Exploring and Exploiting Wire-Level Pipelining in Emerging Technologies

Michael Thaddeus Niemier
University of Notre Dame
Dept. of Computer Science and Engineering
Notre Dame, IN 46556, USA
mniemier@nd.edu

Peter M. Kogge
University of Notre Dame
Dept. of Computer Science and Engineering
Notre Dame, IN 46556, USA
kogge@wizard.cse.nd.edu

Abstract

Pipelining is a technique that has long since been considered fundamental by computer architects. However, the world of nanoelectronics is pushing the idea of pipelining to new and lower levels – particularly the device level. How this affects circuits and the relationship between their timing, architecture, and design will be studied in the context of an inherently self-latching nanotechnology termed Quantum Cellular Automata (QCA). Results indicate that this nanotechnology offers the potential for "free" multi-threading and "processing-in-wire". All of this could be accomplished in a technology that could be almost three orders of magnitude denser than an equivalent design fabricated in a process at the end of the CMOS curve.

1. Introduction

Pipelining as a technique has been with us every day for almost 40 years. It has been designed into our basic circuits, and governed the way we partition larger logic functions. Technology and market forces have pushed clock speeds to realms where speed of light dictates smaller and smaller amounts of logic per stage. More and more we even rely on transitory storage effects (as in wave pipelining) to push the cycle time to lower and lower limits. This in turn has forced us to introduce new instruction and architectural level artifacts such as vectors, systolic arrays, and more recently multi-threading. As we look toward the near future, we see this trend continuing, especially as we approach the supposed end of CMOS at 0.05 microns (50 nanometers).

When we look at alternative technologies, we often see this same phenomena of shrinking stage size appearing at the basic technology level. Older technologies such as Charge Coupled Devices and newer self-latching technologies such as Josephson junctions and Rapid Flux Single Quantum devices bring pipelining down to the device level. To date, the unique application niches and difficult implementation environments of such technologies has discouraged researchers from seriously considering their effects on design. However, it may be time to reconsider the interaction between timing, architecture, and design. The driving force is the rapidly expanding world of nanoelectronics, where devices as small as a molecule may be feasible.

This paper is the outgrowth of design work with one such technology just recently demonstrated, but for which there is strong evidence that room temperature systems made from 2 nm devices may only be a few years away. This technology, termed Quantum Cellular Automata (QCA), is at its base, self-latching, where information is stored at each device by the positions of single electrons, and logic functions are performed not by electron flow but by Coulombic interactions. The result is a technology where even the interconnect is made out of the same self-latching devices as the logic functions, bringing "pipelining" down well below the level of even a simple logic gate.

Thus, while focused on QCAs, our goal in this paper is to reopen the topic of pipelining at an extremely low level, and begin to explore how what today are high level techniques – such as multi-threading – may in fact in the future become part and parcel of micro-organizational levels of design. To support this, Sec. 2 first reviews the QCA technology to introduce where this ultra low level pipelining originates. Sec. 3 discusses the concept of a "clock" in such a context. Sec. 4 provides a motivational example of what the technology may mean in terms of comparison at the dataflow level to the CMOS roadmap. Sec. 5 then explores what is perhaps a more fundamental issue - what does such a technology mean to the design of the state machines that control such data flows? Sec. 6 introduces a canonical approach to their design and gives examples. Sec. 7 will describe a primitive architecture. Sec. 8 concludes with a discussion of what are the most important near term research issues.

2. The QCA Device, Circuits, and Experiments

2.1. The Device

A high-level diagram of a four-dot QCA cell appears in Fig. 1. It consists of four quantum dots that are positioned

1063-6897/01 $10.00 © 2001 IEEE

to form a square. (Note: Future QCA cells have the potential to shrink dots to regions within specially designed molecules). Exactly two mobile electrons are loaded in the cell and can move to different quantum dots in the QCA cell by means of electron tunneling. Coulombic repulsion will cause the electrons to occupy only the corners of the QCA cell resulting in two specific polarizations (see Fig. 1). These polarizations represent the places where the electrons are as far apart from one another as possible (which hence minimizes the Coulombic repulsion between them) without escaping the confines of the cell. Electron tunneling is assumed to be completely controllable by potential barriers that can be raised and lowered between adjacent QCA cells by means of capacitive plates [14].

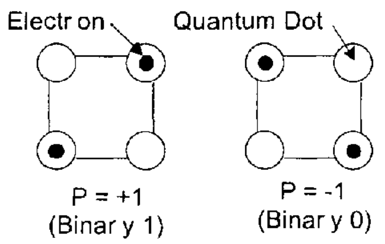

Figure 1. QCA cell polarizations.

2.2. Circuits

The fundamental QCA logical circuit is the three-input majority gate (Fig. 2) [14]. Computation is performed by driving the *device cell* (cell 4 in the figure) to its lowest energy state. This occurs when it assumes the polarization of the majority of the three input cells. The device cell will always assume the majority polarization because in this polarization, electron repulsion between the electrons in the three input cells and the device cell will be at a minimum.

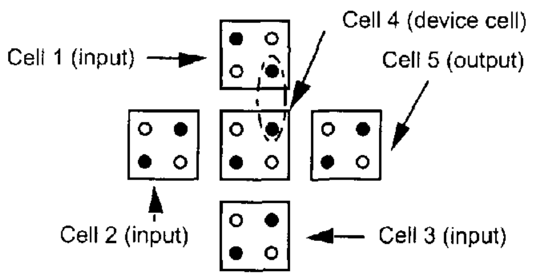

Figure 2. The fundamental QCA logical device.

Fig. 3 illustrates a representation of what is called a "90-degree wire". Assume that the middle cell in Fig. 3 was originally in a polarization opposite of that of the first cell. Thus, if the first cell represents an input cell, the electrons would move as illustrated by the arrows in the figure to take on a polarization that minimizes the Coulombic repulsion between the neighboring cells [14].

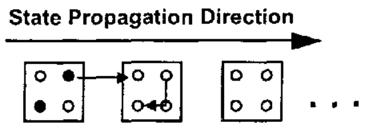

Figure 3. Interaction between 2 cells.

A QCA wire can also be comprised of cells oriented at 45-degrees [14]. With the 45-degree orientation, as the binary value propagates down the length of the wire, it alternates between polarization P = +1 and polarization P = -1. A complemented or uncomplemented value can be ripped off this wire by placing a "ripper cell" at the proper location and considering the direction of signal propagation (this is determined in part by electron position and will not be elaborated upon here) [14], [6]. The significant advantage of the 45-degree wire is that both a transmitted value and its complement can be obtained from a wire without the use of an explicit inverter! Also, QCA cells do not have to be in a perfectly straight line to transmit binary signals correctly. Cells with a 90-degree orientation can be off-center, and a binary value will still be transmitted successfully. Finally, QCA wires possess the unique property that they are able to cross in the plane without the destruction of the value being transmitted on either wire. However, this property holds only if the QCA wires are of different orientations (i.e. a 45-degree wire crossing a 90-degree wire) [14].

2.3. Experiments

The previous two subsections have described how a theoretical QCA device and theoretical QCA circuits would function. It is this theory on which many of the designs for QCA dataflows and state machines are based – and from which we will be able to comment on the effects of a self-latching technology on "conventional" dataflows and state machines. However, it is imperative to emphasize that QCA has moved beyond the realm of theory as actual devices and circuits have been constructed.

For instance, QCA cells made of metal islands with tunnel junctions have been fabricated. Cell operation has been demonstrated at very low temperatures (70 mK) [7], [11]. 3-input majority gate logic [1], [11], a QCA wire [8], clocked QCA cells [9], single-bit memory [10], and power gain have also all been demonstrated. Finally, work is currently underway to raise the cell operating temperature to 20K - 70K. In addition to work with "metal dots", researchers are also working to build QCA cells using chemical molecules

[3]. The benefit of chemical molecules is two fold. First, their operating temperature would be much closer to room temperature. Second, a QCA circuit built from chemical molecules should lead to an almost two-order of magnitude density increase over an equivalent circuit built with metal dots. Currently, several promising candidate molecules that could function at room temperature exist.

3. The Clock

Unlike the standard CMOS clock, the QCA clock has more than just a high and a low phase, but rather, four phases. Also, individual QCA cells are not clocked/timed separately. The support required to clock each cell individually could easily overwhelm the simplification won by the inherent local interconnectivity of the QCA architecture [4]. However, an array of QCA cells can be divided into zones that offer the advantage of multi-phase clocking and group pipelining. For each zone, a single potential modulates the inter-dot barriers in all of the cells [4].

This clocking scheme allows one zone of QCA cells to perform a certain calculation, have its state frozen by the raising of its interdot barriers, and have the output act as the input to a successor zone (i.e. clocking zone 1 can act as input to clocking zone 2). It is this mechanism that provides the inherent self-latching associated with QCA. During the calculation phase, the successor zone is kept in an unpolarized state so it does not influence the calculation. Each of the four clocking zones corresponds to one of four different clocking phases. Physically neighboring zones concurrently receive temporally neighboring clocking phases [4].

Finally, it is important to stress what exactly is meant by the QCA "clock". As mentioned above, the QCA clock has more than a high and a low phase. Also, it is not a "signal" with four different phases. Rather, the clock changes phase when the potential barriers that affect a group of QCA cells (referred to as a *clocking zone*) are raised or lowered or remain raised or lowered (thus accounting for the four clock phases). Furthermore, all of the cells within a clocking zone obviously are in the same phase. One clock cycle occurs when a given clocking zone cycles through the four different clock phases. What exactly the "clock" does is to trap one zone of cells in a specific polarization which in turn allows other cells in neighboring zones to make appropriate changes.

During the first clock phase, the *switch phase*, QCA cells begin unpolarized and their interdot potential barriers are low. The barriers are then raised during this phase and the QCA cells become polarized according to the state of their driver (i.e. their input cell). It is in this clock phase that the actual computation occurs. By the end of this clock phase, barriers are high enough to suppress any electron tunneling and cell states are fixed. During the second clock phase, the *hold phase*, barriers are held high so the outputs of the subarray can be used as inputs to the next stage. In the third clock phase, the *release phase*, barriers are lowered and cells are allowed to relax to an unpolarized state. Finally, during the fourth clock phase, the *relax phase*, cell barriers remain lowered and cells remain in an unpolarized state [4]. As can be seen from the example in Fig. 4, clocking zones clearly "latch" data which allows it to be transferred from one clocking zone to another.

In conclusion, it should be mentioned that clocking signals can be distributed by metal lines underlying the array, with a feature size much larger than the cells. This kind of "hot clock" acts as an energy source to replace the unavoidable energy lost to the environment through dissipative events. Clocked QCA cells exhibit power gain, essential to restoring logic levels in real circuits. Also, a brief word should be said about clock rate. In the design in Fig. 6, the most common wire length consists of 5 cells while the longest wire consists of 51 cells. Assuming a potential device switching speed of 10^{-12} s (resulting in a power dissipation of approximately 10^{-10} W per device) [13], the five cell wire could be clocked at a rate of 2 THz while the 51 cell wire could be clocked at a rate of about 20 GHz. (Of course the clock rate for the whole design would obviously have to be the 20 GHz rate).

4. Motivational Examples and a QCA Dataflow

This section will show how inherent latching or "pipelining" at the device level can affect an actual design. It will begin by discussing methods designed to counter and exploit the need for implicit latching, and will end with full designs that have been simulated. These designs will then be analyzed to show both the strengths and weaknesses of pipelining at the device level.

4.1. Floorplanning

As was seen in Sec. 3 it is the nature of the clock that leads to the inherent self-latching in QCA. In particular, Fig. 4 illustrates that even a simple 5 cell wire can be pipelined. Given this constraint, and before attempting any large scale designs, we felt the need to develop methods to successfully factor self-latching out of the "equation" of a design and furthermore find a means to exploit it [5]. Thus, a study of floorplanning was performed in the context of a microprocessor called Simple 12 (Fig. 6). Our goal was to design it entirely in QCA. Again, although simple, it exhibits almost all of the attributes of a more complex design. While many interesting results arose from this study [5], two were most fundamental (Fig. 5). The first was a floorplan that allows a feedback path to be generated given the constraints of the QCA clock.

The shaded areas in Fig. 5a represent clocking zones each in a specific clock phase. The white arrows show the direction of information propagation through the logic in

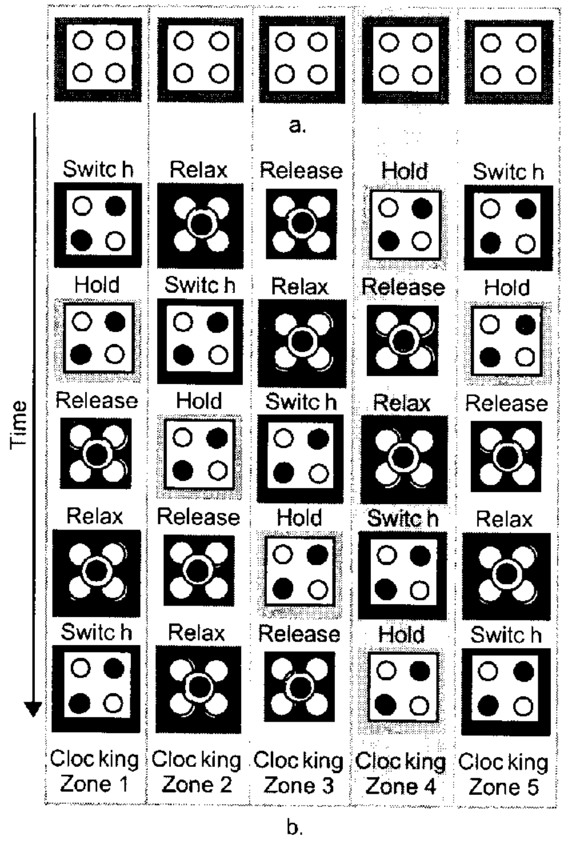

Figure 4. Part (a.) shows a physical 5-cell wire while part (b.) shows a value propagating down the wire.

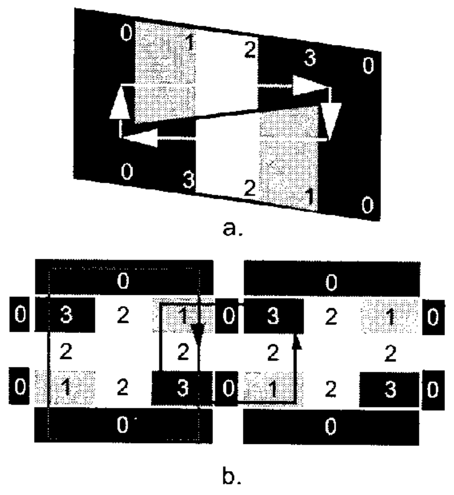

Figure 5. Two floorplans with clocking zones.

each zone. The numbers in each zone reflect the relative phase of the clock in that zone so that at time i a zone labeled j is in phase $(i + j)$ mod 4. Assume for example that the bottom "trapezoid" is computational logic. Data could be fed back to the input of the top trapezoid assumed to be in clocking zone 0 at the far left after "switching" to the computed value in clocking zone 0 at the far right. It can easily be seen that the clocking phases are traversed in the proper order. Also, a signal can start at a given point and an in-phase path exists to return to that point – the definition of feedback. The second floorplan result came from questioning whether or not the clocking zone arrangement of Fig. 5a could be extended to allow efficient and easy 2D wire routing. Thus, could the clocking zones be arranged or tiled so that there are multiple "wire" loops and "wire" crossings and still allow feedback? Such a pattern is illustrated in Fig. 5b. Thus, these general floorplans provide a means for the easy routing of wire and circuit components given the overhead of self-latching.

4.2. Complete Designs

The floorplans discussed above provided the foundation for the design of the Simple 12 dataflow. Simple 12 operates on 12-bit pieces of data and such a design was constructed, measured, simulated, and even fabricated in CMOS [6]. For clarity, one bit of that design appears in Fig. 6 with each subcomponent matched up with the Simple 12 dataflow. As discussed below, Fig. 6 allows many tasks that are "architecturally desirable" to be done "for free". This is by-in-large due to inherent self-latching. However, there is also a minor problem with this design that must be and is taken into account in future work.

By far, the two most interesting architectural "innovations" of an inherently self-latching technology at the gate level are first, something that we call "processing-in-wire" and second, the ability to multi-thread a circuit for "free".

To understand how processing-in-wire works, Fig. 7 shows part of the Simple 12 logic unit plus intermediate signal generation logic that zeros an ALU input so it can perform a specific function. This "Zero A" logic is placed "directly after" the output of the ALU – or more specifically, above it in the "feedback trapezoid". Thus, the A input to the ALU is zeroed on the way back to the input of the ALU. Essentially, useful computation is being performed "in wire" for free! Why is it free? Well, even if there was no logic in the feedback path from the output of the ALU back to the input of the Simple 12 dataflow, a wire would be spread out over the clocking zones back to the input. Thus, whether or not computation is done in these clocking zones, the same n clocking zones still must be traversed.

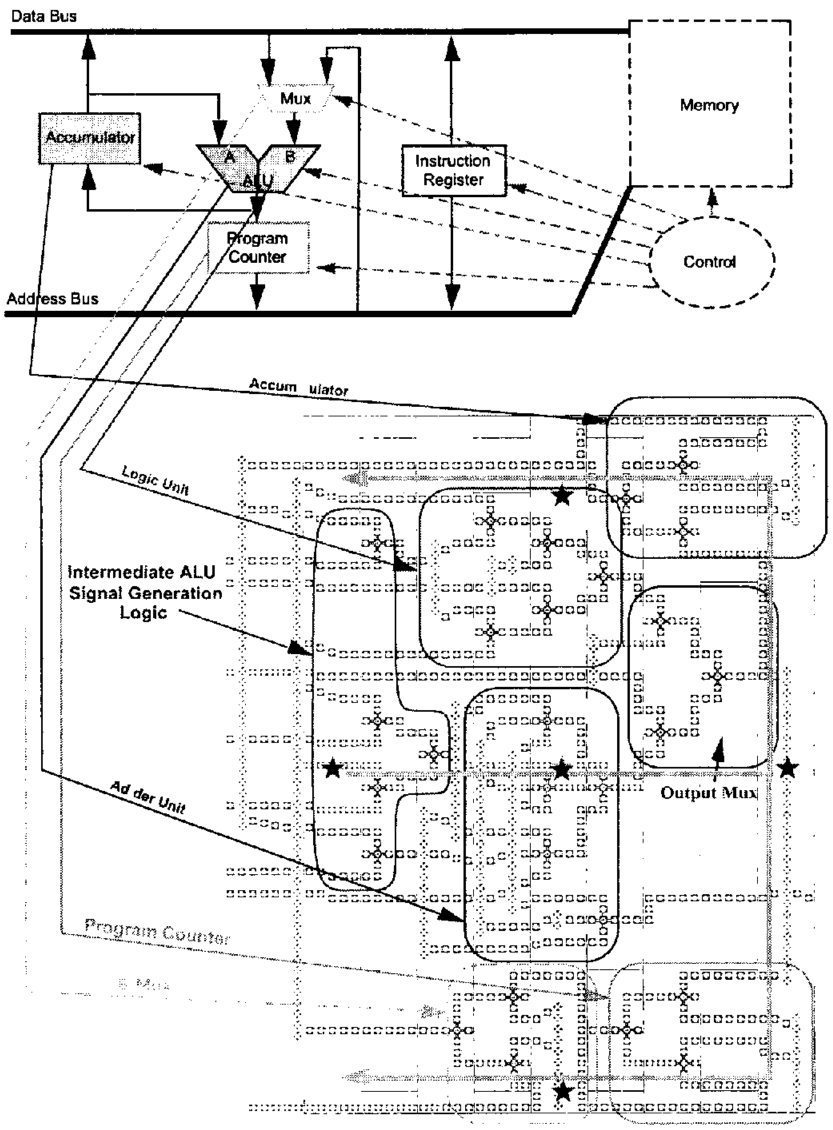

Figure 6. A complete 1-bit dataflow of the QCA Simple 12.

Now, what about the notion of "free" multithreading? Before answering this question it is best to point out a "problem" in Fig. 6 – namely the longest path from input-to-input is spread out over 16 clocking zones. Thus, given that a bit of data can move through four clocking zones during one clock cycle, it will take four clock cycles for one very simple operation to finish. While, this would appear to be bad, as it turns out, the inherent self-latching of QCA should allow multiple computations/instructions to execute simultaneously. In Fig. 6, potential wave-fronts or "threads" are represented by ⋆'s in respective clocking zones. Essentially, each shaded region can represent a different possible ongoing instruction. Thus, by examining Fig. 6 one can see that it is possible to have 4 computations/instructions executing simultaneously. Plus, the inherent self-latching of QCA allows the multithreading to be done "free" without explicit registers, latches, or timing.

There is one additional "benefit" of QCA – its size. Area measurements were taken for the 12-bit QCA design and the equivalent area was determined from an actual MOSIS CMOS implementation of Simple 12. (Note: it should be stressed that both of these designs were hand-crafted. For

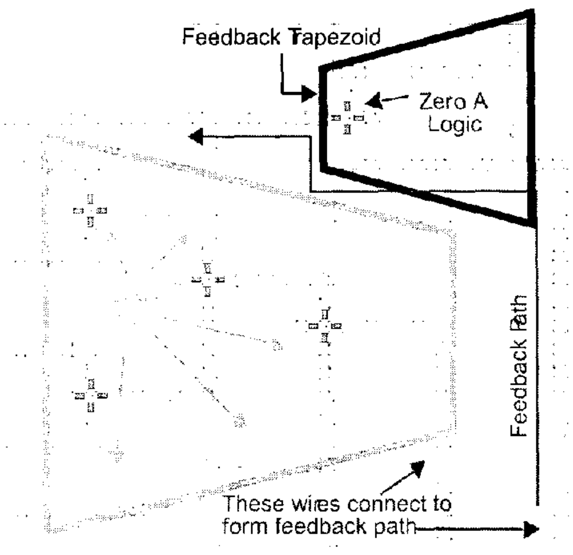

Figure 7. An example of "processing in wire".

the CMOS design, each transistor in the dataflow was laid out by hand. Similarly, in the QCA version, each QCA cell was laid out by hand. This was done in an effort to create the densest possible design.) It was determined that QCA offers at least almost an order of magnitude area density increase over the equivalent CMOS design when scaled to 0.05 micron. With molecular dots, potential density gains approach three orders of magnitude [6]!

It is now imperative that we point out a potential "problem" with Fig. 6. Examine the two vertical wires to the left of the "Intermediate ALU Signal Generation Logic". These two wires are necessarily long (as, for example, output from the accumulator must be sent to the adder unit which is at the "bottom" of this design). When generating designs in QCA, a significant effort should be made to keep a length of a wire within a given clocking zone to a minimum. There are two very important reasons to do this. First, the probability that a QCA cell will switch successfully decreases in proportion to the distance a particular cell is from a frozen input at the beginning of the "wire". Thus, simply, for shorter wires there is a higher probability that all cells making up the wire will switch successfully [4]. This should also explain why in Fig. 6 one could not simply have one long wire in one clocking zone that constituted the feedback path and instead that wire must be broken up over multiple clocking zones (which processing-in-wire then exploits). Additionally, wire length will determine the clock rate – the rate at which clocking zones can change clock phases. This is so because, before a given zone can change phase, every cell within the zone must make the appropriate polarization changes. Obviously, the longest path dictates the time for a signal to propagate down the length of it. As one will see in the next section, minimizing wire length and exploiting inherent self-latching will be the two driving forces behind additional designs.

5. One-Hot State Machines

Given that a dataflow for a self-latching architecture has been constructed, we concluded that the next logical step in additional circuit development would be to generate control logic/state machines for the dataflow in Sec. 4 and to study the effects of a self-latching architecture on state machines in general. Thus, the work discussed in this section will begin with the development of the one-hot state machine that controls the state transitions for Simple 12. Before discussing actual designs, it is important to point out two things. First, a Simple 12 instruction can be in one of three states – *stopped*, *iFetch* (instruction fetch), and *execute*. Thus, a Simple 12 one-hot will obviously need three flip-flops. The second item deals with the nature of one-hot state machines: for each state, S_i, the corresponding state variable y_i is set to 1, or is "hot", while all other flip-flops are set to 0. These transitions should take place *simultaneously during the same clock cycle*. It is this notion of simultaneous switching that will define much of our work with state machines in QCA.

5.1. The Simple 12 One-Hot

A schematic of the three state Simple 12 one-hot appears in Fig. 8, with one flip-flop for each state. Now, when this design is implemented in QCA, there is **no need** for an explicit flip-flop circuit. Why? Clocking zones will make the QCA cells act as inherent latches, controlled by clocking zones changing clock phases. Thus, state information will be represented in three different clocking zones with each clocking zone representing a bit of "state". A first cut of this machine in QCA appears on the right of Fig. 8.

As one can see, three clocking zones hold the computed information that would be stored in three flip-flops. Note that there are two *execute* feedback paths. This wire was duplicated because both the *stopped* state and the *iFetch* state depend on *execute* state information. Also wire routing was actually *simplified* by having the *execute* state information branch in two directions. It is actually these two "wires" that are the most important feature of this particular design. Like the feedback path for the QCA dataflow, they are spread over 4 clocking zones in an effort to break-up and eliminate long wires. However, unlike the QCA Simple 12 dataflow, this technique will not work for this (or most) one-hot state machines. Why is this? The problem centers around the next state of this one-hot state machine depending on information from the previous state. Assume for example that a computation has occurred in Fig. 8 during one

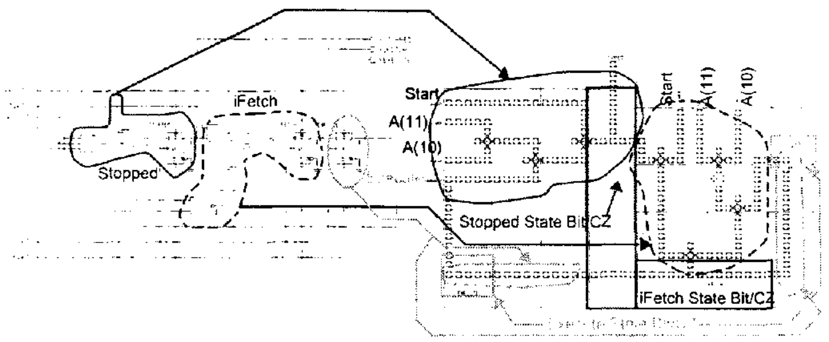

Figure 8. A first-cut of the QCA Simple 12 one-hot.

clock cycle. Thus, data has moved from the inputs to the clocking zones labeled "stopped", "iFetch", and "execute". In theory, a new set of data bits could now enter the inputs of the design so that another state transition calculation can begin. And as mentioned, this state transition calculation requires information about the previous state. However, a problem arises given the fact that the feedback paths containing information about the *execute* state will take another clock cycle to arrive back at the inputs (because the wire is divided up over four clocking zones!). This will only lead to erroneous state data!

Now, for this simple design, the problem is not that difficult to correct. A revised design appears in Fig. 9. As one can see, several clocking zones have been eliminated in the feedback paths that provide information about the previous *execute* state. Now, new data can enter this state machine after one clock cycle occurs and the correct data from feedback paths will arrive at the same time. While the wire length has increased in the clocking zone that contains information about the *iFetch* state, the longest wires in this zone are 15 and 16 cells, which is not unreasonable.

However, the two important things to think about when considering this corrected design do not involve wire length at all. But rather that first, a "true" one-hot design (i.e. one where all state bits switch simultaneously) is possible in a technology with inherent latching. And second, that this true one-hot design has been constructed for a *simple* state machine with only 3 states. Admittedly however, the combinational logic the precedes each state is minimal and can thus fit in 4 clocking zones – and thus be processed in one "clock". State machines with a greater number of states and combinational logic that requires more than one clock to process must be studied further in an effort to see if "true" one-hots with inherent self-latching are still feasible.

5.2. A More Complicated One-Hot

In an effort to investigate one of the two concerns mentioned above, we moved to implement another one-hot state machine in QCA. This was a controller for a last-in, first-out stack [2]. The controller has 5 states and hence 5 flip-flops, with each state's value depending on its neighbor(s) and itself (i.e. state $y0$ would therefore depend on $y0$ and $y1$). The combinational logic that precedes each flip-flop consists of a network of three 2-input AND gates (whose inputs are control signals and state feedback) which in turn feed into a 3-input OR gate. However, two of the states require the control signals' complements while another state ($y1$) requires an additional level of combinational logic (a 2-input NOR gate for 2 input signals PUSH and POP). Finally, a small amount of combinational logic uses output from several of the state flip flops, plus control signals, to generate some final output signals.

Efforts to generate a "circular" design analogous to the 3-state one-hot for Simple 12 (i.e. a design where the clocking zone that contained state information directly abutted a clocking zone that required it as input to the combinational logic that would compute the next state) for the LIFO controller quickly proved to be infeasible. It became quickly apparent that there were simply too many state dependencies to generate a design with reasonably shaped clocking zones and wire lengths that would still function as a true one-hot. In other words, while such a "circular" design theoretically could have been generated, it would never be implementable.

6. Canonical State Machines Implementations

The problem that arose in Sec. 5.2 was that QCA one-hot state machines with a significant number of states and excess combinational logic would not necessarily exhibit an important property of a logically correct one-hot state machine. Namely, the bits representing individual states would not switch simultaneously. Because of this, we began to investigate other ways in which a one-hot state machine could be implemented in QCA in which this most important property would still hold.

Now, one way to *ensure* that the QCA cells represent-

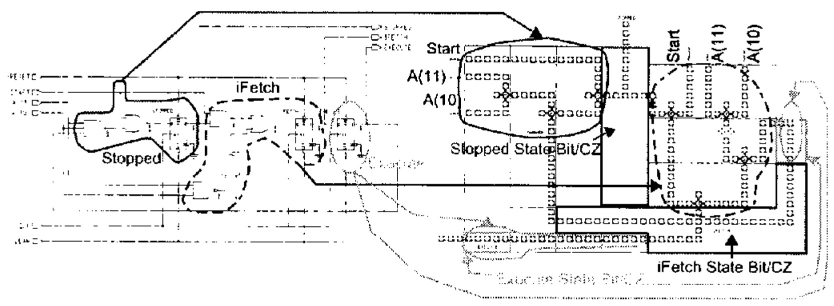

Figure 9. A revised QCA Simple 12 one-hot.

ing state information for any one-hot state machine would switch simultaneously would simply be to place the cells representing each state in the same clocking zone. One manner in which such a design might be created would have the QCA wires representing each state stacked linearly in a given clocking zone. Thus, a different bit of data for each state would exist; and that fact that they were all in one clocking zone would cause all bits to change at the same time.

6.1. An Updated Design

This idea was tested with the LIFO controller. A QCA equivalent appears in Fig. 10. There are three important things to point out about this design. First, there is a comb-like structure that contains state information and feedback paths to the combinational logic that precedes each state. Second, the NOR gate (mentioned above) that precedes the second state's block of combinational logic is conspicuously missing from the QCA implementation. Third, in Sec. 4.2 we mentioned that long wires were undesirable. It can clearly be seen that in the clocking zones that follow the "latched" state information there are several very long wires. These three issues will be discussed below.

As mentioned, the comb-like structure in this design contains state information and feedback paths to the combinational logic that determines each state. There are two important things to point out about it. First, the "comb" is one large clocking zone. Thus, when it enters the *switch* clocking phase, all state information will be updated simultaneously. Second, each feedback wire in this comb-shaped clocking zone is no longer than 25 cells long and a significant effort has been made to keep wire length at a minimum. And more importantly, feedback information arrives at the combinational logic preceding each state at the proper time – i.e. so that adjacent clocking zones are in the appropriate phases to propagate data from one to the other. Thus, this comb-like structure solves the two major problems that we encountered earlier. First, all bits are now guaranteed to switch simultaneously. Second, data arrives at the proper place at the proper time taking into account the inherent latching that the QCA clock enforces.

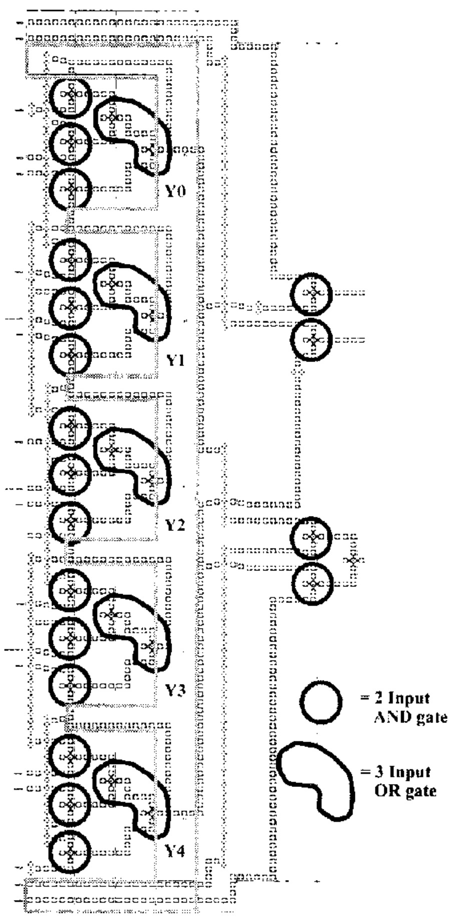

Figure 10. A QCA controller for a last-in, first-out stack.

Now, as mentioned, a NOR gate precedes the combinational logic before state *y1*. It should also be noted that there are inverters that precede states *y0* and *y4*, but as was discussed in Sec. 2.2, it is possible in QCA to generate the complement of a signal without the use of an inverter. However, the NORing of the PUSH and POP signal must still somehow be accomplished. An additional problem arises given the fact that if the NOR gate were to directly precede the combinational logic before state *y1*, the combinational logic would be spread over more than 4 clocking zones and thus could not be accomplished in one clock (see Fig. 10). Our solution to this problem lies in processing-in-wire. In particular, if this design were to actually be implemented, the PUSH and POP inputs would undoubtedly be coming from some other part of this design. There is also a high probability that they would have to traverse *n* clocking zones. Thus, as PUSH and POP should have definite values, there is no reason why they could not be "NORed" together "in transit" to the state machine. This way, output of the NOR gate could simply act as another input to the LIFO controller. This allows combinational logic to finish in one clock cycle and minimizes area.

If one examines Fig. 10 it is easy to see the small amount of combinational logic that follows the 5 states. This determines some final output signals needed for this particular design, depending on state $y0$ and $y4$ of the design. Furthermore, in Fig. 10 there are several long wires are required to bring the information from states $y0$ and $y4$ up to the combinational logic for processing. Now, given the nature of this design (i.e. that all states are positioned vertically in the same clocking zone), there is little that can be done to avoid this (except perhaps breaking up the wire into separate clocking zones as was done in the feedback path of Fig. 6 – however, this will only complicate timing). While there is little that can and will be done about this problem with this design – it was simply laid out in an effort to test another means for implementing a more complicated true one-hot – it illustrates a problem that must and will be addressed in yet another design. Namely, the ordering of states within a latch can affect wire length if the state outputs are needed for other combinational logic.

6.2. The 6-State Simple 12

The final design to be discussed in this paper will be a six state version of the Simple 12 microprocessor's state machine. The three states that have been added are *EAGen* (Effective Address Generation), *no jump*, and *operand*. As this design consists of up to 5 levels of combinational logic preceding flip-flops that hold state information, it is readily apparent that it will provide all of the "challenges" discussed in previous subsections – namely that combinational logic will have to be spread out over more than 4 clocking zones/one clock, there is a large number of states, and there is complex wiring.

The first attempt at translating this design attempted to make use of the idea discussed above. All of the cells representing state were placed in the same clocking zone in an order that represented their logical transition progression. Thus, the *stopped* state was first, the *iFetch* state was second, followed by *EAGen*, *No Jump*, *Operand*, and *Execute*. This led to a very complicated schematic with jumbled logic, routing paths, and long wires. Upon examination, we realized that the reason this happened centered around state dependencies. For instance, the *iFetch* state depends on the previous *stopped*, *no jump*, *operand*, *execute*, and *EAGen* states. As the *iFetch* state was second from the top in our preliminary design, it is easy to see that long wires would be required to provide the combinational logic preceding the *iFetch* state with previous state information. Now, the idea of having all state information represented in a single clocking zone is appealing as it guarantees the properties of a true one-hot. Thus, is there a way to minimize long wires and complicated routing paths so that it still might be used?

Long wires and complicated routing paths arose in our preliminary design largely because of state dependencies. And given the arrangements of state information in a single latch, long wires were required to move this data to other parts of the design. This fact led us to ask the question of whether or not the physical locations of state information could be rearranged to solve these two problems? We began this investigation by representing the state information for the 6-state Simple 12 one-hot as a matrix. Essentially, the rows and columns of the matrix each represent one state of the one-hot. There is a one in the column of the matrix for a given state if it depends on the state for the corresponding row. Otherwise, a zero is placed in the matrix. An initial representation of such a matrix for the 6-state Simple 12 appears in Fig. 11a while that for the LIFO controller is in Fig. 11b.

It should be readily apparent that the 1s of Fig. 11b form a *band* down the diagonal of the matrix. It should also be noted that the states of the LIFO memory controller only depend upon their neighboring states (i.e. state $y1$ only depends on $y0$ and $y2$). Because of this property, wire length and routing paths for the actual state machine is reasonable. Thus, it would seem that rearranging the order of the states in the 6-state Simple 12 to place states that are dependent on one another closer together could help reduce wire length and routing complexity.

This is in fact the case. Fig. 12 shows a second representation of state ordering for the 6-state Simple 12. Bandedness is represented both by shading and a number which is calculated simply by summing the distances of every one from the diagonal. Obviously a lower number is better. Fig. 11a shows a virtually non-existent band with a sum of 20. However, rearranging the order of the states (Fig. 12) indicates signs of a banded matrix and a lower sum of 15. Thus,

	S	I	E	N	O	X
S	1	1	0	0	0	0
I	0	0	1	0	0	0
E	0	1	0	1	0	0
N	0	1	0	0	1	0
O	0	1	0	0	0	1
X	1	1	0	0	0	0

S = Stopped
I = iFetch
E = Execute
N = No Jump
O = Operand
X = Execute
Sum from diagonal = 20

a.

	y0	y1	y2	y3	y4
y0	1	1	0	0	0
y1	1	1	1	0	0
y2	0	1	1	1	0
y3	0	0	1	1	1
y4	0	0	0	1	1

b.

Figure 11. A state matrix representation.

a smaller/narrower band means easier wire routing. (And if the band cannot be minimized beyond a reasonable level, you can save performance by multi-threading.) This information was then used to create a QCA schematic for the 6-state Simple 12 in Fig. 13. As one can see, the routing requirements for the state machine portion of the design are simple and short.

	S	X	O	I	E	N
S	1	0	0	1	0	0
X	1	0	0	1	0	0
O	0	1	0	1	0	0
I	0	0	0	0	1	0
E	0	0	1	1	0	1
N	0	0	0	1	0	0

S = Stopped
I = iFetch
E = Execute
N = No Jump
O = Operand
X = Execute
Sum from diagonal = 15

Figure 12. A new matrix representation.

One issue concerning Fig. 13 still must be addressed: there are very long wires that make up the feedback paths of this design. However, for this particular one-hot state machine this simply *cannot* be avoided. A comb-like structure cannot be used because of combinational logic and state dependencies. Also, the combinational logic is spread out over more than one clock/4 clocking zones. Thus, there is naturally a larger physical distance between the output and the input of this design. The only possible solution to breaking up long wires is to spread them out over more clocking zones. However, as discussed in Sec. 5.1 this can lead to incorrect timing.

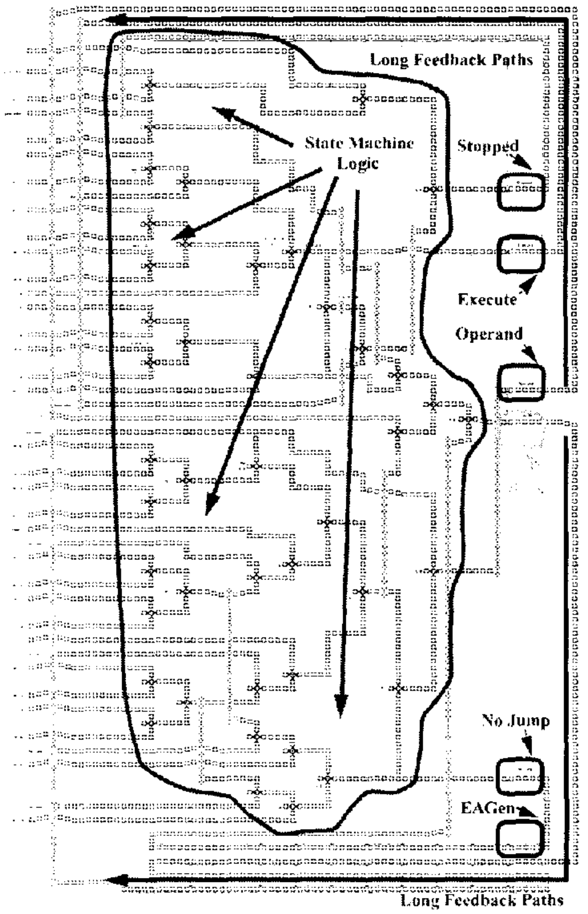

Figure 13. A QCA six-state Simple 12.

6.3. Relative Correctness

To expound upon the above idea, we introduce the idea of "relative correctness". Relative correctness does not mean that output of a state machine would be almost correct. Rather, it refers to the fact that the output of a state machine would be correct relative to the time of execution. This idea can best be explained with an example.

Let us revisit the 3-state Simple 12 one-hot state machine. A different representation/schematic of it appears in Fig. 14. Notice that the feedback path for the *execute* state is again spread out over 4 clocking zones/latches. Before, this would lead to incorrect clocking zone phase alignment. However, now, wire (and hence clocking zones/latches) have been added to the inputs of the design. Thus, it will now take an extra clock cycle for the next bit of input to the state machine to reach the combinational logic. This in turn will allow information about the last state to traverse the feedback path and arrive at the proper time when the clocking zones containing the necessary combinational logic are in the proper phases. Thus, long wires have been all but eliminated.

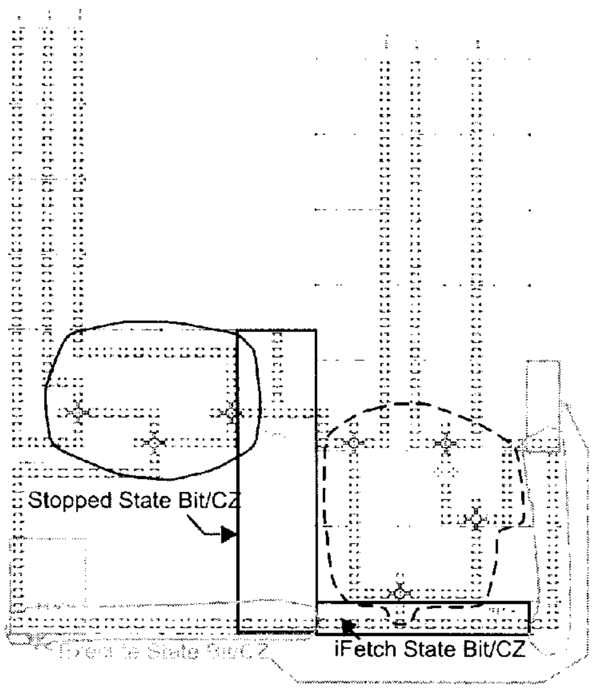

Figure 14. A "relatively correct" 3-state Simple 12 one-hot state machine.

The one negative is that it will now take an extra clock cycle to process information through this one-hot state machine. However, as was discussed in Sec. 4.2 and will be discussed in Sec. 7 it may be possible to inherently multithread such a self-latching state machine.

7. A Primitive Architecture

The work detailed thus far essentially discusses the components of a QCA architecture individually. This section will briefly describe a potential organization for all of them. This potential QCA Simple 12 organization will not consider memory. However, every other significant component of an architecture will be represented. In particular, this organization will consider the individual bits of the dataflow, a state machine, logic that uses state information to generate relevant control signals, any necessary delays required for the control signals, and the feedback delay imposed by the fact that the control logic will require information about the state of the accumulator at the end of the computation – i.e. whether or not it has a 0 value or is negative for conditional jumps. Each of these components will be discussed below – particularly within the context of the inherent "delay" imposed by the number of clocking zones required to hold the logic and/or wire for each.

The first thing to consider are the bits of the dataflow. By examining Fig. 6, we see that 8 clocking zones (2 clocks) are required for a value to be calculated by the ALU. However, before any dataflow computation can occur, some amount of "preprocessing" must be done. Obviously, the appropriate state of the processor must be determined and control signals must be generated. A state machine for Simple 12 was illustrated in Fig. 9. One can obviously see that 4 clocking zones (i.e. one clock) must be traversed to determine the state of the processor. Thus, before any dataflow processing can occur, a delay of 1 clock cycle must be incurred.

However, this is not the only delay to consider. Control signals must also be generated. While the control logic for the Simple 12 dataflow has not yet been designed in QCA, it has been synthesized. And it is estimated that the delay (call it B) will be at least 10 clock cycles. Thus, now no dataflow processing can begin until the delay caused by the state machine and control logic has been incurred. Additionally, the control logic and the state machine for Simple 12 require information about the state of the accumulator – i.e. if it is 0 or negative. This information will not become available until the last bit of the Simple 12 dataflow finishes computing. Now, given that each bit of the current dataflow uses a ripple carry adder, if the dataflow bits are appended linearly onto one another, there is a potentially huge feedback path and associated delay D that must also be factored into the design.

Despite the seemingly large number of delays that need to be considered/balanced a reasonable architecture for this processor is possible and a potential organization is illustrated in Fig. 15. First, there will be no need to delay the control signals after they are generated. These can simply be "pipelined" in the same lane with the dataflow bits so that they arrive at the appropriate logic at the appropriate time. Also, ideally, the delays B and D should be made as close to one another as possible to avoid stalls. Finally, if necessary, the potentially long feedback path for the zero and negative flags can be altered/eliminated by "folding the dataflow over" onto itself. This way, the most significant bit of the dataflow will be closer to the state machine and control logic. In fact, it can even be manipulated to help equalize the B and D delays.

Finally, there are two other things worth mentioning. First, the delays for the control logic (B), the zero and negative flag feedback paths (D), the state machine (C), and

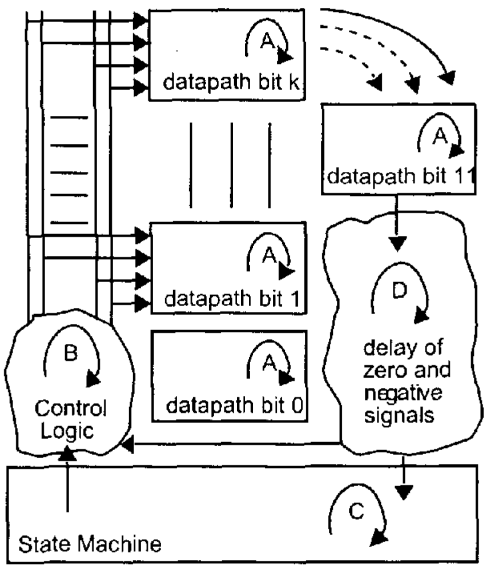

Figure 15. A potential QCA Simple 12 architecture.

the dataflow bits (*A* per bit) will affect the degree of multithreading that is possible. Essentially, the number of possible threads will be equal to at most the number of clocking zones modulo 4 in the longest closed loop comprised by the delays *A, B, C, and D* in Fig. 15. Second, future work will study the counterflow pipeline processor architecture as a means for future QCA designs. Specifically, this architecture has the potential for geometric regularity in processor chip layout and emphasizes local control to avoid complex pipeline stall signals – exactly what is needed in QCA [12].

8. Conclusions and Future Work

This work has succeeded in illustrating that with an inherently self-latched/self-clocked technology, ideas of layout and timing are more closely tied than ever before. Now, an inefficient layout does not just result in longer clock cycles – it introduces **more** clock cycles to a circuit path. However, despite this and other potential problems, opportunities such as "free" multithreading and processing-in-wire offer the potential to make up for delays that a self-latching device can create. In fact, in cases where combinational logic, etc. can fit into four clocking zones (and hence one clock), multi-threading and processing-in-wire can exploit the characteristics of a self-latching device. Additionally, techniques from systolic arrays such as retiming may allow automated rebalancing. Searching for and exploring such opportunities will be an extensive area of future work. Memory will also be studied. Placing these developments in the context of a potential three orders of magnitude density gain over the end of the CMOS curve equivalent certainly indicate that both QCA and other self-latching technologies deserve further study.

Finally, the authors would like to acknowledge the National Science Foundation and the Notre Dame Center for Nanoelectronics.

References

[1] I. Amlani, A. Orlov, G. Toth, G. H. Bernstein, C. S. Lent, and G. L. Snider. Digital logic gate using quantum-dot cellular automata. *Science*, 284:289–291, 1999.

[2] J. Hayes. *Introduction to Digital Logic Design*. Addison-Wesley Publishing Company, New York, 1993.

[3] C. Lent. Molecular electronics: Bypassing the transistor paradigm. *Science*, 288:1597–1599, 2000.

[4] C. S. Lent and P. D. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85:541, 1997.

[5] M. Niemier and P. Kogge. Logic in wire: Using quantum dots to implement a microp rocessor. In *Proceedings of 6th International Conference on Electronics, Circuits and Systems*, 1999.

[6] M. Niemier, M. Kontz, and P. Kogge. A design of and design tools for a novel quantum dot bas ed microprocessor. In *Proceedings of the 27th Design Automation Conference*, pages 227–232, 2000.

[7] A. Orlov, I. Amlani, G. Bernstein, C. Lent, and G. . Snider. Realization of a functional cell for quantum-dot cellular automata. *Science*, 277:928–930, 1997.

[8] A. Orlov, I. Amlani, C. Lent, G. Bernstein, and G. . Snider. Experimental demonstration of a binary wire for quantum-dot cellul ar automata. *Applied Physics Letters*, 74:2875–77, 1999.

[9] A. Orlov, I.Amlani, R. Kummamuru, R. Ramasubramaniam, G. Toth, C. Lent, G. Bernstein, and G. Snider. Experimental demonstration of clocked single-electron switching in quantum-dot cellular automata. *Applied Physics Letters*, 77:295–297, 2000.

[10] A. Orlov, R. Kummamuru, R. Ramasubramaniam, G. Toth, C. Lent, G. Bernstein, and G. Snider. Experimental demonstration of a latch in clocked quantum-dot cellular automata. *Applied Physics Letters*, 78:1625–1627, 2001.

[11] G. Snider, A. Orlov, I. Amlani, X. Zuo, G. B. stein, C. Lent, J. Merz, and W. Porod. Quantum-dot cellular automata: Review and recent experiments. *J. of Applied Physic*, 85:4283–85, 1999.

[12] R. F. Sproull, I. E. Sutherland, and C. E. Molnar. The counterflow pipeline processor architecture. *IEEE Design & Test of Computers*, 11(3):48–59, 1994.

[13] J. Timler and C. Lent. Dissipation and gain in quantum-dot cellular automata. *unpublished*.

[14] P. Tougaw and C. Lent. Logical devices implemented using quantum cellular automata. *Journal of Applied Physics*, 75:1818, 1994.