README

For

“Building Brains with   
ARM processors and FPGAs” PROJECT

By

Felipe Galindo Sanchez

September 2016

# Environments

The source code included is able to implement the spiking neural network using Izhikevich’s model in four different environments:

|  |  |  |
| --- | --- | --- |
| **Environment** | **Description** | **IDE** |
| **Software** | Traditional implementation in C | Any software IDE (e.g Eclipse, Visual Studio, Code Blocks) |
| **HLS** | High-level-synthesis implementation to be executed in Xilinx FPGA devices | Vivado HLS |
| **Zynq** | Implementation containing the drivers required to execute the synthetized version in a Zynq 7000 device | Xlinix SDK |
| **OpenCL** | Version using OpenCL in order to be executed with e.g. GPU and GFX card | Visual Studio or Eclipse |

# The package

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File structure** | **Software** | **Vivado HLS** | **Xlinix SDK (Zynq)** | **OpenCL/GPU** |
| **src:** *Main**source code repository for SNN implementation including for SW, HLS and ZYNQ env* | | | | |
| * **main\_sw.cpp:** *Entry point for a software-only simulation (e.g. Eclipse)* |  |  |  |  |
| * **main\_hls.cpp:** *Entry point for Vivado HLS project* |  |  |  |  |
| * **main\_zynq.cpp:** Entry point for Xlinix SDK project (using Zynq board) |  |  |  |  |
| * **common/** | | | | |
| * **snn\_defs.h:** *Common definitions* |  |  |  |  |
| * **snn\_env.h:** *Definitions for different environments* |  |  |  |  |
| * **snn\_network.h:** *Network/Neuron model specific definitions* |  |  |  |  |
| * **snn\_results.h:** *Methods for saving results into csv* |  |  |  |  |
| * **snn\_start.h:** *Generic entry point “main” for all environments* |  |  |  |  |
| * **snn\_types.h:** *Type definitions* |  |  |  |  |
| * **sw/** | | | | |
| * **snn\_izikevich\_sw.h:** *SNN implementation (software only)* |  |  |  |  |
| * **hw/** | | | | |
| * **snn\_izikevich\_top.cpp:** *SNN top module to be synthetized* |  |  |  |  |
| * **snn\_izikevich.h:** *Processing blocks/methods of SNN* |  |  |  |  |
| * **snn\_izikevich\_axi.h:** *Helper methods for AXI protocol* |  |  |  |  |
| * **snn\_izikevich\_hw\_sim.h:** *Wrapper for simulating HW algorithm* |  |  |  |  |
| * **snn\_izikevich\_hw\_zynq.h:** *Wrapper for executing HW algorithm* |  |  |  |  |
| * **networks/** | | | | |
| * **snn\_network\_defs.h:** |  |  |  |  |
| * **snn\_network\_random.h:** *Random network implementation* |  |  |  |  |
| * **snn\_network\_single.h:** *Network following a specific frequency* |  |  |  |  |
| * **snn\_network\_xor.h:** *Network learning a XOR gate* |  |  |  |  |
| **FeedForwardSpikingneuralNet:** *Source code for OpenCL version* | | | | |
| * **main.cpp:** Entry point for OpenCL version |  |  |  |  |
| * **CL.cpp:** *Drivers for using OpenCL and device required* |  |  |  |  |
| * **CL.h:** *Header file of CL.cpp* |  |  |  |  |
| * **Settings.h:** *Network/Neuron model specific definitions* |  |  |  |  |
| * **kernels.cl:** *Definition of the two processing blocks as kernels* |  |  |  |  |
| **vivado\_hls:** *Vivado HLS Project* files | | | | |
| **vivado\_ip:** *Vivado 2015 (IP Integrator) project files (including Xlinix SDK files)* | | | | |
| **docs:** *Documents supporting the project* | | | | |
| * **README.pdf:** The current document with BKM and How-To | | | | |
| * **Poster.pdf:** Poster version in PDF | | | | |
| * **Thesis.pdf:** Thesis version in PDF | | | | |
| * **source/** | | | | |
| * **README.pdf:** The current Word document with BKM and How-To | | | | |
| * **Poster.pptx:** Poster version for Power Point | | | | |
| * **Thesis.docx:** Poster version for Microsoft Word | | | | |
| * **Figures.xlsx:** Tables and charts created for the thesis and poster documents | | | | |

# FPGA Implementation in a Zynq device

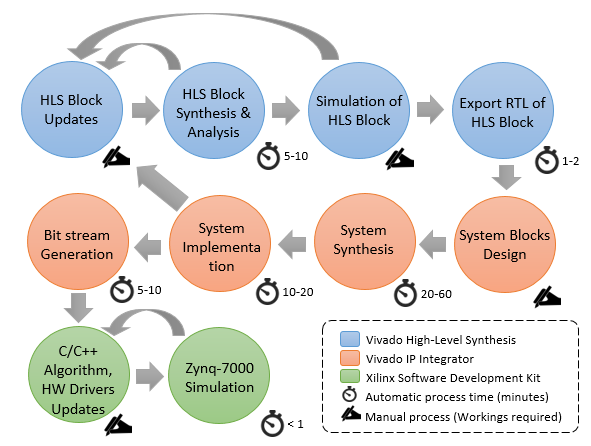
The full workflow for implementing the SNN in a Zynq device involves three tools:

* **Vivado HLS**
* **Vivado IP Integrator**
* **Xlinix SDK**

The general steps are the following:

1. Open **Vivado HLS project** from **vivado\_hls.**
2. HLS need to be synthetized using **Vivado HLS** and the **main\_hls.cpp** entry point
3. Hardware version can be simulated in the same Vivado HLS tool
4. Synthetized version need to be exported to RTL
5. Open **Vivado IP project** from **vivado\_ip**
6. Interconnections between the Zynq device and the exported RTL can be modified in **Vivado (IP Integrator) tool**
7. Synthesis, Implementation and generation of **bit stream** need to be executed in **Vivado tool**
8. Export **bit stream** by goingto **File > Export Hardware**
9. Open **Xlinix SDK** project by selecting **File > Launch SDK** in Vivado IP
10. Implementation can be executed in **Xlinix SDK** with the **exported bit stream** and using the **main\_zynq.cpp** entry point

An overall diagram can be expressed in the following figure:



The precision type can be defined in **src/snn\_config.h** with the **PRECISION\_TYPE** definition as **FLOATING\_POINT** or **FIXED\_POINT.**

The network size to be synthetized can be defined in **src/snn\_config.h** with the **NETWORK\_SIZE** definition.

# Applications

The application to be executed can be defined in **src/snn\_config.h** with the **APP\_TYPE** definition.

Creating a new application:

1. A new file needs to be generated under **src/networks**
2. The **definition for that app** need to be defined in **src/common/snn\_env.h**
3. The created file need to be included along with the other applications in **src/common/snn\_start.h**
4. The created file needs to implement the common application methods:
   1. **uint1\_sw\_t get\_neuron\_type(int32\_t l, int32\_t xl):** Specify if a neuron is exhibitory or inhibitory, with the parameters being the **layer (l)** and the neuron index **(xl)** in the layer
   2. **uint1\_sw\_t get\_spike(int32\_t t, int32\_t x):** Specify if the synaptic input with **index (x)** at a **time (t)** has a spike or not.
   3. **float32\_t get\_weight(int32\_t l, int32\_t xl, int32\_t x, int32\_t y, uint1\_sw\_t feedback):** Specify the **synapses weight** with **index (y)** in the **neuron** in **layer (l)** and the neuron index **(xl)** in the layer. (x) is the index of the neuron along all the layers and feedback indicates if it is after a training iteration or not (initial weight).
   4. **void generate\_inputs():** Offline generation of all inputs over all training iterations
   5. **void persist\_app\_results():** Custom generation of results after the training is completed.
5. **Definitions** of a specific app are encapsulated inside the **src/networks/snn\_network\_defs.h** with its corresponding “if APP\_TYPE == NEW\_APP”

Random network application:

Configuration of the random network can be made in the file **src/networks/snn\_network\_defs.h**

* The size of the network can be modified with **NETWORK\_SIZE**
* The percentage of neurons interconnected can be configured from0.0 to 1.0 with **INTER\_CONNECTION\_PROBABILITY**
* The probability percentage of synaptic inputs can be configured from0.0 to 1.0 with **INPUT\_SYNAPSE\_PROBABILITY**
* The initial weights for synaptic inputs can be configured with **INPUT\_SYNAPSE\_WEIGHT**
* The random weight for interconnection weights can be configured with **SYNAPSE\_WEIGHT**

The **number of synapses** per neuron is **proportionally** to the **network size**, consequently, the input synaptic for each neuron.

* For **small network sizes** (e.g. < 70), the synaptic input may not be enough to produce action potentials, thus **INTER\_CONNECTION\_PROBABILITY** and **SNAPSE\_WEIGHT** may need to be increased.
* For **large network sizes** (e.g. >150), the synaptic input may be high and it may produce high spiking neurons. Thus **INTER\_CONNECTION\_PROBABILITY** and **SNAPSE\_WEIGHT** may be decreased for achieving a regular spiking behavior.

Single application (Firing rate follower):

Configuration of the random network can be made in the file **src/networks/snn\_network\_defs.h**

* The number of trials or training iterations can be configured with **NUM\_TRAINING\_TRIALS**
* Learning rate and STD parameters can be configured with **ALPHA\_PLUS, ALPHA\_MINUS, TAU\_PLUS, TAU\_MINUS, LEARNING\_RATE**
* The number of neurons in the hidden layer can be configured with **SIZE\_NEURONS\_PER\_LAYER**
* The percentage of inhibitory neurons can be configured from 0.0 to 1.0 with **INHIBITORY\_NEURON\_PERC**
* The duration of a trial/iteration is defined by **TRIAL\_TIME\_MS**
* The input and output target frequency to be followed can be defined with **INPUT\_SPIKES, OUTPUT\_SPIKES, INPUT\_FREQ** and **OUTPUT\_FREQ** where **INPUT\_SPIKES/OUTPUT\_SPIKES** is the number of spikes in a trial/iteration time frame.

Further analysis of results/trainings can be done for each iteration inside **void feedback\_error(int32\_t t)** in **src/networks/snn\_network\_single.h**

XOR application:

Configuration of the random network can be made in the file **src/networks/snn\_network\_defs.h**

* The number of trials or training iterations can be configured with **NUM\_TRAINING\_TRIALS**
* Learning rate and STD parameters can be configured with **ALPHA\_PLUS, ALPHA\_MINUS, TAU\_PLUS, TAU\_MINUS, LEARNING\_RATE(progress)**
* The number of neurons in the hidden layer can be configured with **SIZE\_NEURONS\_PER\_LAYER**
* The percentage of inhibitory neurons can be configured from 0.0 to 1.0 with **INHIBITORY\_NEURON\_PERC**
* The duration of a trial/iteration is defined by **TRIAL\_TIME\_MS**
* The encoded delays for each input and output is defined with **DELAY\_INPUT\_LOW\_MS, DELAY\_INPUT\_HIGH\_MS, DELAY\_OUTPUT\_LOW\_MS, DELAY\_OUTPUT\_HIGH\_MS**

Further analysis of results/trainings can be done for each iteration inside **void feedback\_error(int32\_t t)** in **src/networks/snn\_network\_xor.h**