Chapter 7

Simulating Memristive Networks in SystemC-AMS

Dietmar Fey, Lukas Riedersberger and
Marc Reichenbach

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.69662

Abstract

This chapter presents a solution for the simulation of large memristive networks with SystemC-AMS. SystemC-AMS allows simulating memristors both on analogue level and on digital level to link analogue memristive devices to digital circuits and system level specifications. We investigate the benefits and drawbacks of a SystemC-AMS simulation compared to a simulation in SPICE. We show for the example of a two-layer memristive network emulating an optical flow algorithm by the detection of moving edges that large memristive networks can be simulated with a free available SystemC-AMS simulation environment, whereas free available SPICE simulation environment fails. However, it is also shown that commercial SPICE simulators are superior against current SystemC-AMS implementations concerning the size of simulated memristive networks. However, SystemC-AMS simulations of memristive networks offer both still more flexibility and similar run times compared to commercial SPICE simulators for small-sized memristive networks. The flexibility and the powerfulness of a SystemC-AMS solution is demonstrated for a complex network that solves edge detection, filtering and detecting of moving objects. The possible run times of the memristive network are determined in the SystemC-AMS simulation environment and are compared with an optical flow algorithm on classical hardware like a CPU and a GPU.

Keywords: memristive networks, SystemC-AMS, memristor modelling, optical flow, SPICE memristor simulation

1. Introduction

One of the missing things in the research on modelling and simulation of large memristor networks is the availability of an adequate simulation system, which is both fast and flexible. Available SPICE models offer for commercial products, for example, Spectre Circuit Simulator,

fast simulation times but don't offer flexibility. To establish links to higher abstraction levels, for example, to the system level, in order to combine memristive circuits with extensive digital circuits, or even the integration in processor architecture descriptions to execute software in virtual environments is very cumbersome.

Therefore, we established a model for memristors in SystemC-AMS. A SystemC simulation can be carried as fast as SPICE simulations but allows a better linking to higher system levels as it is possible, for example, with Verilog-A, for which already memristor models exist [1]. SystemC-AMS allows also a detailed investigation of the analogue behaviour in the same way as one it is used it in SPICE.

For the modelling of single memristor behaviour in SystemC-AMS, we used the possibility to model variable resistors in SystemC-AMS with electrical linear networks (ELNs) as starting point. The resistance values of such elements can be controlled and modified by a discrete event input signal. We start to demonstrate this possibility with the well-known SPICE memristor model from Biolek et al. [2], which is based on an electronic equivalent circuit of the simple memristor behaviour description from Hewlett-Packard.

However, we do not mimic the electronic equivalent circuit in SystemC-AMS. The memristor model is realized in SystemC-AMS as an own object-orientated class. Using object-orientated programming principles allows simply exchanging the model for the memristive behaviour by another one. In principle, it is possible to use any other model as long as it is specified by a C/C++ code snippet. We have implemented in SystemC-AMS two memristor models, the HP model that is also used in Biolek's SPICE model [2] and a statistic description for a commercial memristor coming from Knowm Inc. [3].

We demonstrate the usefulness and the strength of a SystemC-AMS-based simulation system for a three-dimensional (3D) memristive circuit that implements a detection based on an optical flow. For this application, a memristive network was proposed in Ref. [4]. We adapt this solution and modelled the complete network in SystemC-AMS. We compare the achieved results with an implementation on a GPU to evaluate possibilities and limits of the compute capability of memristive circuits. The chapter is organized as follows. In Section 2, we present our solution for the modelling of memristors in SystemC-AMS. Section 3 gives a brief insight in the optical flow algorithm we used for an implementation on a GPU and a multi-core CPU serving as reference architecture for the simulated memristive network. The corresponding memristor network calculates optical flow gradients as moving edges with a memristive network. We selected exactly this network as a representative complex example for a SystemC-AMS specification of memristive networks. Section 4 specifies the achieved results for the simulation and compares it with a GPU/CPU implementation concerning the run time. Furthermore, the simulation time of a SystemC-AMS specification and a SPICE simulation of the specific memristive network are compared. Finally, the chapter ends with a conclusion.

2. Modelling memristors in systemC-AMS

SystemC-AMS is an extension of the modelling language SystemC about analogue-mixed signals. It allows not only the modelling of digital hardware and corresponding

software in one homogeneous environment but also the combination of discrete and continuous analogue systems. SystemC-AMS contains a solver for the Kirchhoff equations which are used for the computation of the behaviour of electrical networks. SystemC as well SystemC-AMS is not a new language but an extension of C++ about a corresponding library. Therefore, it allows the modelling of analogue and digital systems using a class-orientated structure. This feature is very beneficial for designing complex memristor networks using different models. Just by instantiating another memristor model in the SystemC-AMS program, a whole network can be simulated with another memristor model in a very convenient way. A modification of the electronic network is not necessary.

SystemC-AMS offers three main options for the modelling of discrete and continuous systems on different abstraction levels. Furthermore, these models can also be coupled via matched interfaces. An example scenario for such a coupling of components modelled in different domains consists of, for example, a binary module that is connected to a linear electronic circuit. In this case, the binary module could differ between two states which are used to control a voltage source. According to a state transfer of the binary output, the polarization of the continuous output voltage signal is reversed. For the work presented in this chapter, we used a proof-of-concept implementation of SystemC-AMS from Accellera and Coseda Technologies [5], which is freely available under an Apache 2.0 licence. Since this implementation was developed primarily with respect to its correct implementation of the IEEE standard 1666.1, the main focus was not laid on the simulation speed. Therefore, it is to investigate how other possible SystemC-AMS simulators could offer an alternative in future, resp. a re-evaluation has to be done when the current version of Coseda leaves its current proof-of-concept state.

2.1. SystemC-AMS modelling options

In the following, we briefly present the above-mentioned three modelling options offered by SystemC-AMS and evaluate them for their appropriateness to model memristors.

The *timed data flow* (TDF) model allows the modelling of discrete time steps. Each TDF module has a couple of inputs and outputs, which consume an event at discrete time steps. As result of this occurred event, the module can change its internal state. However, processing at discrete time steps does not help us to model the analogue behaviour of memristors. The *linear signal flow* (LSF) allows solving continuous equations. For that purpose, different basic blocks like adders, multipliers and integrators are offered, which can be connected and are processed by a built-in solver for differential equations. Also, this option does not meet our intention of memristor modelling since LSF is more orientated to model signal-processing algorithms based on pre-built blocks. The third main modelling method is thought to model analogue systems as *electrical linear networks* (ELNs). It allows the set-up and solving of electrical networks by applying the Kirchhoff circuit laws for electronic meshes and nodes. An electrical network consists of modules which are connected via nodes. Using these laws, increasing and decreasing currents and voltages can be determined for the devices. Since memristors are part of electronic circuits, we use this modelling technique primarily for the memristor modelling.

In the following, we describe in detail the SystemC-AMS specification we selected for the memristor modelling. **Figure 1** shows a block diagram of the corresponding model. The basic idea is to model the memristor with a SystemC-AMS built-in data type *variable resistor* which allows changing dynamically its resistance by a discrete value. For each memristor in the simulated network, the voltage, which drops down at its ports *p* and *n*, is read out via *get_voltage*() in each simulation step. Depending on the used memristor model, the new memristor's memristance is calculated. Subsequently, the new value is assigned to the variable resistance via assigning a discrete signal by the method *set_resistance*().

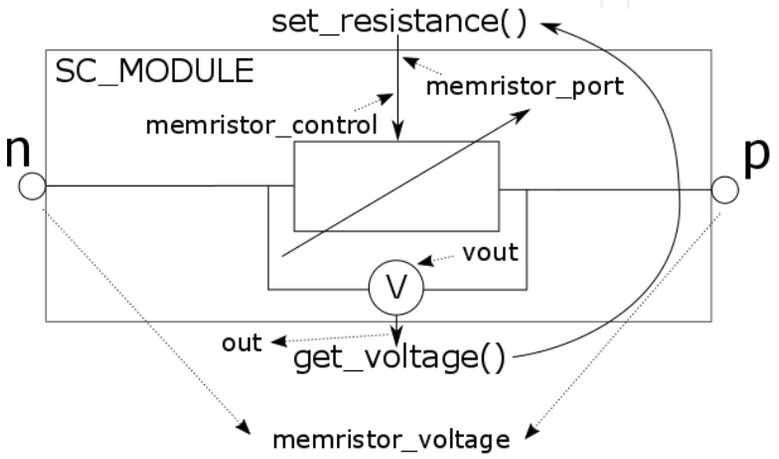

Figure 1. Block diagram for the selected SystemC-AMS model for a memristor.

The code fragment shown below is the corresponding SystemC-AMS specification:

1. The memristor is modelled as an object-orientated *class Memristor* in SystemC-AMS. The memristance is calculated and stored as discrete variable in *R*.
2. The memristor has two ports *p* and *n*.
3. The memristor device, denoted as *memristor_resistor*, is modelled as variable resistor. It inherits its characteristics from the SystemC-AMS built-in type ***sca_eln::sca_de::sca_r***. This variable is used in the circuit to which an instanced memristor element of the *class Memristor* is connected to via the ports *p* and *n*.
4. The voltage drop at the memristor can be measured by a kind of display variable *out*, this is the readable voltage value, that is given out via the virtual voltage metre *vout*. The corresponding voltage value is stored at *memristor_voltage*.
5. SystemC uses a discrete-event simulation, for that it is necessary to define a so-called control port parameter that checks if a signal change occurs at its input. This is the variable *memristor_control*. To this port, a signal has to be attached which is *memristor_port*.

6. The functional behaviour of a memristor, defined by its specific model, is specified by a later instanced virtual function *solve*(), which can be implemented in C/C++ code for each specific memristor model:

```
//Base class
class Memristor {
public:
double R;
//ports of the memristor
sca_eln::sca_terminal p,n;
//resistor controlled by discrete-event input signal, needs input
sca_eln::sca_de::sca_r memristor_resistor;
//converter and voltage meter
sca_tdf::sca_out<double> out;
sca_eln::sca_tdf::sca_vsink vout;
//systemc ams interface to read voltage over the resistor
sca_tdf::sca_signal<double> memristor_voltage;
//control port of controlled resistor
sc_core::sc_in<double> memristor_control;
//systemc ams interface to set the new resistance
sc_core::sc_signal<double> memristor_port;
//solve must be implemented by the specific model
virtual void solve(const double dt) = 0;
};
```

A specific memristor is modelled by an inheritance from the *class Memristor*. This is shown in the following for the specification of a *class MemristorBiolek*, which is inherited by the generic *public class Memristor*. The functional behaviour of the inherited memristor is orientated to the SPICE equivalent model from Biolek given in Ref. [2]. Some physical features for the memristor are defined as constants at the beginning like the *DRIFT_MOBILITY* of the ions and the *LENGTH* of the channel of the modelled memristor. Furthermore, variables for the maximum and the minimum resistance, *R_ON* and *R_OFF*, and the width of the doped region, *w*, are declared. Furthermore, the class constructor and some parameters

(*R_ON, R_OFF, R_INIT*) are defined, which can be passed to the class element when it is instanced to initialize these memristor parameters. The functionality of the memristor type is defined by the method ***solve***. A discrete solution of a differential equation for the memristance change is used; the step width for the integration is defined by *dt*. The method *solve* is the central key of the flexibility in the simulation. It can be changed by another function to implement another model.

The following SystemC-AMS code sequence shows the specification of a class that models a memristor's behaviour specification according to the Biolek model

```
class MemristorBiolek: public Memristor {
private:
  const double DRIFT_MOBILITY = 440000.0 * pow(10.0, -18.0);
  const double LENGTH = 41.0 * pow(10.0, -9.0);
  double R_ON, R_OFF, w;
public:
MemristorBiolek(const double R_ON, const double R_OFF,
const double R_INIT);
void solve(const double dt, std::function<double(double)>
voltage_function = [](const double val) -> double
{
return val;
}
);
std::string name() const { return "Biolek"; }
};
```

The following code snippet specifies the constructor for the inherited class *MemristorBiolek*:

```
Memristor::Memristor(const double R_INIT): R(R_INIT) {} ;
MemristorBiolek::MemristorBiolek(const double R_ON, const double R_
OFF, const double R_INIT): R_ON(R_ON), R_OFF(R_OFF), Memristor(R_INIT)
{
double x = (R_INIT - R_OFF)/(R_ON - R_OFF);
if (x > 1.0) x = 1.0;
```

```
if (x < 0.0) x = 0.0;
w = x * LENGTH;
}
```

The next code sections show the implementation of the method *solve*() to calculate the memristance of the memristor. The nonlinear behaviour of the memristor is modelled by the window function *windowBiolek*() that was set up by Biolek in Ref. [2] in order to modify the changing of the width of the memristor's doped region *w* at the edges of the device

```
inline long double windowBiolek(double x, double I,
double const P_WINDOW) const
{
if (-I >= 0)
return 1 - pow(x - 1, 2 * P_WINDOW);
return 1 - pow(x, 2 * P_WINDOW);
}
void MemristorBiolek::solve(const double dt, std::function<double(dou
ble)> voltage_function) {
double U = memristor_voltage.read(0);
double I = U/R;
R = R_ON * (w/LENGTH) + R_OFF * (1 - w/LENGTH);
double vD = ((DRIFT_MOBILITY * R_ON)/LENGTH) *
I * windowBiolek(w/LENGTH, I, 7.0);
w += vD * dt;
write_resistance();
}
} ;//end of definition of class Memristor
```

Figure 2 shows multiple overlaid hysteresis curves for the I-U relation at the memristor's poles. Throughout, the memristor was simulated with a minimum resistance R_{OFF} = 200 Ω, a maximum resistance R_{ON} = 28 Ω and an initial resistance R_{INIT} = 100 Ω. A sinusoidal voltage source is attached serially to the memristor. The voltage source is oscillating with 1 kHz between −1 and 1 V. The simulated time was set to 1 s with a time resolution of 1 μs. It is to observe that with each oscillation, the hysteresis curve becomes more flat until it ends in a more or less straight line, that is, the non-linear behaviour disappears.

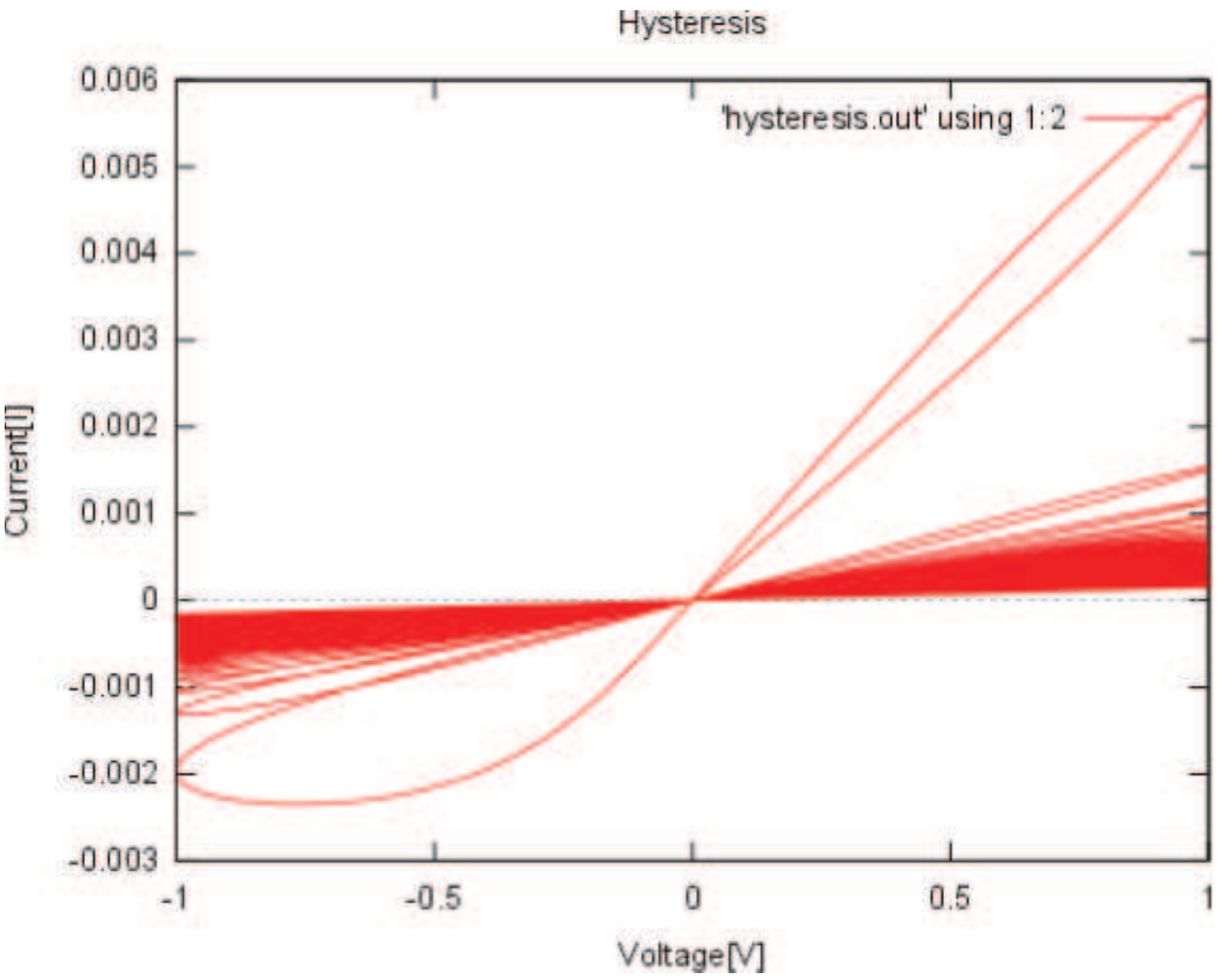

Figure 2. Result of SystemC-AMS simulation of a memristor excited by a sinusoidal voltage signal.

3. Simulation of an optical flow algorithm with a memristor network in SystemC-AMS

In the last chapter, we have shown how to simulate the analogue behaviour of a single memristor element. The next step is to demonstrate the possibility to simulate a much more complex example, namely the simulation of optical flow as detection of moving edges in a grid of memristors. The network mimics the functional behaviour of an artificial retina with a network consisting of resistors and memristors. The network was presented in Ref. [4]. In the following, we describe the set-up of the memristive network and the necessary functions to realize the detection of moving edges. We compare in the following the solution with an optical flow implementation on classical hardware. The optical flow follows the procedure according to Horn and Schunk [6]. For reasons of completeness, we briefly describe this algorithm first and the corresponding memristive network later as well as its SystemC-AMS specification developed by us.

http://dx.doi.org/10.5772/intechopen.69662

3.1. Procedure of the optical flow

The procedure of Horn and Schunk was one of the first optical flow methods. It provides a dense and smooth global result. Global in this sense means that the whole image is considered and not only a local region around a pixel in order to solve the equation motion for pixels in two subsequent images. In an optical flow procedure, a vector field h is computed according to Eq. (1) that describes the translation of pixel (x,y) in a two-dimensional (2D) image over time $\frac{dx}{dt}$ and $\frac{dy}{dt}$.

$$h = (u, v) = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) \tag{1}$$

For the calculation of translating pixels, it is assumed that their intensities remain constant after the translation. That means, a pixel, which is moved between two images $I(x,y,t)$ and $I(x,y,t+dt)$, has to maintain its brightness

$$I(x, y, t) = I(x + u \cdot dt, y + v \cdot dt, t + dt) \tag{2}$$

As a consequence, each algorithm, which is based on this equation, has to calculate with scalar, that is, grey values, and not with colour values. This has to occur also later in the memristive network. Finally, after applying the chain rule, Eq. (2) can be transformed to the central Eq. (3) that is solved in a similar way by detecting moving edges by the corresponding memristive network presented in Ref. [3]. This network is simulated here with SystemC-AMS to demonstrate that complex memristive networks can be simulated with our approach of modelling the dynamic of memristors with variable resistors

$$I(x, y, t) = I(x, y, t) + u \cdot dt \frac{\partial I}{\partial x} + v \cdot \frac{\partial I}{\partial y} + dt \frac{I}{\partial t} \tag{3}$$

$$I_x(x, y) \cdot u + I_y(x, y) \cdot v + I_t(x, y) = 0 \tag{4}$$

3.2. Memristive network and SystemC-AMS specification

In the following, we exemplarily consider the details only for the derivatives in the space to x and y dimension for a corresponding 2D memristor network. The extension to the time domain would be an additional layer in the third direction between corresponding pixels in neighboured images. As mentioned, the algorithm works on grey-scaled images; therefore, the scales have to be inverted in corresponding voltages. For the simulation, it is enough to restrict to an 8-bit resolution. Since the voltage of a photo-sensitive cell is in the range of 0–40 mV, we get the following scaling of the input voltage for each pixel Eq. (4)

$$V_p(x) = x \cdot \left(\frac{40}{255}\right)\text{mV} \tag{5}$$

This scaling has to be carried out for each pixel in the image. In our SystemC-AMS specification, this is done per instruction code, which calculates Eq. (4) and uses the result to

instantiate a DC voltage source. **Figure 3** shows a scheme for a memristive circuit that handles each pixel in the image.

The voltage V, representing a changed grey value, is attached to the network via a resistance R_C, which influences the time behaviour of the network for solving the optical flow. Since the optical flow changes dynamically in the network, the flow is modelled by current flowing through dynamically adapting resistances for which memristors are required. A memristor R_M, which stores the result of the optical flow, again as an encoded grey value, and two further memristors, denoted as outer plexiform layer (OPL) in **Figure 3**, complete the circuit handling a pixel.

A corresponding description of the header file for the pixel (without the OPL) in SystemC-AMS is shown below. Firstly, the parameters are specified for the constructor of a pixel class called *PixelNode*. Since an instance of *PixelNode* is one pixel within a 2D array, it receives two identifiers. The first one is *image_id*. It identifies in which image the pixel is, remember the optical flow requires two subsequent layers connected with each other. The second identifier, *idx*, addresses uniquely the pixel within the image. Then, four resistance values are as follows: R_CONST, the starting value for the top resistor R_C, R_ON, R_OFF and R_INIT for the initial setting of the memristor denoted as *A* in the class, which corresponds to the bottom memristor R_M in **Figure 3**. The parameters *initial_pixel* and *vsource* correspond to the input grey value of the pixel and the input voltage V, which has to be calculated elsewhere in the code according to Eq. (4). The further specifications *eln_pixel* and *neighbours* refer to the virtual electronic network to make a connection to a virtual potentiometer to measure current running through the pixel and the voltage applied at that pixel, respectively, to the connection to the neighboured pixels via the OPL. Both specifications and the ground connection, *gnd*, also require unique identifiers which are passed as strings, *eln_pixel* and *gnd*, in the parentheses to the instances of *A*. Finally, the instructions given within the brackets provide the connections to the memristor as variable resistor analogue to the example given in the previous chapter for a memristor of the class *MemristorBiolek*. The result voltage will adjust at R_C. It is calculated in the method *PixelNode::pixel_value*. This voltage can be used in order to calculate the resulting grey value

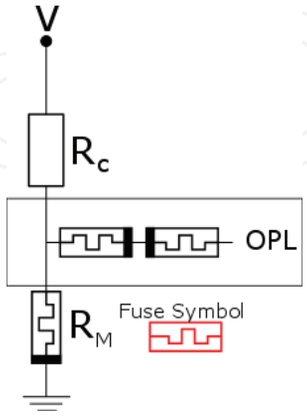

Figure 3. Electrical network for one pixel. The memristive fuse OPL realizes the connection to the neighbour pixel. All pixels correspond to the mid-layer. The resistance R_C controls the speed of the adaption of the memristive network, the voltage over resistance R_M corresponds to the result, that is, if a moving pixel was detected according to a detected optical flow.

http://dx.doi.org/10.5772/intechopen.69662

```
PixelNode::PixelNode(const size_t image_id, const size_t idx,
const double R_CONST, const double R_ON,
const double R_OFF, const double R_INIT,
const unsigned char initial_pixel_value,
const double vsource):
R_CONST(R_CONST),
initial_pixel_value(initial_pixel_value),
vsource(vsource),
A(R_ON,R_OFF,R_INIT),
eln_pixel(("eln_pixel_"+std::to_string(image_id)+"_"
+std::to_string(idx)).c_str(,vsource,R_CONST),
neighbours(("eln_pixel_neighbour_node_"+std::to_string(image_id)+"_"
+std::to_string(idx).c_str()),
gnd(new              sca_eln::sca_node_ref(std::string("gnd"+std::to_
string(image_id)+"_"
+std::to_string(idx).c_str())) {
eln_pixel.memristor_controll(A.get_control_port());
eln_pixel.out(A.get_voltage_port());
eln_pixel.neighbours(neighbours);
A.write_resistance();
}
double PixelNode::pixel_value() const {
double mapped_voltage = A.read_voltage() *
((R_CONST+A.resistance())/A.resistance());
return mapped_voltage;
}
```

If a low-resistance value is assigned to R_C, the voltage drop at the resistance will occur slowly, and due to the higher voltage that is applied to the subsequently attached memristors, in this case, their memristances are changing faster. In opposite, a higher resistance produces a more time-lag reaction in the network since now the memristors need more time to adapt their internal states. This new generated voltage via R_C is now the input for the main layer of the

network. The function of this main layer is adapted to the outer plexiform layer of the retina. This main layer mimics horizontal cells in the retina. Therefore, a connection to the neighbour pixels has to be realized via the so-called *memristive fuses*. These fuses provide an automatic averaging of the voltages connected to neighboured pixels. If there is a high potential difference between two neighboured pixels, then the memristive fuse adjusts faster a higher memristance. This leads to an edge-preserving property of the filter since the influence of the pixel decreases by the time. However, this idea would not work with a single memristor because the potential difference on that memristor could be either positive or negative. In the case of a negative potential, the memristance would decrease. This is the reason why two memristors, which are connected with reversed poles, are seen as depicted in **Figure 3**. Doing this, it does not play a role if the applied voltages are either negative or positive. In case an edge is detected, one of the memristors behaves always different to the other one and we receive as output the voltage that can be detected at resistor R_C.

The following specification in SystemC-AMS shows the code for a memristive fuse. Such a fuse has also a positive and a negative port like a single memristor. Therefore, it can be attached to an electrical network. Furthermore, as already shown in the example for a memristor, we need control signals, *memristor_control_one* and *memristor_control_two*, as discrete input signals to change the resistances of the variable resistors, *memristor_resistor_one* and *memristor_resistor_two*. These memristors are connected via an electrical node called *node*, which is defined in the constructor as well as the binding of their control signals *memristor_control_one/two* to their ports *memristor_resistor_one/two.inp*. Furthermore, the virtual circuit points *memristor_resistor_vout_one/two.n/p* are defined to measure the voltage at these memristors via the signals *memristor_resistor_vout_one/two.outp*. These signals allow displaying the voltages at both memristors

```
SC_MODULE(memristive_fuse)
{
 //negative and positive terminal
 sca_eln::sca_terminal n, p;
 sc_core::sc_in<double> memristor_control_one, memristor_control_two;
 sca_tdf::sca_out<double> memristor_resistor_voltage_one;
 sca_tdf::sca_out<double> memristor_resistor_voltage_two;
private:
 sca_eln::sca_node node;
 sca_eln::sca_tdf::sca_vsink memristor_resistor_vout_one;
 sca_eln::sca_tdf::sca_vsink memristor_resistor_vout_two;
 //two memristors
 sca_eln::sca_de::sca_r memristor_resistor_one;
```

```
sca_eln::sca_de::sca_r memristor_resistor_two;
SC_CTOR(memristive_fuse) :
memristor_resistor_one("memristor_resistor_one",1.0),
memristor_resistor_two("memristor_resistor_two",1.0),
node("node"),
memristor_resistor_voltage_one("memristor_resistor_voltage_one"),
memristor_resistor_voltage_two("memristor_resistor_voltage_two"),
memristor_resistor_vout_one("memristor_resistor_vout_one")
memristor_resistor_vout_two("memristor_resistor_vout_two")
{
//setup memristors
memristor_resistor_one.n(node);
memristor_resistor_one.p(p);
memristor_resistor_one.inp(memristor_control_one);
memristor_resistor_two.n(node);
memristor_resistor_two.p(p);
memristor_resistor_two.inp(memristor_control_two);
//setup voltage measurements for memristor one and two
memristor_resistor_vout_one.p(n);
memristor_resistor_vout_one.n(p);
memristor_resistor_vout_one.outp(memristor_resistor_voltage_one);
memristor_resistor_vout_two.n(node);
memristor_resistor_vout_two.p(p);
memristor_resistor_vout_two.outp(memristor_resistor_voltage_two);
}
};
```

After the definition for a pixel and a memristive_fuse, both these devices can be connected to construct the circuit shown in **Figure 3** by attaching one of the ports *p* or *n* of the memristive fuse to the port *neighbor* of a pixel node. The connection scheme for one pixel detecting the derivatives I_x and I_y in a 2D grid for a direct hexagonal neighbour connection is shown in **Figure 4**.

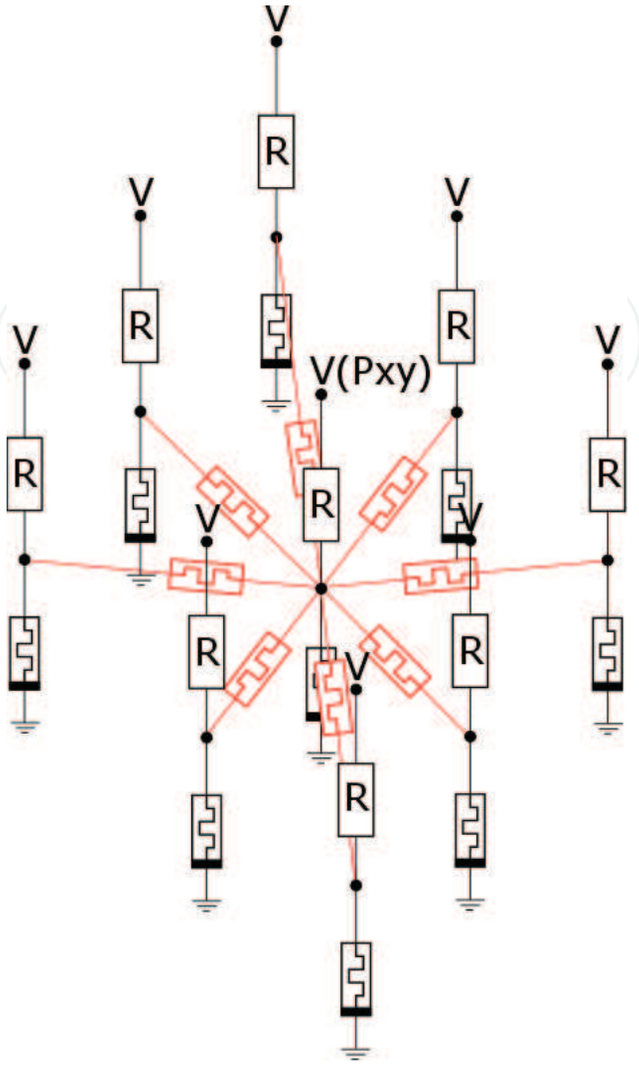

Figure 4. Scheme for the 2D memristive grid with X connection, that is, each pixel has eight connections to four neighbours in rectangular direction (left, right, top, bottom) and to the four diagonals.

This can also be specified in SystemC-AMS which we are unable to present in this paper due to reasons of clarity since the corresponding code is larger. Before we can move to the achieved simulation results, some things concerning the functionality of the network have to be explained before.

The electrical network constructed in this way fulfils several tasks. For example, before the optical flow processing takes place, a Gaussian filtering is carried out on the pixels, which is done by the OPL imitating memristive fuses, too.

An important thing that has to be avoided is that both memristors of a fuse have the same initial mid resistance = $\frac{R_{ON} + R_{OFF}}{2}$. In this case, the changes of memristances in both memristors countermand themselves. If the memristance of one memristors increases, the memristance of the other one decreases. A possible solution for this problem is that both memristors are initialized with a low resistance. In case of a given potential difference between two neighboured pixels, independent of its direction, only one memristor increases, whereas the other one's

memristance stays low and we can detect a corresponding voltage change at R_C corresponding to a given edge pixel.

The result voltage U_G is given to, it can be converted to a grey value x according to Eq. (5):

$$U_G = I_M \cdot \left(R_C + R_M\right) x = U_G \cdot \left(\frac{255}{40\ \text{mV}}\right) \tag{6}$$

Both things, initializing the memristors as described earlier and the grey scale conversion, are performed in our SystemC-AMS specification by appropriate instruction codes, for example, the calculation of the result voltage is carried out with the method *PixelNode::pixel_value* shown above in the class description of *PixelNode*. Besides the automatic filtering of neighboured input voltages, the network as described above allows the detection of edges, too, because edges are nothing else than potential differences. **Figure 5** shows the scheme for a potential propagation if the input voltage is applied left and the ground is applied right. Since this happens also for small differences very fast due to the memristive fuses, a threshold has to be introduced in order to detect real edges. The detection assigns a pixel only then as edge pixel if at least three of the neighbour pixels are above the threshold. This is directly programmed in the SystemC-AMS code which is not shown here. We have not seen a possibility to carry out such thresholding directly in the original analogue network published in Ref. [4].

So far, we have described a solution for determining the derivatives I_x and I_y within the 2D network and its SystemC-AMS equivalent. However, the optical flow requires the input and analysis of input data from two subsequent images in order to detect also the derivative I_t. Therefore, the network has to be extended in the third dimension and we have done that also in our SystemC-AMS specification. This is shown in **Figure 6** in a lateral view for two neighboured pixels located at the same coordinate in two subsequent layers which are connected in the same way as the lateral connections by an additional memristive fuse.

That means we connected together in SystemC-AMS two grids of the size 16 × 12 as shown for one pixel in **Figure 4**. The first gird hosted an image $I(x,y,t)$ and the second one the timely

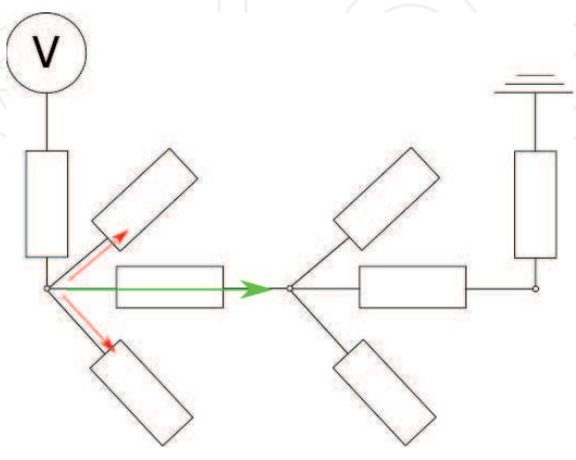

Figure 5. SystemC-AMS network for a detection of a moving edge pixel. The edge pixel disappears on the left (two short arrows) side and moves to the right (long arrow).

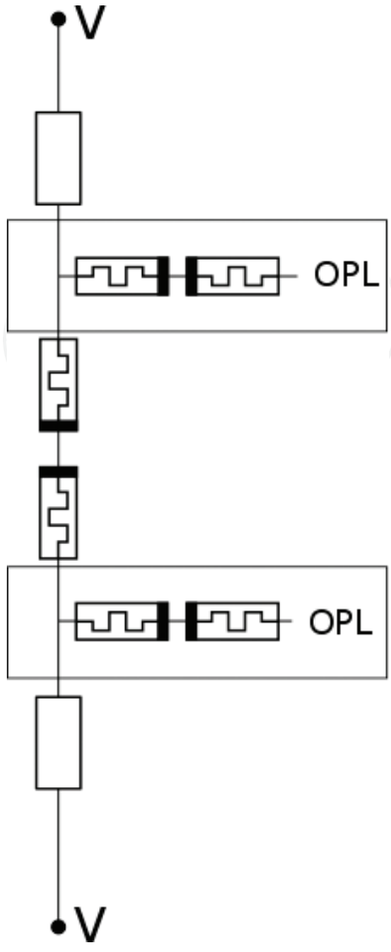

Figure 6. 3D connection of a pixel between two pixels neighboured in subsequent images.

displaced image *I(x,y,t+dt)*. A larger size could not be selected because the free SystemC-AMS version of Coseda did not allow generating more active elements. The SystemC-AMS code we tested extends more than 3000 memristors in all fuses and pixels for two images of size 16 × 12.

4. Results

Figure 7 shows the achieved functional results of the SystemC-AMS simulation for a traffic scene with the memristive network that works on the detection of moving edges with the memristive 2D network compared to a classical solution calculated according to Horn and Schunk on an Inteli5-6600 CPU. It is to recognize that the Horn and Schunk procedure algorithm works much better on a higher resolution, (a) versus (b), whereas the lower resolution is sufficient for the SystemC-AMS detection of moving edges (c). The low resolution was selected since this was the limit for the SystemC-AMS simulation with the proof-of-concept software solution. These moving edges are combined in one object. The grey edges are the disappearing edges, whereas the dark square corresponds to an appearing edge. The assignment

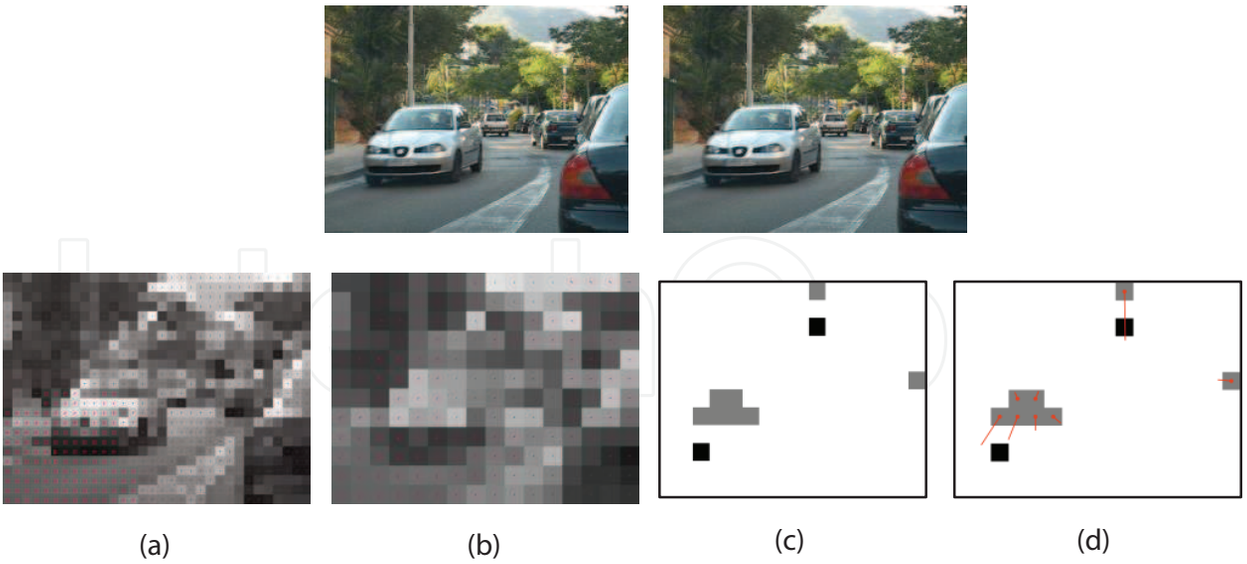

Figure 7. Used test input images (top), the two scenes are slightly displaced. On the bottom left side, (a) solution based on Horn and Schunck procedure calculated on CPU with 32 × 24 image size; (b) solution for 16 × 24 image size. On the bottom right side, the solution for the same scenes determined with the simulated memristive network in SystemC-AMS. Firstly, the detected edges (c), afterwards the detected directions of the moving edges (d).

between the two edges (see arrows in **Figure 7(d)**) can be identified and by this also the moving of the cars shown in front and of the smaller one shown behind in the image.

The results of **Figure 7** demonstrate that it is possible to detect moving objects with the memristive network and its SystemC-AMS model. We are now interested on how fast the network and the SystemC-AMS simulation work. The simulation of the detected moving edges in **Figure 7(c)** shows simulation results for a simulated memristive network for a time interval of 3000 ms. In this case, Biolek model was not used for the memristors but the model from Knowm which produced a significantly higher contrast. At the beginning, only a wave can be observed. After a simulated real time of 3000 ms, a higher contrast is given with that model compared to the input image and the moving objects can be detected. As comparison with existing hardware, we have determined the run times of the detection with the optical flow based on Horn and Schunk on a CPU (corei5-6600 corresponds to Intel's Skylake microarchitecture) and a Jetson TX1-embedded GPU board from Nvidia. The CPU could compute in 3000-ms image sizes of 160×120. It is to expect that the memristive network works also on higher resolution since it is a highly local parallel-processing scheme.

Therefore, the memristive network lies in the same range as the CPU concerning the compute performance. The situation is different compared to the GPU. We measured a time of about 100 ms for an image size of 640 × 480. Hence, the GPU has clear advantages versus the memristive network concerning the run time.

However, the actual interesting point in this paper is the simulation time of the SystemC-AMS specification. In order to get significant values we have carried out a series of possible optimization measures concerning the network topology and the monitoring, resp., the virtual voltage measuring during the simulation. **Figure 8** shows the simulation time

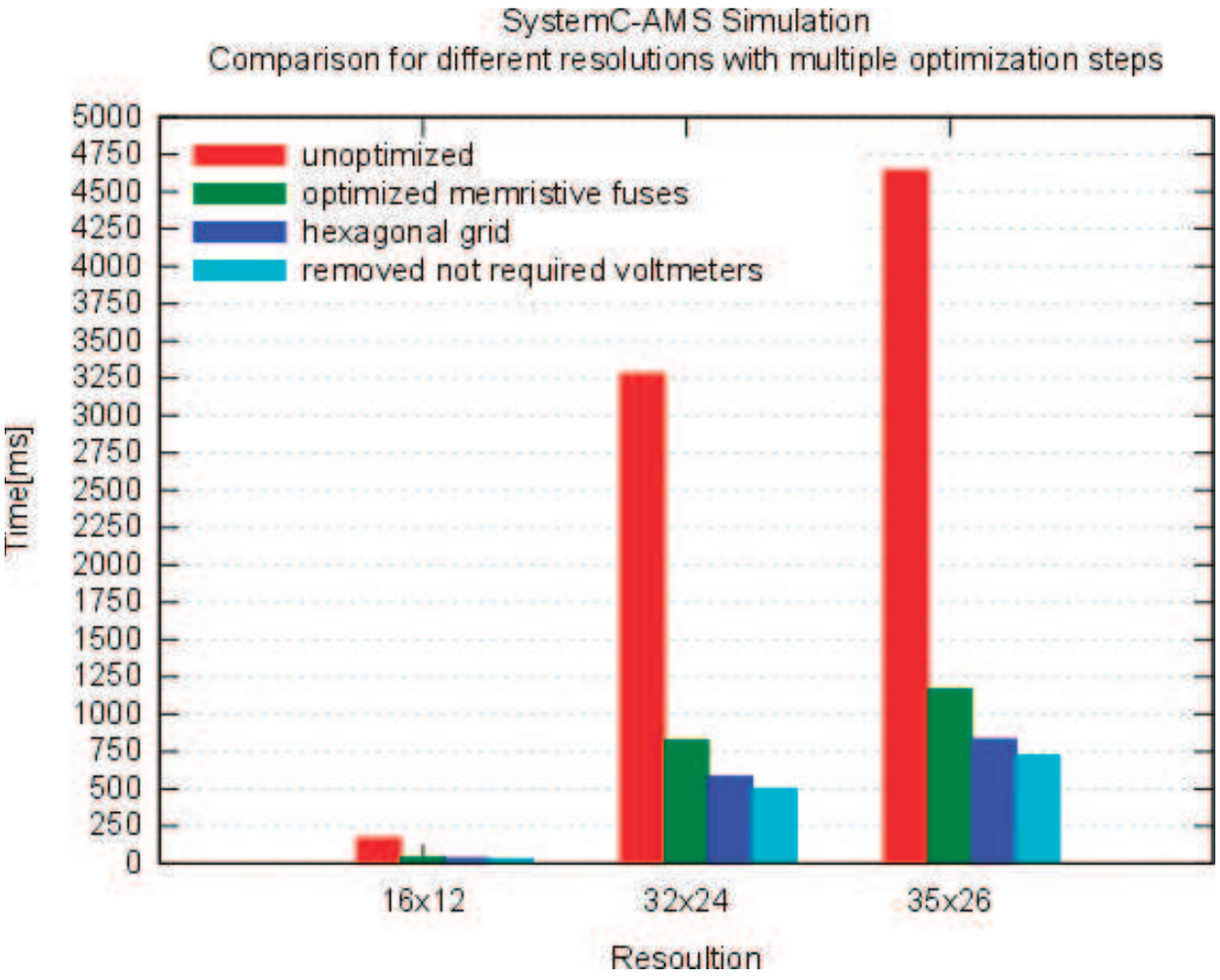

Figure 8. Measuring the simulation time in SystemC-AMS for different options.

for filtering and edge detection in one image for the following different sizes 16 × 12, 32 × 24 and 35 × 26 for a not optimized version (most-left bars—non-optimized), a version, which uses only one memristor in the fuses which is sufficient for edge detection as we found out (second left bar—optimized memristive fuses), using a hexagonal grid instead a 3 × 3 grid as local neighbourhood for a pixel (second right bar—hexagonal grid), and removing the voltage potentiometers for each memristor in the SystemC-AMS code (most right bar—removed not required potentiometers). It is to detect that simulation time can be drastically reduced, for example, for a 35 × 26 image from 4500 ms down to about 750 ms for the largest resolution of 35 × 26 if all optimization steps are applied subsequently.

Our efforts to carry out an equivalent SPICE simulation with LT Spice have been in vain. The LT Spice simulator ended in an endless loop by the trial to simulate this large memristive network. With the PSpice A/D Lite version, the simulation aborted orderly with the message that the symbol table entry is out of bounds. May be the commercial version of Pspice allows to simulate such a large amount of devices. In all, in the 32 × 24-sized grid 768 voltage sources, 1536 resistors and 5461 memristor subcircuits have to be simulated. Further work has been

done using Cadence Virtuoso. While Virtuoso was able to read the network and create a schematic view, it was not possible to start the Spectre simulation due to incompatibilities using the memristor Spice description.

In an older work [7], a similar 32 × 32 array of memristors was simulated in SPICE. Recently, Biolek et al. published a work [8] in which they used a parallel version of a commercial HSPICE simulator which allowed them to simulate extremely large memristor networks. They managed it to simulate a 100 × 100 memristive grid network containing 20,200 memristors in 5.5 s and a 1500 × 1500-sized memristive network containing 4.5 million memristors in 76 min by applying a modified version of the so-called S-model for memristors on a current Intel core i7 architecture.

5. Conclusion

Exploiting the flexibility of a high-level language like SystemC-AMS, the presented simulation environment enables designers to carry out extensive investigations on large memristive circuits to estimate latency and energy consumption just by simple C++ code modifications. Furthermore, such a system allows the simulation of thousands of connected memristors at acceptable simulation times, which is shown by a direct comparison to an equivalent SPICE simulation. A SystemC-AMS description allows faster simulation, but currently the investigated SystemC-AMS implementations do not allow the simulation for networks concerning more than 10 k memristors. Therefore, there seems to be a need for action concerning an extension of SystemC-AMS environments in the future. On the other side, free available SPICE versions failed to simulate memristor networks in the size of 1000 s, whereas the presented SPICE-AMS implementation could handle it in acceptable simulation time of 4–5 s and around 1 s for an optimized version. However, compared to commercial HSPICE simulators only smaller-sized networks of memristors can be investigated. On the other side, a SystemC-AMS solution simplifies a coupling to digital system layers to realize mixed-signal simulations. We demonstrated this flexibility in principle in this paper for the optical flow algorithm.

For the optical flow example, a comparison of a memristive network with real processor architecture like a GPU was carried out. It could be shown by simulations that using a GPU architecture is more efficient for the optical flow problem than a 2D grid memristive network solving the problem by detecting moving edges. The performance of a current CPU solution on the other side offers not more compute power than the memristive network which probably requires less energy consumption than the CPU. At all, we think that mixed-signal solutions are to favour, which combine analogue memristive circuits with digital processors, to unite computational flexibility and the benefits of energy-saving neuromorphic analogue memristor networks. A SystemC-AMS-based simulation environment is generally well suited for the design of such architectures and to estimate the required power and processing time. Our solution laid the foundation for such work in the future.

Author details

Dietmar Fey*, Lukas Riedersberger and Marc Reichenbach

*Address all correspondence to: dietmar.fey@fau.de

Friedrich-Alexander-University Erlangen-Nürnberg (FAU), Computer Architecture, Department of Computer Science, Erlangen, Germany

References

[1] Kvatinsky S, Friedman EG, Kolodny A, Weiser UC. TEAM: ThrEshold adaptive memristor model. IEEE Transactions on Circuits and Systems. 2013;**60-I**(1):211-221

[2] Biolek Z, Biolek D, Biolkov V. SPICE model of memristor with nonlinear dopant drift. Radioengineering. 2009;**18**(2):210-214

[3] Nugent MA, Molter TW. AHaH computing–From metastable switches to attractors to machine learning. PLoS ONE. 2014;**9**(2):e85175. DOI: doi:10.1371/journal.pone.0085175

[4] Lim CKK, Gelencser A, Prodromakis T. Computing image and motion with 3-D memristive grids. In: Adamatzky A, Chua L, editors. Memristor Networks. Cham: Springer International Publishing; 2014. pp. 553-583

[5] Coseda Technologies GmbH. SystemC AMS Proof-of-Concept Download [Internet]. Available from: http://www.coseda-tech.com/systemc-ams-proof-of-concept [Accessed: 20-March-2016]

[6] Horn BKP, Schunck BG. Determining optical flow. Artificial Intelligence. 1981;**17**

[7] Wang Y, Fei W, Yu H. SPICE simulator for hybrid CMOS memristor circuit and system. In: 13th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2012); August 29-31, 2012; Turin, Italy: IEEE; 2012. pp. 1-6

[8] Biolek D, Kolka Z, Biolkova V, Biolek Z. Memristor models for spice simulation of extremely large memristive networks. In: IEEE International Symposium on Circuits and Systems (ISCAS); May 22-25, 2016, Montreal, QC: IEEE; 2016. pp. 389-392