Modeling and Synthesis of mid- and long-term Future Nanotechnologies for Computer Arithmetic Circuits

Bruno Kleinert and Dietmar Fey
Chair of Computer Architecture, University of Erlangen-Nürnberg, Germany
{bruno.kleinert,dietmar.fey}@cs.fau.de

***Abstract*—The paper presents a comparison between two future nanotechnologies that are suitable for arithmetic computation and non-volatile memory. An automatic synthesis procedure of an optical computing design principle onto long-term future Quantom-dot Cellular Automata (QCA) is presented. The goal of this work is to provide a contribution for the elimination of the lack of automatic design procedures for regular build-up QCA arithmetic circuits. A SystemC model of the mid-term future memristor technology is presented, to demonstrate the benefit in space efficiency as a four-value logic memory in a fast signed digit (SD) adder for a hardware implementation of the coordinate rotation digital computer (CORDIC) algorithm. A comparison between QCA and memristor technology presents the advantages of memristors in multi-value logic environments. In this sense, this work is a contribution to ease the automatic synthesis and choice of future nanotechnologies for arithmetic circuits.**

Keywords*—*Nano computing, Memristor computing, Optical Computing, Quantum-dot Cellular Automata.

I. Introduction

Modern computing devices, like processors or Systems-on-a-Chip are getting more and more powerful. Further raising clock frequencies but also energy-saving requirements for embedded and handheld devices, like smartphones and tablet PCs, push the state-of-the-art CMOS technology to its limits, concerning data throughput and manufacturing densities. At the moment of this writing, classic CMOS technology is close to frequency and density limit and new computing and memory technologies need to be developed and researched. A common answer on how to continue in the post-CMOS era, are nanosystems, that are predicted to allow higher manufacturing densities, like self-organization processes, higher clock frequencies and better energy efficiency [1].

Therefore, we investigate two different promising and complimentary nanotechnologies, each of which offer new possibilities for the design of arithmetic circuits in the post-CMOS era. This is, on one side, a mid-term solution, based on new storing capabilities, namely memristor technology, which offers to store multiple different values in a single storage device. This new feature, that is not offered by CMOS memory devices, can be exploited to speed up arithmetic circuits based on signed digit logic, which is not efficiently possible with current technology. On the other side, there is another new nanotechnology, that is to be considered as a long-term alternative, the Quantum-dot Cellular Automata (QCA) [2]. This technology is characterized by the potential of extremely low-power consuming logic cells, based on single electrons entailed in quantum-dots and a possible high-dense arrangement of such cells.

Both technologies lack support by design tools, which is obvious since they are new technologies. Therefore, we contribute in this paper for a removal of this lack. To support an automatic synthesis of arithmetic logic in QCA circuits, we identified an analogy to another unconventional computing technology: Symbolic Substitution Logic (SSL) [3]. It comes from optical computing and shows a lot of similarities concerning regular setup on pixel, respectively QCA cell processing schemes, that can be used to adapt SSL design techniques to synthesize QCA circuits. On the other side, we have the much more mature memristor technology [4] that can be compatibly manufactured with CMOS circuits. Therefore, we consider it worthy, to research on modeling techniques on the digital level, to allow the integrate memristors in an adequate manner to conventional CMOS circuits. We chose the SystemC modeling language for that purpose as it offers enough flexibility, to model the properties of multi-value memristor-based memory with appropriate data structures.

In this paper, we compare both nanotechnologies in the context of automatic design patterns and simulation of basic circuitry to derive building blocks that can be used to build complex logic circuits. We successfully applied Symbolic Substitution Logic (SSL) as a regular design pattern on QCA and present an abstracted prototype model of a memristor for SystemC digital system simulations. We identified challenges for the development of hardware design and synthesis tools to be reusable for the development of memristor-based systems and later on for QCA technology based systems.

The rest of the paper is organized as follows. In Section II, we present the basic principles of digital optical computing based on SSL. In Section III, we present the basic principles of digital optical computing based on SSL. In Section IV, we explain nanotechnology information processing based on QCA. In Section V, we present the mapping process between SSL rules and QCA cells for the example of one stage of a bit-serial QCA adder deduced from an SSL adder. Details and possibilities with memristors are presented and described in Section VI. In Section VII, we present and explain our abstract model of a memristor for digital circuit simulations. Section IX concludes our findings and points out future work.

II. Symbolic Substitution Logic

Symbolic Substitution Logic (SSL) was invented by Brenner et al. [5] in 1986 as a new method for the design of optical computing circuits. It was exactly tailored to the constraints and possibilities of a high-dense pixel parallel processing offered by optical hardware. The idea behind SSL is to search for a certain binary pattern within a binary pixel image and to replace the found patterns by another pattern. This substitution process can be exploited to realize a digital arithmetic in a highly parallel manner. The key features of SSL are characterized by their strong regularity concerning the pixel processing and the focusing on operating on elementary binary information cells, namely pixels, arranged in a grid structure.

In particular, this situation is also given in Quantum-dot Cellular Automata (QCA) [6]. QCA is one of the promising nanotechnologies besides carbon nanotube field effect transistors and further nanodevice technologies based on tunneling effects that are considered as candidates for a new device technique to realize logic circuitry in the post CMOS area. Analogue to an optical computing scheme like SSL QCA are characterized by a highly dense implementation of binary information cells and a regular information flow. Whereas the elementary binary information cell in SSL was a pixel, which is either bright or dark, the binary information cell in QCA corresponds to two electrons, which are arranged in two distinguishable directions in a four dot quantum cell.

In literature, a really large number of proposals for QCA arithmetic circuits can be found, which have been developed largely manually (e.g., [7], [8]). However, there is still a lack of design methodologies that can be used for an automatic design process of arithmetic circuits based on QCA. There is an exception presented in [9], which proposes a methodology how to convert Boolean sum-of-products in an algorithmic way to QCA logic, in particular to QCA majority gates, which is the basic gate structure in QCA (see Section IV). However, most of the QCA arithmetic circuits are still developed in a time consuming try-and-error process by hand.

On the other side there was a lot of research in the 1980s and 1990s in the Optical Computing community on SSL (e.g., [10], [3]), which brought numerous proposals for digital optical computing circuits based on the basic SSL logic building block, the so-called SSL rule (see Section III). Due to this fact and the similarities given in the kind how elementary information is handled in QCA and SSL, we present in the following sections on-going research on developing strategies how SSL rules can be used for an automatic mapping process onto QCA circuits, which can be used in future design tools.

III. Optical computing with SSL

SSL [10], [3] has drawn a lot of attention during the 1980s and 1990s as a method for exploiting the space invariance of regular optical imaging systems for the set-up of digital optical hardware. The base of information processing in an SSL is the implementation of a so-called SSL rule. An SSL rule depicts a pattern substitution process and consists of two parts, a left-hand side (LHS) and a right-hand side (RHS) pattern (see Figure 1). By a corresponding optical hardware each occurrence of the LHS pattern is searched within a binary image and is replaced by the RHS pattern. Figure 2 shows schematically a possible optical set-up for the search process as it was frequently realized in SSL hardware demonstrators. The principle processing works as follows.

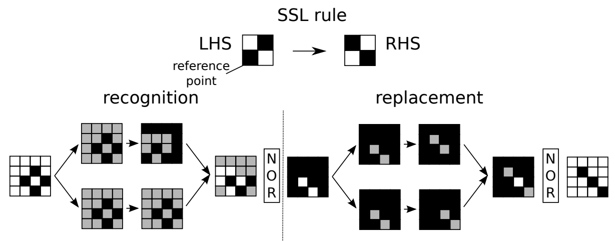

Fig. 1. Principle of SSL

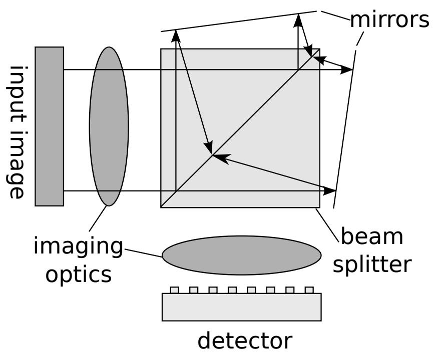

Fig. 2. Implementation of SSL with optical hardware

For each switched-off pixel, i.e., a black pixel, in the LHS of an SSL rule a copy of the image is produced, e.g., by a beam splitter. Furthermore, a reference point is defined within the LHS pattern, e.g., the lower left corner pixel. Each of the copies is reflected, e.g., by tilted mirrors, in such a way that the copies are superimposed and pixels, which have the same relative position to each other as defined in the LHS pattern, meet at the same location.

For the example of Figure 1, this means that one copy of the image is not tilted since it corresponds to the set pixel in the LHS pattern, which is already localed in the reference point. Whereas the other copy is shifted by the tilted mirror, such that each pixel in the copy of the input image is shifted one pixel position down and left. At each position, where two dark pixels meet, an occurrence is given of the LHS pattern in the original input image. The superimposed image is mapped onto an array of optical threshold detectors. Each detector operates on one pixel of the superimposed image as a NOR device. The detector output is used for switching on a LED or laser diode. As a result, one gets a high light intensity at each pixel position, which corresponds to the occurrence of the LHS search pattern in the input image. We denote this new image as a detector output image.

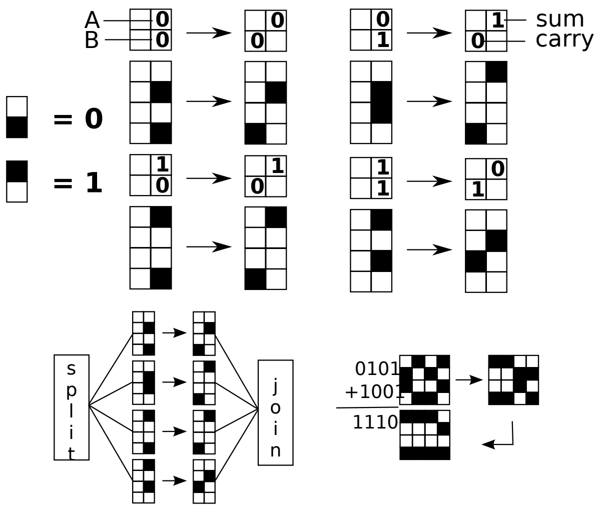

Fig. 3. Realization of a ripple carry adder with SSL. For reasons of improved robustness a dual rail coding is used for 0 and 1.

The recognition step is followed by a replacement step, which works analogue to the recognition step but in opposite direction. For each switched-on pixel in the RHS pattern a copy of the detector output image is again produced by optical beam splitter hardware in such a way that the copies are shifted towards the switched-on pixel in the RHS pattern. This means for the example of Figure 1 that two copies from the detected output image are generated and each of the copies are shifted one pixel up resp. right before superimposing the copies. Once again, the superimposed image is mapped onto a pixel-by-pixel operating NOR detector and LED/laser diode array. The reproduced output is a new image, in which each occurrence of the LHS pattern in the original input image is substituted by the corresponding RHS pattern.

Implementing appropriate SSL rules by splitting the input image into multiple optical recognition and replacement paths, which are applied simultaneously and joined at the end, have been used for the proposing and realizing of digital optical computer arithmetic circuits. Figure 3 shows this schematically for an optical ripple carry adder based on SSL. A large number of further arithmetic circuits using SSL or similar techniques like optical shadow logic [11] have been published in the past for optical adders, multipliers or image processing tasks. All these proposals can be used to transfer them to QCA due to the similarities between SSL and QCA we outlined above.

IV. Nanocomputing with QCA

The elementary information cell in a QCA is a kind of container that groups a few quantum dots, at which charged particles, i.e., electrons, are fixed (see Figure 4). Mostly a QCA cell consists of four dots, in which two electrons are grouped in opposite order. Consequently, the cell knows exactly two polarization adjustments, which are assigned to the binary values 0 or 1. Due to quantum mechanical rules it is possible that a cell can switch between the two states by tunneling of the charged particles between the dots. Concerning the two different particle arrangements one distinguishes between type 1 and type 2 cells (see Figure 4).

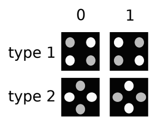

Fig. 4. Binary coding in QCA cells. White circles correspond to empty quantum-dots, gray ones represent dots occupied with electrons.

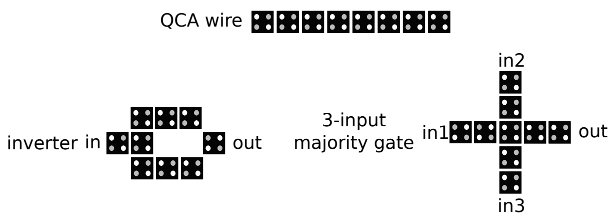

Fig. 5. QCA logic building blocks

A QCA cell serves not only as an information storage cell but also as a transport cell since neighboring QCA cells interchange by Coulomb forces. This means that a cell, which is fixed to a certain polarization, transfers its state to a neighboring cell because this arrangement shows the minimum electrical field energy between neighboring particles of the same charge. Consequently, a QCA wire can be built up, in which information is transported not by an electric current flow but by subsequent reordering of the quantum states in neighboring QCA cells. Due to the fact that no current is flowing and due to the small dimensions of a QCA cell, this technology offers very low power dissipation. Besides information transport one also needs logical gates to realize computing circuits. QCA logic utilizes an inverter and a so-called majority gate for this purpose. Figure 5 shows an inverter built with cells of type 1. In both circuits, the output cell adopts the opposite state of the input cell state, again due to Coulomb forces. In contrast to CMOS circuits, QCA gate logic is not based on the switching of parallel and serial connected transistors but on the states of the cells surrounding a certain QCA cell, serving as output cell of the gate. The majority of the states in these surrounding cells determines the state of the output cell. In Figure 5, a 3-input majority QCA gate is shown. The output cell adopts the same state, which at least is stored in two of the three neighboring cells. By fixing one of the inputs to a certain polarization 2-input AND, OR, NAND and NOR gates can be built.

Based on these three building blocks, QCA wire, QCA inverter and QCA majority gate, various proposals exist in literature for different typical digital circuits like adders, multipliers, shifters, multiplexers and registers, which have been found in a more or less try-and-error procedure. A very impressive collection of computer arithmetic QCA adder and multiplier circuits can be found in the work made by Hänninen [12]. The solutions proposed in this work are distinguished by their regular set-up that helps to realize QCA cells in the future.

This is an important feature since QCA technology has a long-term perspective concerning its realization with real hardware.

Also design tools, which support the automatic synthesis of regular built-up QCA circuits, will encourage and give hints to device technologists how QCA technology should develop in the best way. In this sense, we propose to use optical computing SSL design procedure as a design entry point for the systematic design of nanocomputing QCA logic. How this mapping can be done is presented in the next section.

V. Mapping SSL rules to QCA logic

The procedure to map SSL logic to a regular built QCA layout is subdivided in three steps. These steps correspond (i) to the core of the logic circuitry, namely the synthesis of an SSL rule into an equivalent QCA circuit, (ii) the realization of the splitting process, because we want to realize systems, which apply multiple SSL rules simultaneously, and (iii) the realization of the join at the end of the recognition-substitution stages. We will demonstrate the generic approach for these mapping steps in the following subsections without loss of generality on the example of the ripple carry adder from Figure 3. Furthermore, we will use this example also to show generic applicable optimization measures for mapping SSL rules, which saves otherwise necessary QCA logic resources.

A. *Mapping the split stage to QCA cells*

As shown in Figure 3, the applying of multiple SSL rules starts with a split function. The mapping of the split stage onto QCA logic can be done in a straightforward manner. Producing copies of input cells can be simply done with branches of QCA wires running orthogonally to the input QCA wires. If one has to copy more than one input, as for example for the LHS rules in a ripple carry adder, one has to observe that crossing branches can interchange without conflicts. This can be done by crossing lines between QCA cells of type 1 and type 2. To connect both types of cells, a QCA cell has to be shifted by half height of a cell (see Figure 6, part split).

B. *Mapping SSL rules to QCA cells*

The mapping of SSL rules onto equivalent QCA layouts is divided in two substeps, (i) the mapping of the recognition step and (ii) the mapping of the replacement step. The recognition of an LHS of an SSL rule is mapped to an equivalent QCA majority gate realizing an appropriate AND gate. The number of inputs of this AND gate depends on the number of values in the LHS. For example, the number of relevant inputs for the rules of the ripple carry adder is two. This means that a three-input QCA majority gate can be used, if one of the three inputs is fixed to 0 (see Figure 6). For rules with a higher number of input values an appropriate majority AND gate has to be used. A lot of solutions for QCA gates with more than three inputs can be found in literature, e.g., in [13] an optimized solution for a five-input majority gates is presented. If the value in the LHS is 0, then an inverter has to be included in the path of QCA cells that leads the input value corresponding to the LHS entry to the input of majority gates. The output of the majority

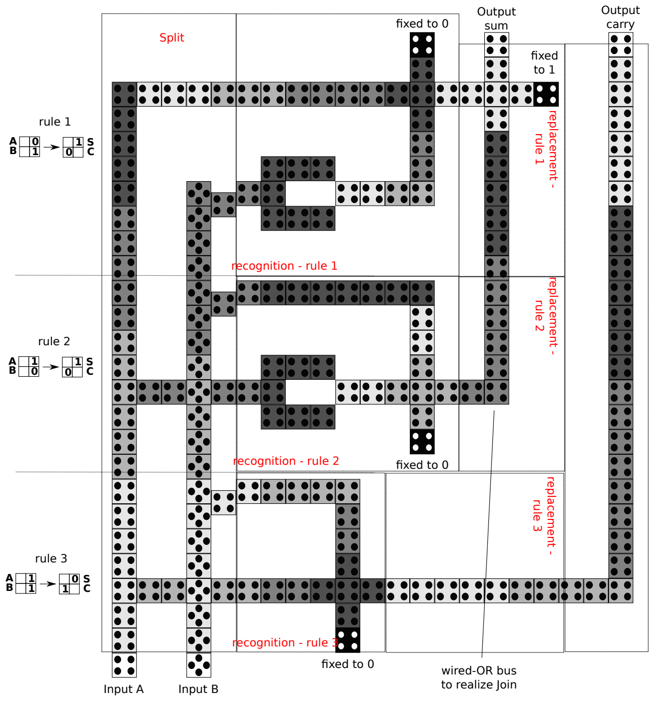

Fig. 6. Result of the mapping process of SSL logic onto QCA logic for the ripple carry adder. To synchronize the changing of QCA cell states' four different clock zones have to be defined. In the figure these four clock zones are marked with a different gray level in QCA cells. Electrons are not shown in this figure.

cell is exactly 1, if the LHS pattern is detected. In this sense the majority gate works analogous to the photo detector NOR device used in SSL (see Figure 1).

The following explanations correspond to the replacement stage in SSL. If the output of the majority gate is 0, then 0's are produced for all 1 values in the RHS of the corresponding SSL rule. If the output of the majority gate is 1, i.e., the LHS pattern was detected, a 1 is produced for each value 1 given in the RHS by an additional majority gate operating as an OR gate (see Figure 6, majority gate in replacement part with one input fixed to 1). If the RHS is 0 no majority gate is necessary since we will work with wired-OR buses in the join stage that carry already the 0 value, which is possibly inserted by the replacement stage located at the lowest position in the wired-OR bus (see rule 2 in Figure 6).

C. *Mapping the join stage to QCA cells*

As just mentioned the principle of the join stage in SSL is the realization of an optical wired-OR. The same idea was pursued for the equivalent QCA logic. If a 1 has to be inserted in the wire due to a relevant 1 from an RHS, which is output from a replacement stage, this can be done with 3-input majority gates with one input fixed to 1 (see Figure 6, wired-OR bus in block rules 1). In this case, a 1 is only injected in the wire if the output of the attached recognition stage to the wire is 1 or the third input coming from the wire is already 1. This functioning corresponds exactly to a wired-OR bus. A logical

1 is injected if an LHS was found and a 1 in the corresponding output of the RHS is given. If the detected rule requires a 0 in the RHS this is automatically given by the fixed injection of a 0 in the QCA wire by the lowest replacement stage attached to the wired-OR bus. If the rules are not in conflict, i.e., only the LHS of exactly one rule was found, then only the output of the RHS belonging to the LHS is injected. This can be either a 0 or a 1. If it is a 0 an explicit injection is not necessary. This causes that rules, which have only 0's in the RHS, must not be implemented if it is secure that exactly one of the rules is always valid. This is given for the case of the ripple carry adder. Therefore, the rule corresponding to (A,B)=(0,0) has not to be implemented with corresponding QCA cells. If it is the only rule that holds, then the corresponding 0's in the output are already on the QCA wires. Utilizing this a priori knowledge the requirements to QCA hardware can be optimized during the synthesis process from SSL logic to QCA logic.

VI. The memristor

In 1971, [14] stated that there must exist a fourth basic circuit element, that he called the memristor. He derived the word memristor from memory + resistor = memristor, a two-connectors circuit element, that works as an adjustable resistor. The resistance value is "memorized" by memristors without the need of energy.

Though this element was predicted to exist in 1971, several years passed by until an operational memristor was successfully built for the first time in the HP laboratories [4]. HP is the current leader in building nanoscale 3D layered memristors [15] and is still ongoing in research about this basic circuit element. At the moment of this writing, literature states that memristors can be built in size of down to 3nm^2, which theoretically gives very promising densities of non-volatile memory. Industry is currently working on the manufacturing of memristor-based memory chips as drop-in replacement for flash memory.

Not only the resistance of a memristor can be used to control the flow of a current, the resistance value can be used to store information. When certain information is mapped to a certain resistance value, information can be stored in memristors, by setting a memristor to this resistance. The memristor then keeps this resistance value, i.e., the information, until is is later "read-out". Reading out means to find out the current resistance value, to which the memristor was previously set. Due to the mapping, the stored information can be obtained from the resistance of a memristor.

E.g., mapping 0 to the lowest possible resistance and 1 to the highest possible resistance, binary information, as known from current computers, can be stored. As it is possible to set a memristor to an arbitrary resistance value, not only binary information can be stored, but also multi-value information or encodings. E.g., when the range of resistance from lowest to highest resistance is divided into 10 specific resistance values, the values 0 to 9 could be stored.

Another area in which memristors can be used, are programmable nano-scaled crossbars. Crossings of nanowires, e.g., assembled from carbon nanotubes, can be connected or disconnected in a switchable manner by the layered 3D memristors from HP [15] with connectors on the bottom and at the top of each memristor.

Not only switchable connections or non-volatile memory can be build from memristors, but also computing, by building basic gates, is possible. [16] describes how the very basic, in CMOS technology widely used, NAND gate can be copied with a circuit build from three memristors. As for QCA, this also means for memristor-based computers, that computing unit and memory meld together.

VII. Memristor-based circuit simulation

Newly developed circuits are typically simulated before expensive prototypes are produced. This should also apply to future circuits that are based on memristors. Ideally, the use of new technology should be completely transparent for hardware developers. Though, it is possible to build arithmetic circuits, computation logic and memory from memristors in theory, their characteristics affect the design process of the whole system. I.e., they can not transparently replace transistors in existing circuits.

To simulate memristors in certain circuits, we used the SystemC hardware modeling language. We chose this language to analyze simulations of digital systems that make use of memristors and its challenges. SystemC has the advantage to be quick and easy to use and does not require a large toolchain assembled from a variety of different software tools.

A. *Abstraction of analogue behavior in digital simulators*

As presented in the previous section, we use SystemC to develop simulations of digital systems that are among others build from memristors. In our first step, we use memristors as a register.

Without going too deep into detail, which can be found in other literature (see [17]), we will only explain those characteristics of memristors in this paper, that are relevant to our research. The resistance of a memristor is set by (i) a current flow through the memristor, (ii) the time interval of this current flow and (iii) the polarization of the current. I.e., a higher current and a longer time interval of that current change the resistance in a greater amount in contrast to a low current for a short time interval.

To "read out" the resistance value of a memristor, [17] suggests to apply an alternating-current to the memristor. An alternating-current has the advantage that the resistance of the memristor does not get changed, disregarding a small delta, as the alternating-current, flowing through the memristor changes its resistance value up and down by approximately the same amount. By the help of comparators the current resistance value of a memristor can be obtained.

As we want to work with digital simulators, this analogue behavior has to be abstracted to model a memristor. As mentioned above the write procedure needs a current and time. A current can not be modeled in any way in a digital simulation, as a result we propose to drop it from the model. Though, the time interval in which current has to flow through the memristor can be modeled. We propose to require time intervals in the memristor model from literature and use the clock frequency of the simulation as a reference. We propose that the input value has to be applied to the memristor, i.e., its model, for exactly as many clock cycles as necessary. If the input signal is applied for a shorter or longer amount of time, the memristor should store a lower or higher resistance value, as it would happen in reality.

Another challenge that memristors introduce, is that they should not be "blindly" set into a new state of resistance as this could burn the device if a high current flows through the memristor when it is in a low resistance state. This is especially important for newly produced memristors, as their initial resistance value is almost unpredictable, due to tolerances during manufacturing. For simulated memristors we suggest to set randomly picked resistance values to them during the initialization phase at the beginning of a simulation run. By doing so the unpredictable state of a new memristor can be simulated.

Furthermore, we suggest to use the simulators debug capabilities to display warnings about erroneous behavior to the user, to point out errors as obvious as possible. As the complete field of applications for which memristors can be used are unlikely to be ever known, simulators should not decide whether an access to a memristor at a certain point in time should lead to an error or not.

B. Impact on real circuits

In Section VII, we propose an analogue read-out circuit, to obtain the current resistance value of a memristor. This analogue circuit is hidden, i.e., not visible to the designer, in a high level hardware description language, like VHDL, Verilog or SystemC. When a hardware description is synthesized, these analogue read-out circuits have to be added implicitly to the later real hardware, which, of course, has extra costs of energy consumption of these chips and the required extra space on them.

We propose that hardware development software and analyzation and debugging tools have to be made aware of the extra added read-out circuits, otherwise the analogue circuits have to be added in a by-hand process which is typically erroneous. This gives hardware developers the ability to better understand and analyze memristor-based circuits before first prototype samples of chips are produced.

C. Memristors as memory in four-value logic

Multi-value logic memory is a promising field to build space efficient memory arrays. To demonstrate a possible use case for multi-value, i.e., four-value memory in our case demonstration, we chose the CORDIC [18] algorithm as an example. For this algorithm, a successive multiplicity of additions have to be computed. To obtain high performance, we do not use a binary representation of addends, but a signed digit (SD) logic representation, for fast signed digit adders. The high performance is achieved as no carry bits have to be computed in SD adders. Though the conversion from a number in SD representation to binary representation is expensive, this will not affect the overall performance of the CORDIC implementation significantly, as the repeated additions outweigh the expensive conversion.

TABLE I. BINARY ENCODING OF SD DIGITS

SD	Binary
-1	10
0	00
0	11
1	01

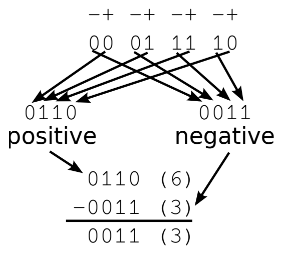

Fig. 7. Conversion from an SD number into a binary or decimal number.

TABLE II. MAPPING OF 4-VALUE LOGIC

resistance	SD	Binary
lowest	-1	10
low	0	00
high	0	11
highest	1	01

In SD logic each digit can take the values -1, 0, and 1. These digits are assembled of a positive and a negative *weight*. To obtain the decimal value of a SD number, the negative weight has to be subtracted from the positive one. As a result, a value of 0 can be composed from 0 negative and 0 positve weight, or from 1 negative and 1 positive weight. I.e., in SD logic exactly 4 values or states are necessary to store an SD digit. To encode one SD digit in binary, two bits have to be used to encode its value. Table I depicts the binary encoding of SD digits. The higher bit stores the *negative weight*, the lower bit stores the *positive weight*. To convert an SD number into binary or decimal representation, the negative weight has to be subtracted from the positive weight of the whole number as shown in Figure 7.

Memristors can be set to an arbitrary resistance value. We take advantage of this capability and define four resistance values for our case demonstration as follows:

- lowest resistance
- low resistance
- high resistance
- highest resistance

We map these for states to the SD values -1, 0 and 1. Of course the difference in resistance between each pair of encodings should be large enough to avoid faulty read-out results, that would lead to a misinterpretation. The four mappings are necessary, because there are two valid representations of the value 0 in SD logic, which can be expressed in a binary encoding as 00 and 11. As a result a possible mapping is shown in Table II.

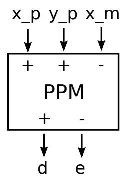

Fig. 8. PPM cell with its three input and two outputs connectors.

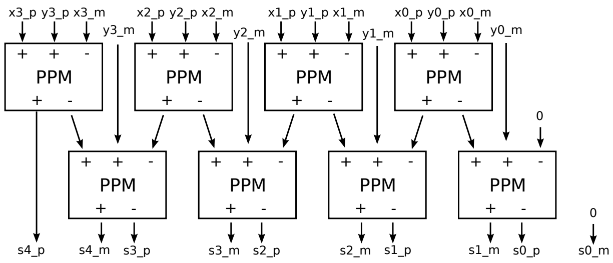

Fig. 9. 4 digit SD adder built from PPM cells.

D. SystemC memristor implementation

In this section, we describe our implementation in SystemC of the necessary modules (circuit components) to evaluate the use of memristors as memory or registers in fast SD adders. This adder should later be used in a simulation for a DSP that implements the CORDIC algorithm to compute trigonometric functions.

For our case demonstration, we implemented the SD adder as presented in [19]. This SD adder is assembled from so-called Plus-Plus-Minus (PPM) cells and its advantage is, that addends can be SD numbers, but also regular binary numbers with 0 as negative weight in all digits. A PPM cell is depicted in Figure 8. Its three inputs are from left to right the positive weight of the addend x (x_p), the positive weight of the addend y (y_p) and the negative weight of the addend x (x_m). The outputs are a positive weight d and a negative weight e, whereas d is computed by

$$d = x\_p \cdot y\_p \vee x\_p \cdot \bar{x}\_n \vee y\_p \cdot \bar{x}\_n$$

and e is

$$e = x\_p \oplus y\_p \oplus x\_n$$

(see also [19]).

The whole SD adder is assembled from two PPM cells per digit, e.g., for a four digit SD adder eight PPM cells are necessary. To perform the SD adding, the PPM cells have to be connected as shown in Figure 9 to build a 4 digit input SD adder (see also [19]).

Since the memristor is an analogue circuit element, we can not implement it in classic SystemC directly, and, as already mentioned in Section VII-A, it is not of interest to our research to obtain an accurate analogue simulation model, but a digital equivalent.

We implemented the memristor as a SystemC module. Its interface has three input signals and one output signal, whereas the input signals are a clock signal *clock*, a boolean input signal *w_en* to signal a "write" access and an input signal *in* of type uint8_t that characterizes the resistance value to be stored in the memristor. Internally the memristor module stores its state in a private member *state* of type uint8_t. Another private member is a boolean lock variable *lock*, its use is described later in this section.

At first glance the clock input signal might seem unnecessary as the memristor is not a naturally clocked circuit element, but as mentioned in Section VII-A, an input value should only be stored correctly if the input that is to be stored is constantly available for a certain time interval. Otherwise, the memristor should not store the resistance value or a different one from what was set at the input. The clock is used to trigger an incremental counter on each rising clock edge. It allows the memristor module to observe if the input signal is available unchanged for the correct time interval, that corresponds to the value to be stored, i.e., for the correct number of clock cycles.

Our SystemC model of a memristor implements an endless loop in a SystemC thread, which immediately blocks and gets woken up every rising clock edge. The loop checks if the write signal is set to *true* and if the lock is free. If that is the case the lock is taken and the loop blocks for 5ns with the SystemC *wait()* instruction. After waiting for 5ns the input value is copied into the internal state variable *state* and the lock is released so that the resistance value of the memristor can be changed again. As we propose in Section VII-A, a warning is displayed during the SystemC simulation, if the input is changed while waiting for 5ns or if the write signal gets set to *false*.

For a sole digital system, it is sufficient to use type bool, sc_bit or sc_logic for the input signal of the memristor. Though, we want our model to be able to store four-value logic but also very fast simulation for very large memristor-based systems in the future. The SystemC documentation states, that users should use C++ data types, where possible if one wants to achieve high speed simulations. For that reason, we chose uint8_t to store more than only binary information and to use a C++ primitive data type for good simulation performance in the future.

The choice of the input data type affects the way, how to use our memristor model in circuit models. A typical hardware description use boolean or sc_logic types that store and transport binary information. To attach our model to such a circuit a transformation between digital logic and the memristor inputs and outputs has to be performed. For this purpose we implemented two connector modules to fulfill this requirement for four-value logic: A conversion module that takes two-wire binary input and is to be connected to the memristor input, and a module that is connected to a memristor output and transforms it to two-wire binary value. The transformation modules have no memristor as a private member. We expect the user to connect memristor and transformation modules by herself, which leaves the option to use single transformation modules for a cluster of memristors and place multiplexers between the input and output ports of memristor modules. This causes an extra effort to the user, but we considered it more worth to save redundant conversion modules in situations when they could be reused to access a cluster of memristors.

In our model, the conversion modules model the analogue

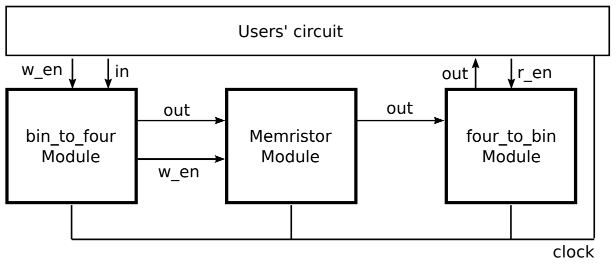

Fig. 10. Schematic depiction how memristor and conversion modules are to be connected.

read-out and write-in circuits as there have to be in a real chip that is based on memristors (see VII-A). Figure 10 depicts schematically the architecture, how to use our memristor model and the transformation modules in users' hardware descriptions. Data flow directions are pointed out with arrows. From the user's circuit, binary input *in*, data is stored in the memristor, when the write enable signal *w_en* is set to high. Then the data is transformed and output via the left *out* signal into four-value logic and stored in the memristor and the write enable input *w_en* of the memristor module is set to high to signal the write procedure. To read the stored value from the memristor, the user has to set the read enable input signal *r_en* to high and the *four_to_bin* module will output the value stored in the memristor binary encoded on the very right output signal *out*.

Above we presented all necessary modules to build the fast SD adder with a four-value register for successive additions for the CORDIC algorithm. The prototype model we have implemented in SystemC is depicted in Figure 11. Both addends x and y can be in SD representation and in binary representation with negative weights set to 0. The multiplexer *MUX* selects between the second addend or using a previously calculated sum *sum*, stored in the memristor register. The SD sum is computed by the PPM cells and available at the output *sum*. When the sum is valid at the output of the SD adder, it is also stored in the memristor register and can be reused as addend at a later time.

E. Memristor simulation results

We present our results of the memristor and the fast SD adder simulation in this section. By the help of a test bench that we implemented in SystemC, we verified our memristor SystemC model to be correct. To do so, we wrote a test bench that attempted to store all possible values in the memristor module, read it back and compared it to the previously stored input. In order to test faulty accesses, we interrupted the input to the memristor module during the write-in phase and verified that the stored value differed from the input data.

Furthermore, we proved the SD adder module to be correct. This implies that we also proved the PPM cell modules to be correct while verifying the complete SD adder. In order to verify the SD adder, we had it perform additions of all possible input permutations to verify that the output sum corresponds to the correct addition.

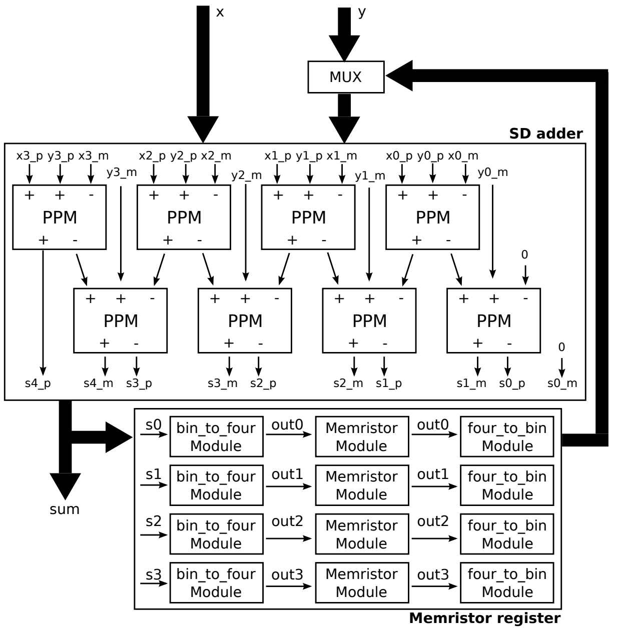

Fig. 11. Architecture of our fast SD adder with memristor register for CORDIC.

Due to the connections between the output ports d and e of the top PPM cells to the input ports x_p and x_n of the bottom PPM cells, a delay of one clock cycle is introduced. This results from the limitation that ports can not be connected directly or like wires, but are always transformed into Flip-Flops that present the input value to its output connector with a delay of one clock cycle.

Concerning the write access performance to the memristor it depends on the simulated clock speed, how many clock cycles are necessary to finish the successful storing of the value. In our simulation, we used the standard clock frequency of SystemC 2.3.0 that is 1GHz, i.e., a period of 1ns. As a result, a write access to the memristor model lead to a delay of five clock cycles in our simulation.

Both transformation modules, as presented in Section VII-D, added a delay of one clock cycle, as information has to pass through one stage of Flip-Flops that are connected to the output ports.

For both data paths through our SD adder with memristor registers, the data throughput is delayed only by Flip-Flops. The worst case were two additions, in which the intermediate sum was stored in the memristors an reused for a second addition. In that case, the overall delay, until the final result was displayed at the output *sum* (see Figure 11), composed as follows: During the first addition 1 clock cycle delay appears due to Flip-Flops in the interconnection between the PPM cells and a 2nd delay until the final computation result is displayed at the sum output ports. During the conversion from SD to four-value logic, a delay appeared in the *bin_to_four* transformation module and another 5 clock cycles delay until data is stored in the memristors. Information was written in parallel to the memristors, so the delay always remains constant at 5 clock cycles, unless the frequency of the clock is not changed. When the data is read out from the memristors, 1 clock cycle delay is added in the *four_to_bin* transformation module. During the second addition, when an addend was added to the intermediate sum, the delay was limited to the same 2 clock cycles, exactly as during the previous addition. All in all, the clock cycles sumed up to 11 clock cycles delay in our worst case scenario.

Since we used the standard clock frequency of SystemC 2.3.0, 1ns was equivalent to one clock cycle. This allowed us to use the SystemC built-in function *sc_time_stamp()* to retrieve the delayed clock cycles.

VIII. Comparison of QCA and memristor technology

While both technologies are two very differing approaches to overcome the CMOS limitations, we identified overlapping similarities in both technologies. In this section, we will present our findings.

Though the memristor works by the transport of electrons and QCA relies on the propagation of Coloumb force impulses between electrons, both devices are stateful. While the memristor can theoretically be put into an infinite number of different states, i.e., it can be set to an arbitrary resistance, it is important to put it into a well-chosen limited number of states, e.g., in a digital system to low resistance for logic 0 and high resistance for logic 1. By its nature, the memristor will keep its resistance, to which it was previously set, without the need of energy. It will remain in this state until it is changed to another resistance.

Regarding QCA, which needs a clocked electric field as a clock signal [20], QCA cells keep their state, i.e., the arrangement of the electrons in the potential wells, without the need of energy. This is a common ground between QCA and memristor technology. Regarding the state-of-the-art efforts for current CMOS based computers, to put them into an energy-saving “sleep” state, we identify the possibility of future QCA- or memristor-based computers, this capability is automatically available by the underlying technology. QCA cells and memristors remain in their state without the need of energy. I.e., a computer based only on QCA or memristor technology is put into an energy saving state as soon as the power supply is disconnected. As soon as it is reconnected to its power supply it will continue computation at the very same point when it was disconnected from power.

Simulations of QCA systems predict very high clock frequencies, up to THz scale, while memristors limit data throughput by the necessary time interval to put the memristor in a specific resistance. This is an advantage of QCA over memristors. On the other hand, the memristor can be used in multi-value logic environments, whereas QCA can only compute and store binary information. In our SD adder model, the memristor is used to store four-value logic information, which is an advantage over a binary memmory. The four-value logic allows to reduce the number of memory devices to the half, in contrast to binary memory. In larger systems than our model, four-value memristor memory can improve the space efficiency on a chip enormous.

IX. Conclusion

We presented a generic design procedure for mapping digital optical computing circuits based on SSL onto nanocomputing QCA circuits. This will form both the base for future design tools for compact, regular build-up QCA circuits and supports the direct mapping of optical computing circuits to QCA technology. For example, we intend to map an integer arithmetic unit based on SSL, designed by us [21], onto a complete QCA integer unit. In addition, we have to verify the schematically shown QCA circuit of Figure 6 by simulation with the QCADesigner tool [2], the standard for simulating QCA layouts. Furthermore, the insertion of an exact clocking scheme for the QCA cells has to be considered in the synthesis procedure. Nevertheless, the basic step for an automatic synthesis of SSL arithmetic circuits to QCA layouts is established.

Furthermore, we presented our developed SystemC model of a memristor and the characteristics of the model for a digital circuit simulation, which we derived from its analogue behavior. We have shown that the memristor can work as a four-value logic memory in a fast SD adder circuit, which is our prototype of a building-block for a future implementation of a CORDIC implementation. Though the prototype needs some further improvement, we also demonstrated that it is possible to model this typical analogue device for a digital simulation and design process. For our prototype we also modeled in SystemC the analogue write-in and read-out circuits for a digital simulator. The prototype was completely verified and with some improvement will form a building-block for the implementation of memristor-based arithmetic circuits.

Our findings pointed out, that development and synthesis software tools for memristor-based circuits have to be aware of the analogue extra circuitry. We proposed that the hardware designer must be given the ability on a high level hardware description, to influence and optimize the utilization and reusability of underlying analogue circuitry, in order to gain maximum space efficiency on a chip.

We compared both nanotechnologies and found challenging differences in the requirements to automated design tools. If memristor-based arithmetic units become state-of-the-art in the mid-term future and hardware design tools get adopted to this technology, our findings point that further research on design tools is necessary to make them reusable for the long-term QCA technology.

Despite the differences, we found a promising common ground among both technologies for energy efficient future computers. In contrast to current CMOS-based computers, both technologies keep remain in their current state without the need of energy. We propose to leverage this natural property for mid- and long-term future computers to save energy. We suggest to cut off power supply during idle states of these devices as systems built from QCA cells and memristors will continue their computations exactly when the were powered off.

We identified the advantage of memristors over QCA technology, to be suitable for multi-level logic environments. With our model of a fast SD adder with a memristor-based intermediate register, we demonstrated the advantage of space efficiency of a four-value logic memory, that is implemented with memristors. Our model needs only half of the memory elements, compared to a binary memory.

References

[1] D. Fey and B. Kleinert, "Using Symbolic Substitution Logic as an Automated Design Procedure for QCA Arithmetic Circuits," in *FUTURE COMPUTING 2012, The Fourth International Conference on Future Computational Technologies and Applications*, 2012, pp. 94–97.

[2] K. Walus, T. J. Dysart, G. A. Jullien, and R. A. Budiman, "QCADesigner: A rapid design and simulation tool for quantum-dot cellular automata," *Nanotechnology, IEEE Transactions on*, vol. 3, no. 1, pp. 26–31, 2004.

[3] K.-H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," *Appl. Opt.*, vol. 25, no. 18, pp. 3054–3060, Sep 1986. [Online]. Available: http://ao.osa.org/abstract.cfm?URI=ao-25-18-3054

[4] R. Williams, "How we found the missing memristor," *Spectrum, IEEE*, vol. 45, no. 12, pp. 28–35, 2008.

[5] K. H. Brenner, W. Eckert, and C. Passon, "Demonstration of an optical pipeline adder and design concepts for its microintegration," *Optics & Laser Technology*, vol. 26, no. 4, pp. 229–237, 1994.

[6] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, "Quantum cellular automata," *Nanotechnology*, vol. 4, no. 1, p. 49, 1993. [Online]. Available: http://stacks.iop.org/0957-4484/4/i=1/a=004

[7] V. A. Mardiris and I. G. Karafyllidis, "Design and simulation of modular 2n to 1 quantum-dot cellular automata (QCA) multiplexers," *International Journal of Circuit Theory and Applications*, vol. 38, no. 8, pp. 771–785, 2010. [Online]. Available: http://dx.doi.org/10.1002/cta.595

[8] F. Bruschi, F. Perini, V. Rana, and D. Sciuto, "An efficient Quantum-Dot Cellular Automata adder," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, 2011, pp. 1–4.

[9] R. Zhang, K. Walus, W. Wang, and G. A. Jullien, "A method of majority logic reduction for quantum cellular automata," *Nanotechnology, IEEE Transactions on*, vol. 3, no. 4, pp. 443–450, 2004.

[10] A. Louri, "Parallel implementation of optical symbolic substitution logic using shadow-casting and polarization," *Applied optics*, vol. 30, no. 5, pp. 540–548, 1991.

[11] Y. Ichioka and J. Tanida, "Optical parallel logic gates using a shadow-casting system for optical digital computing," *Proceedings of the IEEE*, vol. 72, no. 7, pp. 787–801, 1984.

[12] I. Hänninen, "Computer Arithmetic on Quantum-dot Cellular Automata Technology," Ph.D. dissertation, Tampare University of Technology, http://dspace.cc.tut.fi/dpub/handle/123456789/6337?show=full, 2009.

[13] R. Akeela and M. D. Wagh, "A Five-input Majority Gate in Quantum-dot Cellular Automata," 2011.

[14] L. Chua, "Memristor-the missing circuit element," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 5, pp. 507–519, 1971.

[15] G. S. Snider, "Self-organized computation with unreliable, memristive nanodevices," *Nanotechnology*, vol. 18, no. 36, p. 365202, 2007.

[16] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive'switches enable 'stateful'logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[17] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[18] J. E. Volder, "The CORDIC trigonometric computing technique," *Electronic Computers, IRE Transactions on*, no. 3, pp. 330–334, 1959.

[19] B. Kasche, "Entwurf eines optoelektronischen Rechenwerkes," Ph.D. dissertation, 1999.

[20] S. E. Frost, "Memory Architecture for Quantom-dot Cellular Automata," Ph.D. dissertation, University of Notre Dame, 2005.

[21] D. Fey and K. H. Brenner, "Digital optical arithmetic based on systolic arrays and symbolic substitution logic," *Opt. Comput*, vol. 1, pp. 153–167, 1990.