltage $V_{mem}$ is **reset**.

Back to Tutorial

# Q&A 2 Spiking Neural Networks

Answer to L2: The two figures are connected through the operation of the integrate-and-fire dynamics.

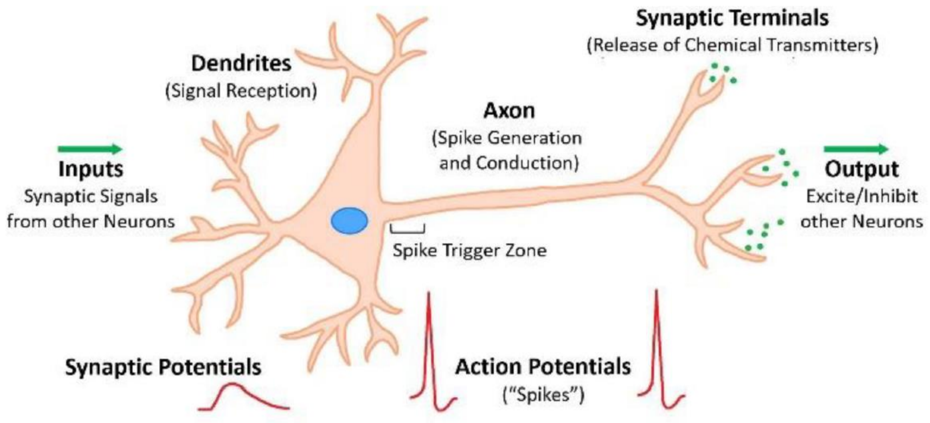



**Fig. 8**: Biological neuron cell applying spike transmission [2].

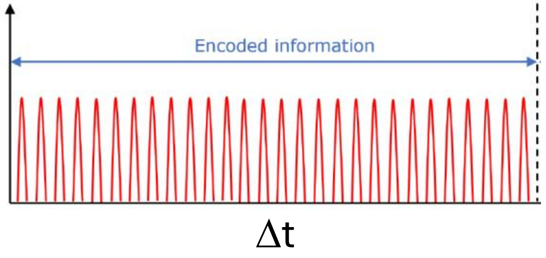
**Explanation:**

Fig. 8 illustrates a simplified diagram of a biological neuron and of the events associated with neuronal activity.
The neuronal characteristics serve as inspiration for how SNNs are set up and operate, as the incoming synaptic signals are received by **dendrites** building up a **membrane potential** that triggers the **emission of spikes** once a **threshold value** is reached.
A possible implementation of these neuronal characteristics using electrical signals is given by the **integrate-and-fire** process outlined at the answer of L1.

**Back to Tutorial**

# Q&A 2.1 Rate Encoding of Spikes in SNNs

Answer to L2: The <u>advantage</u> of rate encoding is that the timing between the individual spikes does not need to be very accurate. Hence timing jitter does not become a major issue. The <u>disadvantage</u> though is latency and power consumption. The latency is increased because each spike of the train must be followed by an idle period to distinguish it from the previous spike. Moreover, the larger the code range is the longer the spike train becomes. Also, power consumption might become a problem because of the high number of spikes to be generated.




Number of spikes $n_s$ within $\Delta t$:

$$\frac{n_s}{\Delta t} \Rightarrow \underbrace{D[0:N-1]}_{\text{N-bit wide binary data transmitted via rate encoding}}$$

Note that a counter is an integrator in the digital domain. The leaky term of the integrate-and-fire process is not modelled here.

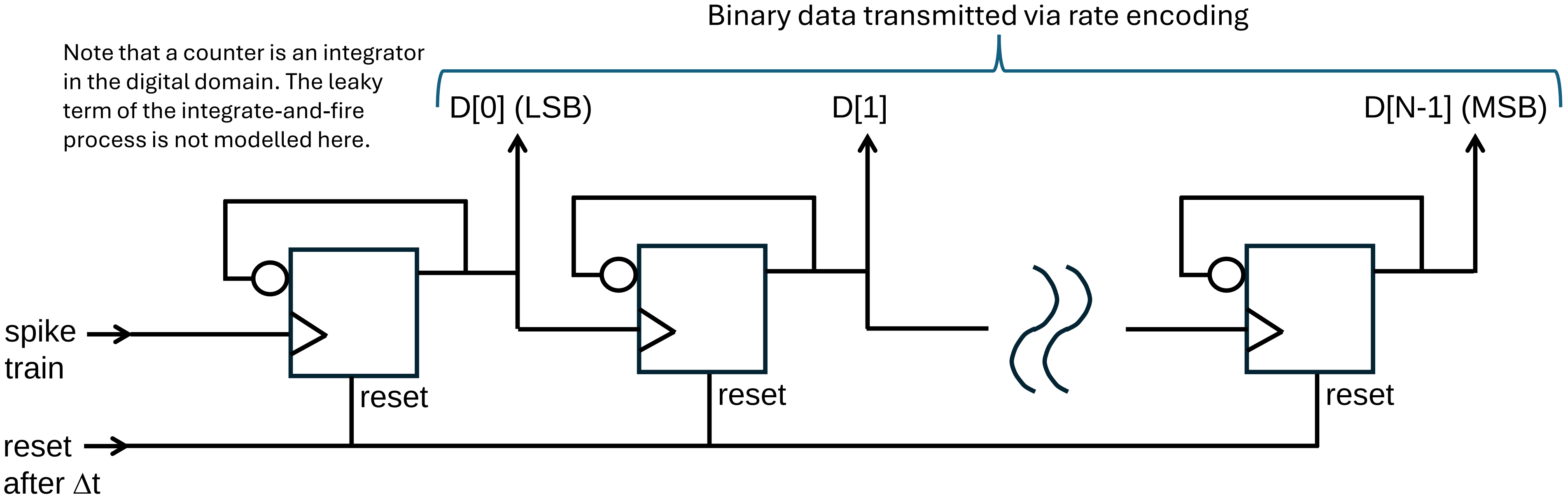



**Fig. 5**: Resetable counter used to measure the rate of the spike train.

Back to Tutorial

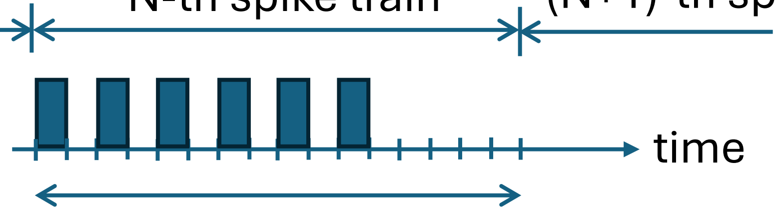
# Q&A 2.2 Time Encoding of Spikes in SNNs

Answer to L2: In **rate encoding** the information is transmitted by counting the number of spikes $n_s$ within a given observation interval $\Delta t$, followed by mapping the spike density $n_s/\Delta t$ to a numerical value, e.g., a binary number D[0:N-1]. This type of encoding requires that individual spikes can be distinguished from one each other, i.e., each spike must be followed by **the absence of a spike** to define it as a spike.

$$\frac{n_s}{\Delta t} \Rightarrow D[0{:}N\text{-}1]$$

(N–1)-th spike train N-th spike train (N+1)-th spike train

Rate encoding:

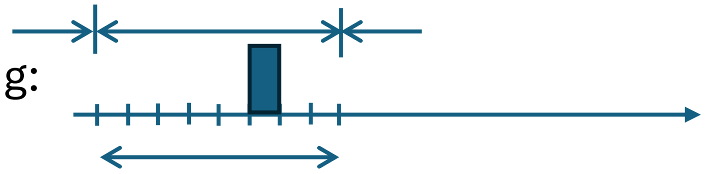



$\Delta t$ providing 16 positions for max. 8 spikes, i.e., code range is 2^3=8, 3 bits

Example shows D[0:2]=b110 (decimal 6, little endian)

For **time encoding** the absence of spikes **is not required.** Thus, the observation interval $\Delta t$ can be halved for the same code range and the data rate doubles w.r.t. rate encoding.

Time encoding:



$\Delta t$ providing 8 positions (3bits) for time encoded spikes

Example shows a spike at time position 6. Hence, the data transmitted is D[0:2]=b110

Back to Tutorial

# Q&A 2.2.1 Inter-Spike Interval Encoding

**Advantage:**
Because the information is defined by the time interval between two spikes, it is not necessary to synchronize the time base of individual neurons within a layer.

**Drawback:**
To detect the transmitted information, a high-resolution clock generator is required. This creates costs in terms of latency and power consumption.

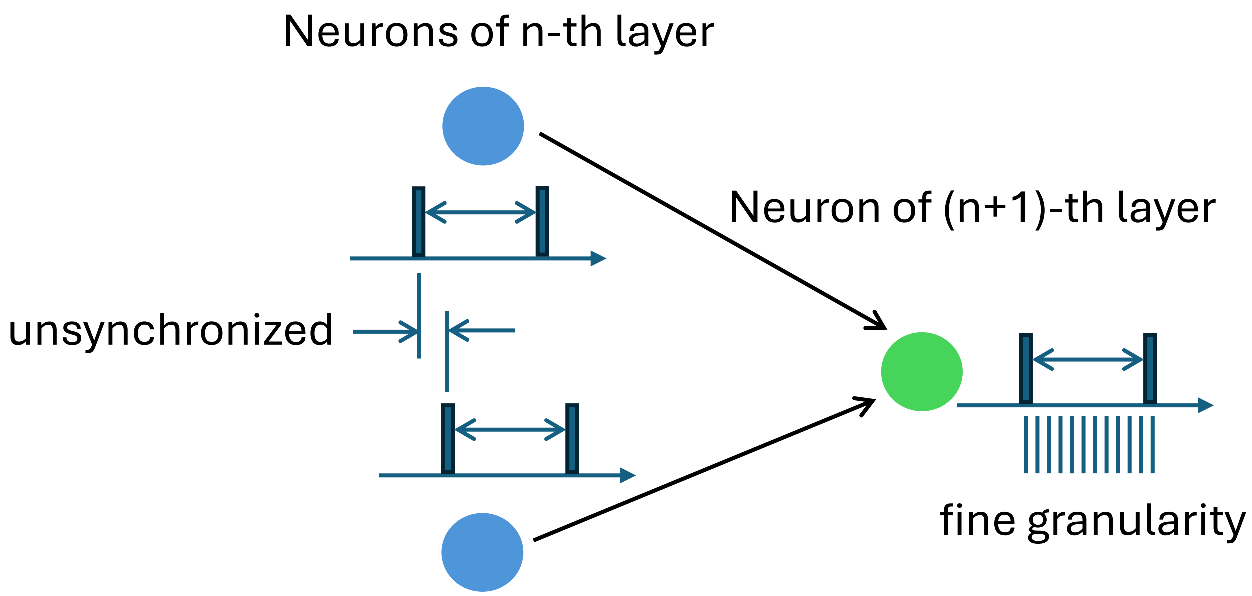


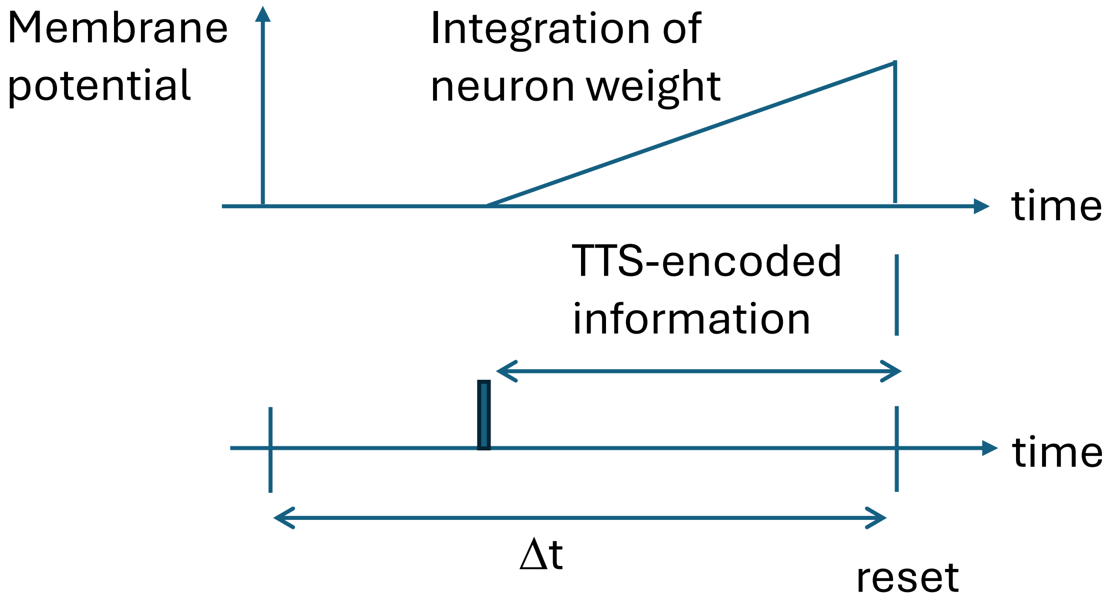

**Fig. 11**: Inter-spike interval encoding does not require the synchronization of neurons within a layer and requires high-resolution clock sources to measure the length of the time interval between two adjacent spikes.

[Back to Tutorial]

# Q&A 2.2.2 Time-to-Spike Encoding (TTS)

Answer to L1: The purpose of the observation interval at time-to-spike encoding is twofold: a) its length defines the code range, i.e., the number of time ticks covered, and b) the end of the observation interval defines the evaluation and reset of the membrane potential.

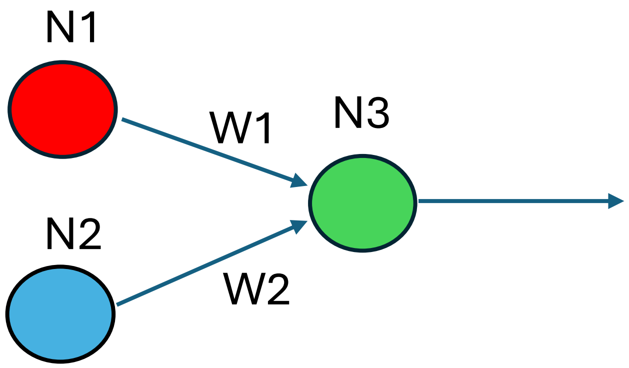
Answer to L2: The decoding of a time-to-spike encoded signal in a neuron receiver is performed by integrating the neuron weight in the time interval between the occurrence of the incoming spike and the end of the observation interval as illustrated below.



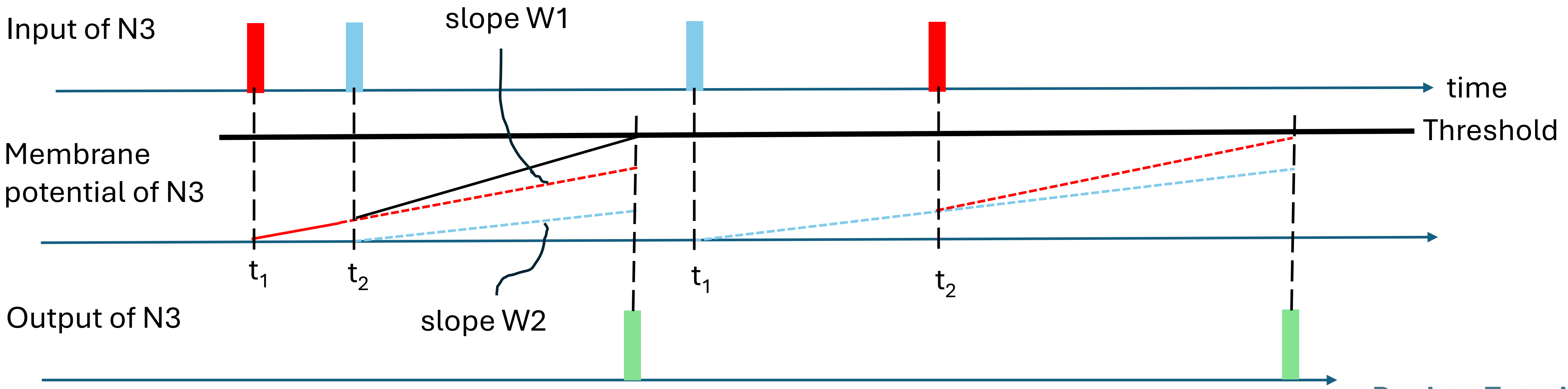

Back to Tutorial

# Q&A 2.2.3 Time-to-First-Spike Encoding (TTFS)

Answer to L2:




- N1 sends out a spike at $t_1$ that is integrated with a slope according to the weight W1. See red curve.
- N2 sends out a spike at $t_2$ that is integrated with a slope according to the weight W2. See blue curve.
- The integration ramps are superimposed. As soon as the threshold of the membrane potential is reached, N3 sends out a spike. See green spikes.



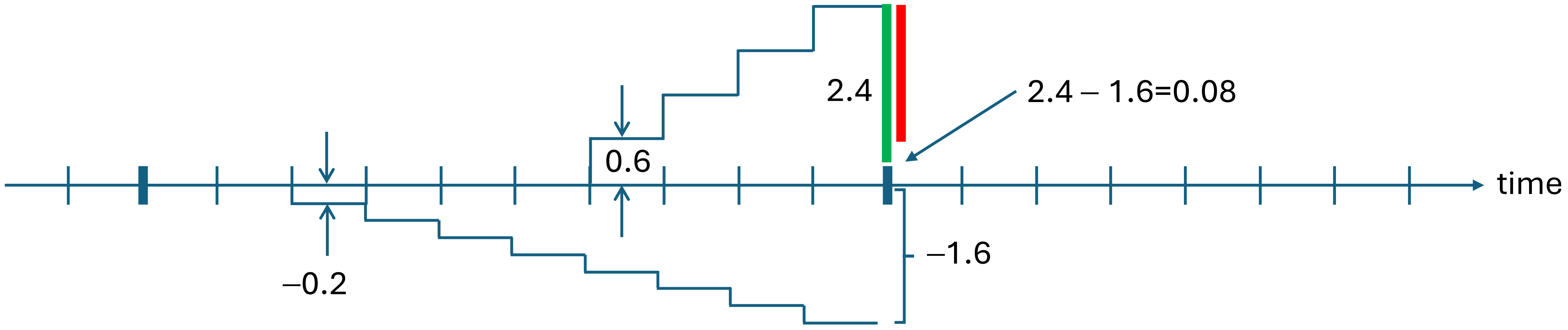

**Back to Tutorial**

# Q&A 2.3 Data Processing within Neuron

Answer to L1: Assumptions: $x_1$=0.4, $w_1$=0.6
$x_2$=0.8, $w_2$=–0.2

The data ($x_1$, $x_2$) must be positive.
The weights ($w_1$, $w_2$) can be positive or negative.

- ANN MAC: Sum of partial products = $x_1w_1 + x_2w_2 = 0.4 \cdot 0.6 + 0.8 \cdot -0.2 = 0.08$
- SNN Integrator: The data of e.g., $x_1$=0.4 is interpreted as clocking the integrator with 4 pulses so that the weight of $w_1$=0.6 is iteratively summed up 4 times.




Back to Tutorial

# Q&A 2.4 TTS Transmission Example

Answer to L1: The membrane potential $V_{m2}$ at the end of the observation period $T_2$ (i.e., the second observation interval in the drawing) is negative because $w_1x_1 < w_2x_2$. The ReLu-operation sets the negative value to zero. Hence, no $t_2$-spike is generated.

Answer to L2: Note that the higher the membrane potential $V_m$ is, the more important the outgoing spike becomes and the earlier it must be transmitted so that it has a higher value during the integration at the successive neuron. Thus, the value of $(T-t_i)$ is **proportional** to $V_m$.
This can be implemented by means of a **down-counter**.

Example: The length of the observation interval is 16. If $V_m$=15, $t_i$ equals 1. Hence, N3 spikes at $t_i$=1. If the down-counter starts counting at t=0 with the value of 16, it reaches $V_m$=15 already at t=1 and N3 sends out a spike.
A thorough understanding of this working principle is essentinal to read the vhdl code of the TTS implementation, which will be presented later in this tutorial.

Back to Tutorial

# Q&A 3 Iris Dataset (2)

Answer to L1: The scatter matrix indicates that the labelling of *Iris-setosa* is easiest to classify because many attributes (e.g., petal width and petal length) of *Iris-setosa* are rather different from those of *Iris-versicolor* and *Iris-virginica*. The complexity is higher to distinguish between *Iris-versicolor* and *Iris-virginica*. Hence, the classification accuracy of *Iris-setosa* will be higher than that of the two other species.

**Back to Tutorial**

# Q&A 3.1 Network topologies for Iris dataset classification

Answer to L2:

The objective is to design a network as small as possible but at the same time as accurate as possible. Network size and accuracy can be opposite, hence, priorities in terms of hardware implementation must be set. The two networks can be characterized as follows:

(4-10-3)-network:

- Number of neurons: **17** (3 input layer neurons, 10 hidden layer neurons, 4 output layer neurons)
- Number of neuron transmitters/receivers: 14 neuron transmitters (4 input TX, 10 hidden TX)
  13 neuron receivers (10 hidden RX, 3 output RX)
- Number of synapses: **70** (=4x10+10x3)
- Latency: **128 $T_{cycle}$** (for SNN with synchronous TTS scheme and 6b resolution)

(4-5-3-3)-network:

- Number of neurons: **15** (3 input layer neurons, 8 hidden layer neurons, 4 output layer neurons)
- Number of neuron transmitters/receivers: 12 neuron transmitters (4 input TX, 8 hidden TX)
  11 neuron receivers (8 hidden RX, 3 output RX)
- Number of synapses: **44** (=4x5+5x3+3x3)
- Latency: **192 $T_{cycle}$** (for SNN with synchronous TTS scheme and 6b resolution)

Back to Tutorial

# Q&A 3.2 ANN Neuron Model for Hardware Implementation

Answer to L1:

The receiver is highlighted in yellow. First, the input features $x_0$ to $x_k$ (i.e., petal/sepal widths/lengths) are latched. They have a 6b resolution. Next the product $w_i x_i$ is calculated, which has 12b resolution (including the sign bit).

In a next step highlighted in red the partial products $w_i x_i$ are summed up. In addition, the latched bias $b_j$ is added to the sum of the products. The multiplications and additions are performed with a 12b resolution. The further processing though is using 6b. Hence the last step of the highlighted red part includes the addition of a half value according to the round half up method for the rounding to 6b.

The neuron transmitter is highlighted in green. The signal first goes through a limiter that implements the ReLu function (i.e., negative values are set to 0 and values higher than a maximum value are set to a predefined maximum value). The successive quantizer cuts off the first 6 bits (note that this operation is valid because of the round half up method previously applied). The quantized value is latched to make it accessible for the next neuron.

The neuron depicted applies to the hidden layer where a neuron receiver and a neuron transmitter are required. For the input layer only the latching part of the transmitter is used. For the output layer the neuron receiver is employed with an additional softmax function.

Back to Tutorial

# Q&A 3.3 TTS Neuron Model (synchronous spiking)

Answer to L1:

In the ANN the information to be transmitted is encoded in the code domain. In the SNN the information is encoded in the time domain by means of spikes. Hence the receiver is implemented as a switch that feeds the weight $w_i$ to the output as soon as a spike is detected; otherwise, a zero occurs at the output.

Similarly to the ANN the transmitter outputs are summed up. However, as opposed to the ANN in which the transmitter performs the multiplication $w_i x_i$, the multiplication in the SNN is performed by means of an integration of the summation of the receiver outputs. This is done by a register whose value is fed back and added to the input via a ripple counter. Analogously to the ANN a bias $b_j$ is added to the integrator output and the sum goes to a quantizer to generate a 6b value.

The transmitter needs to perform a time encoding of the quantized integrator value. This is performed by a down-counter whose value is compared to the quantized integrator output. When they are equal a spike is generated. A down-counter instead of an up-counter is used because of the $(T-t_i)$-dependency, i.e., the higher the quantized integrator output is, the earlier the spike needs to be sent out. This relationship is also illustrated here.

As opposed to the ANN, the SNN integrator and switches need to be reset periodically at the end of the observation interval T.

Back to Tutorial

# Q&A 3.4 TTFS Neuron Model (asynchronous spiking)

Answer to L1:

In TTFS signaling a spike is sent out when the membrane potential reaches a threshold value whereas in TTS signaling the membrane potential is evaluated at the end of the observation interval and the timing of the spike to be sent out is proportional to the value of the membrane potential, i.e., $V_m$ is proportional to $(T-t_i)$, see here. TTS is a synchronous signaling scheme in which a reset signal is generated at the end of the observation interval. TTFS is an asynchronous signaling scheme that generates the reset signal when the threshold value is reached, and a new spike is sent out.

The synaptic receiver and the integrator are the same in TTS and TTFS signaling. TTFS does not have the addition of a bias value. The neuronal transmitter of TTFS differs from the TTS implementation in two aspects:

a) The comparator compares the integrator value (=membrane potential) to a fixed threshold in TTFS.
b) The comparator output is hold by a 1-bit register that controls the pulse generator that sends out the spikes.

Back to Tutorial

# Q&A 4.1 Floating-point Training and Inference of ANN (1)

Answer to L1: The program code is structured into the following parts:

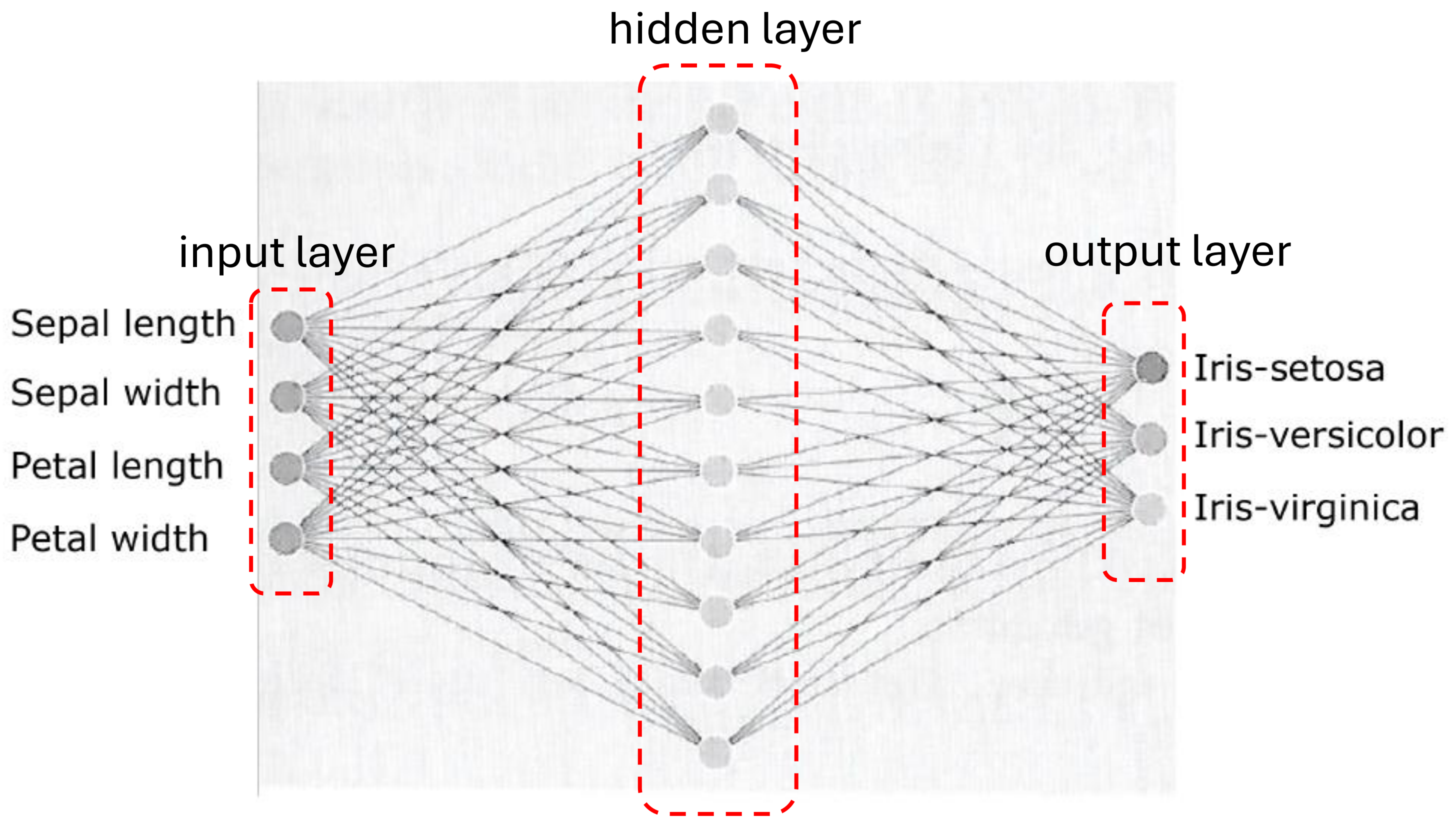
- import the Iris flower dataset
- replace the text labels by numbers and shuffle the data
- define the neural network model
- perform the training with 80% of the data
- perform inference based on the trained weights and biases with 20% of the data
- report the accuracy and generate the convolution matrix

Answer to L2: The class ;Iris-setosa’ yields 100% accuracy. This was to be expected by the scatter matrix.

Back to Tutorial

# Q&A 4.1 Floating-point Training and Inference of ANN (2)

Answer to L2: The following network is modeled by the code: (4,10,3) – input layer with 4 neurons, hidden layer with 10 neurons and output layer with 3 neurons




Back to Tutorial

# Q&A 4.1 Floating-point Training and Inference of ANN (3)

Answer to L3:

The line

```
ann_model.add(Dense(10, activation='relu')) #, use_bias=False, fully connected hidden layer of 10 neurons
```

needs to be replaced by

```
ann_model.add(Dense(5, activation='relu')) #, use_bias=False, fully connected hidden layer of 5 neurons
ann_model.add(Dense(3, activation='relu')) #, use_bias=False, fully connected hidden layer of 3 neurons
```

and the following lines need to be added

```
np.savetxt("Weights_Biases/ANN_weights_hiddenLayer_2.csv", ann_weights[2], delimiter=",")
np.savetxt("Weights_Biases/ANN_biases_hiddenLayer_2.csv", ann_weights[3], delimiter=",")
```

The accuracy of the network (4,5,3,3) is 97.3% whereas the network (4,10,3) has an accuracy of 98%.
Note that the calculated accuracy numbers slightly vary depending on the initialization (data shuffling).

Back to Tutorial

# Q&A 4.3 Quantization-Aware Training of ANN

Answer to L3:

Load the Iris flower dataset (4 features, 1 label per row)

↓

Load the trained floating-point data for the weights and biases

↓

Define the list of resolutions for the quantized weights and biases according to the format (total number of bits, number of fractional bits)

↓

Loop through the list of resolution definitions by
- quantizing the weights and biases accordingly
- building the model (4-10-3) based on calling the sub-routines ANNModel.py
- performing the inference and determining the accuracy

This step may take a long time (~ 1 hour)

↓

Generate a plot with the heatmap of the accuracies obtained from the quantized inference

Back to Tutorial

# Q&A 5.1 VHDL-Implementation of TTS SNN (1)

Answer to L1:

4 1 0.25

Example of (6,2)-quantization of feature data: "010111“ corresponds to 5.75cm

sign 2 0.5

0.25

1 0.0625

Example of (6,4)-quantization of weight data: " 110111“ corresponds to -0.5625

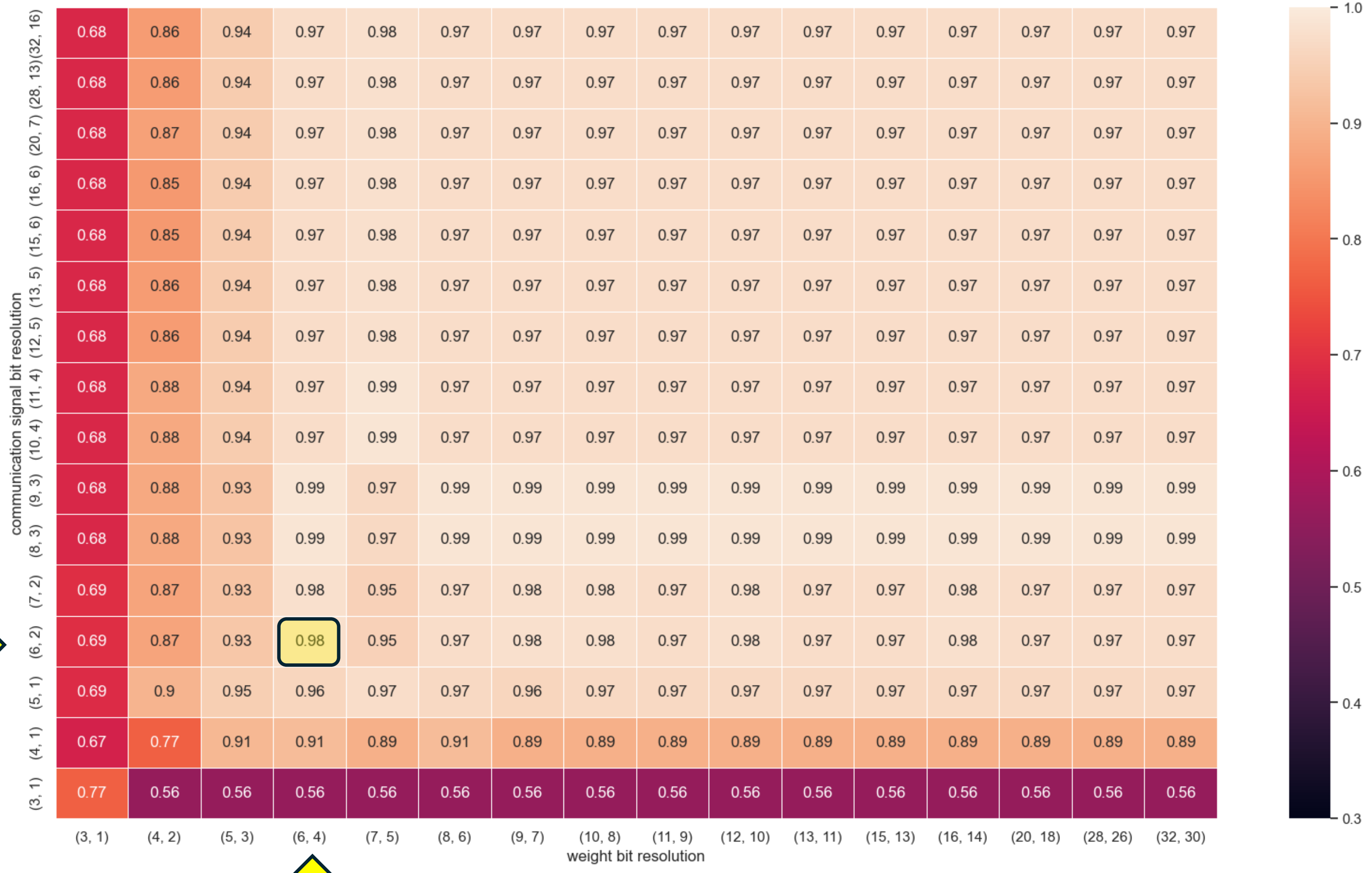
sign 0.125

0.5

**Back to Tutorial**

# Q&A 5.1 VHDL-Implementation of TTS SNN (2)

Answer to L2:

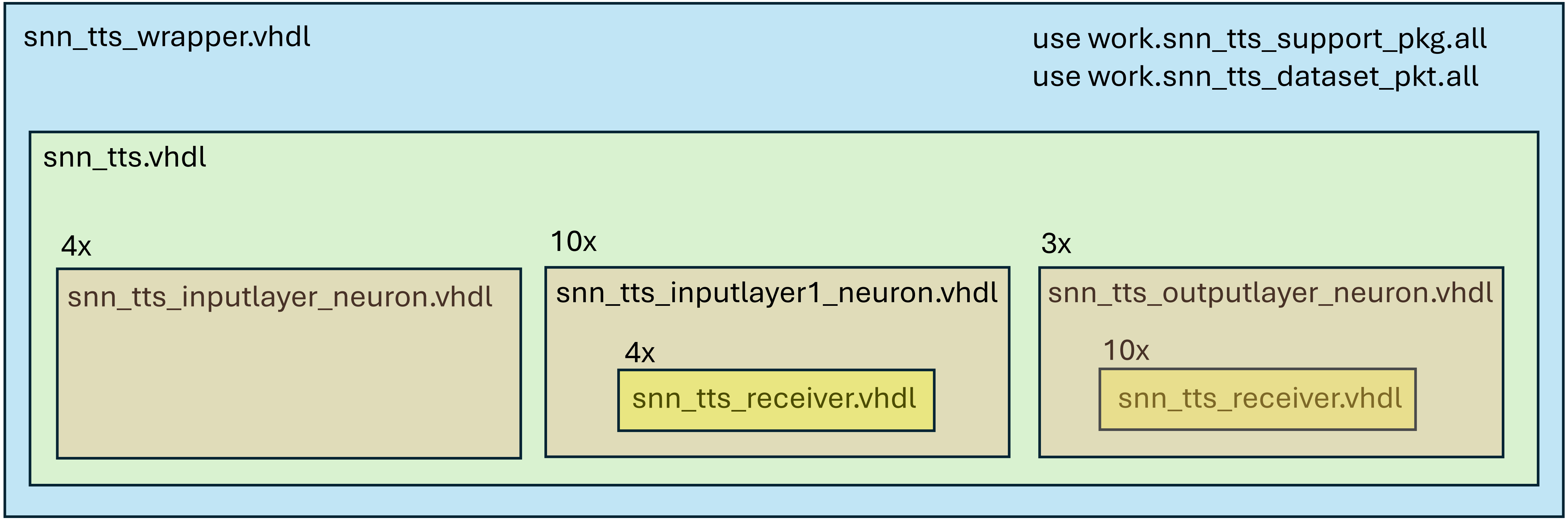



98% accuracy can be achieved with (6,4) and (6,2) resolution

Back to Tutorial

# Q&A 5.1 VHDL-Implementation of TTS SNN (3)

Answer to L3:




[Back to Tutorial]

# References

[1] “Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook 1: Cognitive Domain“, published by David McKay Company, Inc., in 1956. It is often referred to as “Bloom’s Taxonomy” and was authored by a committee of educators chaired by Benjamin Bloom.

[2] S. Widmer *et al.*, "Design of Time-Encoded Spiking Neural Networks in 7-nm CMOS Technology," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 9, pp. 3639-3643, Sept. 2023.

[3] E. Adrian and Y. Zotterman. “The impulses produced by sensory nerve-endings”. In: J. Physiol. 61.2 (Apr. 1926), pp. 151–171.

[4] Simon Thorpe, Denis Fize, and Catherine Marlot. “Speed of Processing in the Human Visual System”. In: Nature 381 (July 1996), pp. 520–2. doi: 10.1038/381520a0.

[5] R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems". Annals of Eugenics. 7 (2): 179–188.

[6] Edgar Anderson (1936). "The species problem in Iris". Annals of the Missouri Botanical Garden. 23 (3): 457–509.

# Acknowledgement

The TensorFlow and vhdl code presented in this tutorial has been written by Sandro Widmer during his Master Thesis at IBM Research Zurich in 2022.

# Glossary and Acronyms

ANN: Artificial neural network

ASIC: Application specific integrated circuit

CSA: Carry save adder

MAC: Multiply and accumulate

MPW: Multi-project wafer

RLM: Random logic macro (IBM's terminology for RTL)

SNN: Spiking neural network

TF: TensorFlow

VHDL: Very High-Speed Integrated Circuit Hardware Description Language