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

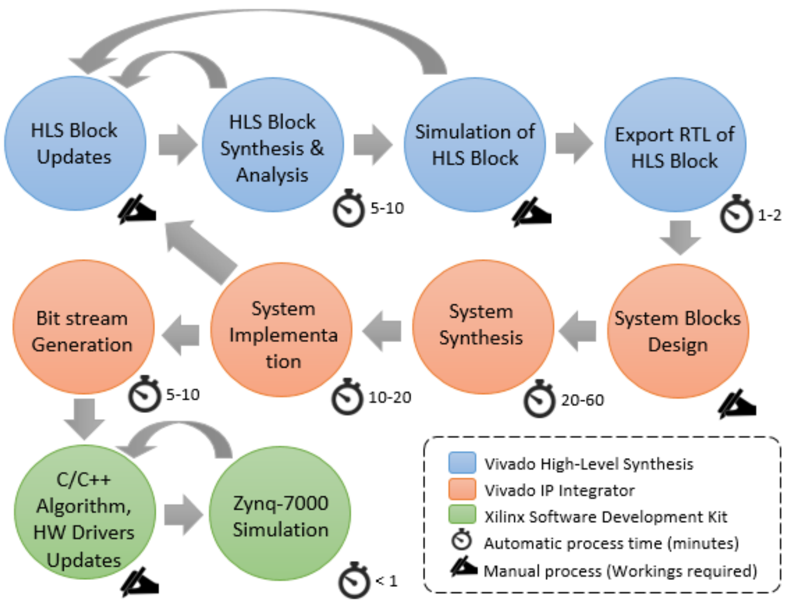
The full workflow for implementing the SNN in a Zynq device involves three tools:

- **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 going to **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.

<u>Creating a new application:</u>

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:
   a. **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
   b. **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.
   c. **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).
   d. **void generate_inputs():** Offline generation of all inputs over all training iterations
   e. **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"

<u>Random network application:</u>

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 from 0.0 to 1.0 with **INTER_CONNECTION_PROBABILITY**
– The probability percentage of synaptic inputs can be configured from 0.0 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**