Spring 5-2-2015

A Signal Distribution Network for Sequential Quantum-dot Cellular Automata Systems

Hayden M. Hast
Valparaiso University, hayden.hast@valpo.edu

Douglas Tougaw
Valparaiso University, douglas.tougaw@valpo.edu

Sami Khorbotly
Valparaiso University, sami.khorbotly@valpo.edu

A Signal Distribution Network for Sequential Quantum-dot Cellular Automata Systems

Hayden Hast, Sami Khorbotly, and Douglas Tougaw

Abstract—**The authors describe a signal distribution network for sequential systems constructed using the Quantum-dot Cellular Automata (QCA) computing paradigm. This network promises to enable the construction of arbitrarily complex QCA sequential systems in which all wire crossings are performed using nearest neighbor interactions, which will improve the thermal behavior of QCA systems as well as their resistance to stray charge and fabrication imperfections. The new sequential signal distribution network is demonstrated by the complete design and simulation of a two-bit counter, a three-bit counter, and a pattern detection circuit.**

***Index Terms*—Nanoelectronics, quantum-dot cellular automata (QCA), quasi-adiabatic switching, wire crossing, sequential systems.**

I. Introduction

Quantum-dot Cellular Automata (QCA) is an emerging nanoscale computing paradigm that offers many benefits over traditional transistor-based computing paradigms, such as reduced power consumption, increased speed, and reduced surface area [1]-[6].

QCA systems are composed of cells, each of which includes two electrons that are shared among four quantum dots that lie at the corners of a square. Mutual Coulombic repulsion between the two electrons causes them to occupy diagonally opposite quantum dots. There are two bistable configurations of the cell, one along the left-leaning diagonal axis (encoded as a binary 0), and the other along the right-leaning diagonal axis (encoded as a binary 1) [1]-[2]. Electrons contained within adjacent cells repel each other, so that cells placed along a line tend to align in the same state [3]. Figure 1 illustrates this geometry as well as the typical dimensions used for the calculations in this paper.

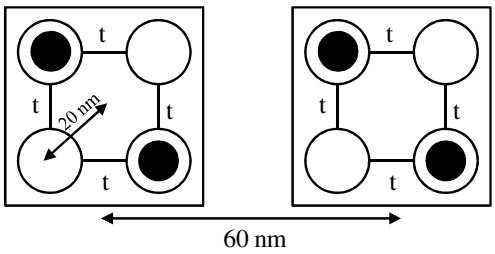


Fig. 1. The QCA geometry used in this paper. Each cell is composed of four quantum dots placed at the corners of a square, and the distance from each dot to the center of the square is 20 nm. Tunneling is allowed between each pair of adjacent quantum dots. Two electrons occupy each cell, and the distance between the centers of the adjacent cells is 60 nm. [After reference 7]

These cells can be arranged into particular geometric patterns to implement majority logic functions, which can in turn be reduced to AND and OR gates. Inverters can also be constructed by arranging the cells in a particular pattern, which means that the QCA paradigm can be used to implement a complete set of combinational logic functions [4]-[5].

The tunneling among the four dots can be controlled by the height of the tunneling barriers between each pair of dots, and this can in turn be controlled by the voltage applied to a nearby metal lead under the plane of the QCA cells. When the barriers are raised very high, the cell is in a "locked" state, meaning the electrons are not allowed to move among the dots and are effectively isolated on their current dot. When the barriers are lowered significantly, the cell is in a "relaxed" state, meaning that the electrons are largely free to spread out evenly among the four cells. By quasi-adiabatically switching from a relaxed state to a locked state and then back to a relaxed state, the system is able to respond to inputs applied in a specific manner. Allowing different groups of cells to pass through this cycle at different times allows data to propagate through the system in a predetermined manner [6].

As shown in Figure 2, four different clock signals (all repeating the same pattern, but offset from each other in time) can be applied to four different regions of the system in order to allow data to flow continuously through a QCA system [8]-[9]. In addition to controlling the directionality of the data flow, the use of quasi-adiabatic switching and clocking

H. Hast is with the Electrical and Computer Engineering Department, Valparaiso University, Valparaiso, IN 46383 USA (phone: 219-464-5000; fax: 219-464-5065; e-mail: Hayden.Hast@ valpo.edu).

S. Khorbotly is with the Electrical and Computer Engineering Department, Valparaiso University, Valparaiso, IN 46383 USA (phone: 219-464-5183; fax: 219-464-5065; e-mail: Sami.Khorbotly@ valpo.edu).

D. Tougaw is with the Electrical and Computer Engineering Department, Valparaiso University, Valparaiso, IN 46383 USA (phone: 219-464-5027; fax: 219-464-5065; e-mail: Doug.Tougaw@ valpo.edu).

regions also allows each QCA cell to store one bit of data during its locked phase [10]. This is equivalent to the functionality of a D flip-flop. Combined with the complete set of combinational logic described above, this memory functionality further enables QCA systems to implement generalized sequential logic functionality as well.

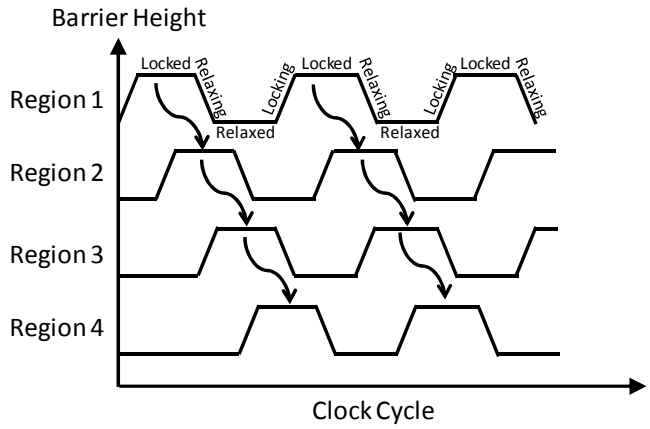


Fig. 2. Data propagates from one region (in the "locked" phase) to its neighbor (in the "locking" phase). Once a region has propagated its information, it relaxes in preparation for receiving its next input. A typical region of cells will repetitively cycle through these four phases, allowing data to be transferred in the desired direction as indicated by the arrows. [After reference 7.]

One challenge facing the QCA paradigm is that it is particularly difficult cross two lines of cells without allowing the signals to interact. Because QCA systems are coplanar, and because their interactions are based on physical proximity, it is very difficult to pass signals through each other without interference. One solution to this problem, relying on rotated cells and next-near neighbor interactions, was proposed in [5]. Unfortunately, it has since been shown that this wire crossing negatively impacts the excitation energy between the ground state and the first excited state, which has been shown to degrade the overall system's thermal behavior and its resistance to fabrication imperfections and stray charge [11]-[15]. A great deal of research has been performed to investigate methods to minimize wire crossings [16]-[19] and to use multiple parallel crossings to strengthen the interaction of the next-near neighbor wire crossing described above [20-21]. One solution even allows an arbitrary set of wire crossings, although it requires the use of several additional clock signals [22].

II. Improving the Combinational Signal Distribution Network

One method for implementing a generalized set of wire crossings (referred to as a "signal distribution network") was presented in [7] and [23]. This network allows a set of N inputs to be arbitrarily duplicated and distributed to the inputs of a combinational logic system. Such a signal distribution network (SDN) for combinational systems is shown in Figure 3, where it is used to implement the wire crossings necessary for a one-bit full adder.

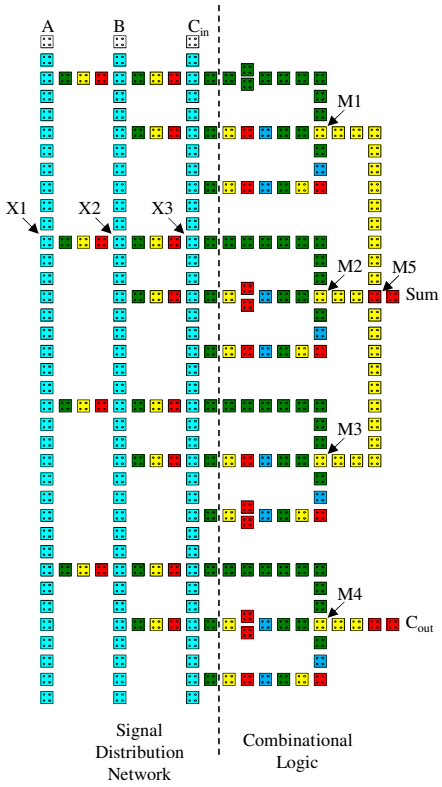


Fig. 3. (Color Online) The combinational signal distribution network (SDN). The three signals propagate down the vertical wires and then work their way toward the combinational logic. Signals (such as C_{in}) that have a shorter distance to travel must move more slowly, while those that have a longer distance to travel (such as A) must move more quickly. All signals arrive at the first level of majority gates (M1-M4) simultaneously. [After 7]

While the SDN presented in [7] offers the ability to perform any necessary number of wire crossings while only relying on near-neighbor interactions, it does require a relatively large amount of surface area to perform the signal distribution. As shown in Figure 3, the three parallel vertical wires and the horizontal wires that connect them take up approximately as much surface area as the combinational logic needed to perform the addition.

It turns out that this may not be strictly necessary. The multiple horizontal lines connecting between the leftmost and rightmost vertical lines are redundant with each other, an artifact of the Programmable Array of Logic technology that was used as an initial model for this network. Furthermore, the multiple vertical wires are only needed to place the correct values into the right-moving data pipeline. As shown in Figure 4, which is functionally identical to Figure 3, the same result can be achieved with only one vertical wire that is driven by a short input wire that receives all the results in parallel. Making this small change to the SDN reduces the size of the overall network by almost half while removing dozens of redundant cells. It also illustrates that the key to the operation of the SDN is the serial transmission of the data values to the input of the vertical line.

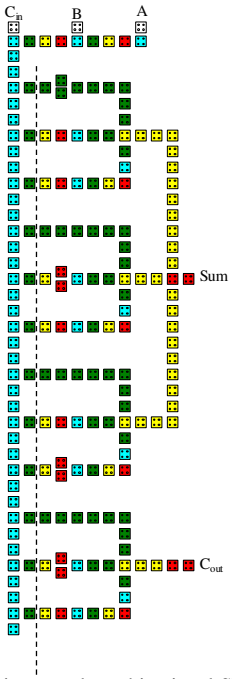


Fig. 4. (Color Online) An improved combinational SDN. Here, the multiple long vertical lines are no longer necessary, and the signals simply work their way to the left along the top line until it is their turn to drive the entire vertical line. This small change significantly reduces the surface area and number of cells required for the combinational SDN.

III. A Signal Distribution Network for Sequential Systems

Figure 5 shows a schematic representation of a generalized sequential digital system. Such a system is distinguished from a combinational system in that it contains a "current state" that is a function of all previous inputs to the system. This current state is routed back to the Next State Decoder (NSD), which determines the new value of the current state. In addition, the current state may need to be distributed to multiple inputs of an output decoder, which can be used to assert one or more outputs when the system is in a particular state. Thus, the sequential system will require one or possibly two combinational SDNs, as well as finding a way to perform the wire crossings between the current state and the outputs of the system. These wire crossings (15 for the case shown here) will occur in a very regular pattern that suggests a common solution.

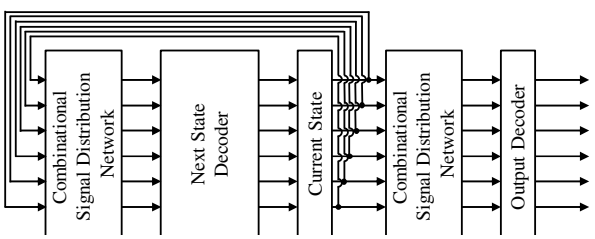


Fig. 5. A schematic representation of the wire crossings and signal distribution networks required by a typical sequential system. Even neglecting the wire crossings that are eliminated by the use of two separate combinational SDNs, this system still requires 15 wire crossings (which occur in a very regular, standard pattern) between the current state block and the second SDN.

Thus, a sequential system requires one or two signal distribution blocks and one fixed wire crossing block. Figure 6 shows an alternative implementation of the functionality present in Figure 5 that addresses all three of these needs simultaneously. In this case, there is no need for an explicit "current state" block, since each QCA cell can provide the functionality of a D flip-flop. Instead, the signals leaving the NSD in parallel pass through a series of delay blocks (composed of four QCA cells each), and they are shifted vertically upward. This serial data stream is then delivered to the inputs of the next state decoder as well as to the output decoder, if present. If the current state of the system is to serve directly as the output (as in the case of a counter, for example), then the output (current state) can simply be read at the location indicated.

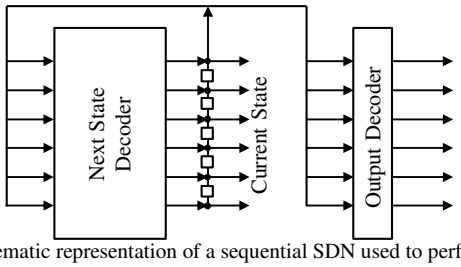


Fig. 6. A schematic representation of a sequential SDN used to perform all the necessary wire crossings for a sequential QCA system. Since each QCA cell acts as a D flip-flop, there is no need for an explicit "current state" block. Data leaving the next state decoder works its way vertically along a series of delay blocks (implemented by clocking regions). Once they reach the top, they are transmitted to the inputs of the next state decoder and the output decoder. As before, the first data to arrive must be delayed until the later data has also propagated so that they arrive at the combinational logic gates simultaneously.

This "Sequential Signal Distribution Network" (SSDN) can be used to significantly simplify the distribution of signals in sequential systems. The next three sections of this paper illustrate specific examples of sequential systems constructed using this method.

IV. Sequential SDNs to Implement Counter Circuits

In previous sections of this paper, the basic block diagram and data flow of sequential systems have been described. In this section, an implementation of the SSDN will be shown and described in detail. In order to exhibit the full functionality of the SSDN, only complete examples will be examined. By using full examples of sequential systems, it can be demonstrated that the SSDN can handle multiple wire crossings, which pass the current state to the output decoder as well as routing it back to the next state decoder.

A counter circuit, one of the most common and basic sequential systems, has been chosen to exhibit the implementation of the SSDN. Counter circuits can be seen in a wide variety of applications and are also easily scalable for high volume calculation tests. The counter circuit maintains the current state of the system, distributes those signals as necessary to the next-state decoder, and forwards the current state to an optional output decoder. The schematic

representation of a two-bit counter circuit can be seen in Figure 7. The sequential system contains two wire crossings used to distribute the signals to the next-state decoder as well as a wire crossing needed to route the current-state signals back to the inputs of the next-state decoder.

The QCA implementation of the schematic from Figure 7 can be seen in Figure 8. Here, it can be observed that the three explicit wire crossings mentioned above are no longer explicitly needed. Instead, the clocked QCA cells are positioned and clocked in such a way that the signal is transmitted serially along the top and left side of the system.

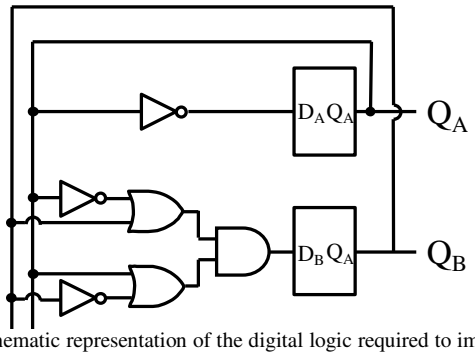


Fig. 7. A schematic representation of the digital logic required to implement a two-bit (modulo-4) counter. Note that two wire crossings are required to distribute the signals to the next-state decoder, and a third is required to feed the current state back to the inputs of the next-state decoder.

This new functionality of the SSDN is made possible by the addition of two new clock phases, shown as orange and purple in the on-line version of the figure. This addition was required in order to pass the data vertically and horizontally in an alternating pattern. The timing of asserting these new clock phases coincide with the timing of the yellow clock phase. However, only one of the orange or purple phases is ever asserted at a time. For example, in one clock cycle the purple and yellow will be asserted at the same time, then in the next cycle, orange and yellow will be asserted. This ensures that the data is driven by the green cells and is then passed either horizontally or vertically through each intersection. The alignment of the yellow, orange and purple clock phases maintains the flow of data through the system.

The purpose of a sequential system is to use previous outputs, or current states, as the next inputs into the system. However, realistic sequential systems have user inputs such as a start, stop, or reset. In order to emulate a realistic system, a reset input was implemented. Figure 8 shows two grey reset cells. At the beginning of the simulation, the reset signal will place the counter into a known state (‘00’ in this case). It is important to note that this extra clock signal is not strictly necessary, but is only included to allow the system to be initialized. Thus, the true number of clock signals required for this system is four (as with all pipelined QCA systems) plus two (the orange and purple signals described above). Adding a SSDN to a QCA network will increase the number of clock signals required from four to six.

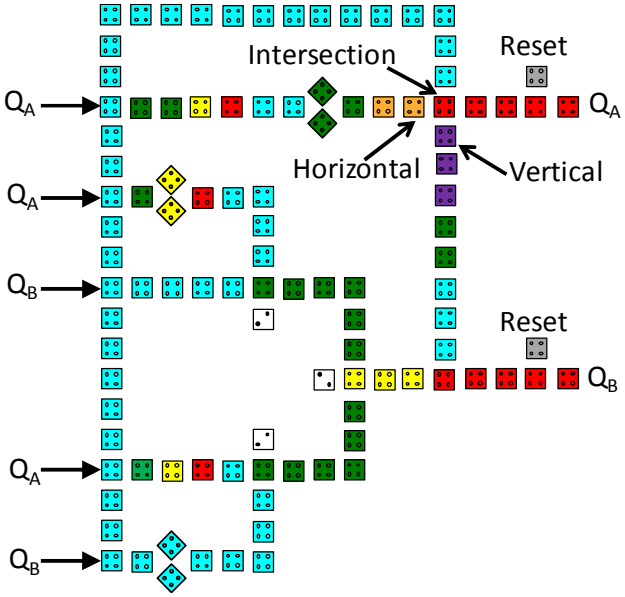


Fig. 8. (Color Online) An implementation of the two-bit counter from Figure 7 using QCA cells and a sequential SDN. The orange region is asserted when the signal needs to pass horizontally through the intersection cell, while the purple region is asserted when data needs to flow vertically through the intersection cell. The labels and arrows on the left side of the figure are only intended as an aid to the reader in understanding which signal is being used on each horizontal line.

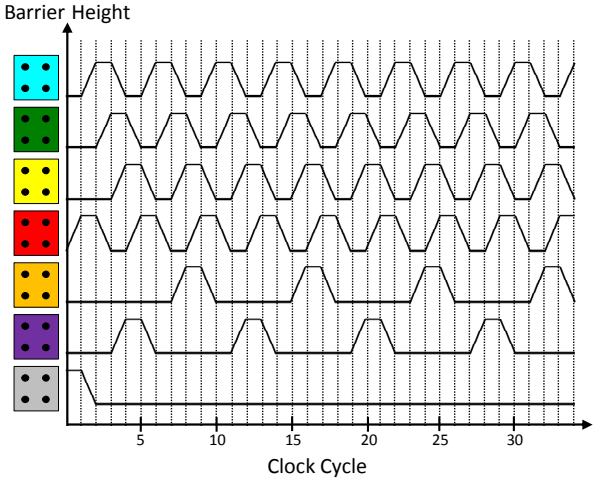


Fig. 9. (Color Online) The clock signals applied to the regions shown in Figure 8. The colors of the cells along the vertical axis correspond to the color of each cell show in Figure 8. Notice that the four primary regions (blue, green, yellow, and red) continuously cycle as shown in Figure 2. The orange and purple regions alternate, and each always asserts simultaneously with the yellow region.

Using the clock cycle chart in Figure 9, the flow of the data through the sequential system depicted in Figure 8 can be examined. The counter is first put into a known state of ‘00’ from the assertion of the grey, or reset phase. The data from Q_A then flows up and around, through the distribution network using the blue, green, and yellow cells, in that order. At the same time, data from Q_B moves vertically through its own blue and green cells. However, instead of entering a yellow cell, the data flows into the purple cells. This is because the purple and yellow clock cycles are both asserted at cycle four. The red phase is then asserted, acquiring the data driven by the

purple cells while disregarding the bogus data in the relaxed orange cells. This is the first instance of passing the correct data through the wire crossing.

The data from Q_B is then quickly passed along through another cycle of blue, green, and yellow while the Q_A data moves in much smaller steps, often only one cell at a time. By phase number eight, both sets of data have passed through the next state decoder and are ready to be output. Again, the utilization of the added clock phases allows the data to be passed through the wire crossing. In clock cycle number eight both the orange and yellow phases are asserted, locking the Q_A data and Q_B data into their respectively colored cells, waiting for the red phase to lock. The red cells for output Q_A are only driven by the valid data locked in the orange cells, and not the invalid data in the adjacent purple cells. Output Q_B is also determined only by the locked adjacent yellow cells. Thus, after nine clock cycles, the first valid output, '01' is seen.

The process described above is repeated, but without the use of the reset. This causes the next output, '10' to appear eight cycles later, on clock cycle 17. The sequential system continues this pattern through 25 total cycles, counting from '00' to '11.'

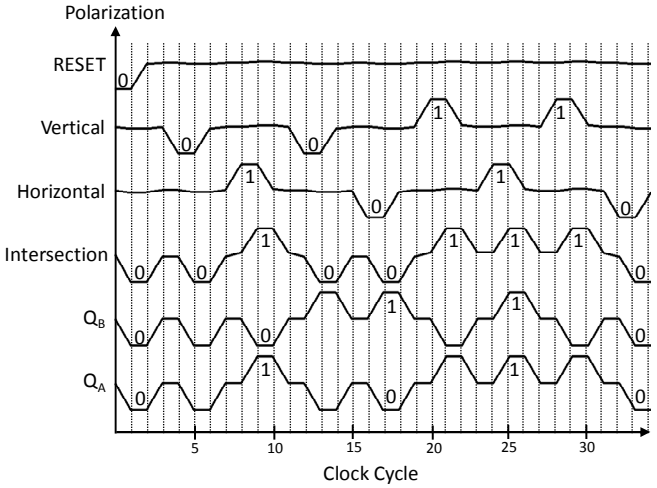


Fig. 10. Simulation results of the system shown in Figure 8 when the clock signals of Figure 9 are applied. Notice that the RESET signal is applied to the cells in the gray region in the first clock cycle, which causes Q_A and Q_B to go to zero. Over the next 34 clock cycles, those two signals progress through the sequence 00, 01, 10, 11, and back to 00. It is important to note that the outputs are only valid at the times labelled. It is also possible to observe how the cells labelled "vertical" and "horizontal" alternate control of the "intersection" cell, allowing data to flow in both directions.

Applying the clock signals depicted in Figure 9 to the two-bit counter of Figure 8 produces the simulation results shown in Figure 10. These results were obtained by performing a self-consistent simulation of the array of QCA cells, applying the Intercellular Hartree Approximation. Each cell was modeled using the Time-Independent Schrodinger equation and second-quantization operators. The ground state of each cell was calculated multiple times until the entire array of cells had reached a self-consistent ground state. The expectation value of the charge density on each site was then calculated, and this yielded the polarization of each cell, as shown in Figure 10. The same method was used to generate the results in Figure 14 and Figure 19.

Figure 10 displays the reset signal, the state of the vertical (purple) cells, the state of the horizontal (orange) cells, the state of the intersection cell, and the counter outputs. Beginning from the top, it can be seen that the reset signal is only asserted at the beginning of the sequence in order to start the counter into a known state.

Also very noticeable is that the vertical and horizontal signals are mutually exclusive. This is due to the fact that there is only valid data passing in one direction every fourth cycle. Although the purple and orange signals are only locked alternatively every four cycles, the intersection signal locks every four cycles because it is being driven by either the vertical or horizontal data. Further examination confirms this because the value of the intersection signal is always the previous value of the vertical or horizontal signals, whichever one was asserted last.

Finally, the output of the two-bit counter can be seen in signals Q_A and Q_B. The Q_A output signal matches the intersection signal because they are both located in the same row of red cells. The output is first driven by the reset and puts the counter into '00.' Eight cycles later, the first legitimate output of '01' is displayed. It is important to note that the outputs are only valid once every eight clock cycles. Although they lock every four clock cycles, the locked results for Q_A and Q_B are invalid for cycles in which no label appears in Figure 10. The process continues, outputting '10' during cycle 17, '11' during cycle 25 and finally rolling back to '00' on cycle 35. These simulation results confirm that a QCA two-bit counter sequential system can be constructed using the implementation of the proposed SSDN.

As previously mentioned, counter circuits work well for examples because of their scalability. In order to examine how the SSDN performs on a larger scale, it was decided to expand the previous example from a two-bit counter to a three-bit counter. Figure 11 shows the schematic representation of the three-bit counter that will be implemented with the SSDN. It can be seen that the three-bit counter contains six wire crossings used to distribute the signals to the next-state decoder as well as three wire crossings needed to route the current-state signals back to the inputs of the next-state decoder.

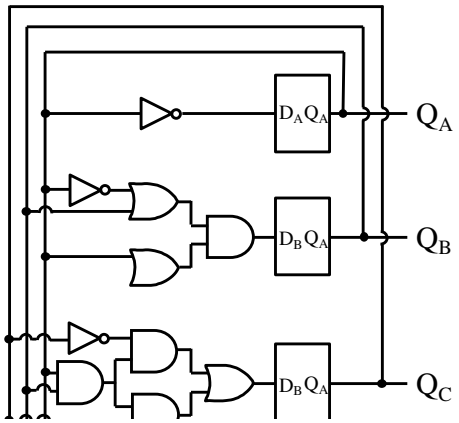


Fig. 11. A schematic representation of the digital logic required to implement a three-bit (modulo-8) counter. Note that six wire crossings are

required to distribute the signals to the next-state decoder, and three more are required to feed the current state back to the inputs of the next-state decoder. These nine wire crossings will be implemented using the sequential SDN.

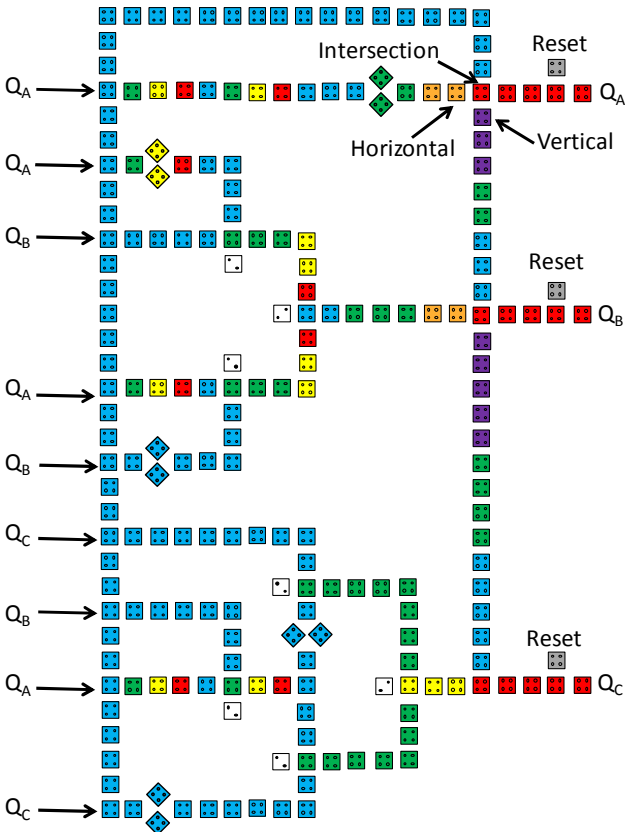


Fig. 12. (Color Online) An implementation of the three-bit counter from Figure 11 using QCA cells and a sequential SDN. Again notice the use of two special clock phases to implement the serial wire crossings. It is now more evident that lines carrying Q_A must be designed to transmit that value more slowly, whereas Q_C moves the data as quickly as possible toward the beginning of the combinational logic.

Figure 12 displays the QCA implementation of the three-bit counter circuit depicted in Figure 11. The QCA implementation of the next state decoder has become visibly more complex, accounting for the increased number of inputs and outputs to the sequential system. However, the same data flow principles that were utilized in the two-bit counter can be seen at work here.

The current states of Q_A, Q_B, and Q_C are successively transmitted vertically, passing through the three wire crossings with the use of the newly created purple and orange clock phases. These signals are then passed serially along the top and down the left side of the diagram. To the left side of the diagram there are labels depicting where each of the current state signals enter the next state decoder. As with the two-bit counter, the Q_A signal must slowly propagate for four clock cycles in order for the Q_B signal to catch up. Then, both the Q_A and Q_B signals (or the logical combination of the two) must propagate for four more signals, allowing all three signals to align before continuing through the decoding logic. The next-state outputs are then passed horizontally through the wire crossing, labeled "intersection" in Figure 12, to their respective outputs.

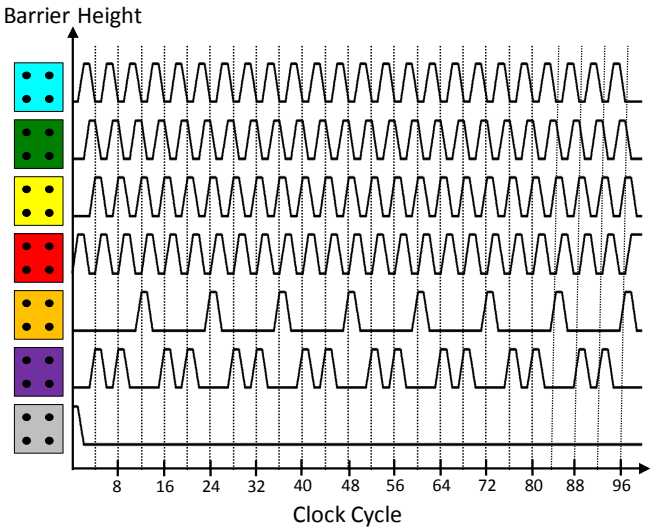


Fig. 13. (Color Online) The clock signals applied to the regions shown in Figure 12. The colors of the cells along the vertical axis correspond to the color of each cell show in Figure 12. This figure is very similar to Figure 9, except that each horizontal data transmission (the orange region) is followed by two vertical transmissions. As shown in Figure 11 and Figure 12, this is required to enable Q_C to first cross Q_B and then to cross Q_A while moving up the right edge of the system.

Figure 13 depicts the sequence of the applied clock signals to the corresponding regions in Figure 12. The assertion of the reset signal at the beginning of the sequence puts the sequential system into a known state of '000'. The data is then propagated through the system, flowing through the red, blue, green, and yellow regions in order. However, exceptions to this data progression occur when a signal must pass through a wire crossing. As in the two-bit counter example, the addition of the orange and purple signals make this wire crossing possible. However, instead of the purple and orange regions being asserted alternatively every four cycles as described in the two-bit counter, a pattern of purple-purple-orange can be seen instead. This is because Q_C must pass through two wire crossings, utilizing the vertical purple regions, before it can reach the next-state decoder. Once it reaches the next sate decoder, it is then quickly propagated through the required logic, after which the outputs are passed horizontally through the orange regions. It is the behavior that results in two purple region assertions to every one orange regions assertion. The simulation data depicted in Figure 14 verifies that there are two vertical assertions to every horizontal assertion as well as the correct outputs of the QCA implemented three-bit counter circuit.

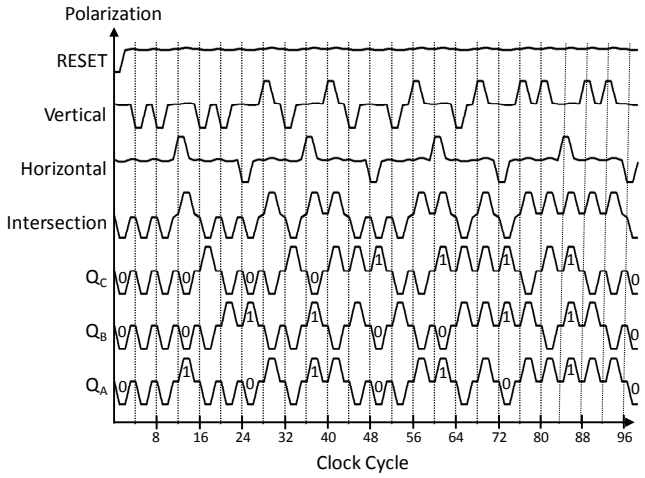


Fig. 14. Simulation results of the system shown in Figure 12 when the clock signals of Figure 13 are applied. Over the course of 98 clock cycles, the output signals Q_A, Q_B, and Q_C cycle through the sequence 000, 001, 010, 011, 100, 101, 110, 111, and back to 000. It is important to note that the outputs are only valid at the times labelled. The pattern of one horizontal data pulse followed by two vertical data pulses can also be observed in this figure.

V. Pattern Detector Using Sequential SDN

The SSDN can also be used to construct a pattern detector, which monitors a serial bit stream for a particular binary sequence. This example is particularly interesting because it illustrates a *Mealy* machine, in contrast with the *Moore* machines described in the previous examples. In computing theory, a *Moore* machine is a sequential system where the output is determined exclusively by the state of the machine. A *Mealy* machine, on the other hand, is a sequential system where the output is determined by both the state of the machine and the input signal.

The chosen example is a pattern detector designed to detect the pattern "11010." The system has a single input line *x* and a single output line *z*. The output value is always a logic '0' except when the last 5 bits of the serial input stream have been "11010."

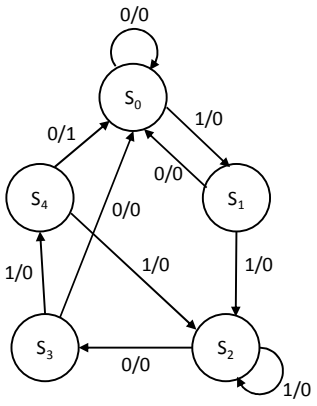


Fig. 15. State diagram for detection of the pattern 11010. Labels along each transition arrow show the input signal that will cause the system to move along that arrow, followed by the output signal that will result from the current state and that input being applied. Note that the only time the output will be asserted is when the system is in state S_4 and receives a 0 input.

The logic design for this system results in the state diagram shown in Figure 15. As can be seen in the figure, five distinct states $[S_0 - S_4]$ are needed for this implementation. A three-bit binary number is required to represent these five states.

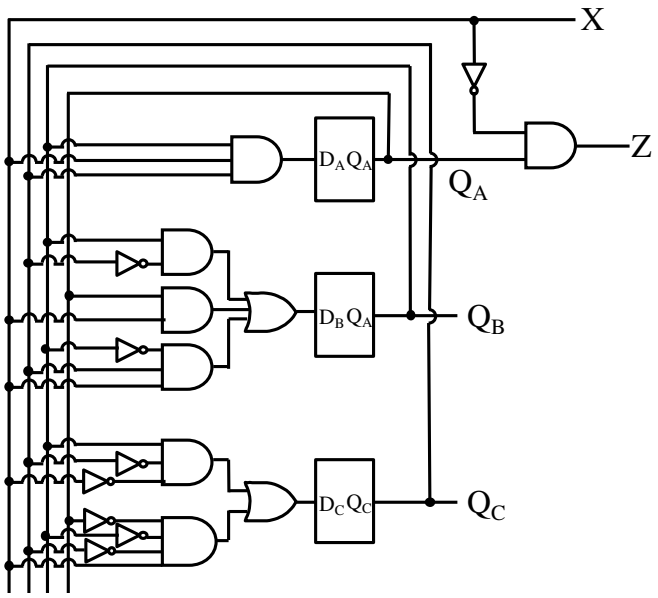


Fig. 16. Digital logic required to implement the state diagram shown in Figure 15. This system requires 31 wire crossings to distribute the signals to the next-state decoder, along with 4 wire crossing to feed the current state back to the next state decoder. Note the input X at the top right corner, as well as the output Z.

The three state variables used to store the state information are labeled Q_A, Q_B, and Q_C. Using the single output design methodology [24], the next state decoder equations were minimized in the sum of products format as follows:

$$D_A = Q_B \cdot Q_C \cdot x$$
$$D_B = Q_B \cdot \overline{Q_C} + Q_A \cdot x + \overline{Q_B} \cdot Q_C \cdot x$$
$$D_C = Q_B \cdot \overline{Q_C} \cdot \bar{x} + \overline{Q_A} \cdot \overline{Q_B} \cdot \overline{Q_C} \cdot x$$

Where D_α denotes the future/next state value of the state variable Q_α.The minimized logic equation for the output of the system was similarly found to be:

$$Z = Q_A \cdot \bar{x}$$

The logic circuit diagram corresponding to the pattern detector is shown in Figure 16. The circuit includes three D-flip flops to store the state variables, 15 combinational logic components used to implement the next state decoder, and two combinational logic components used to implement the output decoder.

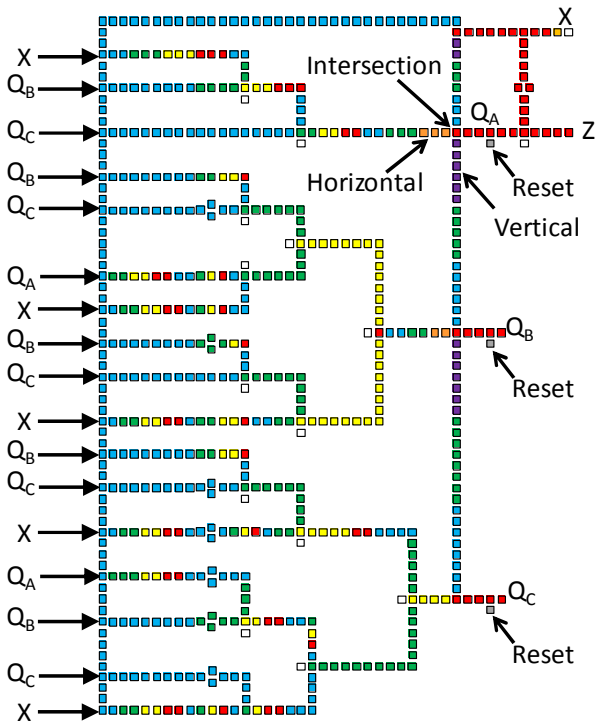


Fig. 17. (Color Online) An implementation of the pattern detector from Figure 16 using QCA cells and a sequential SDN. The three state variables complete a full loop and update their values every 20 clock cycles.

An implementation of the pattern detector from Figure 16 using QCA cells and a sequential SDN is shown in Figure 17. Initially, the present state value is determined by the three "Reset" cells associated with each of the three state variables Q_A, Q_B, and Q_C. In this particular example, only Q_A is used, along with the input signal *x*, in the output decoder. However, all three state variables are used, along with the input signal *x*, to determine the next state value. For this purpose, the values of these variables all propagate through the rightmost vertical line starting in the same clock phase. These values then complete the loop by moving leftward along the top horizontal line, and then copies of the signals are distributed as needed by twelve horizontal lines. Similarly, the input signal *x* propagates along the same path where five copies of it are distributed to help determine the next state value.

Seeing that four signals are propagating via the same path, the pattern of clock phases is carefully selected to ensure that all four signals are routed to their destinations at the proper times. The input signal *x*, controlled by an external source signal, will appear on the far-left vertical line after two clock cycles, and it is never updated internally. The three state variables, on the other hand, complete a feedback loop and periodically update their own values every 20 clock cycles. It is therefore imperative that the values of the three state variables and the input signal reach the end of the loop simultaneously to update the state value. For example, Q_A has a shorter time to reach the far-left vertical line than Q_B, so its propagation is slowed down along the horizontal lines. Similarly, Q_C has the longest path to the far-left vertical line but is caught up by having the fastest propagation in the horizontal lines.

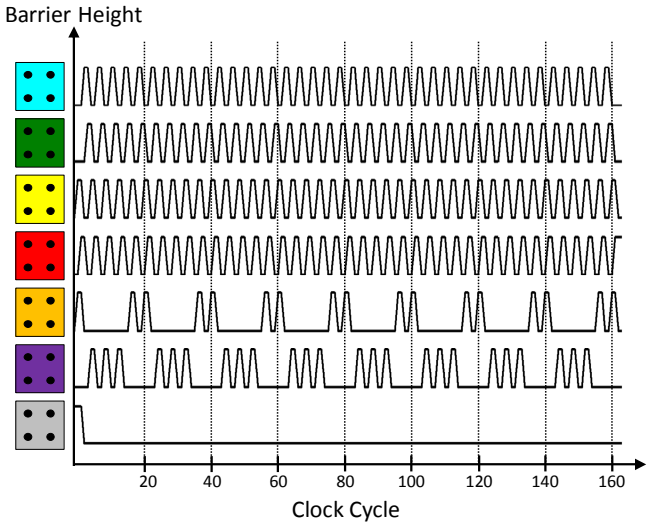


Fig. 18. (Color Online) The clock signals applied to the regions shown in Figure 17. The three purple pulses allow the data to move upward along the right edge of the next-state decoder, while one of the orange pulses enables data to move horizontally to the right of the next-state decoder. The second orange pulse is not strictly necessary, because the data is working its way through the next-state decoder. It is included here to ensure that the "intersection" cells are in a well-defined state for the simulation.

The clock signals shown in Figure 18 illustrate the clocking pattern used to ensure the appropriate routing and propagation of all signals. The gray signal is only activated to reset the system in order to determine a specific initial state. It is then de-activated during the synchronous operation of the system. Once the system is initialized, a four cycle "normal" pattern is set for the signals to propagate to the blue cells in one clock cycle, followed by the green cells, then the yellow cells, and finally the red cells. The only exception to this pattern is the intersection lines where more sophisticated clocking is needed to handle the wire crossing. At these intersections, the yellow cells are replaced by either orange cells for the horizontal lines or purple cells for the vertical lines. The yellow clock signal is de-multiplexed between the purple cells and the orange cells clock in order to propagate the correct signals at the correct times. It all starts by activating the orange cells to horizontally propagate the input signal *x*. Then, the purple cells are activated for three consecutive cycles to vertically propagate Q_A, Q_B, and Q_C all the way to the top horizontal line. The orange cells are then activated to ensure that Q_A, Q_B, and Q_C are horizontally updated with the next state values. The pattern is then periodically repeated.

The simulation results of the system are shown in Figure 19. During the first two clock cycles, the system is reset, and an input value is applied. As was shown in Figure 17, the state variables are updated once every 20 clock cycles. Therefore, it is reasonable to evaluate the output of the system only during the $(20 \cdot N + 2)^{\text{th}}$ clock cycles where N is a positive integer. The results show that the only time an output of '1' was detected was at $N = 6$ in response to the input segment $x[2{:}6]$ being "11010." This input sequence is highlighted by the dotted box shown in Figure 19, which is immediately followed

by an asserted output.

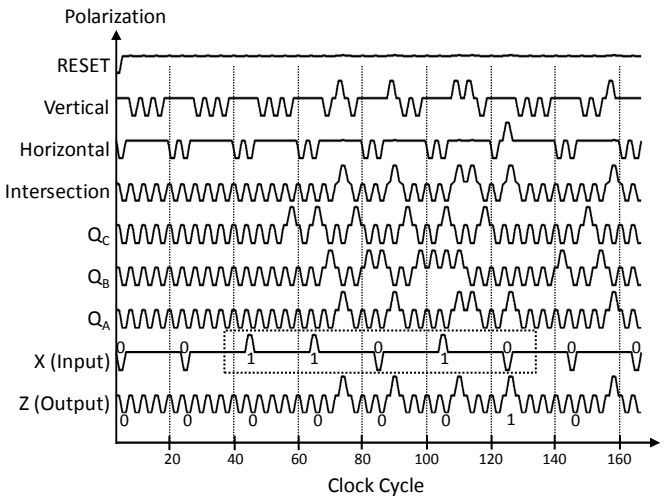


Fig. 19. Simulation results of the system shown in Figure 17 when the clock signals of Figure 18 are applied. It is especially important here to remember that the output Z is only valid in clock cycles 2, 22, 42, 62, etc., as shown by the labels adjacent to that signal. The desired pattern is applied to the input X (as highlighted by the dotted line), resulting in an asserted output on Z in the following clock cycle.

VI. Conclusions and Observations

In this paper, the authors have introduced a specialized Sequential Signal Distribution Network (SSDN) that can be used to route and distribute the signals as needed in a generalized sequential system. This functionality has been demonstrated by the complete design and simulation of three example systems: a two-bit counter, a three-bit counter, and a sequence detector. The SSDN allows designers to apply a generalized strategy for implementing the wire crossings necessary to route signals from the current state to the required inputs of the next state decoder and the output decoder, if one is required.

The examples presented in this paper have been selected to illustrate the features of the SSDN, but the method is entirely general. There is nothing about the examples presented here that make them particularly advantageous to the methods presented. The SSDN is a general tool that can be used to implement any Mealy or Moore finite state machine. Designers who wish to incorporate this tool into their own designs only need to adjust the clock phases of the horizontal lines leading up to the majority logic gates in order to ensure that all input signals arrive at the majority logic gates simultaneously, as illustrated in the three examples presented in this paper.

Although this is a significant step forward in overcoming the wire crossing challenge, this solution still exhibits some undesirable characteristics. In particular, it requires 4·(N-1) clock cycles to route the signals to the appropriate inputs, where N is the number of bits in the current state. Another concern is that it is necessary to be able to control the clocking regions very precisely, with several regions containing only a single cell. In practice, this will be difficult to implement. A third concern is that this network requires six different clock signals (in addition to the Reset signal, which is only used to ensure a well-defined initial state). Since it is possible to implement combinational QCA systems with only four clock signals, this adds complexity to the clocking system required.

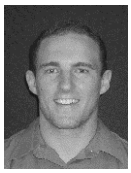
Acknowledgments

This work was supported by the Frederick Jenny Professorship of Emerging Technology and the Leitha and Willard Richardson Professorship of Engineering, both of which are provided through the Valparaiso University College of Engineering.

References

1. C. S. Lent, P. D. Tougaw, and W. Porod, “Bistable saturation in coupled quantum dots for quantum cellular automata,” *Applied Physics Letter*s, vol. 62, no. 7, pp. 714-716, 1993.
2. C. S. Lent, P. D. Tougaw, and Wolfgang Porod, “Bistable saturation in coupled quantum-dot cells,” *Journal of Applied Physics,* vol. 74, no. 5, pp. 3558-3566, 1993.
3. C. S. Lent and P. D. Tougaw, “Lines of interacting quantum-dot cells: a binary wire,” *Journal of Applied Physics*, vol. 74, no.10, pp. 6227-6233, 1993.
4. P. D. Tougaw and C. S. Lent, “Logical devices implemented using quantum cellular automata,” *Journal of Applied Physics,* vol. 75, no. 3, pp. 1818-1825, 1994.
5. C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, “Quantum cellular automata,” *Nanotechnology,* vol. 4, no. 1, pp. 49-57, 1993.
6. C. S. Lent and P. D. Tougaw, “A device architecture for computing with quantum dots,” *Proceedings of the IEEE*, vol. 85, no. 4, pp. 541-557, 1997.
7. D. Tougaw and M. Khatun, “A Scalable Signal Distribution Network for Quantum-Dot Cellular Automata,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 215-224, 2013.
8. K. Hennessy and C. Lent, “Clocking of Molecular Quantum-dot Cellular Automata,” *J. Vac. Sci. Technol*., vol. 19, no. B, pp. 1752- 1755, 2001.
9. C. S. Lent and B. Isaksen, “Clocked molecular quantum-dot cellular automata,” *IEEE Transactions on Electron Devices*, vol. 50, no. 9, pp. 1890–1896, 2003.
10. V. Vankamamidi, M. Ottavi, and F. Lombardi, “Two-Dimensional Schemes for Clocking/Timing of QCA Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.27, no.1, pp.34-44, 2008.
11. G. A. Anduwan, B. D. Padgett, M. Kuntzman, M. K. Hendrichsen, I. Sturzu, M. Khatun, and P. D. Tougaw, “Fault-tolerance and thermal characteristics of quantum-dot cellular automata devices,” *J. Appl. Phys.*, 107, 114306, 2010.
12. M. Khatun, T. Barclay, I. Sturzu, and D. Tougaw, “Fault Tolerance Properties in Quantum-dot Cellular Automata Devices,” *J. Phys. D: Appl. Phys*. 39, pp. 1489-1494, 2006.
13. M. Khatun, T. Barclay, I. Sturzu, and D. Tougaw, “Fault Tolerance Calculations for Clocked Quantum-dot Cellular Automata Devices,” *Journal of Applied Physics*, 98, 094904, 2005.
14. M. Khatun, B. D. Padgett, G. A. Anduwan, I. Sturzu, and D. Tougaw, “Defect and Temperature Effects on Complex Quantum-dot Cellular Automata Devices,” *Journal of Applied Mathematics and Physics*, vol. 1, no. 2, 2013.
15. M. LaRue, D. Tougaw, and J. D. Will, “Effect of Stray Charge in a QCA System: A Validation of the Intercellular Hartree Approximation,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 225-233, 2013.
16. A. Chaudhary, D. Z. Chen, X. S. Hu, K. Whitton, M. Niemier, and R. Ravichardran, “Eliminating Wire Crossings for Molecular Quantum-dot Cellular Automata Implementation,” *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005.
17. B. Smith and S. K. Lim, “QCA Channel Routing with Wire Crossing Minimization,” *Proceedings of the Great Lakes Symposium on VLSI*, 2005.
18. H. Chen and D. Lee, “On crossing minimization problem,” *IEEE Transactions on Computer-Aided Design*, 1998.

19. W.J. Chung, B. Smith, and S. K. Lim, "QCA Physical Design with Crossing Minimization," *Proceedings of the 2005 5th IEEE Conference on Nanotechnology*, 2005.
20. S. Bhanja , M. Ottavi , F. Lombardi , S. Pontarelli, "Novel designs for thermally robust coplanar crossing in QCA," *Proceedings of the Conference on Design, Automation and Test in Europe*, 2006.
21. S. Bhanja, M. Ottavi, F. Lombardi, and S. Pontarelli, "QCA Circuits for Robust Coplanar Crossing," *J. Electron. Test.* 23, pp. 193-210, 2007.
22. C.R. Graunke, D.I. Wheeler, D. Tougaw, and J.D. Will, "Implementation of a crossbar network using quantum-dot cellular automata," *IEEE Transactions on Nanotechnology*, vol.4, no.4, pp. 435-440, 2005.
23. D. Tougaw, "A Clocking Strategy for Scalable and Fault-Tolerant QDCA Signal Distribution in Combinational and Sequential Devices," a chapter in *Field Coupled-Nanocomputing: Paradigms, Processes, and Perspectives*, ed. N. G. Anderson and S. Bhanja, Springer, 2014.
24. S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, McGraw-Hill, 2013.



Hayden Hast is currently a senior electrical engineering student at Valparaiso University. He is a Hesse Scholar and a member of Tau Beta Pi. He plans to join Texas Instruments Inc. as a member of the Global Application Rotational Program after graduation.

Sami Khorbotly (M'06–SM'13) received the B. Eng. degree in electrical engineering from Beirut Arab University, Beirut, Lebanon in 2001. He then received the M.S. and Ph. D. degrees both in Electrical and Computer Engineering from the University of Akron, Akron, OH in 2003 and 2007, respectively. He is currently an Associate Professor of Electrical and Computer Engineering and the Frederick F. Jenny Professor of Emerging Technologies at Valparaiso University. His research interests include digital circuits design, fixed-point DSP, and robotic systems.

Douglas Tougaw (M'91–SM'02) received the B.S. degree in electrical engineering from the Rose-Hulman Institute of Technology, Terre Haute, IN, in1991, and the M.S. and Ph.D. degrees in electrical engineering from the University of Notre Dame, South Bend, IN, in 1994 and 1996, respectively, under the guidance of Dr. Craig Lent. He also received the MBA degree from Valparaiso University, Valparaiso, IN, in 2005. He is currently a Professor of Electrical and Computer Engineering and the Richardson Professor of Engineering at Valparaiso University. His research interests include nanotechnology, engineering education, and engineering ethics.