ELSEVIER

Microelectronics Journal 38 (2007) 525–537

Microelectronics Journal

www.elsevier.com/locate/mejo

Design of sequential circuits by quantum-dot cellular automata

J. Huang*, M. Momenzadeh, F. Lombardi

Department of Electrical and Computer Engineering, Northeastern University, USA

Received 3 January 2007; accepted 3 March 2007
Available online 25 April 2007

Abstract

This paper proposes a detailed design analysis of sequential circuits for quantum-dot cellular automata (QCA). This analysis encompasses flip-flop (FF) devices as well as circuits. Initially, a novel RS-type FF amenable to a QCA implementation is proposed. This FF extends a previous threshold-based configuration to QCA by taking into account the timing issues associated with the adiabatic switching of this technology. The characterization of a D-type FF as a device consisting of an embedded wire is also presented. Unique timing constraints in QCA sequential logic design are identified and investigated. An algorithm for assigning appropriate clocking zones to a QCA sequential circuit is proposed. A technique referred to as stretching is used in the algorithm to ensure timing and delay matching. This algorithm relies on a topological sorting and enumeration step to consistently traversing only once the edges of the graph representation of the QCA sequential circuit. Examples of QCA sequential circuits are provided.

Keywords: QCA; Sequential design; Emerging technology; Delay matching; Adiabatic switching

1. Introduction

At nano regimes, computation is substantially different from conventional VLSI. In recent years, the so-called emerging technologies have been investigated to take advantage of extremely small feature size and high device density for implementing new computational paradigms at physical and logic levels for manufacturing, design and assembly. Among emerging technologies, *quantum-dot cellular automata* (QCA) [1] relies on novel design concepts to exploit new physical phenomena (such as Coulombic interactions), and implement unique paradigms (such as *memory-in-motion* and *processing-by-wire* [2,3]). In QCA, gates (such as the inverter, INV and majority voter, MV) and other devices (such as the binary wire and the inverter chain) have been proposed as primitives for combinational circuit design. As a combined methodology for *computation and communication* [3], various designs of combinational as well as memory circuits have been proposed for QCA implementation [3–5]. Clock circuitry design and floor planning have been addressed in previous research [2,6].

Recent developments in QCA manufacturing (involving molecular implementations) have substantially changed the nature of processing. At very small feature sizes, it is anticipated that either *self-assembly* or large-scale cell deposition on insulated substrates will be utilized. In these implementations, QCA cells (each made of two dipoles) are deposited on parallel V-shaped tracks. The QCA cells are arranged in a dense layout and computation occurs among adjacent cells [7]. These manufacturing techniques are well suited for molecular implementation.

The design and characterization of sequential circuits in QCA have not been fully addressed in the technical literature. While sequential elements can be implemented using QCA memory cells [3], such an approach would be prohibitive in terms of hardware (due to its extensive control circuitry) and very slow in performance. As QCA memories rely on the paradigm of memory-in-motion [3], a longer access time should be expected due to the latency in the storage elements. Moreover, sequentiality in QCA does not have the same requirements as in CMOS-based

*Corresponding author. Tel.: +1 617 373 7780.
E-mail addresses: hjing@ece.neu.edu (J. Huang), mmomenza@ece.neu.edu (M. Momenzadeh), lombardi@ece.neu.edu (F. Lombardi).

0026-2692/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2007.03.013

circuits. In CMOS, operation is controlled through a clock that controls information flows. Latching is implicitly implemented in QCA as sequential behavior is dependent on adiabatic switching and the layout of the QCA cells. The four-phase adiabatic clocking scheme for QCA introduces timing by dividing the QCA circuit into zones; this unique feature imposes strict timing constraints on QCA sequential circuits. In QCA, sequential behavior must be strictly controlled as feedback paths traversing different zones may cause uneven delays among signals.

The objective of this paper is to propose a detailed analysis and design strategy for sequential circuits for QCA. This analysis encompasses flip-flop (FF) devices as well as circuits. Initially, a novel RS-type FF as well as D-type FF is presented as basic elements in sequential QCA design. These FFs have been introduced in [8] by the same authors. Unique timing requirements of clocked QCA are identified and explored. Sequential QCA circuit design is then discussed in detail, with emphasis on timing and delay matching. As a further contribution of this work, a novel algorithm is proposed to assign appropriate clocking zones to QCA cells, such that timing constraints are met. The algorithm is applicable to general FF-based QCA sequential circuit inclusive of the coplanar crossing. Examples are provided, all circuits have been simulated and verified using QCADesigner [17].

This paper is organized as follows: Section 2 describes the basic preliminaries inclusive of QCA technology and clocking requirements. Section 3 presents two novel FF arrangements (D- and RS-types) as basic elements for QCA sequential design. Section 4 discusses the different constraints encountered in QCA with respect to timing and clocking due to adiabatic switching. In Section 5, the stretching algorithm for delay matching is presented. Section 6 extends this algorithm to the so-called coplanar device of QCA. Section 7 provides three examples as an application of the proposed technique to different sequential circuits. Conclusions are given in the last section.

2. Preliminaries

A QCA cell can be viewed as a set of four charge containers or "dots", positioned at the corners of a square. The cell contains two extra mobile electrons that can quantum mechanically tunnel between dots, but not cells. The electrons are forced to the corner positions by Coulombic repulsion. The two possible polarization states represent logic "0" and logic "1". Unlike conventional logic circuits in which information is transferred by electrical current, QCA operates by the Coulombic interaction that connects the state of one cell to the state of its neighbors. This results in a technology in which information transfer (interconnection) is the same as information transformation (logic manipulation).

One of the basic logic gates in QCA is the MV with logic function $\mathrm{MV}(A,B,C) = AB + AC + BC$. MV can be realized by five QCA cells, as shown in Fig. 1(b). Logic

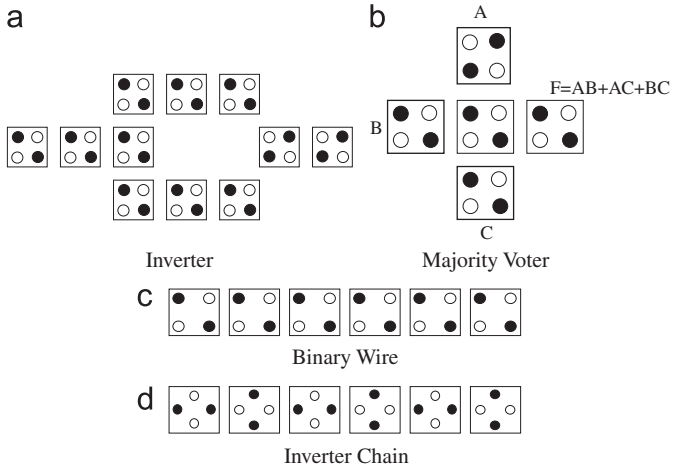


Fig. 1. Basic QCA devices: (a) inverter; (b) majority voter; (c) binary wire; (d) inverter chain.

AND and OR functions can be implemented from the MV by setting an input (the so-called programming or control input) permanently to a 0 or 1 value. The INV is the other basic gate in QCA and is shown in Fig. 1(a). The binary wire and inverter chain (as interconnect fabric) are shown in Fig. 1(c) and (d). Differently from CMOS, QCA cells of different orientation can be used to accomplish coplanar crossing.

A QCA circuit operates by mapping the ground state to the logic solution that the circuit is designed to generate [9]. In this system, thermodynamic effects must be considered. If thermal fluctuations excite QCA cells above ground state, erroneous computation may appear at the outputs of the circuit. For robust operation, the excitation energy must be well above $k_B T$, where k_B is Boltzmann's constant and T is the operating temperature. As cell size decreases using molecular implementations, then the energy separation between states increases and higher temperature operation is possible [10].

Thermodynamic effects thus impose limitation on the number of cells that can operate correctly in a QCA system. Consider a binary wire of N cells. Let E_{kink} be the energy required for a QCA cell to be affected by kink (i.e. to align differently from its expected polarization). As E_{kink} is independent of location, then for large values of N, E_{kink} remains the same while the locations in which kink can occur, increase. So multiple kinks may be possible [9]. Therefore, the probability of having kink is a function of N. As shown in [9], for fixed values of T and E_{kink} the maximum number of cells is given by $\exp(E_{\mathrm{kink}}/k_B T)$. Therefore, a QCA system must be partitioned into zones such that the length of any binary wire inside a zone exceeds $\exp(E_{\mathrm{kink}}/k_B T)$.

Two types of switching are possible in the operation of QCA: abrupt switching and adiabatic switching. In abrupt switching, the inputs to the QCA circuit change suddenly and the circuit can be in some excited state; subsequently,

the QCA circuit is relaxed to ground state by dissipating energy to the environment [9]. This inelastic relaxation is uncontrolled and the QCA circuit may enter a metastable state that is determined by a local rather than a global energy ground state. Therefore, adiabatic switching is usually preferred; in adiabatic switching, the system is always kept in its instantaneous ground state.

A clock signal is utilized to ensure adiabatic switching for QCA. The clock signals for QCA are generated through an electric field that is applied to the cells to either raise, or lower the tunneling barrier between dots within a QCA cell. When the barrier is low, the cells are in a non-polarized state; when the barrier is high, the cells are not allowed to change state. Adiabatic switching is achieved by lowering the barrier, removing the previous input, applying the current input and then raising the barrier [9]. If transitions are carried out gradually, the QCA system will remain close to the ground state. This clocking scheme consists of four phases: switch, hold, release and relax, as shown in Fig. 2(a). The QCA circuit is partitioned into so-called *clocking zones*, such that all cells in a zone are controlled by the same clock signal. Cells in each zone perform a specific calculation; the state of a zone is then fixed so that input signals can be provided to the next zone [2]. At the beginning of the switch phase, the cells in a zone are in an unpolarized state. During the switch phase, the barrier is raised, and the cells in a zone begin to polarize according to the input (as provided by the output of the previous zone). At the same time, the next zone is in an unpolarized state, thus not affecting its computational state. Then, the cells are placed in the hold phase, in which the barrier is sufficiently high such that the state is unchanged and can be used as input to the next zone. After lowering the barrier, the cells are placed in a release phase in which they relax to an unpolarized state. The last clock phase is the relax phase in which the cells are again unpolarized. The binary wire in Fig. 2(b) can be used to illustrate this clocking scheme. Initially, subarray1 is switching as according to the fixed input. Then, subarray1 enters the hold phase; at this time subarray2 starts switching. As subarray3 is in the relaxed state, then it will not influence the computational state of subarray2. At the next phase, subarray1 is moved to a release phase; subarray2 is in the hold state and provides the input to subarray3 (that is in the switch phase). Thus, information transfers in a pipelined fashion. The clock signals (electric field) can be generated by embedded CMOS wires below the QCA plane.

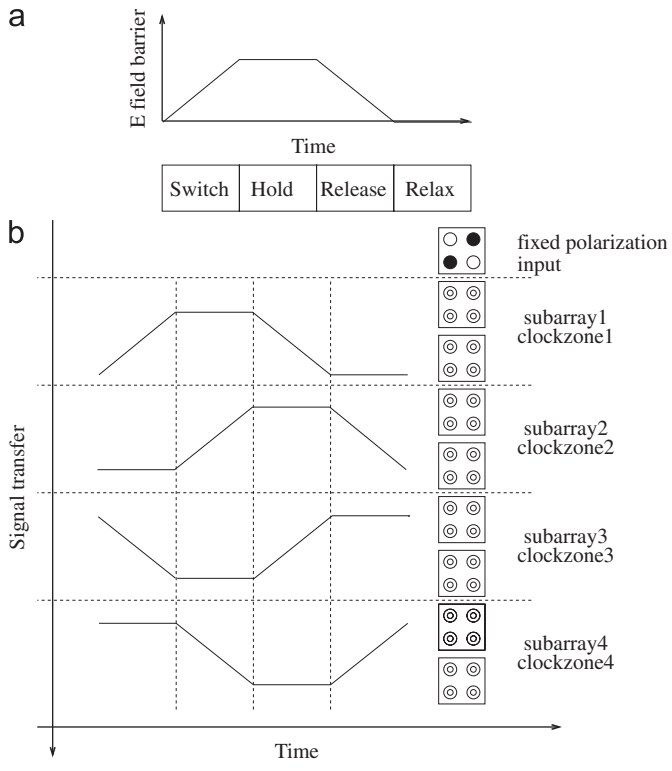
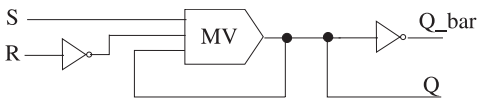


Fig. 2. Clocking in QCA: (a) four-phase clocking and (b) switching of a binary wire.

Clocking not only permits control over the information flow, but also provides true power gain to QCA devices. As information moves from cell to cell, signal energy is lost to the environment due to dissipative processes. In clocked QCA, this energy is originated from the clock with no flow of current. The four-phased clocking scheme has been used to design various QCA combinational circuits [5,11]. In a clocked QCA circuit, information is transferred and processed in a pipeline fashion. All cells within the same zone are allowed to switch simultaneously, while cells in different zones are isolated. Consider again the binary wire in Fig. 2(b); initially, subarray1 switches according to the fixed input, and subarray2 shows no definite polarization at this time. The signal is "latched" when subarray1 enters the hold phase and acts as input to subarray2. subarray2 processes its information, i.e. effectively a latch is present between the two subarrays and they perform independent computation in their cells. However, very little investigation has been reported on sequential circuits in QCA. Apart from memory design, no sequential device can be found in the technical literature for QCA.

3. QCA flip-flops

In this section, the novel QCA designs of two FFs are presented: RS and D types, respectively.



R	S	Q
0	0	Q_old
0	1	1
1	0	0
1	1	Q_old

Fig. 3. Schematic diagram of the QCA RS flip-flop.

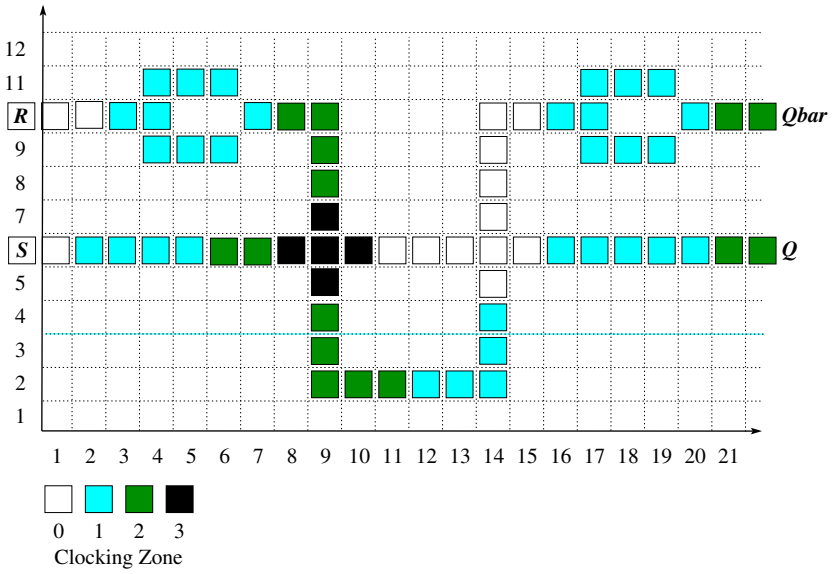


Fig. 4. QCA layout of the RS flip-flop.

3.1. RS flip-flop

This new device is shown in Fig. 3 and represents a novel QCA extension of the original scheme given in [12] for threshold logic. The basic component in the RS FF is the MV. If the setting input S is 1 and the resetting input R is 0, then the stored value of the FF is 1. The output value is changed to 0 if R is 1 and S is 0. When both S and R have the same value, then the output value remains unchanged. Fig. 4 shows the QCA layout of the RS FF. The threshold scheme of [12] requires a three-phase synchronization process; although it is possible to use three-phase synchronization in QCA, the four-phase clocking scheme commonly employed in QCA is used here. In QCA, a clock cycle consists of four clocking zones. In this design, the number of phases for synchronization is limited by the architecture of the inner loop (feedback) in the RS FF. The delay of the inner loop must be a multiple of a complete clock cycle, i.e. the number of clocking zones in the inner loop must be a multiple of four. In this case, the old value of Q can be made available during the next computation, i.e. after k complete clock cycles, where k is an integer. In the RS FF of Fig. 4, the x and y coordinates are used to identify the QCA cells in the Cartesian layout. The inner loop of the FF has a delay of one clock cycle; at the output, Q is available six clocking zones after R and S have been applied.

3.2. D flip-flop

If adiabatic switching is employed (as discussed in a previous section), latching is effectively accomplished through timing by using a four-phase clocking arrangement. Therefore, a device with an equivalent behavior as a D-type FF (D FF) can be constructed by a QCA binary wire with four clocking zones (i.e. it can be buried in a design). In this case, the input signal is delivered to the output after at least one complete clock cycle delay and control is accomplished by timing. The relative simplicity of a D FF over the RS FF (no MV required in the former arrangement) seems to suggest that sequential design in QCA could be achieved at ease within the Cartesian layout. However as shown in the next section, timing and signal delay must be carefully considered.

4. Timing constraints in QCA sequential design

In conventional logic design, synchronous operation is usually implemented in a sequential circuit. This circuit can be represented by a Mealy machine model that consists of two parts: the FFs and the combinational logic. This general model can also be used for a QCA implementation of sequential logic [13]. However, in QCA the four-phase clock signal controls not only the FFs, but also the combinational gates. As described previously, the entire QCA circuit is pipelined and latched by the clock signals; an important timing constraint in a QCA design is that for every logic gate, all inputs must arrive at the same time, i.e. all inputs must be in the same clocking zone. In synchronous sequential logic, not only the inputs need to be in the same clocking zone, but also all FFs should compute at the same time; therefore when designing this type of circuit in QCA, it is necessary to ensure that all paths from the outputs of the FFs (passing through the combinational logic) to the inputs of the FFs have the same delay (i.e. the same number of traversed clocking zones), thus enforcing the condition that signals arrive at the inputs of the FFs at the same time. This process is, however, dependent on the FF-type and the layout of the QCA circuit.

4.1. Timing constraints using RS FF

For QCA sequential designs with RS FFs, the timing constraints are as follows:

- all state variables must be updated at the same time, as required by a synchronous sequential design. If the state variables are chosen to be the output of the MVs in the RS FFs, then all MVs in the RS FFs must be in the same clocking zone;
- for each MV, all inputs must arrive at the same time, i.e. all paths from the output of an MV in the RS FF to the input of an MV in the RS FF must have the same delay (as given by the number of clocking zones).

These timing constraints are illustrated in the example shown in Fig. 5. This circuit is a 2-bit gray code counter. Two RS FFs are employed in the design whose state transition diagram is shown in Fig. 5. The circuit functions as follows. When RESET $= 0$, $Q1Q2 = 11$; when RESET $= 1$, $Q1Q2$ counts the gray code sequence: $00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00 \ldots$. For the two FFs to compute at the same time, MV1 and MV2 must be placed in the same clocking zone, i.e. $Q1$ and $Q2$ are in the same clocking zone. The corresponding timing constraint in the signals consists of ensuring that for each MV, all three inputs must arrive at the same time. This corresponds to the condition by which the paths p1, p2, p3, p4, p5 and p6 must have the same delay (given by the number of clocking zones). Two of these paths (p1 and p2) are the inner loops of the RS FF. As the inner loop must have a delay that is a multiple of the clock cycle (for adiabatic switching, one clock cycle consists of four clocking zones), then a strict timing arrangement must be implemented in the QCA design.

The RS FF shown in Fig. 3 has one complete clock cycle delay for the inner loop. However, the number of QCA cells that can be placed in the same clocking zone, is bounded. In a complex sequential design, a path that goes through a number of combinational logic gates may require more than one complete clock cycle. In this case, the delay is determined by the longest path; the other paths must be increased through a so-called stretching process by which these paths match the longest delay (including the inner loops of the FFs). If all paths have a delay of exactly k cycles, then a valid output will be produced every k cycles. However, k should have a small value (preferably 1) to maintain the data flow in the pipeline of the QCA circuit.

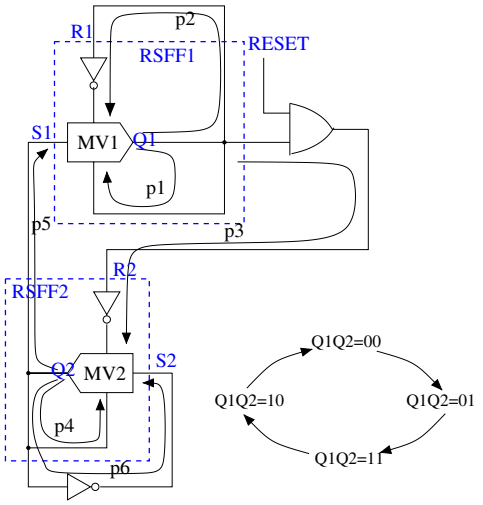


Fig. 5. Schematic diagram of devices in QCA 2-bit gray code counter.

4.2. Timing constraints using D FFs

Similar timing constraints are applicable to the QCA design of sequential circuits using D FFs. As outlined previously, a D FF in QCA is a binary wire operating over one (or multiple) full clock cycles. In a D FF-based design, the D FF is effectively "buried" in other logic gates. An example is shown in Fig. 6, this sequential circuit is the so-called *traffic light*. The pair $Q1Q2$ defines the state variables of the traffic light as follows: $Q1Q2 = 00$ is green, $Q1Q2 = 01$ is yellow and $Q1Q2 = 11$ is red. W is the pedestrian crossing (input) signal. $W = 1$ denotes a pedestrian's request to cross. The circuit functions are as follows:

- when the light is green, at the next step the light becomes yellow;
- when the light is yellow, at the next step the light becomes red;
- when the light is red, and there is a pedestrian request to cross, the light remains red; otherwise, the light becomes green.

The corresponding QCA layout of the sequential circuit is shown in Fig. 7. In this example, the first D FF is "buried" inside the loop p1, while the second D FF is "buried" inside the loop p2. For example, p1 consists of one INV, two MVs as well as the binary wires connecting them; the D FF is embedded inside the INV, MVs and wires.

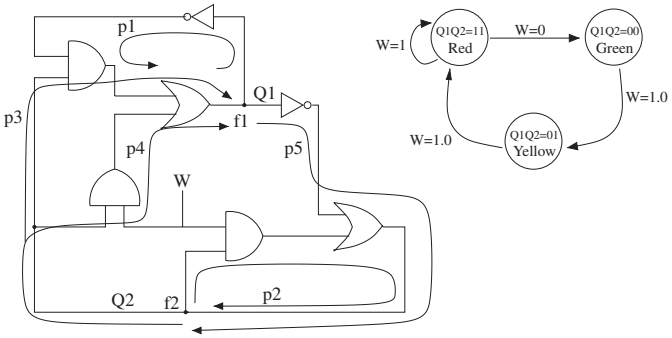


Fig. 6. Schematic diagram of the traffic light.

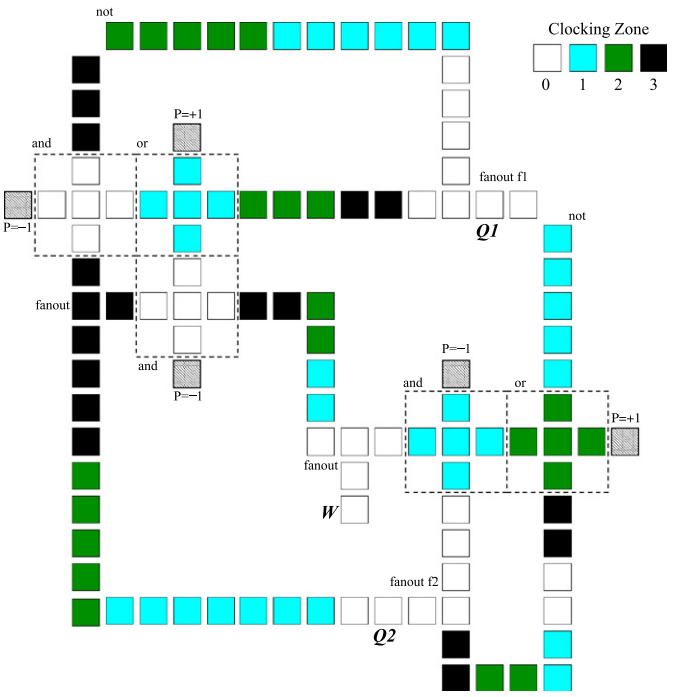


Fig. 7. QCA layout of the traffic light.

The state variables $Q1, Q2$ can be chosen anywhere in the cells of the loops. In this example, they are chosen to be the output at the fanout points ($f1$ and $f2$) as shown in Fig. 7. Thus, $f1$ and $f2$ must be in the same clocking zone such that the two state variables are updated simultaneously. Furthermore, it is required that all inputs of an MV must arrive at the same time. In summary, the paths p1, p2, p3, p4 and p5 must have the same delay (number of clocking zones). Note that in this design, the two paths p3 and p4 share a common part. From the QCA layout of Fig. 7 the longest delay path is incurred in p3 and p4, both of which have a delay of two clock cycles. Therefore, the other paths must be adjusted (for matching a two clock cycle delay); this process is referred to as *stretching* and it will be described in detail next.

5. Stretching algorithm for delay matching

In this section, a clocking *zone assignment* algorithm to meet the timing constraints in QCA sequential circuits, is proposed. This algorithm is a novel modification of the algorithm introduced in [14] for satisfying the timing constraints in reconvergent paths for QCA circuits.

In this algorithm, the gates consist of MVs, INVs (both as active gates), fanouts and wires. For example, a wire gate is basically a binary wire that performs the identity function. It is assumed that each gate is placed in a clocking zone. If the length of the wire (number of cells) is greater than $\exp(E_{\text{kink}}/K_{\text{B}}T)$, then the wire is further partitioned such that the length of any wire in a single zone does not exceed $\exp(E_{\text{kink}}/K_{\text{B}}T)$ (as previously described).

The algorithm assigns clocking zones to each gate by *enumerating* them. The proposed method can be explained as follows. Each gate (active gate, fanout, wire) is initially assumed to be in its own clocking zone. Hence, a *directed cyclic graph* $G' = (V', E')$ can be used to model the sequential circuit. Each gate is represented by a vertex in G'. If the output of gate u drives the input of gate v, a directed edge $(u, v) \in E'$. Next, G' is transformed into a *directed acyclic graph* (DAG) G by breaking (opening) the feedback loops. Then, a vertex (as starting point in the execution of the algorithm) referred to as the *Super Source*, is added to the graph representation of the QCA circuit. The preliminary step of this process consists of adding edges between the Super Source and the vertices that represent the FFs. The algorithm takes this transformed graph as input and finds for each vertex of the DAG the longest path from the Super Source. Since the graph is a DAG, the vertices can be arranged in a *topologically sorted order*. The Bellman–Ford algorithm can then be applied in this order to find the longest paths [15]. The topological sorting step ensures that a vertex is processed only when all its parents have been processed. Each node is given a label for its clock number (this corresponds to the number of zones from the Super Source). The clock number of the child node is one more than the largest clock number of its parents.

Next, the paths are stretched (by adding edges) for delay matching. In the stretching process, if the number of clocking zones of two nodes with a common edge differs by i, then $i-1$ vertices are added between these nodes. When adding vertices, the algorithm considers the effect of shared paths to reduce the number of additional clocking zones. Fig. 8(a) illustrates an example. Assume that clocking zones U_1, U_2 and U_3 are required between node A and nodes B and C, respectively. Using the proposed algorithm for stretching (Fig. 8(b)), node U_1 is added to the shared path and only two clocking zones are inserted.

Let the *center gate* of a FF be defined as the gate (MV, INV, fanout) whose output is the state variable. The state variables in a synchronous sequential design must be updated at the same time, therefore all center gates must be in the same clocking zone. The center gate can be chosen arbitrarily inside a FF, for convenience the center gate for an RS FF is chosen to be the MV inside the FF itself (Fig. 3). For a D FF, the center gate is the gate whose output is the state variable. The center gate can be chosen anywhere in the feedback loop. For example the fanouts $f1$ and $f2$ in the traffic light example are the center gates (Fig. 6). From the previous discussion, the following

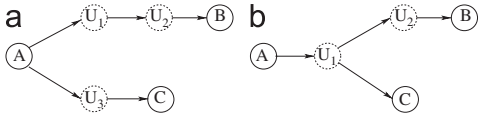


Fig. 8. Stretching considerations for path sharing.

conditions must be met at circuit level to meet the timing constraints:

- all center gates must be in the same clocking zone;
- all paths from a output of a center gate to an input of a center gate must have the same delay;
- all paths from the primary inputs to a center gate must have the same delay.

A synchronous sequential circuit can be represented by a Mealy machine. Timing constraints must also be imposed on any combinational logic prior to the Mealy machine itself. This input logic block is shown in Fig. 9 as CL1. Therefore, another timing requirement must be considered such that all primary inputs are in the same clocking zone as the FF. In this arrangement, the primary inputs are synchronized with the state variables of the sequential machine. This is achieved by adding an edge from the Super Source to each primary input.

A new *graph-based model* is proposed for the QCA sequential circuit. This is formally given by an unweighted directed acyclic graph $G' = (V', E')$. This graph is transformed into a DAG $G = (V, E)$ with the so-called *vertex splitting* step as follows:

- Each center gate CG is represented by two vertices $u' \in V$ and $u'' \in V$ (vertex splitting).
- All inputs to each CG are modeled by edges entering u'.
- All outputs of CG are modeled by edges leaving u''.

u' is called an *input center vertex*, while u'' is called an *output center vertex*. There are two types of loop inside a graph in the proposed model for QCA circuits: (1) self-loops (from a FF to itself) and (2) connecting loops (from one FF to another). The process of splitting the center gates cuts both these types of loops, thus breaking all feedback paths in the circuit. Consider Fig. 9; if all FFs in a sequential circuit are open-looped, then the resulting circuit is combinational. Therefore, the corresponding graph is a DAG because no loop exists in a combinational circuit by definition (no feedback path).

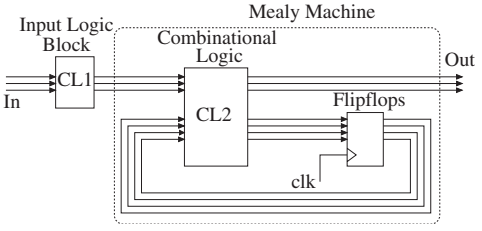


Fig. 9. Mealy model of a sequential machine with additional combinational logic at the primary inputs.

In the next step, a Super Source vertex (denoted by ss) is added to G. An edge is added from ss to each of the output center vertex as well as the primary input vertex. After this modification, G is still a DAG. Let the clocking zone of gate u be denoted as clk(u). The proposed algorithm (denoted as AssignClk(), see Algorithm 1) assigns the clocking zones to each gate $u \in V$. The algorithm starts at ss and initially assigns clk(ss) $= -1$. A sorting step is done for the DAG such that the vertices are arranged in a topologically sorted order. Next Function NumerateDAG() (see Algorithm 2) is executed such that each vertex u is assigned a clk(u). In NumerateDAG() the vertices are processed in a topologically sorted order, such that u is processed only after all its parents have been processed. After the execution of Function NumerateDAG(), the clock assignment satisfies the following condition: let the parents of vertex u be $v1, v2, \ldots, vn$, then clk(u) is assigned the maximum value of clk($v1$), clk($v2$), ..., clk(vn) plus 1. As all center output vertices as well as the primary input vertices are children of ss, then they will be assigned to clocking zone 0.

After executing the Function NumerateDAG(), further processing is needed for the center vertices. For each center vertex CG, the two vertices u'' and u' represent the same gate; therefore, they must be in the same clocking zone. This is clock zone 0 because clk(u'') $= 0$ for all output center vertices u''. So the first requirement is that the clocking zone of all input center vertices must satisfy the condition clk(u') modulo $4 = 0$. However, this is not necessarily applicable after the execution of the Function NumerateDAG(). Hence, an adjustment may be required. The second requirement is that all input center vertices must have the same clock number. Let k' be the maximum of clk(u') among all input center vertices. Let k be the smallest integer that is greater than k' and is a multiple of 4. k is assigned to clk(u') for all input center vertices u'. For example, if k' is 6, then $k = 8$.

After all vertices have been numerated, the algorithm AssignClk() stretches the short paths to match the longer ones as follows. The function minchild(v) finds the minimum clk among v's children, where children(v) denotes

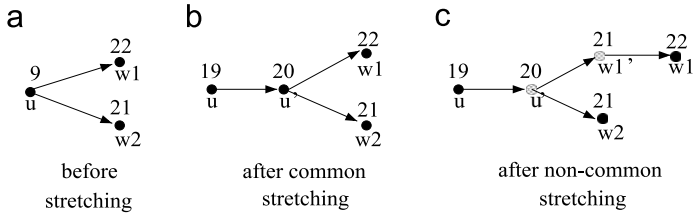


Fig. 10. Example of stretching of common and non-common paths: (a) before stretching; (b) after common stretching; (c) after non-common stretching.

the set of v's children. Initially, the path between the input center vertices u' and its parent v is stretched. If $\mathrm{clk}(u') > \mathrm{clk}(v) + 1$, then node u is extended, such that a number of nodes (given by $\mathrm{clk}(u') - \mathrm{clk}(v) - 1$) are added between v and u'. Next, stretching is performed on all other vertices u. Stretching of a common path is considered first. Let w be the child of u with the smallest clk value among all u's children. Node u is extended such that a number of nodes (given by $\mathrm{clk}(w) - \mathrm{ckl}(u) - 1$) are added between u and the children of u. Finally, stretching of non-common paths between u and its children is performed. This is illustrated by the example as shown Fig. 10. u has two children, $w1$ and $w2$. Stretching is required because $\mathrm{clk}(u) = 19$, $\mathrm{clk}(w1) = 22$ and $\mathrm{clk}(w2) = 21$. Initially common path stretching is performed (in which u' is added to the graph). Then non-common path stretching is performed, in which $w1'$ is added.

The final graph that satisfies all timing requirements, has the following characteristics:

1. for every center gate CG, $\mathrm{clk}(u'') = 0$, $\mathrm{clk}(u') = k$ where k modulo $4 = 0$;
2. for all primary input vertex u, $\mathrm{clk}(u) = 0$;
3. for each edge (u, v) from u to v, $\mathrm{clk}(u) + 1 = \mathrm{clk}(v)$.

The algorithm and corresponding subroutine are given in Algorithms 1 and 2, respectively. As four-phased clocking is used in QCA, the number of the clocking zone of any gate u must be $\mathrm{clk}(u)$ modulo 4. For example if $\mathrm{clk}(u) = 7$, u should be in zone 3.

The complexity of the proposed algorithm is computed as follows. The topological sorting step of G can be

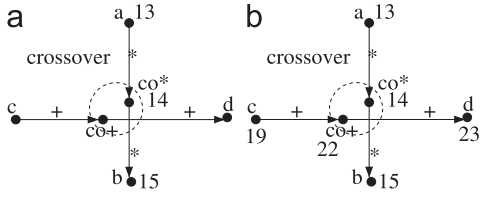


Fig. 11. Example of crossover by using the coplanar device.

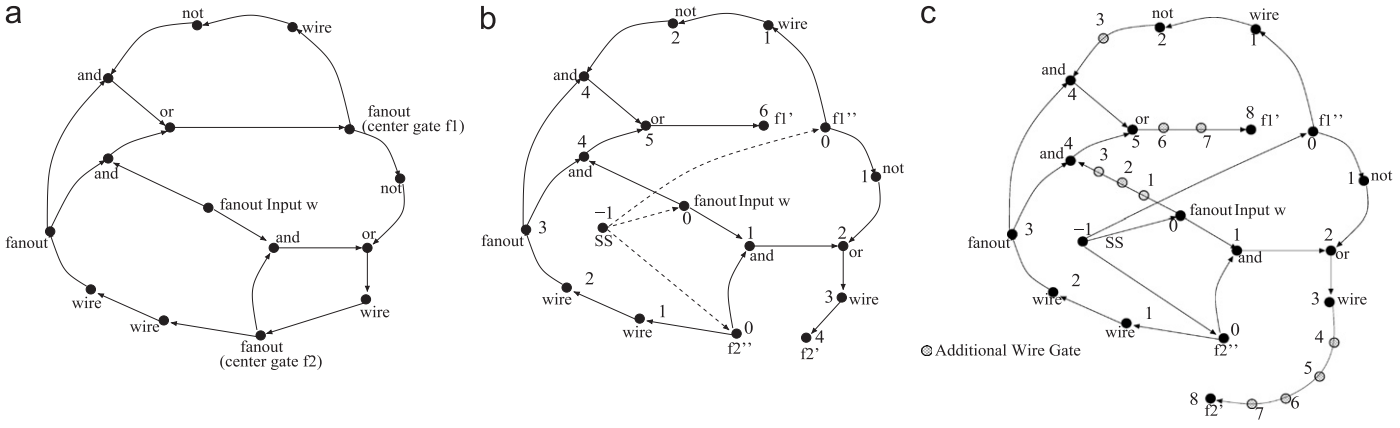


Fig. 12. Graph model of the QCA traffic light circuit.

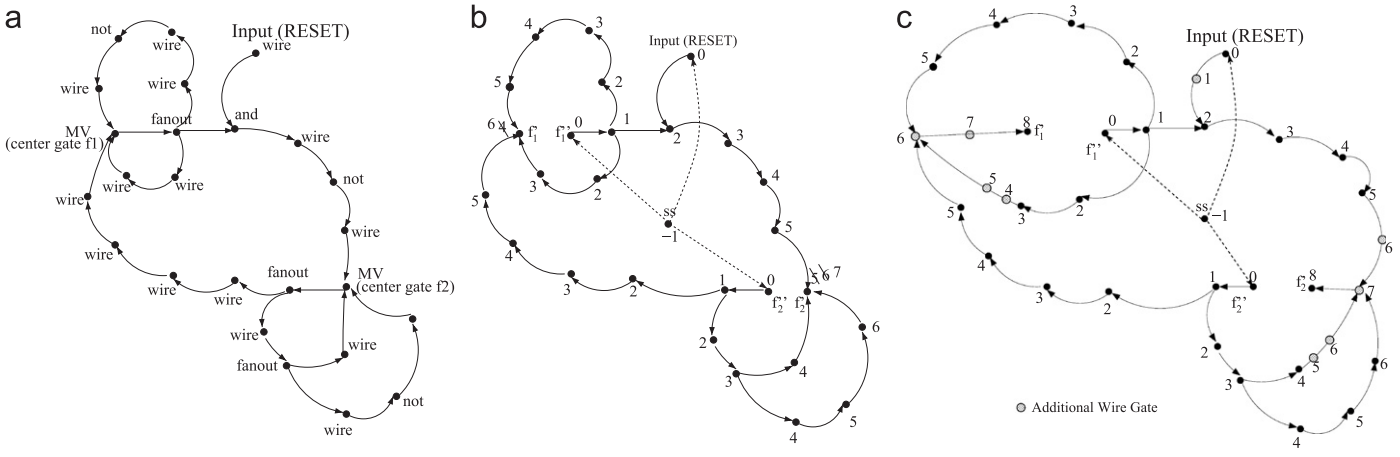


Fig. 13. Graph model of QCA 2-bit gray code counter circuit.

performed in $O(|V|+|E|) = O(|E|)$ time [15]. Function NumerateDAG() has a complexity of $O(|E|)$, because each edge is traversed exactly once. Stretching considers each edge in G once by inserting the required vertices and edges. Since for each original edge, at most $O(|V|)$ vertices need to be added, then stretching has a complexity of $O(|V||E|)$. Thus, Function AssignClk() has a time complexity of $O(|V||E|)$.

6. Algorithm for coplanar device

In this section, an assignment algorithm is introduced to satisfy the timing constraints imposed by the so-called coplanar device in QCA circuits. In this device, two separate QCA wires cross on the Cartesian plane without affecting each other. This operational feature is valid provided the wires are in the same clocking zone at the crossing point. Therefore, the values of the clock zones of the wires at the crossing point must be modulo 4 of each other.

The proposed algorithm can be explained as follows. The crossover is identified in the graph by a pair of vertices (denoted as co* and co+). An example is shown in Fig. 11; a is the parent of co*, while c is the parent of co+; b is the child of co*, while d is the child of co+. No cycle is introduced because the crossover is treated as two separate vertices (hence, with the crossover device the graph G is still a DAG). The algorithm introduced in the previous section is applicable to crossover vertices with a slight modification to Function NumerateDAG(). The topological sort as well as the stretching steps in Algorithm AssignClk() can be used for the crossover vertices with no modification by treating each crossover as two separate vertices. However, in Function NumerateDAG() co* and co+ must be in the same clocking zone, i.e. $(\text{clk}(\text{co}^*) - \text{clk}(\text{co}^+))$ modulo $4 = 0$. During the execution of Function NumerateDAG(), if a crossover vertex is encountered, Function Crossover() (see Algorithm 3) is called. When reaching a crossover vertex for the first time (say co*), the execution of the algorithm is unchanged and clk is assigned to the crossover vertex (co*). When the crossover vertex is accessed for the second time (i.e. co+), then co+ is numbered such that it is placed in the same clocking zone as co*. For example, in Fig. 11 assume co* is reached first. Since $\text{clk}(a) = 13$, so $\text{clk}(\text{co}^*) = 14$. The crossover point is then processed for a second time when co+ is reached. As clk(co*) modulo $4 = 2$, then clk(co+) must satisfy clk(co+) modulo $4 = 2$ too. Assume $\text{clk}(c) = 19$, then clk(co+) is assigned 22, which is the smallest integer i bigger than 19 for which i modulo $4 = 2$.

7. Examples of QCA circuits

In this section, three sequential circuits are presented as examples of the application of the proposed algorithm.

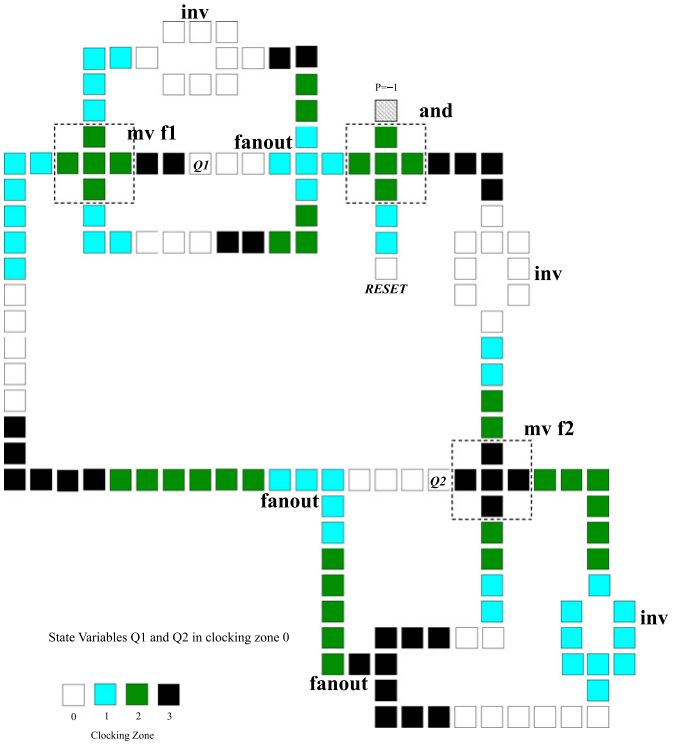


Fig. 14. QCA layout of 2-bit gray counter.

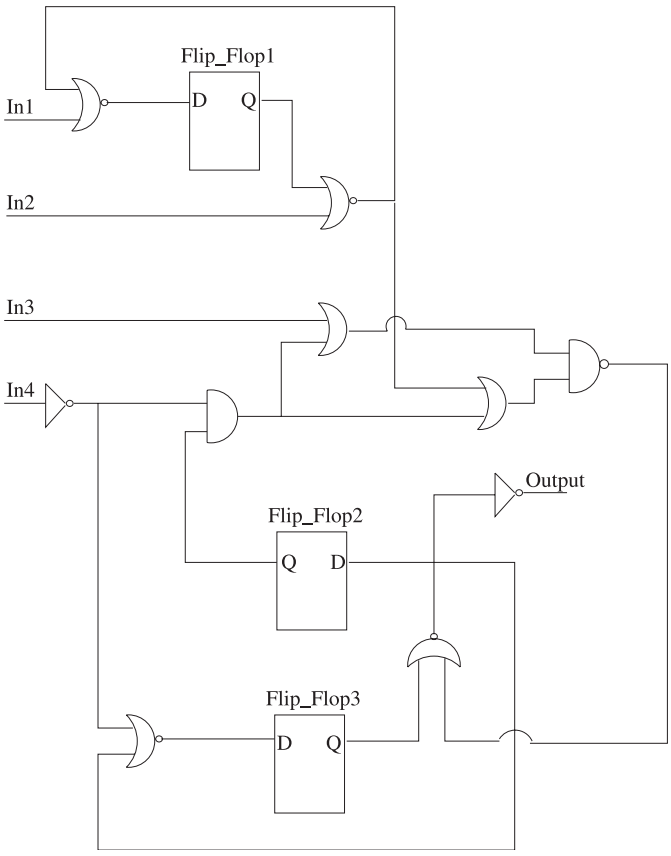


Fig. 15. Schematic of S27 in ISCAS89.

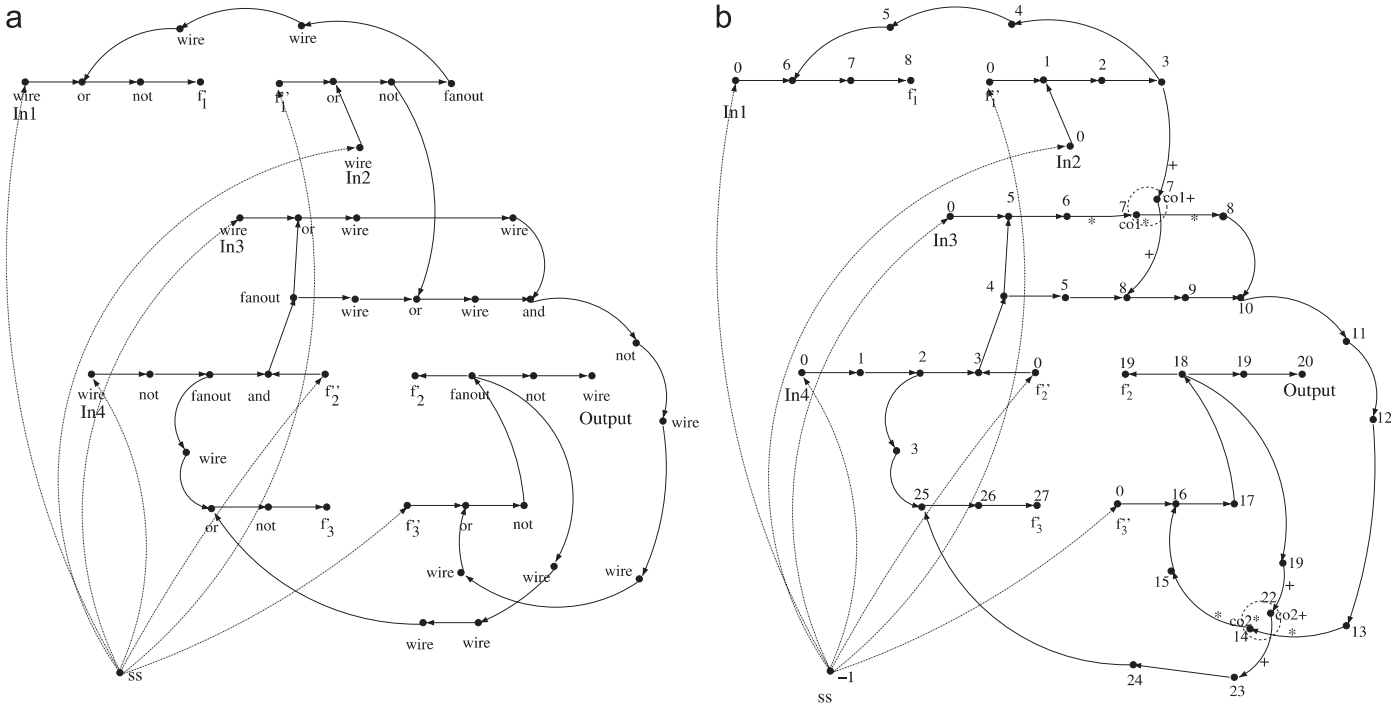


Fig. 16. Graph model of QCA S27 benchmark circuit, ISCAS89.

- The first example is the traffic light discussed previously, the graph model of which is shown in Fig. 12. The center gates are the fanout gates $f1$ and $f2$. The original graph is given in Fig. 12(a). It can be seen that the center gates are the fanout gates. The graph after applying Function NumerateDAG() is shown in Fig. 12(b). The center gates have been split and shown as $f1'$, $f1''$ and $f2'$, $f2''$. In this graph, the labels given by integers indicate the clocking zone assignment. As $f1' = 6$, $f2' = 4$, then in the next step the algorithm assigns $f1' = f2' = 8$ as a multiple of four. Two additional wires are added between the parent of $f1'$ (an OR gate) and $f1'$, while four additional wires are added between the parent of $f2'$ (a wire gate) and $f2'$. The final graph after stretching is shown in Fig. 12(c); this matches the layout of the traffic light given previously in Fig. 7. Note that in the layout the state variables are Q1 and Q2, both in clocking zone 0.
- The second example is the 2-bit gray code counter discussed previously in Section 4. The original graph is given in Fig. 13(a); the center gates are the MVs $f1$ and $f2$. The graph after applying the Function NumerateDAG() is shown in Fig. 12(b). The center gates have been split and shown as $f1'$, $f1''$ and $f2'$, $f2''$. The clock numbers 6 and 7 are ultimately assigned to $f1'$ and $f2'$, respectively. In the next step, the algorithm assigns $f1' = f2' = 8$ and additional wires are added. The final graph is shown in Fig. 13(c), and the QCA layout is given in Fig. 14. In the layout MV $f1$ is in clocking zone 2 while MV $f2$ is in clocking zone 3. This is due to stretching to ensure that the state variables $Q1$ and $Q2$ are in the same clocking zone. In this example $Q1$ and $Q2$ are in clocking 0.
- The last example is the QCA circuit for S27 from the ISCAS89 sequential benchmark set; this is shown in Fig. 15. S27 has four inputs and one output. A QCA implementation requires three D FFs, 10 active gates (two inverters, one AND, two NANDs, two ORs, four NORs). The original graph is given in Fig. 16(a). The center gates are f_1, f_2, f_3. The graph representation of this circuit after applying NumerateDAG() is shown in Fig. 16(b). In the proposed algorithm, $f1' = f2' = f3' = 28$. The details of the stretching step are not shown due to lack of space. The final QCA layout of this circuit is shown in Fig. 17.

8. Conclusion

This paper has presented a detailed analysis for designing sequential circuits in QCA. This analysis encompasses FF devices as well as circuits. Initially, a novel RS-type FF amenable to a QCA implementation has been proposed. This FF extends the threshold-based configuration of [12] to QCA by taking into account

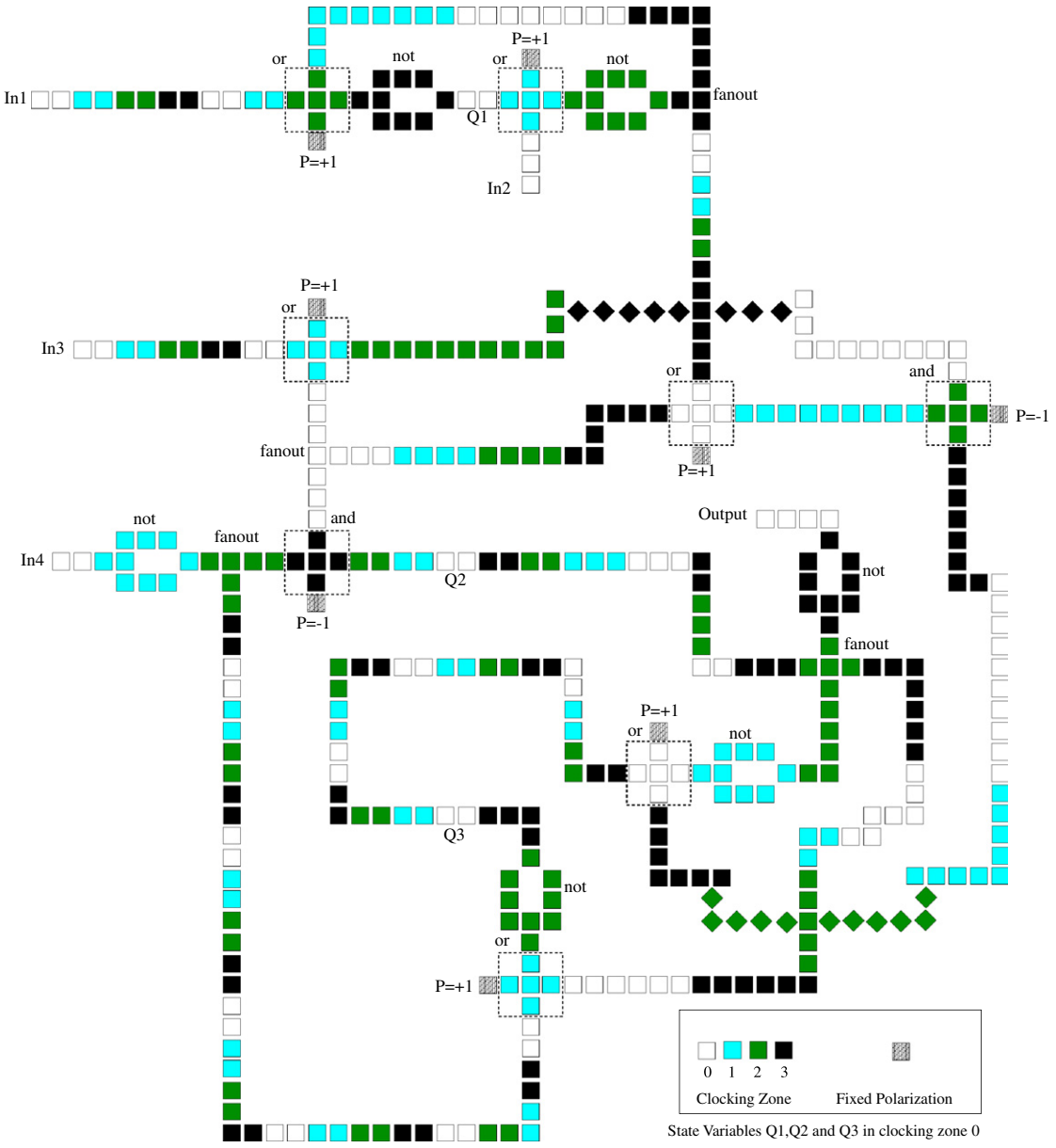


Fig. 17. QCA layout S27 benchmark circuit, ISCAS89.

the timing issues associated with the adiabatic switching of this technology. The characterization of a D-type FF as a device consisting of an embedded wire, has been also provided. The unique timing constraints in QCA sequential design have been explored in detail. Timing requirements for both RS FF-based design as well as D FF-based design are presented. A circuit-level characterization of sequentiality in QCA has been pursued. An algorithm has been proposed to assign appropriate clocking zones to QCA circuits so that timing constraints are met. This algorithm relies on a topological sorting and enumeration step to consistently traversing only once the edges of the graph representation of the QCA sequential circuit. Timing considerations are accounted by matching the delays along all paths by inserting QCA wires. Unique QCA devices (such as the coplanar crossing) have also been considered. Three examples of QCA sequential circuits have been described.

Finally, it should be noted that the proposed algorithm does not guarantee to be optimal for inserting the wire gates. Also the logic level design rather than the physical layout has been considered in this algorithm. Therefore if the resulting final graph cannot be drawn into the QCA layout, further stretching may be required to match the delays.

Algorithm 1. Clock assignment algorithm.

```
Function AssignClk(G(V, E))
Data : G as DAG graph of the Circuit;
       u' and u'' input and output center vertices;
       ss super source vertex

begin
    topological sort G(V, E)
    clk(ss) ← −1
    NumerateDAG(G)
    - - assign same clock to all input center vertices
    for all input center vertices u' do
        k ← maximum of clk(u')
    end
    if k modulo 4 ≠ 0 then
        k ← k + 4 − (k modulo 4)
    end
    for all input center vertices u' do
        clk(u') ← k
    end
        - - stretch any edge (v, u') where u' is an input
    center vertex
    for each edge (v, u') do
        while clk(v) + 1 < clk(u') do
            i ← clk(u') − clk(v) − 1
            V ← V ∪ x_i                    - - create new vertex
            x_i
            clk(x_i) ← clk(v) + 1
            E ← E ∪ {(v, x_i), (x_i, u')} − {(v, u')}
            v ← x_i
        end
    end
        - - stretch other vertices
    for each u ∈ V do
                                - - stretching the common path
        between u and u's children
        while clk(u) + 1 < minchild(u) do
            i ← minchild(u) − clk(u) − 1
            V ← V ∪ x_i                    - - create new
            vertex x_i
            clk(x_i) ← clk(u) + 1
            E ← E ∪ {(u, x_i), (x_i, children(u))} −
            {(u, children(u))}
            u ← x_i
        end
                                        - - stretching the
        non-common path
        for every child v of u do
            while clk(u) + 1 < clk(v) do
                i ← clk(v) − clk(u) − 1
                V ← V ∪ x_i
                clk(x_i) ← clk(u) + 1
                E ← E ∪ {(u, x_i), (x_i, v)} − {(u, v)}
                u ← x_i
            end
        end
    end
end
```

Algorithm 2. Algorithm to numerate vertices in a DAG.

```
Function NumerateDAG(G(V, E))
Data : G as DAG graph of the Circuit
       ss super source vertex

begin
    for each u ∈ V do
        clk(u) ← −∞
    end
    for each u taken in topologically sorted order do
        for each of u's child v do
            if clk(v) < clk(u) + 1 then
                clk(v) ← clk(u) + 1
            end
        end
    end
end
```

Algorithm 3. Crossover algorithm.

```
Function Crossover(G(V, E), co, v)
Data : G as DAG graph of the Circuit;
       co crossover vertex;          v current
       parent vertex of crossover vertex; passed ∈
       {False, True};                a ∈ {∗, +} is
       attribute of current edge (v, co)
begin
                                        - - crossover first
    pass
    if co.passed = False then
        co.passed = True
        if a = ∗ then
            clk(co*) ← clk(v) + 1
        end
        else
            clk(co+) ← clk(v) + 1
        end
    end
                                    - - crossover second
    pass
    else
        if a = + then
            x← (clk(co*) − clk(v) − 1)modulo4 where
            x= 0, 1, 2, 3
            clk(co+) ← clk(v) + 1 + x
        end
        else
            x← (clk(co+) − clk(v) − 1)modulo4 where
            x= 0, 1, 2, 3
            clk(co*) ← clk(v) + 1 + x
        end
    end
end
```

References

[1] C.S. Lent , P.D. Tougaw, W. Porod, Quantum cellular automata: the physics of computing with arrays of quantum dot molecules, in: PhysComp '94: Proceedings of the Workshop on Physics and Computing, IEEE Computer Society Press, 1994, pp. 5–13.

[2] M.T. Niemier, P.M. Kogge, Problems in designing with QCAs: layout = timing, Int. J. Circuit Theory Appl. 29 (1) (2001) 49–62.

[3] S.E. Frost, A.F. Rodrigues, A.W. Janiszewski, R.T. Rausch, P.M. Kogge, Memory in motion: a study of storage structures in QCA, in: First Workshop on Non-Silicon Computation, 2002.

[4] K. Walus, R.A. Budiman, G.A. Jullien, Effects of morphological variations of self-assembled nanostructures on quantum-dot cellular automata (QCA) circuits, in: Frontiers of Integration, An International Workshop on Integrating Nanotechnologies, 2002.

[5] K. Walus, A. Vetteth, G.A. Jullien, V.S. Dimitrov, RAM design using quantum-dot cellular automata, NanoTechnology Conf. 2 (2003) 160–163.

[6] M. Ottavi, V. Vankamamidi, F. Lombardi, Clocking and cell placement for QCA, in: Proceedings of the IEEE Nanotechnology Conference, 2006, pp 343–346.

[7] Personal communication with Professor Marya Lieberman, Department of Chemistry and Biochemistry, University of Notre Dame, IN, USA.

[8] M. Momenzadeh, J. Huang, F. Lombardi, Defect characterization and tolerance of QCA sequential devices and circuits, in: Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), 2005, pp. 199–207.

[9] C.S. Lent, P. Douglas Tougaw, A device architecture for computing with quantum dots, Proc. IEEE 85 (1997) 541–557.

[10] J. Jiao, G.L. Long, F. Grandjean, A.M. Beatty, T.P. Fehiner, Building blocking for the molecular expression of QCA, isolation and characterization of a covalently bounded square array of two ferrocenium and two ferrocene complexes, J. Am. Chem. Soc. (JACS Communications) 125 (25) (2003) 7522–7523.

[11] M.T. Niemier, A.F. Rodrigues, P.M. Kogge, A potentially implementable FPGA for quantum dot cellular automata, First Workshop on Non-Silicon Computation (NSC-1), 2002.
[12] S. Muroga, Threshold Logic and Its Applications, Wiley-Interscience, New York, 1971.
[13] N.N. Biswas, Logic Design Theory, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[14] M.T. Niemier, P.M. Kogge, Exploring and exploiting wire-level pipeling in emerging technologies, in: Proc. ISCAS, 2001, pp. 166–177.
[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, McGraw-Hill, New York, 2001, pp. 643–693.
[17] QCADesigner Home Page. ⟨www.qcadesigner.ca⟩.