Adam SOKOLNICKI[1], Giovanny SANCHEZ[2], Jordi MADRENAS[2], Manuel MORENO[2]
Bartosz SAKOWICZ[1]

Technical University of Lodz(1), Universitat Politecnica de Catalunya (2)

Graphical representation of data for a multiprocessor array emulating spiking neural networks

Abstract. *It is crucial for any hardware platform emulating neural networks to have tools that give valuable insight to emulated network. A solution that adresses these needs for the specific multiprocessor array used in the Perplexus project [1] will presented in this paper. It consists of a software that exploits interface between the platform and outside world. It instructs and fetches the needed data from the device. The precise data source is defined by user-defined JavaScripts' expressions that are evaluated by the program and presented simultaneously using waveform, histogram and raster plots in real time.*

Streszczenie. *Dla każdej sprzętowej platformy emulującej sieci neuronowe bardzo istotne jest aby posiadała ona narzędzia pozwalające monitorować jej stan. Oprogramowanie które rozwiązuje ten problem dla dedykowanej wieloprocesorowej platformy stosowanej w projekcie Perplexus [1] zostanie zaprezentowane w tym artykule. Wykorzystuje ono interfejs platformy do komunikacji ze światem zewnętrznym, wykonuje kod oraz pobiera dane z urządzenia. Dokładne położenie danych jest definiowane przez wyrażenia JavaScript pisane przez użytkownika, które następnie zostają wykonane przez program. Dane mogą zostać zaprezentowane równolegle na wykresach falowych, histogramowych oraz rastrowych. (**Graficzna reprezentacja danych dla wieloprocesorowej płyty emulującej impulsywną sieć neuronową**)*

Keywords: Spiking neural network, Simulation, Visualizaton, Embeddable script engine, Hardware Emulation.
Słowa kluczowe: Impulsywna sieć neuronowa, Symulacja, Wizualizacja, Osadzalny silnik skryptowy, Sprzętowa emulacja

Perplexus project

A main objective of the European Comission-funded Perplexus project was to develop the so-called Ubichip - an intergrated circuit endowed with bio-inspired capabilities. This digital chip has been designed to simulate complex systems. Among other features and applications, the ubichip operating as a SIMD (Single-Instruction Muliple-Data) multiprocessor is able to emulate large-scale spiking neural network models [1].

The building blocks of Ubichip operating in the SIMD multiprocessor mode are shown in figure 1. The multiprocessor architecture has been designed taking into account the specific characteristics of this kind of networks, in particular the binary spike characteristic, the large number of interconnects and the low frequency of spikes. The core element is the Processing Element (PE) array. Each of the PEs consists of a simple 16-bit ALU and two 8-register banks. Each PE can emulate a single spiking neuron and all its incoming synapses. The sequencer governs the PE execution algorithm and it fetches the data and instructions from SRAM to the array. It means that implementation of the network's model is stored in SRAM that is the variables mapped to each neuron and the algorithm of the computation. While emulating SNN Ubichip works in Single Instruction Multiple Data (SIMD) mode - the same instructions are executed by all of the PEs but with different data [2]. The instruction set of the multiprocessor has been optimized to allow the efficient emulation of SNN, being general enough to permit the programming of different SNN models [3], [4]. The issue of spike distribution among neurons has been addressed by means of a special-purpose AER (Address Event Representation) bus [5] and a CAM (Content-Addressable Memory) that detects the postsynaptic spikes and generates the presynaptic spikes.

The ubichip is embedded in the so-called Ubidule [1], a board whose purpose is to make communication and controlling of Ubichip possible. It is equipped with a Colibri board [6] which includes a high-end XScale embedded processor. In the present application, the processor runs GNU/Linux operating system. The connection between Ubidule and outside world can be established using USB or Ethernet.

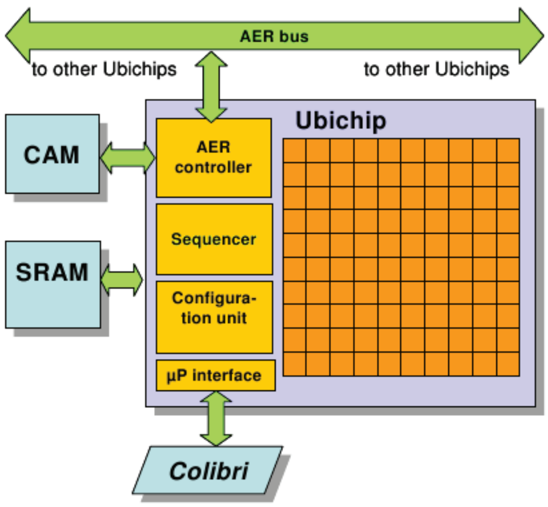


Fig. 1. Ubichip block diagram.

The workflow

The chip operation phases are presented in figure 2. There are two main phases. In the first one, the sequencer is enabled and normal execution of the code that emulates neurons and synapses is taking place. After it, the sequencer is disabled and control is given to the AER bus/CAM controller [5]. In this phase, the spikes are propagated. The controller scans the PEs in search of fired neurons. When such is found it uses CAM (Content Adressable Memory) to find the connected neurons and apropriate variables in the memory are updated. After the second phase there is HALT instruction in the program, so the sequencer stops. A software that needs to fetch data can check if the chip is in HALTed state by polling technique. In case it is it can read the data and resume the execution. The software presented in this paper is accessing the data in this manner.

Spiking neural networks

As previously indicated, the type of the neural networks targeted by the chip are spiking neural networks. SNN are realistic models of biological neurons. One of their main characteristics is that every membrane voltage of the neuron has a fixed treshold. When the value of the membrane potential reaches the threshold, the neuron fires a spike. This spike

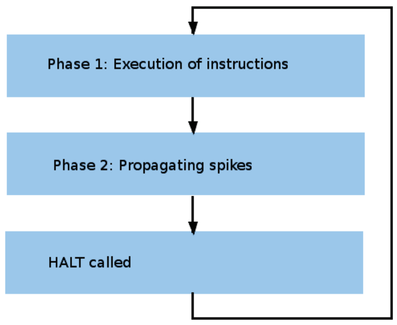

Phase 1: Execution of instructions
Phase 2: Propagating spikes
HALT called

Fig. 2. Workflow of Ubichip.

propagates via synapses to all connected neurons altering their membrane potential [7]. The neuron's spike is depicted in figure 3.

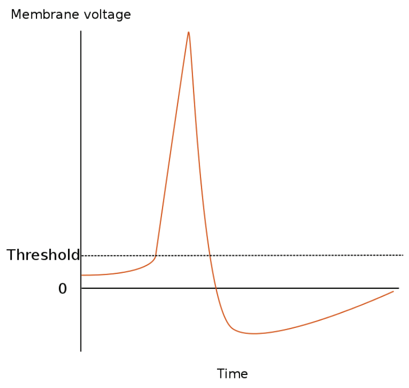


Fig. 3. Neuron's spike. Based on [8]

Objectives

In the initial state of the development the objectives for the desired application were defined as follows:

- enable communication with Ubichip from a desktop PC,
- enable controlling of the execution of the emulation and fetching results from the platform via this channel,
- enable visualising the results of the emulation through waveform, histogram and raster plots in real time.

In order to achieve stated objectives two possible solutions were taken under the consideration. One of them was to write standalone server for the Ubichip and application client for desktop PC. The other one was to exploit the existing interface that is used for configuration and execution of the emulation from the PC by means of the Ethernet connection.

The advantage of the first solution is that it gives more freedom when it comes to possible communication channels. The programmer can make use of Ethernet and USB interfaces, where in the second solution the transmission is taking place through Ethernet only.

On other hand usage of the existing interface that is constantly updated with the hardware changes makes the application more flexible and guarantees its effectiveness in the future. It also cuts the devolopment time since the needed software on the Ubichip end point is already created and working. Those were the key factors for choosing this solution over another.

Software base

Among created software supporting programming and communication with the Ubichip, there are three applications that should be mentioned.

- SpiNDeK - generates assembly with implementation of Iglesias-Villa [12] model of spiking neural network for Ubichip.[9]
- UbiColibri - TCP/IP server that runs on Ubidule and listens for incoming connections from Ubimanager. It is using Ethernet as a transmission channel.[10]
- Ubimanager - desktop application for configuring and controlling the execution on Ubichip.[11]

Ubimanager and UbiColibri constitutes mentioned in the previous section interface between PC and the platform. One of the important features of Ubimanager is that it has got a plug-in facility so that 3rd party can create plug-ins that can make use of this interface. That is how the final decision was taken to write Ubimanager's plug-in later named as Ubiplot. This decision was also dictated by the will to keep overall software in Perplexus project consistent and to match the way other software applications were written for Ubichip. Ubimanager was enchanced with plug-in facility for more or less this kind of tasks [11].

Ubiplot

It is a plug-in for Ubimanger whose task is to control the emulation on the Ubichip and display the data stored in its SRAM using waveform, histogram and raster plots.

Because Ubimanager had been written using Qt framework Ubiplot had to be created with it as well. The used language is C++.

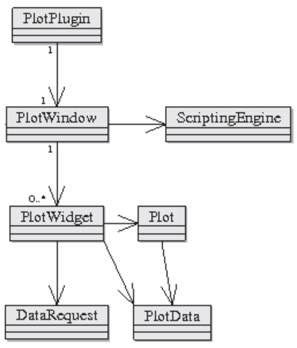


Fig. 4. Collaboration of the main classes of Ubiplot.

The collaboration of the main classes of Ubiplot is depicted in figure 4. It tries to be simple and obvious. By following this pattern extending plug-in with new kind of plots should be a relatively easy task. Every class accomplishes some elementary task such as requesting and storing data, plotting, supplying user interface etc.

The object of class *PlotPlugin* is created by Ubimanager itself when it is loading the plug-in. It inherits from the *UbiSimplePlugin* - class created by Ubimanager's development team for plug-ins to derive from. It supplies methods for remotely controlling the execution and accessing the Ubichip SRAM. *PlotPlugin* also spawns *PlotWindow*. This is the main window and the user interface, it holds also *PlotWidget* objects which encapsulates single plot. The *DataRequest* objects hold all the needed information for accessing the SRAM via calls to the methods of Ubimanagers's *SimplePluginInterface*. The *PlotData* objects store the incoming data and *Plot* objects are using it to draw the actual plot.

Every supported kind of the plot has got its classes derived from *PlotWidget*, *Plot* and *PlotData* parents. The classes that inherit from *PlotData* are template classes. The reason behind this is that the incoming data can be treated as coded in simple or two's complement so the datatypes of the templates are *unsigned int* or *int* respectively.

Variables' map

The main window of the plug-in has got a "Variables" menu. By this menu, the user can create a plot for observing specific variables of the model. The position of the current value of the variable in the memory depends on the order generated by SpiNDeK assembly code [9]. The example map for 100 neurons and 300 synapses [9] can be seen in figure 5. This mapping is applied to a particular SNN model proposed by Iglesias and Villa [12], although, as indicated before, the ubichip permits the implementation of any SNN model.

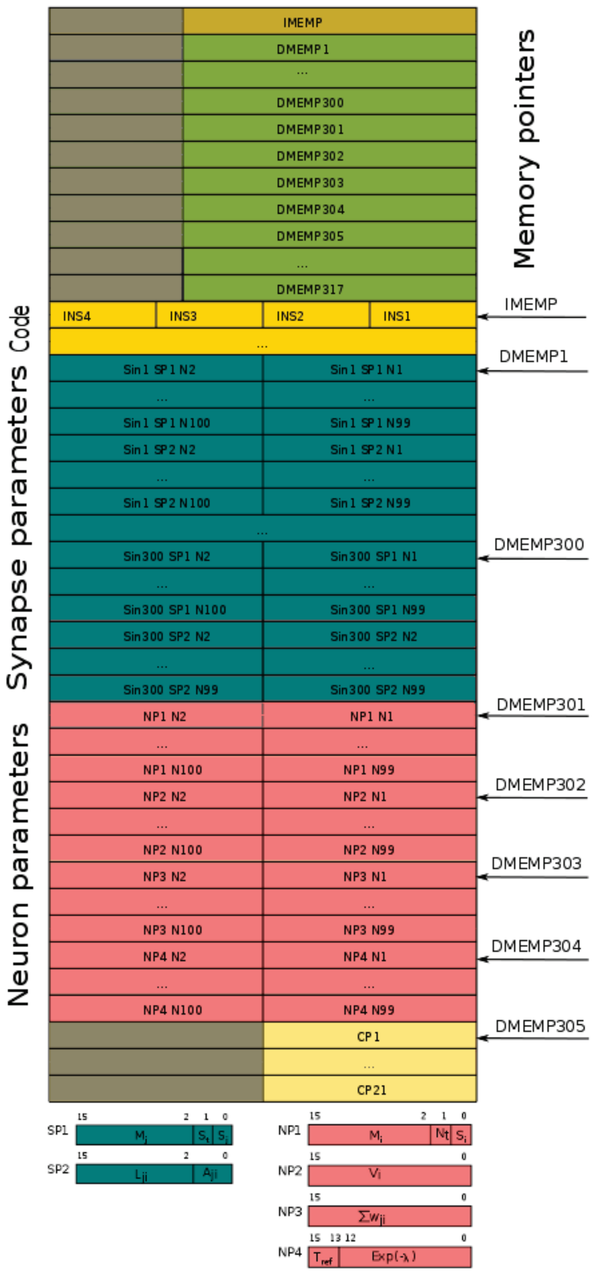


Fig. 5. Memory structure of a SpiNDeK implementation of Iglesias-Villa model [9].

The described menu is created based on the user-entered text file. There each variable of the network model is defined by name, pointer address or absolute position in the memory, value for masking out the memory row and some other needed information for feeding the Ubimanager-UbiColibri interface.

The most canonical approach to implement such feature would be to create some facility that would parse XML document whose elements represent that information. But the serious drawback of this solution is that it assumes that the generated memory map will stay the same in the future and most probably it will not. If such scenario would take place the plug-in would not be useful for other mappings or algorithms.

To overcome this issue, *QScriptEngine* class was incorporated into Ubiplot. This class is part of the *QScript* module of Qt libraries. Essentialy it is a JavaScript engine that is able to evaluate expressions and give back the results. This engine was wrapped with *ScriptEngine* class and it parses the user-input file in order to obtain the name of the variables and their placement for creating the aforementioned "Variables" menu. The format of this file is YAML since it is more readable and easier to write for humans than XML, thus it seemed more suitable for this task.

The example variable map file can be analyzed in figure 6. It has two sections "neuron" and "synapse" for neuron and synapse parameters respectively. Every section holds variables definition that starts with minus sign, in it all the information regarding parameter that is needed by the plug-in is written. It will be explained more in depth further in this section.

For simplicity the map in figure 6 defines only two variables: a neuron variable M_i and synapse variable L_{ji}. M_i is memory of the latest inter-spike interval that is used to compute L_{ji}. L_{ji} corresponds to an internal real-valued variable for activation level of jth synapse of ith neuron.

```
neuron:
    − name: Mi
      pointer: SYNAPSES + 1
      offset: (N + 1)/2 − 1
      mask: "N%2 == 0 ? 0xFFFC0000 : 0x0000FFFC"
      delta: 1
      binary code: simple
      magnitude: 1.0
synapse:
    − name: Lji
      pointer: S
      offset: NEURONS/2 + (N + 1)/2 − 1
      mask: "N%2 == 0 ? 0xFFFC0000 : 0x0000FFFC"
      delta: NEURONS
      binary code: simple
      magnitude: 0.01
```

Fig. 6. Simple variables' map

The plug-in sets 4 JavaScript variables that can be used in the file - "NEURONS", "SYNAPSES", "N" and "S". The "NEURONS" and "SYNAPSE" are equal to the number of neurons and synapses of the network (they are entered by the user when map is being loaded). While evaluating expression, "N" and "S" are, respectively, the indices of the neuron and synapse for which the request is being created. Assuming the user wants to plot L_{ji} for 1st synapse of 3rd neuron in the network of 4 neurons each having 3 synapses, the definition of L_{ji} should be read as follows:

- Pointer to the variable is located at address that is equal to the index of the synapse (1 in this example).
- After dereferencing it add value computed using this formula: NEURONS/2 + (N + 1)/2 - 1. For this example the value would be: 4/2 + (3 + 1)/2 - 1 = 3
- The mask will be 0x0000FFFC since the condition $N\%2 == 0$ for $N = 3$ is not met.
- The difference between each address when plotting for more than one unit is equal to number of neurons in the network (4).
- Data is binary encoded (not in two's complement code).
- Finally, multiply the incoming values by 0.01.

The quotation marks are needed when the string contains reserved keywords according to YAML specification (expression for mask in this case). If the entries "delta", "magnitude" or "binary code" are omitted, the sensible default values will be used (1, 1.0, "simple" respectively).

The map in figure 6 can be less verbose if the "define" section is used and the entries with default values are omitted, as seen in figure .

```
define:
    -
        "odd_even_neuron_idx = (N + 1)/2 - 1"
    -
        "sixteen_bits_mask = N%2 == 0 ? 0xFFFF0000 : 0x0000FFFF"
neuron:
    - name: Mi
      pointer: SYNAPSES + 1
      offset: odd_even_neuron_idx
      mask: sixteen_bits_mask
synapse:
    - name: Lji
      pointer: S
      offset: NEURONS/2 + odd_even_neuron_idx
      mask: sixteen_bits_mask
      delta: NEURONS
      magnitude: 0.01
```

By means of the proposed syntax, writing a variable map file should be an easy task since the C-like syntax of JavaScript is widely known and intuitive.

Variable's map feature makes the plug-in suitable for working with different sets of variables, memory mappings and different neural networks models.

Real-time emulation results

Some results are presented for a simple network of 4 neurons with 3 synapses each connecting the neuron with every other one.

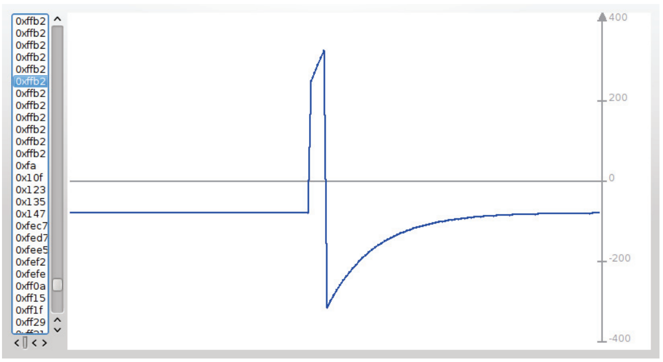
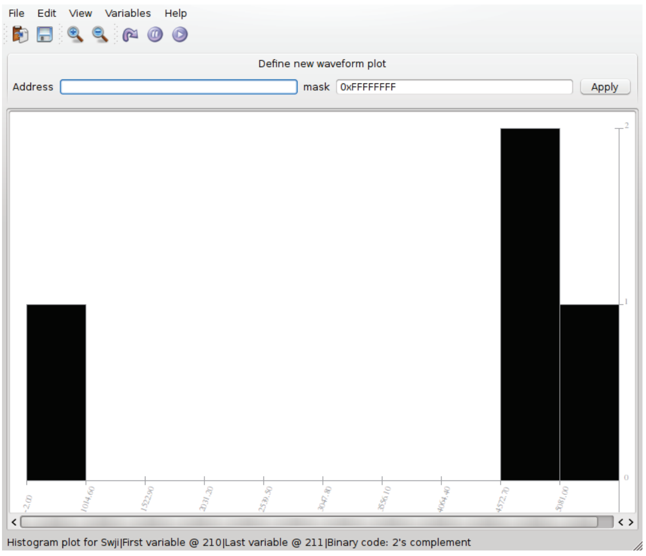


Fig. 7. Waveform showing membrane potential of first neuron while spiking, list on the left shows values in the hexadecimal base.

Figure 7 shows the waveform plot. It displays the membrane potential of a spikinig neuron. One can observe that neuron's potential was at resting state then it rised rapidly (spike), then it decreased and was exponentialy increasing back to resting potential. The list widget on the left shows data fetched in every execution step in the hexadecimal base. The right click on this widget will invoke context menu from which user can change the base to decimal base.

Figure 8 shows an example usage of the histogram plots. The plot is depicting the state of the synaptic weights of all the neurons in the current step. Due to the reduced number of variables the plot is discretized. The y axis shows ranges of the values of the synaptic weights in current step. The resolution was set to 10 while creating plot. The x axis is depicting how many neurons of given values was observed.

Ubiplot displaying four plots in the main window and one plot in the separate window is showned in figure 9. The plots in the main window are waveforms for membrane potentials, where spikes are visualised in the separate window using



Histogram plot for Swji|First variable @ 210|Last variable @ 211|Binary code: 2's complement

Fig. 8. Histogram plot for synaptic weights. The resolution is set to 10.

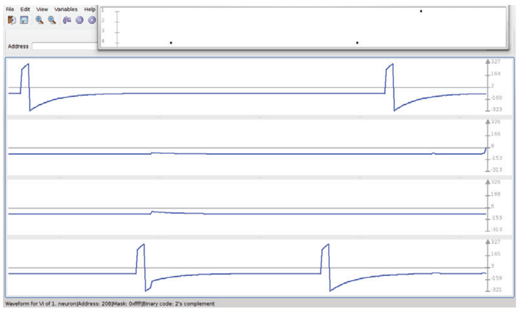


Fig. 9. Visualising membrane potentials in the waveforms and spikes using raster plot in the separate window.

raster plot. The first spike of the neuron in the top waveform is not visible in the raster plot since the width of the window holding raster was decreased and is showing less samples than waveforms in the main window.

Conclusions

Currently, the created software is being used successfully. It gives valuable debugging information and serves well for demonstration purposes.

The Qt framework, as expected, proved its usefulness. It is very reliable, easy to use and its API is complete. The only thing missing was the support for YAML format thus the external yaml-cpp library was introduced. Being written in Qt Ubiplot is cross platform, it is obviously a key advantage for end users since they can use it in different operating systems.

During the development question concerning the language to use arose. As it turned out C++ is error-prone and for such specific task some other language seems to be a better choice. Especially use of Python language that integrates easily into C, together with PyQt looks promising. Replacing C++ with a less verbose and error-prone language would benefit in shorter development time and more maintainable code.

The presented plug-in is looking promising in the context of future development. The usage of the *QScriptEngine* and the pseudo-scriptibility introduced by it makes the plug-in flexible. Thus it should be applicable to future implementation of any spiking neural network model. Also straightforward yet sufficient structure of Ubiplot makes the plug-in easy to extend with new features or new kind of plots.

Ubiplot is distributed under the GNU license version

2 the same as Ubimanager. Ubimanager's and Ubiplot's source codes are publicly available [10] [13].

REFERENCES

[1] A. Upegui, Y. Thoma, E. Sanchez, A. Perez-Uribe, J. M. Moreno, J. Madrenas and G. Sassatelli: "The PERPLEXUS bio-inspired hardware platform: A flexible and modular approach", International Journal of Knowledge-Based and Intelligent Engineering Systems, IOS Press, Vol 12, Number 3, pp. 201-212, 2008.

[2] J. Madrenas, J.M. Moreno: "Strategies in SIMD Computing for Complex Neural Bioinspired Applications", Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems, pp. 376-381, San Francisco, USA, July 29 - August 1, 2009.

[3] J. Madrenas and J.M. Moreno, "Strategies in SIMD Computing for Complex Neural Bioinspired Applications", Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on, IEEE, 2009, p. 376381.

[4] G. Sánchez, J. Madrenas, and J. Moreno, "Performance Evaluation and Scaling of a Multiprocessor Architecture Emulating Complex SNN Algorithms", Evolvable Systems: From Biology to Hardware, 2010, pp. 145-156.

[5] J.M. Moreno, J. Madrenas, and L. Kotynia, "Synchronous Digital Implementation of the AER Communication Scheme for Emulating Large-Scale Spiking Neural Networks Models", Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on, IEEE, 2009, p. 189196.

[6] Toradex Company website, http://www.toradex.com/En/Products/Colibri_Boards

[7] J. Iglesias, A. E. P. Villa, "Recurrent spatiotemporal firing patterns in large spiking neural networks with ontogenetic and epigenetic processes", Journal of Physiology-Paris: Volume 104, Issues 3-4, May-September 2010, pp. 137-146

[8] W. Gerstner and W. M. Kistler, Spiking Neuron Models - "Single Neurons, Populations, Plasticity", Cambridge University Press, 2002, p. 8.

[9] Michael Hauptvogel, J. Madrenas and J.M. Moreno, "SpiNDeK: an integrated design tool for the multiprocessor emulation of complex bioinspired spiking neural networks", CEC'09 Proceedings of the Eleventh conference on Congress on Evolutionary Computation, 2009

[10] Repository of the ubimanagertools source code, http://gitorious.org/ubimanager/ubimanagertools

[11] Y. Thoma and A. Upegui, "UbiManager: a software tool for managing ubichips", Adaptive Hardware and Systems, 2008. AHS 2008. NASA/ESA Conference on, IEEE, 2008, pp. 213-219.

[12] J. Iglesias, J. Eriksson, F. Grize, M. Tomassini and A.E.P. Villa, "Dynamics of pruning in simulated large-scale spiking neural networks Biosystems", 2005, Volume 79, Issues 1-3.

[13] Repository of the ubiplot source code, http://gitorious.org/ubiplot

***Authors**: M. Sc. Adam Sokolnicki, Ph.D. Bartosz Sakowicz, Department of Microelectronics and Computer Science, The Faculty of Electrical, Electronic, Computer and Control Engineering, Technical University of Lodz, ul. Bohdana Stefanowskiego 18/22 90-924 Lodz, Poland, email: sakowicz@dmcs.pl, M. Sc. Giovanny Sanchez, Ph.D. Jordi Madrenas, Ph.D. Manuel Moreno, Department of Electronic Engineering, Universitat Politecnica de Catalunya, Jordi Girona 1-3, 08034 Barcelona, Catalunya, Spain, email: madrenas@eel.upc.edu,*