Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 352428, 11 pages
doi:10.1155/2010/352428

Research Article

Design of a Reconfigurable Pulsed Quad-Cell for Cellular-Automata-Based Conformal Computing

Mariam Hoseini,[1] Zhou Tan,[1] Chao You,[1] and Mark Pavicic[2]

[1] *Electrical and Computer Engineering Department, North Dakota State University, Fargo, ND 58102, USA*
[2] *Center for Nanoscale Science and Engineering, North Dakota State University, Fargo, ND 58102, USA*

Correspondence should be addressed to Mark Pavicic, mark.pavicic@ndsu.edu

Received 22 November 2009; Revised 1 April 2010; Accepted 28 June 2010

Academic Editor: Paul Chow

Copyright © 2010 Mariam Hoseini et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper presents the design of a reconfigurable asynchronous computing element, called the pulsed quad-cell (PQ-cell), for constructing conformal computers. Conformal computers are systems with an exceptional ability to conform to the physical and computational needs of an application. PQ-cells, like cellular automata, are assembled into arrays, communicate with neighboring cells, and are collectively capable of general computation. They operate asynchronously to scale without the limitations of a global clock and to minimize power consumption. Cell operations are stimulated by pulses which travel on different wires to represent 0's and 1's. Cells are individually configured to perform logic, move and store information, and coordinate parallel activity. The PQ-cell design targets a 0.25 μm CMOS technology. Simulations show that a single cell consumes 15.6 pJ per operation when pulsed at 1.3 GHz. Examples of multicell structures include a 98 MHz ring oscillator and a 190 MHz pipeline.

1. Introduction

In recent years there has been widespread interest in making things out of very large numbers of very small parts. These parts could be special molecular structures, microfabricated devices, or even living cells. The parts are so small and numerous that new approaches are sought for assembly, programming (defining local interactions to achieve global behavior), dealing with faults, and so on. There are many ideas about what such an ensemble might be useful for. It could be some form of programmable material, "smart matter", swarms of tiny robots, or simply a computer. Related research areas that have computing as a desired outcome include molecular computing [1, 2], biomolecular computing [3], bioinspired computing [4, 5], and amorphous computing [6].

For computer systems with many small parts, the programming models tend to be quite different from what is used in conventional computers. For example, in amorphous systems [7], information essentially diffuses through the system. This is similar to node-to-node "hopping" in wireless sensor networks. In both cases information moves in steps that are much shorter than the dimensions of the system. How to deal with such issues is of interest because it may enable the realization of systems that are superior to today's programmable systems in important ways. In particular, it would be very useful to be able to perform brain-like tasks with systems that are much smaller and more efficient than what can be expected from today's computing architectures.

Our interest is in nonbiological cellular arrays in which the parts are densely packed in a regular structure and the need for power and communication is met by electrically conductive wires or planes. In particular, we envision subarrays fabricated on CMOS chips, and the chips, in turn, are arrayed on large thin flexible substrates or sheets. Sheets may be cut, joined, bent, and stacked to conform to the physical and computational needs of an application. We refer to this flexible and scalable form of computing as "conformal computing" [8].

Our long-term vision is to help make progress toward systems capable of efficiently performing brain-like tasks. Conformal computing moves in that direction by exploring a computational medium assembled from "cells" that are much simpler than conventional instruction-processing

nodes. Although the cells can be used to assemble conventional structures, our desire is to explore alternatives that are more similar to cellular automata [9], crystalline computing [10], cell matrices [11], BLOB computing [12], cellular neural nets [13], and the like. Therefore the cell designs emphasize simplicity (small multiplexors, 2-input logic units) and scalability (clock-less synchronization, multichip arrays) combined with features of cellular automata (regular structure, local communication) and FPGAs (reconfigurable function and initial state).

This paper presents a particular cell design, called the pulsed quad-cell (PQ-cell), for constructing a conformal computer. The PQ-cell is a quad cell because it consists of four orientations of an elementary cell called a quarter. The PQ-cell is the latest in a series of cell designs that include a clocked cell [14] and a triggered cell. These designs are distinguished by the source of the stimulus that causes a cell to perform an operation. Clocked cells use pulses generated from a central source and distributed throughout the array. Triggered cells use pulses generated by other cells in response to previous pulses and routed along computational paths. In the clocked and triggered schemes, a transferred data bit is accompanied by a pulse which stimulates the receiving cell to accept and process the bit. Because the data and stimulus are conducted on separate wires, it is necessary to design for worst-case delays to ensure that the data is set up before the pulses arrive. This necessity is eliminated if the data itself is the stimulus. This is the idea in the PQ-cell design. The PQ-cell design uses dual-rail encoding in which a unit of data is a single pulse that appears on one of two rails (wires): one rail for "0" pulses and the other rail for "1" pulses. The cells route the data pulses along computational paths.

Rather than a theoretical design, or one based on a future technology, the PQ-cell design is targeted for fabrication in a 0.25 μm CMOS technology. Therefore the PQ-cell array can serve as a concrete example of a novel computational host for new and beneficial forms of computation. The remainder of this paper is as follows. Section 2 uses comparisons with Cellular automata (CA) and FPGAs to describe the general features of PQ-cell arrays. Section 3 presents the specifics of the PQ-cell design. Section 4 shows simulation results for a single cell and a variety of useful multicell structures. Section 5 contains a summary and conclusions.

2. Background

To conform to a wide range of computational needs, a computing system needs to scale from small to very large sizes. This need can be met by an extensible system of small computational elements. CA have these properties [15–17]. A CA cell is simple and the cells are arranged on a lattice that can have a periphery to which cells can easily be added. Similarly, PQ-cell arrays are scalable because, like CA, the cells are simple and are arranged on a two-dimensional lattice.

PQ-cell arrays are also like CA in that a cell has a state, and state transitions follow rules that are based on the states of nearby cells and possibly its own state. The state transitions occur when cells perform an update. In CA, updates are performed throughout the array in a parallel fashion, which may be either synchronous or asynchronous [18]. If synchronous, all the cells update their states once within each of a series of discrete time steps. Each step involves two phases: input and output. During the input phase, the cells input the states of certain nearby cells. During the output phase, the cells update their states. All the cells complete a phase before any of the cells move on to the next phase. If asynchronous, there is no global synchronization and updates depend on other factors. The PQ-cells update asynchronously in response to pulses sent by neighboring cells. By eliminating the need for global synchronization, the PQ-cell architecture is easier to scale to large sizes.

Computing with PQ-cell arrays can use any of the methods in use for CA. The two most common methods are digital circuit emulation and spatiotemporal modeling of a dynamic system. This paper focuses on circuit emulation; however, specially designed cells may be emulated by PQ-cell subarrays and used to model dynamic systems. The emulated cells then become the building blocks for larger computational structures such as cellular neural nets.

CAs are typically uniform, which means that all the cells follow the same rules for state transitions. Therefore, for digital circuit emulation, the usual approach is to create patterns of cells to perform the functions of wires, logic gates, and registers [19]. These patterns involve multiple cells and may take many cycles to advance a signal. PQ-cell arrays, however, are like nonuniform CA. The cells may have different rules. This allows a more effective method in which a single PQ-cell can perform the function of a wire, a logic gate, a storage element, or simple combinations thereof. By directly implementing these circuit elements, computation is faster and more compact. Furthermore, PQ-cells can be cascaded without intermediate storage elements. This optimizes the performance of multilevel combinatorial logic. Customization of a PQ-cell is achieved by loading a set of configuration bits. A similar initialization step is needed for CA, but only the initial state is specified. The configuration of a PQ-cell specifies its initial state, its transition rules (inputs and functions), and how it will synchronize parallel activity.

The ability to configure and reconfigure the cells is a feature that PQ-cell arrays share with FPGAs. The origins of the FPGA include the cellular arrays surveyed by Minnick in 1967 [20]. An early and enduring motivation for cellar arrays was to be able to use low-cost batch processing methods. An accompanying idea is some form of configurability, which is needed to customize the array to a particular application. It was also recognized early on that it would be desirable to do this configuring "in the field", as opposed to in the factory and ultimately to be able to configure repeatedly without removing or even having physical access to the device. These desires are now met by FPGAs and similar devices.

The PQ-cell explores a variation on the FPGA theme in which emphasis is on support for computational paradigms that deal directly with the spatiotemporal realities of a physical computing system. For some problems, including brain-like tasks like pattern recognition, this approach may lead to significantly improved performance and scalability. For this reason the cell designs developed so far have not adopted some of the features that optimize the mapping

of arbitrary circuits onto an array. In particular, the PQ-cells route pulses through cells rather than through an interconnection network. Also, since it may be used for routing, and it is not yet clear what functions are needed, each quarter of a PQ-cell uses a simple 2-input logic unit rather than a 4- to 6-input look-up table.

Another difference is explicit support for asynchronous operation. Each PQ-cell contains a unit for synchronizing pulses. This unit also enables each cell to be configured as a stage in a pipeline for processing and transporting information. Pipelines can cross chip boundaries. This supports extreme scalability and allows portions of the array to operate at different speeds (for purposes such as local heat management). So a PQ-cell array is like an extensible FPGA whose reconfigurable elements are simple cells that communicate asynchronously with nearby cells to update their states.

3. The PQ-Cell Design

This section presents the PQ-cell design, beginning with basic features and then focusing on facilities for processing and storing bits, routing pulses, coordinating parallel activity, maintaining pulse integrity, satisfying timing requirements, and configuring PQ-cell arrays.

3.1. Basic Features: Pulsed Operation and Quarters. At a high level, a PQ-cell is a unit that receives and sends pulses. It can receive a pulse at any one of four inputs and respond by sending a pulse on any number of four outputs. The received pulse may be interpreted as a bit of data or as a control signal. The sent pulses are always in response to a received pulse. So, without stimulation by a pulse, a PQ-cell does nothing. This is one reason for why PQ-cell arrays are expected to be efficient consumers of power.

A PQ-cell is called a quad-cell because it consists of four elementary cells called quarters. A quarter is the basic operational unit of a cell.

Figure 1(a) shows a quarter (shaded box) and its connections to four neighboring quarters (open boxes). A quarter receives pulses from two quarters, one internal to the cell and one external. Likewise it sends pulses to two quarters, one internal and one external. Each pulse appears on one of a pair of wires, which is what allows the pulse to be interpreted as a bit. Figures 1(b) and 1(c) show how four quarters are combined to form a quad-cell and how the connections between quarters form the connections between cells.

3.2. Processing and Storing Bits: Logic Units and Data Latches. In response to a pulse from the internal quarter, a quarter records which wire the pulse arrived on. This record is stored in a data latch (RS latch) and becomes the B-input to a logic unit (LU) within the quarter. The LU also has an A-input, which receives pulses from an external quarter. Each pulse at the A-input causes the LU to form a result based on the A and B inputs. This result is represented by a pulse that is sent to an internal quarter and to an external quarter.

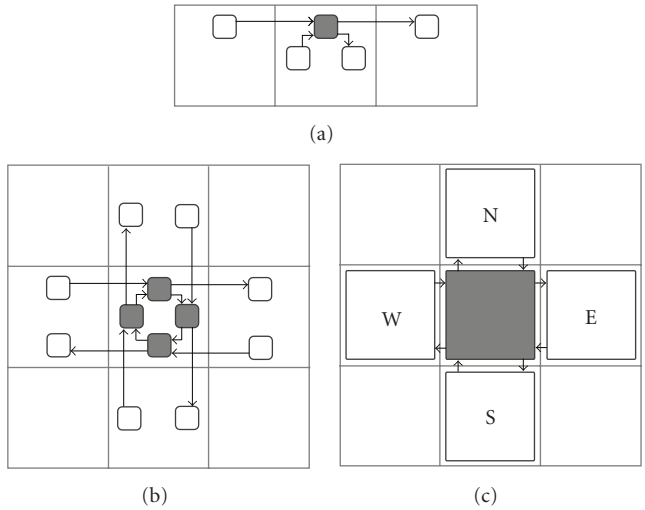


Figure 1: Where pulses are received from and sent to by (a) a quarter, (b) the four quarters of a cell, and (c) a cell. Each arrow represents a pair of wires.

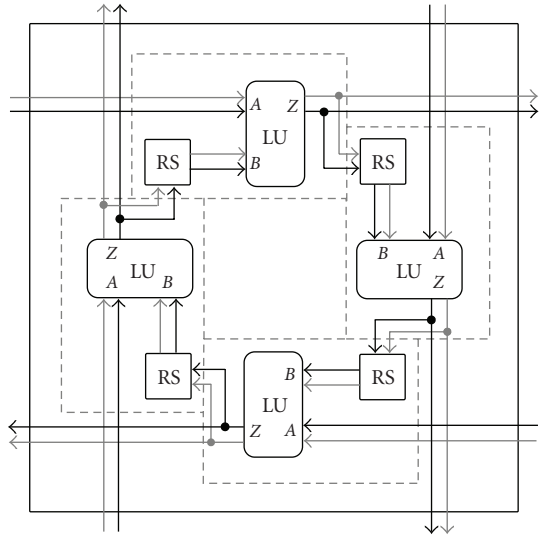


Figure 2: Logic units (LUs) and data latches (RS latches). To the A input and from the Z output, the black wires carry "1" pulses and the gray wires carry "0" pulses. To the B input, the black and gray wires are the Q and $\overline{Q}$ outputs, respectively, of the RS latch.

Figure 2 gives internal views of the quarters that show the connections between the LUs and the data latches. The dashed lines outline the quarters (shown as shaded boxes in Figure 1(b)). Each data latch is implemented as an RS latch. The result formed by an LU is a logical combination (Boolean function) of the A and B inputs and appears as a pulse at the Z-output.

Table 1 lists the LU functions. Columns D, E, F, and G correspond to configuration bits that select which one of the 16 functions is performed by the LU. Z and $\overline{Z}$ are the "1" and "0" output wires, respectively, of the LU. The expressions in the Z and $\overline{Z}$ columns specify which wire will carry the output

Table 1: The PQ-cell LU functions.

D	E	F	G	Z	$\overline{Z}$
0	0	0	0	0	1
0	0	0	1	$A \cdot B$	$\overline{A} + \overline{B}$
0	0	1	0	$A \cdot \overline{B}$	$\overline{A} + B$
0	0	1	1	A	$\overline{A}$
0	1	0	0	$\overline{A} \cdot B$	$A + \overline{B}$
0	1	0	1	B	$\overline{B}$
0	1	1	0	$A \oplus B$	$\overline{A} \cdot \overline{B} + A \cdot B$
0	1	1	1	$A + B$	$\overline{A} \cdot \overline{B}$
1	0	0	0	$\overline{A} \cdot \overline{B}$	$A + B$
1	0	0	1	$\overline{A} \cdot \overline{B} + A \cdot B$	$A \oplus B$
1	0	1	0	$\overline{B}$	B
1	0	1	1	$A + \overline{B}$	$\overline{A} \cdot B$
1	1	0	0	$\overline{A}$	A
1	1	0	1	$\overline{A} + B$	$A \cdot \overline{B}$
1	1	1	0	$\overline{A} + \overline{B}$	$A \cdot B$
1	1	1	1	1	0

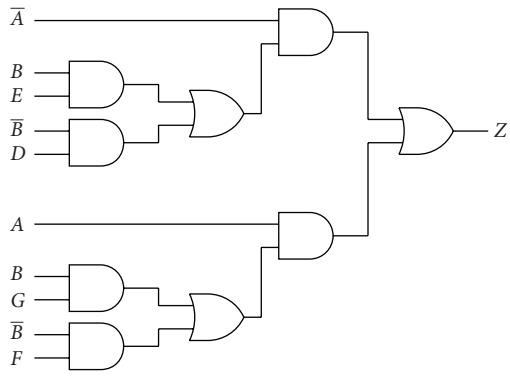


Figure 3: Implementation of half of the LU.

pulse. For example, if $DEFG = 0011$, then $Z = A$ means that a pulse that enters the LU on the A wire will exit the LU on the Z wire. Likewise, $\overline{Z} = \overline{A}$ means that a pulse that enters the LU on the $\overline{A}$ wire will exit the LU on the $\overline{Z}$ wire. Another example is that if $DEFG = 0101$, then $Z = B$ and $\overline{Z} = \overline{B}$. In this case there is no dependence on where the pulse enters the LU. If $B = 1$, the pulse exits on the Z, wire and if $B = 0$, the pulse exits on the $\overline{Z}$ wire.

Figure 3 shows an implementation of half of the LU using combinatorial logic. It is essentially a selector that chooses some combination (either, neither, or both) of $\overline{A}$ and A to exit at Z. The choice is based on four configuration bits and the state of the data latch. The circuit for $\overline{Z}$ is equivalent.

3.3. Routing Pulses: Paths, Turns, Crossovers, and Forks. In its response to a pulse, a PQ-cell may send a pulse to one or more of its neighbors. A neighbor may, in turn, send a pulse to one or more of its neighbors, and so on. This sequence of operations forms a path through the array. It is necessary that a means be provided for steering pulses along these paths.

Pulse steering in a PQ-cell is achieved by using a selector to insert a right turn. Figure 4 shows the cell with the

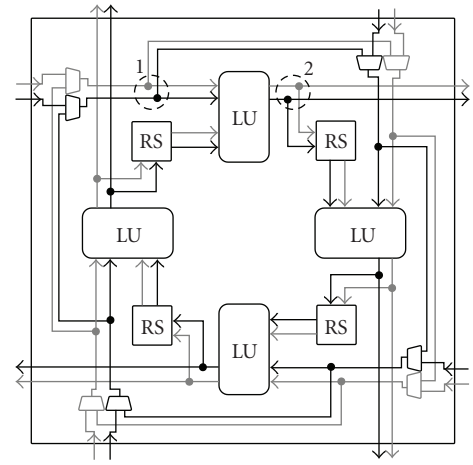


Figure 4: Adding selectors for making right turns. The numbered circles locate forks.

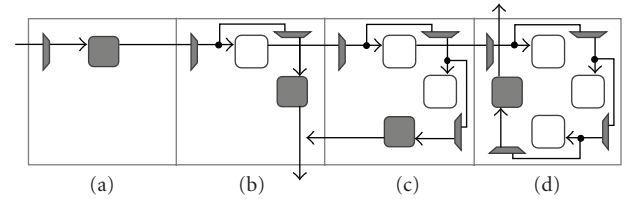


Figure 5: Using selectors to make turns of (a) 0 degrees, (b) 90 degrees, (c) 180 degrees, and (d) 270 degrees.

selectors. Each selector chooses one of two sources for the A input of the LU. Since each quarter has a selector, up to three successive right turns can be made in a cell. Examples are shown in Figure 5.

The cell configuration determines which source is connected to the A input. Since there are two wires coming from each source, there are two 1-of-2 selectors. For added flexibility, these selectors are configured independently. This is useful when routing control signals. To prevent a pulse from initiating further activity, a cell may configure its input selectors so that it does not accept pulses from that source.

A path may need to cross itself or another path. In a PQ-cell, this need is met by the connections between the quarters, which include four crossovers (Figure 1(b)).

If a cell, in response to a single pulse, sends pulses to multiple neighbors, the cell is initiating parallel activity. This is called a fork. In a PQ-cell, a fork results when a quarter uses the turn selector to accept an internal input (Figure 5(b)). This can happen at most three times in succession because there are four quarters and the input pulse must be accepted by one of them. So one input pulse could result in up to four output pulses, each headed in a different direction.

3.4. Coordinating Parallel Activity. To coordinate parallel activity, the PQ-cell includes a synchronizing operation called a join. (It could also be called a rendezvous.) A join involves two or more quarters within a cell. A quarter participates in a join if it is configured to do so.

A participating quarter is either ready or not-ready to send an output pulse. The join condition is satisfied when every participating quarter is ready. If a participating quarter is ready, it was either initialized to be ready or it became ready by performing an operation in response to an input pulse. Once the join condition is satisfied, each of the participating quarters outputs a pulse and becomes not-ready.

The PQ-cell implementation of the join operation is shown in Figure 6. The additional circuitry is collectively referred to as the synchronizer. It has four configuration bits, $J1$, $J2$, $J3$, and $J4$, which indicate which cells are participating in the join. These bits also control the output selectors, choosing either the path from the LU (if a quarter is not participating in the join) or the switched path from the synchronizer (if a quarter is participating in the join). Each quarter has an RS latch that is set when a pulse exits the LU. This is called the event latch because it records an event of interest to the synchronizer. Also associated with each quarter is an OR gate whose output feeds into a 4-input AND gate.

The join condition is satisfied when the output of the AND gate, labeled R (for reset), is high. Therefore the outputs of all the OR gates must be high. For each quarter, the OR gate output is high if the corresponding configuration bit is high (the quarter is not participating in the join) or the event latch output is high (the quarter is ready to output a pulse). When the join is satisfied, R is used to reset the event latches, which also causes R to return low. This produces a reset pulse that passes through the switch closed by an output of the data latch. So the effect of a join is to delay the LU outputs of participating quarters until the join condition is satisfied.

3.5. Maintaining Pulse Integrity. As a pulse travels along a path, its amplitude is restored each time it is redriven. However, its width may get shorter or longer, depending on the relative speeds with which leading and trailing edges are generated by the circuitry. If a pulse becomes too short, it may fail to stimulate further logic and vanish. If a pulse becomes too long, it may interfere with other pulses. Therefore some means is required for maintaining pulse width. This is the purpose of the pulse regenerator (PR).

The PR outputs a pulse of width W in response to an input pulse that may be shorter or longer than W. Figure 7 shows one form for the PR. It has two delays, $D1$ and $D2$. This circuit outputs a pulse of width $W = D2 - \delta$ (where δ is the delay through the first NOR gate) in response to an input pulse whose width is at least $D1$, where $W/2 < D1 < W$. Choosing $D1$ near $W/2$ allows for narrower input pulses.

A PR is at every output from a cell. Figure 8 is a composite diagram of the PQ-cell that shows where the PRs are located.

3.6. Satisfying Timing Requirements. Correct operation of computations in PQ-cell arrays requires certain timing requirements to be observed. Pulse width requirements are managed by the pulse regenerator. Pulse separation is managed by handshaking such as in pipelines. A remaining requirement is to ensure that the B-input to an LU is set before a pulse arrives at the A-input. The output of the LU is produced in response to a pulse at the A-input. Since the B-input prepares the LU to produce this response, it must arrive a short time before the pulse at the A-input. Figure 9 shows three ways to achieve this.

The first solution depends on other cells to delay the A-input relative to the B-input (Figure 9(a)). The other solutions use the join operation and are independent of delays external to the cell. In these solutions, the path leading to the A-input passes through the cell before reentering the cell and delivering a pulse to the A-input. This path includes an LU that participates in a join with the LU that delivers the B-input (Figure 9(c)). If one LU is the source of both inputs, then only that LU participates in the join (Figure 9(b)).

The solutions using the join are dependent on the design of fork 3. Fork 3 is one of five forks within the PQ-cell whose locations are circled and numbered in Figures 4 and 6. Each fork is a point at which a signal simultaneously enters two or more paths. If these paths interact within the cell, and the result of this interaction depends on the order in which the signals arrive, then the cell needs to be designed to ensure a consistent outcome. Fork 3 creates two paths, one that goes to the B-input of the LU and one that exits the cell under the control of the synchronizer. By involving the synchronizer, the delay in the shortest path from fork 3 to the A-input (which involves exiting then reentering the cell) is guaranteed by design to exceed the time required for the B-input to set up the LU. Therefore, even if there are no external delays, the input order is still correct.

Another design consideration for fork 3 results from an internal interaction between its two paths. One path sets the data latch and the other path could cause the synchronizer to generate a reset pulse. Since the data latch sets up the route by which the reset pulse exits the cell, the path through the data latch must be shorter than the path through the synchronizer.

Fork 1 is also of interest. This fork is located at the input of the LU in the N quarter and creates two paths that may come together at the LU in the E quarter. One path leads to the B-input and the other path (if chosen by the input selector) leads to the A-input. The path to the A-input is shorter; so the LU should be configured to perform a function that uses only the A-input; that is, $Z = 0$, 1, A, or $\overline{A}$. Forks 2, 4, and 5 create independent paths and therefore present no special timing issues.

3.7. Configuring PQ-Cell Arrays. The behavior and initial state of a cell are determined by a set of configuration bits that are loaded into the cell before it is used. A simple way to load these bits is to shift them serially into a long shift register that contains all the bits of all the cells in the array. However, this would be a slow process, especially for large arrays. A faster and more flexible scheme is envisioned for PQ-cell arrays.

A PQ-cell array has many short shift registers. Each register holds the configuration bits for a subset of the array, which may be a single PQ-cell. Each quarter has 9 configuration bits: 4 for the LU, 2 for the input selector, 1 for the data latch, 1 for the event latch, and 1 to specify whether or not the quarter participates in the join.

Each register is at a node of a 2D mesh network. Serial streams of configuration bits pass through the network

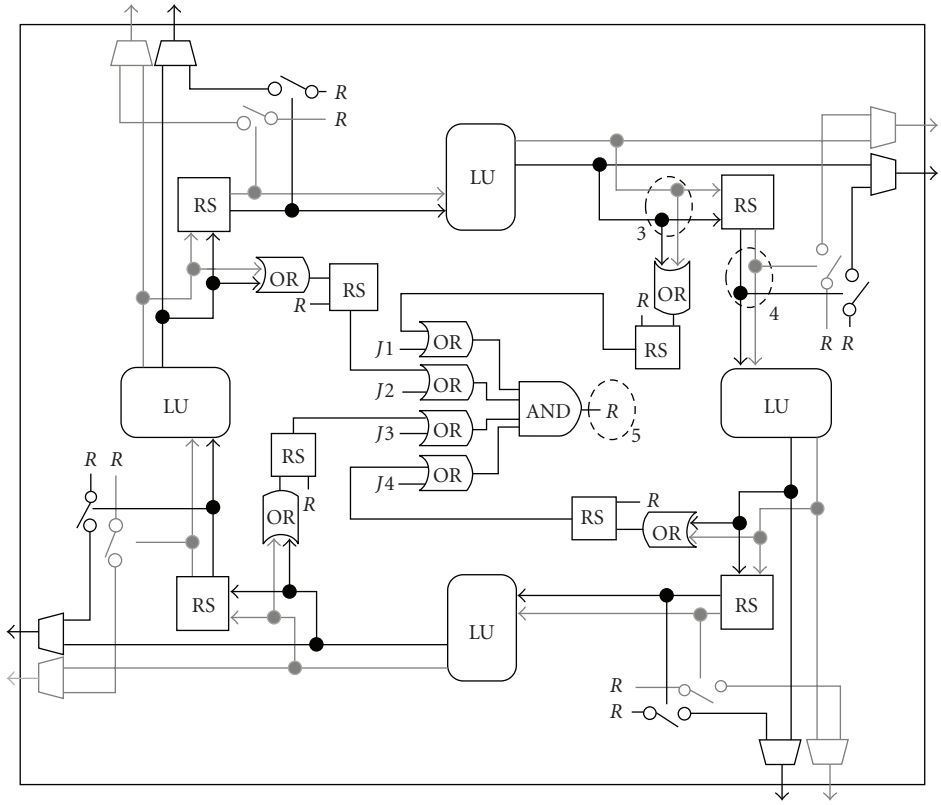


Figure 6: PQ-cell circuitry involved in performing the join operation.

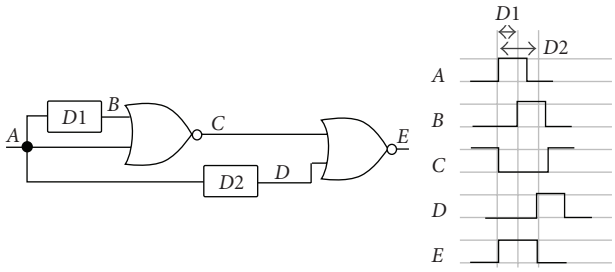


Figure 7: A design for the pulse regenerator (PR).

to reach selected registers, bypassing registers that are not configured by that stream. Multiple streams may be in the network at the same time. Furthermore, cells that are not being configured may continue to operate.

Figure 10 shows a single node in the configuration network. In this case the node is associated with a single PQ-cell. The controller routes incoming data bits to one of three shift registers or to another node. The first bit to arrive determines whether or not this node will receive configuration bits. The second bit identifies the end of the stream. The next two bits select the next node to be visited by the stream. Then, depending on the first bit, configuration bits for this node will follow. Finally, depending on the second bit, any additional bits are passed on to the selected node.

The benefits of selectivity can be illustrated by a numeric example. Based on simulation results, the configuration network is expected to transport bits at roughly the speed of a pipeline structure (Section 4.4), which is about 200 MHz in the target technology. With 36 configuration bits and 4 bits for routing, a single cell requires 200 ns to program. To reach a neighboring cell adds 5 ns of propagation time. Therefore a block of 1000 contiguous cells takes 205 μs to program. If this block is part of a much larger array of $N \times N$ cells, and configuration bits originate from the periphery of the array, the configuration stream will pass through an average of N cells to reach the block. Since each node consumes 4 routing bits, the delay through a cell is 25 ns. Therefore N can be as large as 8200 before the stream transport time exceeds the block programming time. Using a single configuration stream to program this entire array would require 13.8 seconds. This time can be reduced by using multiple concurrent configuration streams, but it is still clearly advantageous to be able to program a large array selectively.

4. Simulations of the PQ-Cell

A small PQ-cell array is being designed for fabrication in TSMC's 0.25 μm CMOS technology. Each PQ-cell uses about 2000 transistors. Approximately one half of these is for the configuration circuitry. The other half is divided between the synchronizer (120 transistors) and the four quarters (each of which uses 216 transistors). Simulations have been performed for single and multiple cell structures. These simulations verify that the cell functions as expected and supply

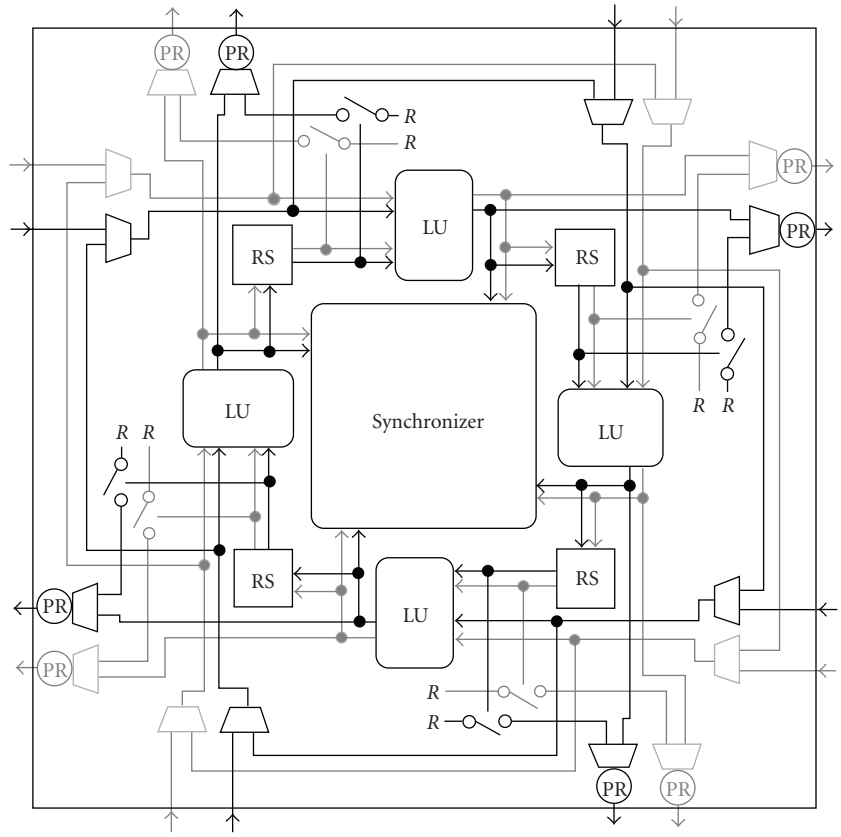


Figure 8: Composite diagram of the PQ-cell including a pulse regenerator (PR) at each output.

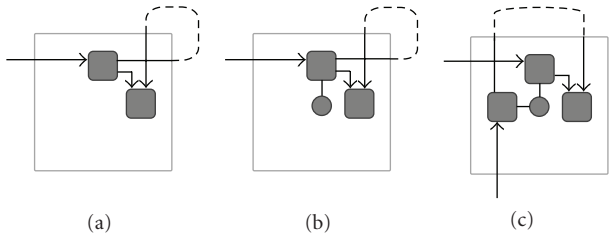


Figure 9: Three ways to properly order the two inputs to an LU: (a) introduce sufficient external delay, (b) supply both inputs from an LU within the same cell that participates in a join, and (c) supply the A-input from an LU that participates in a join with the LU that supplies the B-input.

approximate measures of performance and power dissipation. This section describes four of these simulations: a single cell, a full adder, a ring oscillator, and a pipeline. At the end of this section, PQ-cell performance is compared with that of asynchronous FPGA logic cell designs of other researchers.

4.1. Single Cell. A single cell was simulated to find the minimum pulse width, the propagation delay through a cell, and the energy consumed in a single operation and to explore the effect of supply voltage on these measures. The minimum pulse width was determined to be about 550 ps for supply voltages between 1.8 V and 2.5 V. The temperature was 25°C. This result was independent of the cell function.

For the other measures, the cell was configured to perform the XOR function. The B-input was set to zero and pulses were supplied to the A-input. The input pattern was a series of pulses alternating between the "0" wire and the "1" wire. The pulses were 700 ps wide and were separated on each wire by 800 ps. This is an input pulse rate of 1.3 GHz. The results are shown in Figures 11 and 12.

Figure 11 shows the input and output waveforms when operating at 2.5 V. The propagation delay through the cell is 1.1 ns. At 1.8 V the propagation delay is about 1.5 ns. Figure 12 shows the current profiles when operating at 2.5 V and 1.8 V. At 2.5 V the average current is approximately 2.5 mA over a period of 2.5 ns; so the energy consumption per pulse is 15.6 pJ. At 1.8 V the average current was approximately 1.5 mA over a period of 3.2 ns; so the energy consumption per pulse is 8.6 pJ. This is significantly less than the consumption at 2.5 V. The tradeoff is a modest increase in propagation delay. Note that energy consumption is essentially zero when there are no pulses. This is one of the benefits of asynchronous circuits.

4.2. Full Adder. The full adder is an example of using multiple PQ-cells to perform combinatorial logic. Figure 13 shows how the adder was constructed. The inputs to the adder are A, B, and C. C is carry in. The Sum is formed as $A \oplus B \oplus C = A \oplus (B \oplus C)$. The Carry (carry out) is formed as $AB + AC + BC = A(B + C) + BC$. The supply voltage was set to 2.5 V. Input pulses representing a "1" were sent

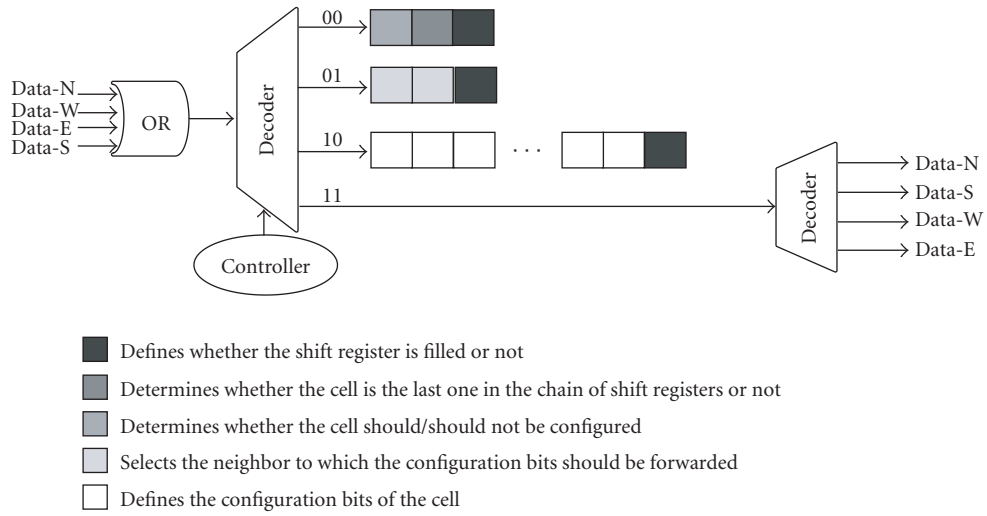


Figure 10: The behavioral model of the configuration circuitry.

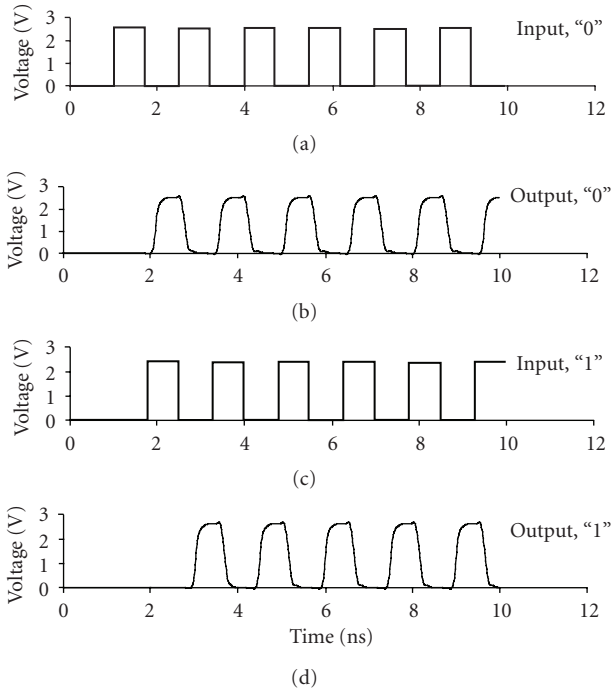


Figure 11: Simulation results of a PQ-cell performing an XOR function.

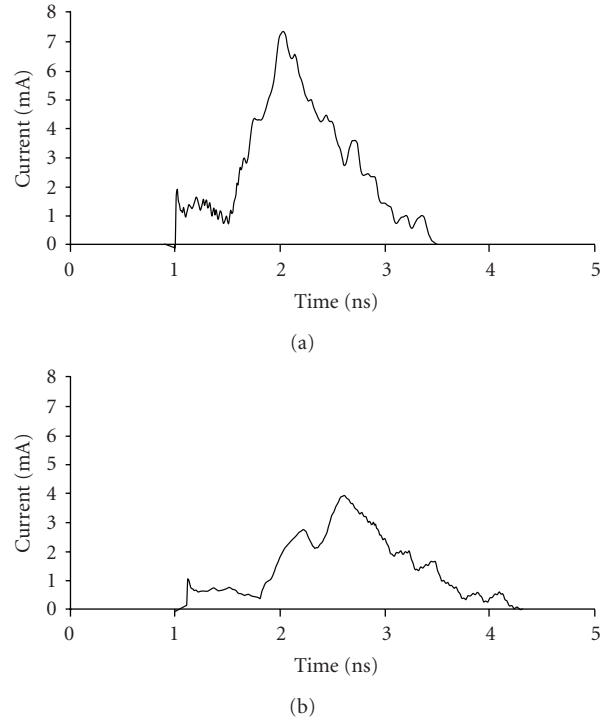


Figure 12: The PQ-cell current profile at (a) 2.5 V and (b) 1.8 V. The profile at 1.8 V is lower (average is 1.5 mA) and longer (by about 700 ps).

to the *C*, *B*, and *A* inputs, in that order, at 2 ns intervals. The output pulse for Sum appeared after a 1.1 ns delay and the output pulse for Carry appeared after a 3.6 ns delay. These results are consistent with what was expected considering the propagation delay through a single cell.

4.3. Ring Oscillator. The ring oscillator is a loop. Four cells were used in its construction (Figure 14).

The oscillator has a start input, a stop input, and an output. The oscillator is started by supplying a "1" pulse at the start input. The input selector is configured to accept "1" pulses from an external source and "0" pulses from an internal source. The input pulse causes the N quarter of cell a to output a "1" pulse, which passes through the N quarter of cell *x* and triggers the N quarter of cell *b*. The N quarter of cell *b* outputs the complement of the latest value it received from the W quarter; initially, this is a 0. Cell *c* receives the output from cell *b*, duplicates it, and sends one copy to the output and the other copy back to cell *b*. Cell *b* duplicates this input and stores one copy as the *B*-input of the N quarter.

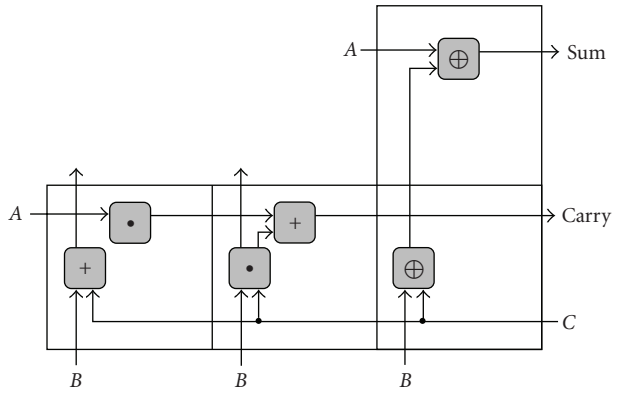


Figure 13: Full adder constructed from four cells in a PQ-cell array.

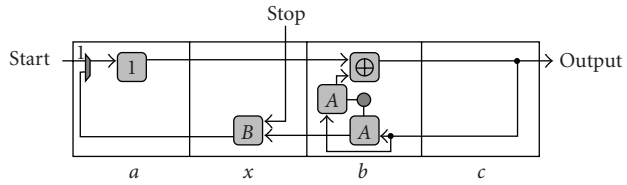


Figure 14: Ring oscillator constructed from four PQ-cells.

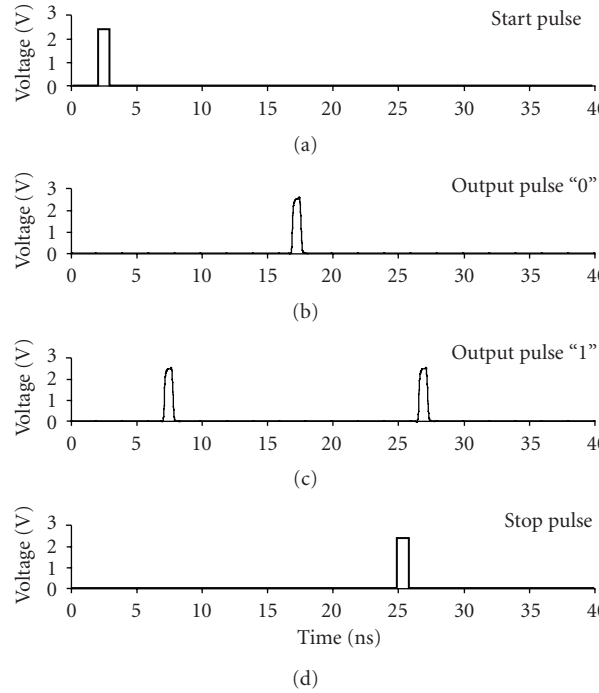


Figure 15: Simulation results of the ring oscillator.

The other copy goes to the S quarter of cell x which outputs a pulse with the value at its B-input. If this value is a 0, the oscillator continues; otherwise it stops. Also note that the S and W quarters of cell b participate in a join. The effect of this is to delay the cell output from the S quarter until the W quarter has readied its output. This ensures the proper arrival order of the inputs to the N quarter.

When simulating the oscillator, the data latches are initially zero. Nothing happens until a "1" pulse enters the start input. This causes the first output, which is the complement of the data latch at the B-input of the LU in the N quarter of cell b. The simulation results are shown in Figure 15. The output pulse rate is approximately 98 MHz at 25°C. The oscillator was also simulated at −25°C and 65°C, and the output pulse rates were 119 MHz and 84 MHz, respectively. This inverse relationship is due to decreases in transistor current as temperature increases.

4.4. Pipeline. Pipelines are important structures for transporting and processing data. In particular, asynchronous pipelines, because of their ability to store variable amounts of data, can form elastic connections between computations at different locations within a PQ-cell array. Event-driven elastic pipelines, with or without internal processing, were the subject of Sutherland's 1988 ACM Turing Award lecture [21].

PQ-cells are readily configured as symmetric pipelines that can operate in either direction. Figure 16 shows a series of PQ-cells forming three stages of a pipeline. Refer to the quarters by their compass locations within a cell: N, S, E, and W. Then this pipeline involves the N and S quarters in each cell. These quarters are configured to participate in a join. To cause the pipeline to operate in the West to East direction,

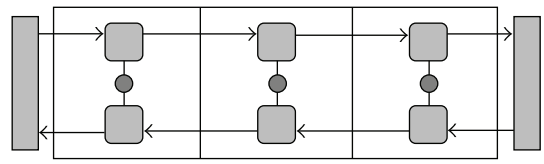


Figure 16: Three stages of a pipeline. Each stage uses two quarters and the synchronizer (filled circle) of a PQ-cell.

the N quarters are initialized to not-ready and the S quarters are initialized to ready.

For any cell in the initial state, if a pulse arrives at N, the join is satisfied and two pulses are sent, one from N and one from S. If the input pulse is interpreted as data, then the pulse from N is interpreted as data being passed to the next stage and the pulse from S is interpreted as a signal being passed to the previous stage. The new state of the cell is that both quarters are not-ready. The next state depends on which event occurs first; the cell receives either a data pulse from the previous cell or a signal pulse from the next cell. But one event will not satisfy the join. Only when both events have occurred is the join satisfied and the cell outputs another pair of pulses: a data pulse to the East and a signal pulse to the West.

A 3-stage pipeline was simulated to see how fast it would run. It was configured as a loop, with a wire connecting the output to the input at each end. By inserting an initial pulse, the pipeline circulated the data that was initially in the data latches. The resulting pipeline speed was 190 MHz. This can be improved by integrating pulse width control (Section 3.5) with other circuitry; so this figure should be viewed as a lower bound.

Table 2: Speed and energy consumption of various asynchronous cell designs.

Design	Technology	Voltage	Speed (MHz)	Energy (pJ/cycle)
Wong	TSMC 180	1.8	190–235	2.1–3.1
Teifel	TSMC 250	2.5	400	18
Mahram	TSMC 180	1.8	280	2.2
PQ-cell	TSMC 250	2.5	190	15.6

4.5. Comparisons. To obtain a view of a PQ-cell's performance compared to other work, the asynchronous FPGA logic cells presented by Wong et al. [22], Teifel and Manohar [23], and Mahram et al. [24] were studied. These were chosen because they are examples of asynchronous cell designs and because the authors included estimates of speed and energy consumption. These estimates are summarized in Table 2, along with those of the PQ-cell. The frequency given for the PQ-cell is the simulated pipeline speed. The PQ-cell energy consumption is from the single-cell simulation when operating at 1.3 GHz.

There are significant architectural differences between the designs; so it is difficult to make meaningful conclusions. Even so, it is interesting to see that the results are relatively close. The biggest difference is the low energy consumption of the Wong and Mahram FPGAs, but at least a factor of 2 can be attributed to the technology and supply voltage. For example, an FPGA cell described in [25] consumed 18 pJ/cycle at 250 nm and was expected to improve to 7 pJ/cycle at 180 nm. A similar improvement can be expected for the PQ-cell. Future work will need to make a more careful comparison to see what can be learned by studying the architectural variations and their ramifications.

5. Summary and Conclusions

The PQ-cell is the latest member of a set of exploratory designs for a simple reconfigurable computing element for cellular-automata-based conformal computing. A single cell includes facilities for performing logic, moving and storing information, and coordinating parallel activity. The PQ-cell is a dual-rail pulse-driven asynchronous primitive that combines stimulus and data in a single pulse that appears on one of two rails (wires). This dual-rail design eliminates any need to maintain a timing relationship between separate stimulus and data signals.

A novel feature of the PQ-cell is its quad-cell design which consists of four elementary quarters, each with a different compass orientation, that share synchronization and configuration circuitry. Each quarter includes a 1-bit storage unit and a logic unit capable of performing any one of the 16 possible functions of two bits. The P-cell, a single cell with 4-fold rotational symmetry, is also being considered and future work will include a careful comparison between these two designs.

The cells are designed to be elements of extensible cellular arrays in which communication takes place directly between neighboring cells. Accordingly, arrays of PQ-cells can be configured to form a wide variety of computational structures including cellular automata and FPGA-like circuits. Because there are no global signals or long wires, and pulse integrity is maintained by the cells, the arrays can be extended to very large sizes. Also, to make the cell configuration process extensible, it was designed to be a selective and highly parallel activity.

In addition to a functional design, this paper presented simulation results for an IC design of the PQ-cell. The design, intended for fabrication in TSMC's 0.25 μm CMOS technology, was used in simulations of basic single and multiple cell structures, including an XOR gate, a full adder, a ring oscillator, and a pipeline. The simulations were important for testing the correctness of the design and for obtaining estimates of speed and power. Comparisons with other work indicate that the PQ-cell is competitive in performance and energy consumption.

The PQ-cell design is a good step toward a reconfigurable asynchronous primitive that can serve as the elemental unit in extensible arrays for conformal computing. The next steps are to refine the design and move on to layout and fabrication of a prototype array chip. Many additional steps are needed to adequately address issues related to the performance, cost, programming, and use of systems employing these arrays. The hope is that this path will lead to an extensible material that is a superior host for computation in applications ranging from small low-power sensors to large high-throughput pattern processors.

Acknowledgments

This material is based on research sponsored by the Defense Microelectronics Activity (DMEA) under Agreement numbers H94003-06-2-0603 and H94003-07-2-0707. The United States Government is authorized to reproduce and distribute reprints for Government purposes, not withstanding any copyright notation thereon.

References

[1] J. Lyke, G. Donohoe, and S. Karna, "Reconfigurable cellular array architectures for molecular electronics," Tech. Rep. AFRL-VS-TR-2001-1039, Air Force Research Laboratory, 2001.

[2] S. Das, G. Rose, M. Ziegler, C. Picconatto, and J. Ellenbogen, "Architectures and simulations for nanoprocessor systems integrated on the molecular scale," in *Introducing Molecular Electronics*, pp. 479–512, Springer, Berlin, 2005.

[3] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, no. 5187, pp. 1021–1024, 1994.

[4] M. Sipper, "Emergence of cellular computing," *Computer*, vol. 32, no. 7, pp. 18–26, 1999.

[5] P.-A. Mudry, F. Vannel, G. Tempesti, and D. Mange, "CONFETTI: a reconfigurable hardware platform for prototyping cellular architectures," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS '07)*, pp. 1–8, Long Beach, Calif, USA, March 2007.

[6] H. Abelson, D. Allen, D. Coore et al., "Amorphous computing," *Communications of the ACM*, vol. 43, no. 5, pp. 74–82, 2000.

[7] H. Abelson, J. Beal, and G. Sussman, "Amorphous computing," Tech. Rep. MIT-CSAIL-TR-2007-030, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, Mass, USA, June, 2007.

[8] M. J. Pavicic, "Wallpaper computers: thin, flexible, extensible and R2R ready," in *Proceedings of the Flexible Electronics and Displays Conference*, pp. 2–5, Phoenix, Ariz, USA, February 2009.

[9] N. Margolus, "CAM-8: a computer architecture based on cellular automata," in *Pattern Formation and Lattice Gas Automata*, pp. 167–187, American Mathematical Society, 1996.

[10] T. Toffoli, "A pedestrian's introduction to spacetime crystallography," *IBM Journal of Research and Development*, vol. 48, no. 1, pp. 13–29, 2004.

[11] N. J. Macias and P. M. Athanas, "Application of self-configurability for autonomous, highly-localized self-regulation," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS '07)*, pp. 397–404, Edinburgh, Scotland, July 2007.

[12] F. Gruau, Y. Lhuillier, P. Reitz, and O. Temam, "BLOB computing," in *Proceedings of the Computing Frontiers Conference (CF '04)*, pp. 125–139, Ischia, Italy, April 2004.

[13] L. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*, Cambridge University Press, New York, NY, USA, 2002.

[14] M. Hoseini, C. You, and M. J. Pavicic, "A cellular automata ASIC for conformal computing," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '08)*, pp. 305–306, Las Vegas, Nev, USA, July 2008.

[15] S. Wolfram, *A New Kind of Science*, Wolfram Media, Pasadena, Calif, USA, 2002.

[16] P. Sarkar, "A brief history of cellular automata," *ACM Computing Surveys*, vol. 32, no. 1, pp. 80–107, 2000.

[17] N. Ganguly, B. Sikdar, A. Deutech, G. Canright, and P. Chaudhuri, "A Survey on Cellular Automata," February 2006, http://www.cs.unibo.it/bison/publications.

[18] S. Adachi, F. Peper, and J. Lee, "Computation by asynchronously updating cellular automata," *Journal of Statistical Physics*, vol. 114, no. 1-2, pp. 261–289, 2004.

[19] F. Peper, J. Lee, S. Adachi, and S. Mashiko, "Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers?" *Nanotechnology*, vol. 14, no. 4, pp. 469–485, 2003.

[20] R. Minnick, "A survey of microcellular tresearch," *Journal of the ACM*, vol. 14, no. 2, pp. 203–241, 1967.

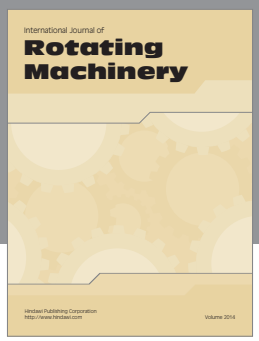
[21] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.

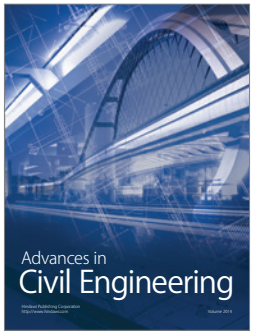
[22] C. Wong, A. Martin, and P. Thomas, "An architecture for asynchronous FPGAs," in *Proceedings of the IEEE International Conference on Field-Programmable Technology*, pp. 170–177, December 2003.

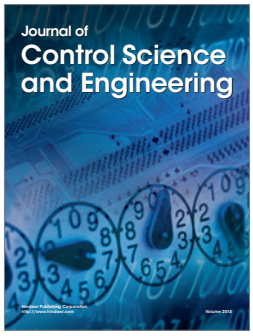
[23] J. Teifel and R. Manohar, "An asynchronous dataflow FPGA architecture," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1376–1392, 2004.

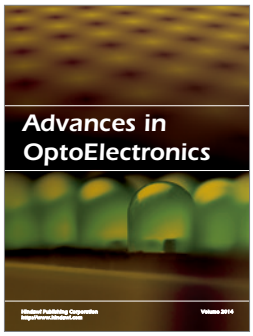
[24] A. Mahram, M. Najibi, and H. Pedram, "An asynchronous fpga logic cell implementation," in *Proceedings of the 17th ACM Great Lakes Symposium on VLSI*, pp. 176–179, Stresa-Lago Maggiore, Italy, March 2007.

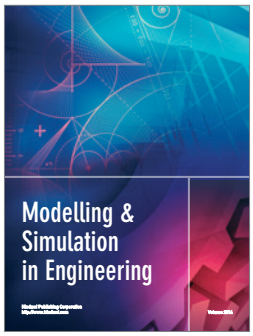
[25] J. Teifel and R. Manohar, "Highly pipelined asynchronous FPGAs," in *Proceedings of the ACM/SIGDA 12th International Symposium on Field-Programmable Gate Arrays*, pp. 133–142, Monterey, Calif, USA, February 2004.

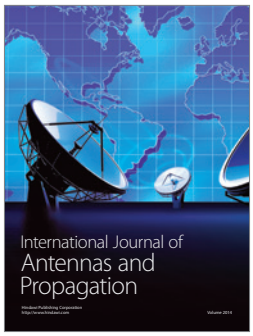


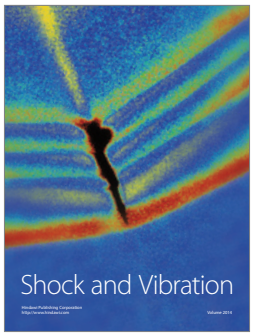


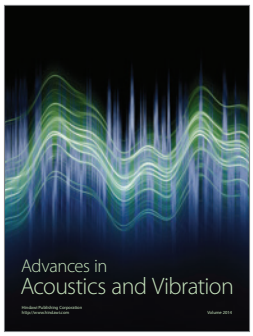












Hindawi

Submit your manuscripts at
http://www.hindawi.com