Synchronous Digital Implementation of the AER Communication Scheme for Emulating Large-Scale Spiking Neural Networks Models

J.M. Moreno, J. Madrenas
Technical University of Catalunya-Dept. of Electronic Engineering
moreno@eel.upc.edu, madrenas@eel.upc.edu

L. Kotynia
Technical University of Lodz-Dept. of Microelectronics and Computer Science
lkotynia@dmcs.p.lodz.pl

Abstract

In this paper we shall present a fully synchronous digital implementation of the Address Event Representation (AER) communication scheme that has been used in the PERPLEXUS chip in order to permit the emulation of large-scale biologically inspired spiking neural networks models. By introducing specific commands in the AER protocol it is possible to distribute the AER bus among a large number of chips where the functionality of the spiking neurons is being emulated. A careful design of the AER encoder module using compact Content Addressable Memories (CAMs) allows for a feasible realization of large-scale models.

1. Introduction

The PERPLEXUS project [1] aims at the development of a flexible and scalable hardware substrate that will enable the efficient emulation of complex, virtually unbounded systems. The major outcome of this project is an integrated circuit, called Ubichip [2]. The internal architecture of the Ubichip has been endowed with specific hardware mechanisms, like dynamic routing [3] or self-replication [4] so as to provide support for a wide range of bio-inspired principles. Additionally, it supports massively parallel SIMD (Single Instruction Multiple Data) like data flows, thus making it a perfect candidate for the efficient emulation of Spiking Neural Networks (SNN) models [5].

The Address Event Representation (AER) is a communication scheme that was conceived specifically to address the issues raised by massive and sparse interconnection patterns. It was initially proposed in [6], [7] and later developed in [8], [9], [10], [11]. It consists in translating a sequence of events (spikes) into an ordered sequence of addresses that correspond to the individual processing elements that produced these events. This sequence of addresses is sent through a bus that communicates the places (usually chips) where the units are physically mapped. Since there is only a single actual channel where the addresses can be transmitted the implementation of this protocol usually involves the use of arbiters based on self-timed asynchronous logic [12].

The SpiNNaker system [13] tries to overcome these issues by implementing a packet-switched network [14] that is used to carry out the communication between the processing elements that emulate the spiking neurons.

The approach taken in the Ubichip consists in extending the basic principles presents in the AER scheme so as to permit a fully synchronous implementation of the protocol. The extension is based in the definition of specific commands sent through the address bus that permit different chips to synchronize their access to it, thus avoiding the need for a dedicated arbiter. Additionally, this will permit to share a single address bus among a virtually unbounded number of Ubichips, and in this way it will be possible to emulate in real time large-scale biologically inspired SNN models like the one presented in [15], [16], which constitutes the target for the PERPLEXUS project.

The structure of the paper is organized as follows. First a brief overview of the Ubichip architecture will be provided, followed by a description of the basic principles on which the AER communication scheme relies. Then the data flow used to carry out the emulation of spiking neural networks will be outlined, including a specification of the commands defined to extend the basic AER scheme. Afterwards the implementation details of the AER encoder and decoder modules will be provided, emphasizing the special care that has to be taken with the Content Addressable Memory (CAM) element that is the core building block of the AER decoder. Finally, the prototyping results of the complete system and our current and future work will be outlined.

2. The PERPLEXUS chip architecture

Figure 1 depicts the overall organization of the Ubichip.

It is constituted by three main building blocks: A MacroCell array, a system manager and a CAM controller. The MacroCell array is a regular, bi-dimensional arrangement of functional elements called Macrocells (MC in Figure 1). Each MacroCell contains four Ubicells, a Dynamic Routing unit and a Self-Replication unit. The final Ubichip will contain a 10 x 10 array or MacroCells, and thus a total of 400 Ubicells.

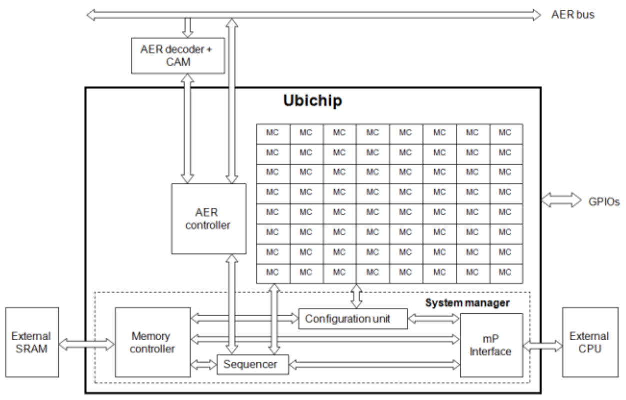

Figure 1. Internal organization of the Ubichip

The Ubicell is the elementary functional block of the Ubichip, and is constructed around four independent memory blocks and a 4-bit ALU. The memory blocks can be configured to provide several combinational and sequential operation modes, apart as being used as conventional SRAM. Additionally, they can be used as a register file for the ALU, so that a single Ubicell can act as an actual 4-bit processor. The Dynamic Routing unit takes care of the physical implementation of in-hardware, on-line path construction mechanisms among Ubicells. The Self-Replication unit permits to implement local and distributed self-configuration processes that are the basis for allowing the physical realization of self-replication mechanisms.

The system manager is composed of a microprocessor interface, a configuration unit, a memory controller and a sequencer. The microprocessor interface takes care of the communication between the Ubichip and an external CPU. The configuration unit manages the configuration process of the Ubichip that can be carried out using either a parallel or a serial interface. Additionally, it contains the necessary resources for facilitating the debugging of the complete system, allowing the designer to perform an edge-by-edge emulation and inspection of the application. The memory controller manages the interface of the Ubichip with an external SRAM memory that stores the program executed by the sequencer and the data used by the MacroCells when the system is configured in SIMD multiprocessor mode. The sequencer included in the system manager is in charge of controlling the MacroCell array when it is configured in SIMD multiprocessor mode. This unit reads the instructions stored in the external SRAM and dispatches them to the MacroCells that act as elementary processors in this mode.

The AER controller included in the Ubichip is the subsystem that carries out the physical implementation of the AER protocol. It is constituted by an AER encoder and a control unit that communicates with an external AER decoder module provided with a CAM unit.

A detailed explanation of the internal architecture of the Ubichip is provided in [17].

3. The AER communication scheme

As it has been previously stated, the basic principle of the AER communication scheme consists in translating a sequence of events produced by a set of processing elements into an ordered sequence of addresses that are sent through a dedicated bus that is broadcasted to the rest of the system. In the receiver side, the sequence of addresses is converted again into a sequence of events that are transferred to the corresponding destinations. Figure 2 depicts the basic mechanism on which AER is based.

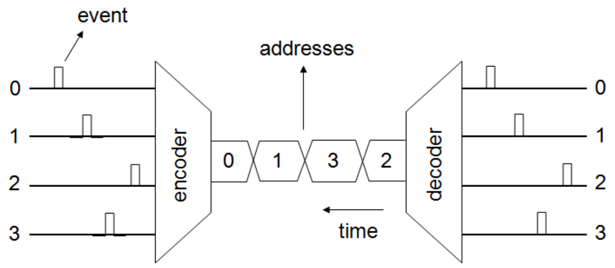

Figure 2. AER principle

This communication scheme is thus quite efficient for overcoming the bottlenecks that appear when information has to be exchanged within a system composed of massively interconnected components, like is the case of SNN models. For these models the events produced in the system correspond to spikes, and the processing elements are the units emulating the functionality of the neurons.

There are several issues to be addressed when an efficient implementation of the AER communication scheme is to be considered. The first one refers to the procedure used for scanning the events to be transmitted. In the case two or more events are produced at the same time it has to be decided in which order they are broadcasted through the address bus, because it can allocate a single address per time slot. The common way to face this problem is by means of asynchronous arbiters that establish a priority in the case of a simultaneous multi-event situation.

Another important issue refers to the access technique implemented for the AER bus. Since this bus is shared by different components working independently and asynchronously, a proper access method has to be established in order to avoid collisions in the bus. Several access techniques [9], including sequential scanning, ALOHA-based access or arbitration access have been proposed an used for implementing the AER communication principles, but there is not a clear guideline for choosing a particular access method for a specific application.

The strategy used in the Ubichip for implementing the AER communication protocol simplifies both the arbitration required in the encoder module as well as the access technique required for the management of the global AER bus. It is based in dividing the emulation of the SNN model implemented in the Ubichips in two phases, an execution phase and a spike transmission phase. The spike transmission phase is performed sequentially by all the Ubichips present in an ordered way, so that only one Ubichip has access to the AER bus at a given time. Once the spike transmission phase is completed all the Ubichips execute in parallel updating their internal state depending on the spikes received from the previous phase.

4. Neural emulation principle

The SNN model considered within the framework of the PERPLEXUS project is that presented in [15] and [16]. It is constituted by Leaky Integrate-and-Fire neuron units connected by synapses with variable weight depending on the time correlation between pre- and post-synaptic spikes. The modification of the synaptic strength is driven by a discrete activation variable, which is in line with recent observations suggesting that synaptic plasticity may be based on discrete dynamics [18]. The target for the PERPLEXUS project is a network constituted by 10000 neurons. The connectivity pattern between neurons is broad (i.e., a neuron may connect with any other neuron in the network), and a single neuron may have up to 300 synaptic inputs.

When the Ubichip is configured in multiprocessor mode the four Ubicells that constitute a MacroCell are joined in order to construct a 16-bit processor which is in charge of emulating a single neuron. Therefore, a single Ubichip is able to emulate the functionality of 100 spiking neurons. Figure 3 depicts the organization of a single processor when the Ubichip is configured in multiprocessor mode. The 5-bit opcode input determines the instruction to be executed by the processor, while the 3-bit source_dest input specifies which register is the source or the destination for a given operation. Each processor contains 16 16-bit register, divided in two banks, the active and the shadow one. Each bank is constituted by 8 registers (from which the first one is the accumulator), and operations executed by a processor affect only the registers constituting the active bank, even though there are instructions for moving data between both banks. The result of any arithmetic or logic instruction is always stored in the accumulator register (labeled ACC in Figure 3, the remaining registers of the active bank are labeled R0 to R6, the shadow register bank is not represented in the figure for the sake of clarity).

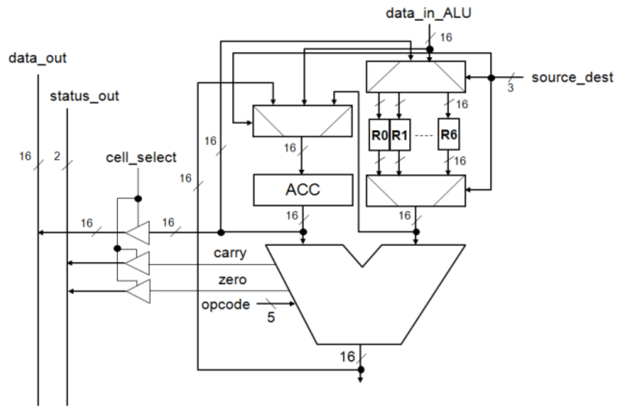

Figure 3. Organization of a 16-bit processor

The contents of the accumulator, as well as the carry and zero flags produced by the execution of an operation may be transferred to the sequencer using the data_out and status_out buses, respectively. These buses are shared by all the processors of an Ubichip. Only a single processor in the Ubichip may write to these buses at any given time, and this is controlled by a cell_select input present in every processor that is driven by the sequencer.

All the processors contained in an Ubichip execute the same sequence of instructions. These are stored in the external SRAM module and are fetched by the sequencer and later dispatched to the processors using an internal global bus. This SRAM unit contains also part of the data (basically, synaptic weights and overall network parameters) needed by the processors to emulate a spiking neuron, and the internal register bank is used to keep those variables that are used frequently, like the membrane potential or the learning parameters, so as to minimize the number of accesses to the external memory.

Since the target network to be emulated is constituted by 10000 neurons and a single Ubichip is able to implement 100 neurons, a total of 100 Ubichips will be attached to the shared AER bus. Every Ubichip contains an internal identifier, called chip_id, which ranges from 1 to 100, and additionally there is an input, called master, that identifies which Ubichip will drive the overall emulation process. This input should be set to '1' for the Ubichip with the highest value of the chip_id identifier.

The width of the AER bus is set to 7 bits, because the maximum value to be sent on it is 100. An additional pull-up driven signal, called ready, is included in the bus for synchronization purposes, as it will be explained later. Figure 4 shows the organization of an Ubichip network constructed in order to emulate a 10000 neurons SNN network.

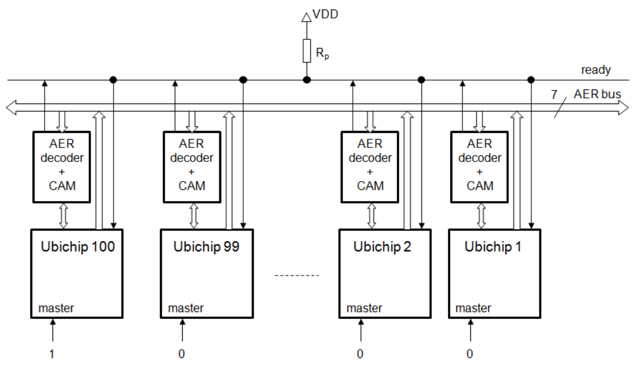

Figure 4. Ubichip network for emulating SNN models

As it has been previously stated, the emulation of a SNN network is divided in two phases: a spike transmission phase and an execution phase. The spike transmission phase is initiated by the Ubichip that is acting as a master in the network. This Ubichip first places a START_TX command on the AER bus, indicating to the network the beginning of the transmission phase. Then it places on the bus its chip_id value, which will be stored by the remaining Ubichips since it will constitute the most significant part of the address provided to the CAM units. Afterwards it will place sequentially on the bus the addresses of its internal processors that produced a spike during the last execution phase (these addresses will be concatenated to the previous chip_id value in order to obtain the complete addresses to be provided to the CAM units). Once all the spikes have been sent a NEXT_FRAME command is issued by the master Ubichip. At this time all the Ubichips will decrement by one unit the chip_id value received just after the START_TX command, and the Ubichip whose chip_id matches this value will take now control of the AER bus, sending its chip_id and the eventual spikes produced by its internal processors. This process is repeated successively until Ubichip 1 sends the NEXT_FRAME command. The master Ubichip knows then that the spike transmission phase is over, since the value obtained after decrementing the last chip_id value sent is 0. Therefore, it will issue the START_PROCESSING command, signaling the start of the execution phase. Due to the fact that all the processors in the Ubichips execute exactly the same sequence of instructions, after completing a new emulation step the master Ubichip is able to send a START_TX command starting a new spike transmission phase. Between the START_PROCESSING and START_TX commands the master Ubichip is placing the PROCESSING command on the AER bus. This command has no effect on the Ubichips, and is used just for network debugging processes.

Figure 5 depicts graphically the evolution of the neural emulation process, while table 1 summarizes the set of commands needed to implement it.

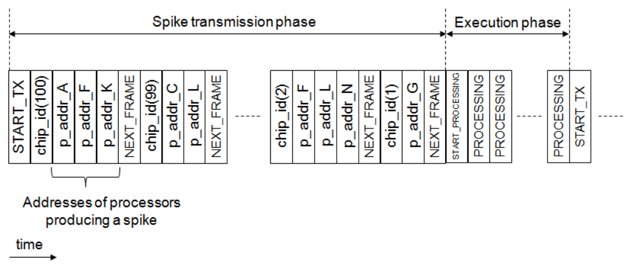

Figure 5. Evolution in time of a SNN model emulation

Table 1. List of commands defined for the proposed AER implementation

Hex value	Mnemonic	Description
0x00 – 0x63	p_addr_n	Address of processor producing a spike
0x01 – 0x 64	chip_id	Ubichip identifier
0x7F	NEXT_FRAME	Gives bus access to the next Ubichip
0x7E	START_TX	Start of the spike transmission phase
0x7D	START_PROCESSING	Start of the execution phase
0x7C	PROCESSING	Execution phase in progress
0x7B	NO_SPIKE	No spikes produced

It is worth noting that the width of the time slot (i.e., a spike transmission frame) during which a given Ubichip is sending spikes is not fixed, but depends on the number of spikes generated by its processors during the last execution phase. Additionally, as indicated in Table 1, there is a specific command signaling the absence of spikes. This helps to optimize the bus occupancy and therefore to minimize the duration of the spike transmission phase.

As it can be easily deduced from the previous explanation, the proposed strategy is easily scalable. Increasing the number of Ubichips in the network or the number of processors per Ubichip implies to add an additional line to the bus every time this number is doubled. Furthermore, reducing the number of Ubichips in the network implies just to reconfigure the internal chip_id values and changing the master input for one of them, being the remaining connections kept the same.

The previous explanation refers to the process of sending spikes through the AER bus. In order to detect if a spike produced by a neuron corresponds to a synaptic connection of another neuron, each Ubichip is interfaced with an AER decoder and a CAM unit. Actually this CAM unit contains as many independent CAM blocks as there are processors in an Ubichip (i.e., 100). The contents of every CAM block is initialized with the address of the neurons driving the synaptic inputs of the neuron it is associated with. Therefore, during the spike transmission phase the AER decoder concatenates the chip_id value sent in a transmission frame with the addresses of the processors sent during this frame, obtaining in this way the final addresses to be used as input for the CAM blocks it is managing. If a given CAM block produces a hit as a consequence of a read access this means that there is an input spike for the corresponding neuron. The address corresponding to the position where the match was identified indicates the input synapse of the neuron affected by this spike.

Since a single AER decoder is responsible for controlling the eventual hits produced by 100 CAM blocks, it may happen that during the spike transmission phase an address present in the bus produces a large number of hits in the CAM unit of an Ubichip, so that processing these hits will take longer than the time allocated for the arrival of a new AER address. In this case the AER decoder will drive low the ready line of the bus, thus indicating the sender Ubichip to stall the spike transmission process until this line is asserted again. The Ubichip that had access to the bus when the ready line was deasserted sends the NO_SPIKE command to the bus until it is set again to VDD.

So as to permit the construction of large Ubichip networks (and therefore the emulation of actual large-scale SNN models) the AER bus is working at a frequency that is 10 times slower than the system frequency used for the Ubichips (in our current implementation the target is 100 MHz for the Ubichip and 10 MHz for the AER bus).

5. AER encoder

The AER encoder is part of the global AER controller included in the Ubichip. It is basically a finite state machine that handles the spike transmission phase and synchronizes with the sequencer in order to maintain the overall SNN emulation process. Table 2 summarizes the states defined for this unit.

Before the addresses corresponding to the spikes produced are sent to the AER bus the encoder has to scan sequentially the processors present in a Ubichip in order to detect if the register storing the output spike is set or not. This scanning process is carried out row by row. Since the Ubichip frequency is 10 times faster than the frequency used for the AER bus, a FIFO queue with depth of 10 lines is used in order to keep the spike transmission process without missing spikes.

Table 2. States of the AER controller

Mnemonic	**Description**
OFF	The AER controller is disabled. It may be enabled by setting a specific configuration bit of the Ubichip
IDLE	The controller is listening to the AER bus
SEND_S_TX	The master Ubichip sends a START_TRANSMISSION command to the AER bus
SEND_ID	The Ubichip with access to the bus sends its chip_id identifier
SEND_SPIKES	The Ubichip with access to the bus sends the addresses corresponding to the spikes produced during the execution phase
SEND_N_F	The Ubichip with access to the bus sends the NEXT_FRAME command
STALL	The ready line is low and the active Ubichip sends the NO_SPIKE command until it is asserted again
SYNCH	Ubichips store the chip_id identifier sent to the bus in the current spike frame
SEND_S_PROC	The master Ubichip sends the START_PROCESSING command that initiates the execution phase
DATA_PROC	The processors contained in the Ubichip perform an execution cycle with the spikes produced during the spike transmission phase

6. AER decoder and CAM unit

The overall organization of the AER decoder and CAM unit is depicted in figure 6. As it can be deduced from this figure, it is constituted by a FIFO queue, a CAM array and a priority encoder. The inputs for this subsystem are the chip_id identifier sent in the current spike transmission frame and the addresses present in the AER bus (p_addr_n in Figure). By combining these two values the read address for the CAM blocks is obtained.

The outputs of this unit are a signal indicating that a hit has happened in one of the CAM blocks (hit signal), the identifier of the neuron for which a spike has been produced (neuron_id signal) and the identifier of the synaptic input of this neuron affected by the spike (synapse_id signal).

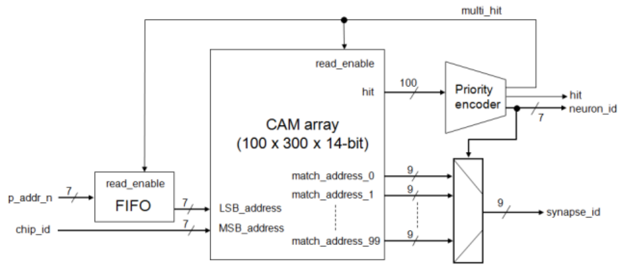

Figure 6. Organization of the AER decoder and CAM unit

The FIFO unit, whose depth is 10 words, is needed in order to allocate some time for handling the eventual matches produced in the CAM blocks.

The CAM array is constituted by 100 CAM blocks, one per processor in the Ubichip. Each CAM block has a depth of 300 words, since this is the maximum number of synaptic inputs per neuron in the SNN model considered. The width of these CAM blocks is 14 bits, since each position in the CAM stores the identifier corresponding to the neuron that produced a spike for the synaptic input corresponding to this position, and this information is encoded using the chip_id and the addresses sent in a given spike transmission frame and both values are encoded using 7 bits.

The priority encoder just encodes the hit lines produced by the CAM blocks into the neuron_id identifier. This signal is also used to select the address of the CAM block for which a hit was found. This subsystem is actually a sequential component, since there may be more than one CAM block producing a hit for the same input address to the CAM array. In this case, by properly setting the multi_hit signal that drives the read_enable input of the CAM blocks and the FIFO, the priority encoder guarantees that all the hits in the CAM array are properly processed in a sequential order.

As it has been stated previously, the AER encoder and the CAM unit are external to the Ubichip. This decision has been taken just for maximizing the number of Ubicells in the prototype, but both blocks couls be easily integrated in the Ubichip to facilitate a compact system implementation. They will be implemented using a commercial FPGA (a Xilinx XC3S5000fg900-4 device), and therefore special attention has been paid to the design of the CAM block in order to allow for a feasible implementation of the system.

7. CAM block design

The CAM unit needed to implement the AER protocol for the current realization of the Ubichip is constituted by 100 CAM blocks whose size is 300 x 15-bit. Since this unit will be implemented using a commercial FPGA, the first approach should be based on the use of the dedicated memory resources present in the device (either distributed or concentrated RAM elements) to create efficient memory structures.

If a BlockRAM (this is the term used for concentrated memory elements in the Xilinx devices) implementation is used it would require 20 blocks of memory for every CAM block, i.e., a total of 2000 blocks for the complete CAM unit. This number exceeds the maximum of 104 RAM blocks available in the target device (by the way, the largest one of the Xilinx Spartan-3 family).

When trying to implement the CAM block using distributed memory (actually the compact shift register mode, SLR16, of the Xilinx logic cells), the realization requires 2400 logic cells per CAM block, i.e., a total of 240000 logic cells for the complete CAM unit. This also exceeds the maximum of 74880 logic cells available in the target device.

Therefore, the approach taken has consisted in implementing a CAM block as a dedicated combinational unit. This means that after the actual connections between the neurons of the network to be emulated are known it is possible to set the contents of the CAM in the form of a table that can be later translated into simple combinational logic. For this purpose a generic and configurable behavioral VHDL description has been created. The synthesis and compilation of this description with a sample connectivity pattern for one neuron demonstrate that it occupies just 0.29 % of the resources available in the target device, thus making it a very efficient solution for implementing the whole system (the complete CAM array will occupy just 29 % of the device). This provides margin for extending the synaptic connectivity of the neurons beyond 300 inputs of for implementing in a single FPGA the AER encoder and CAM unit of two Ubichips, thus simplifying the implementation of the whole AER network.

This generic description can be customized automatically for a given SNN model connectivity pattern using the tools presented in [5]. These tools permit to adjust the network parameters using a graphical user interface, and they also provide as an output not only the program to be executed by the processors contained in the Ubichips, but also the configuration parameters for the CAM array of every Ubichip. This facilitates considerably the prototyping and implementation tasks.

8. Prototyping results

A system prototype constituted by an array of 4 x 4 Ubicells (i.e., 2 x 2 MacroCells or neural processors) has been implemented and physically mapped onto a Xilinx XC3S5000fg900-4 device. The prototype contains also the AER decoder and CAM array. The system occupies 33 % of the resources available, and the behavior of the different subsystems has been successfully tested. The FPGA device is part of a specific board developed within the framework of the PERPLEXUS project whose core is a Marvell PXA270 microprocessor that is used as an overall system controller and debugger.

9. Conclusions and future work

In this paper a synchronous implementation of the AER communication protocol has been presented. It has been implemented within the framework of the PERPLEXUS project as a basic mechanism for supporting the efficient emulation of large-scale biologically inspired SNN models.

The proposed implementation of the protocol includes some commands that alleviate the arbitration and access mechanisms that are required in the original AER proposal. This facilitates the construction of large SNN networks that are emulated by clusters of processors (included in an Ubichip in the case of the PERPLEXUS project) that may work locally independent system clocks and can synchronize its operation by means of the commands sent through the AER bus.

The proposed implementation is scalable, since it is not very sensitive to the number of processor clusters neither to the number of processors included in them.

In the physical implementation of the proposal special attention has been paid to the realization of the CAM blocks on which the AER decoding subsystem is based. By optimizing the behavioral description of an elementary CAM block it has been possible to include a complete 100 CAM array into a single commercial FPGA device occupying just 29 % of the available resources.

A system prototype has been physically implemented and successfully tested in a Xilinx XC3S5000fg900-4 device.

Our current work is concentrated in the final stages of the physical implementation of the Ubichip in the form of an ASIC. It will be fabricated using a commercial 180 nm 6-metal CMOS process, and it will contain an array of 10 x 10 Macrocells (20 x 20 Ubicells).

10. Acknowledgements

The work presented in this paper has been funded by the European Union (PERPLEXUS project, Contract no. 34632) and by the Spanish Ministry of Science and Innovation (Project reference TEC2006-28116-E).

11. References

[1] E. Sanchez, A. Perez-Uribe, A. Upegui, Y. Thoma, J.M. Moreno, A. Villa, H. Volken, A. Napieralski, G. Sassatelli, E. Lawarec, “PERPLEXUS: Pervasive Computing Framework for Modeling Complex Virtually-Unbounded Systems”, Proceedings of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems, pp. 587-591, Edinburgh, UK, August 5-8, 2007.

[2] A. Upegui, Y. Thoma, E. Sanchez, A. Perez-Uribe, J.M. Moreno, J. Madrenas, “The Perplexus bio-inspired reconfigurable circuit”, Proceedings of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems, pp. 600-605, Edinburgh, UK, August 5-8, 2007.

[3] A. Upegui, Y. Thoma, A. Perez-Uribe and E. Sanchez, **“**Dynamic Routing on the Ubichip: Toward Synaptogenetic Neural Networks”, Proceedings of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems, pp. 228-235, Noordwijk, The Netherlands, June 22-25, 2008.

[4] Y. Thoma, A. Upegui, A. Perez-Uribe, E. Sanchez, “Self-replication mechanism by means of self-reconfiguration”, 20th International Conference on Architecture of Computing Systems (ARCS '07), pp. 105-112, VDE Verlag, 2007.

[5] M. Hauptvogel, J. Madrenas, J.M. Moreno, “SpiNDeK: An Integrated Design Tool for the Multiprocessor Emulation of Complex Bioinspired Spiking Neural Networks”, to be published in the Proceedings of the 2009 IEEE Congress on Evolutionary Computation, Trondheim, Norway, May 18-21, 2009.

[6] M.A. Mahowald, “VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function”, Ph.D. dissertation, California Institute of Technology, Pasadena, 1992.

[7] M. Sivilotti, “Wiring Considerations in Analog VLSI Systems With Applications to Field Programmable Networks”, Ph.D. dissertation, California Institute of Technology, Pasadena, 1991.

[8] K.A. Boahen, “Point to Point Connetivity Between Neuromorphic Chips Using Address Events”, IEEE Trans. on Circuits and Systems II, Vol. 47, No. 5, pp. 416-434, May 2000.

[9] E. Culurciello, A.G. Andreou, “A Comparative Study of Access Topologies for Chip-Level Address-Event Communication Channels”, IEEE Trans. on Neural Networks, Vol. 14, No. 5, pp. 1266-1277, September 2003.

[10] K.A. Boahen, “A Burst-Mode Word-Serial Address-Event Link-I: Transmitter Design”, IEEE Trans. on Circuits and Systems I, Vol. 51, No. 7, pp. 1269-1280, July 2004.

[11] K.A. Boahen, “A Burst-Mode Word-Serial Address-Event Link-II: Receiver Design”, IEEE Trans. on Circuits and Systems I, Vol. 51, No. 7, pp. 1281-1291, July 2004.

[12] E. Culurciello, R. Etienne-Cummings, K. Boahen, “High Dynamic-Range, Arbitrated Address Event Representation Digital Imager”, Proceedings of the 2001 IEEE International Symposium on Circuits and Systems, Vol. 3, pp. 505-508, Sydney, Australia, May 6-9, 2001.

[13] L.A. Plana, S.B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, S. Yang, “A GALS Infrastructure for a Massively Parallel Multiprocessor”, IEEE Design and Test of Computers, Vol. 24, No. 5, pp. 454-463, Sept.-Oct. 2007.

[14] L.A. Plana, J. Bainbridge, S. Furber, S. Salisbury, Y. Shi, J. Wu, “An On-Chip and Inter-Chip Communications Network for the SpiNNaker Massively-Parallel Neural Net Simulator”, Proceedings of the second ACM/IEEE International Symposium on Networks-on-Chip, pp. 215-216, Newcastle upon Tyne, United Kingdom, April 7-11, 2008.

[15] J. Iglesias, J. Eriksson, F. Grize, M. Tomassini, A.E.P. Villa, “Dynamics of pruning in simulated large-scale spiking neural networks”. Biosystems, Vol. 79, pp. 11-20, January-March 2005.

[16] J. Iglesias, J. Eriksson, B. Pardo, M. Tomassini, A.E.P. Villa, “Emergence of Oriented Cell Assemblies with Spike-Timing-Dependent Plasticity”, Artificial Neural Networks: Biological Inspirations. W. Duch, J. Kacprzyk, E. Oja, S. Zadrozny (eds.). Lecture Notes in Computer Science, Vol. 3693, pp. 11-20, Springer-Verlag, 2005.

[17] J.M. Moreno, J. Madrenas, “A Reconfigurable Architecture for Emulating Large-Scale Bio-inspired Systems”, to be published in the Proceedings of the 2009 IEEE Congress on Evolutionary Computation, Trondheim, Norway, May 18-21, 2009.

[18] J.M. Montgomery, D.V. Madison, “Discrete synaptic states define a major mechanism of synapse plasticity”, Trends in Neurosciences, Vol. 27, No. 12, pp. 744-750, December 2004.