

## **Fault-Tolerance in Nanocomputers: A Cellular Array Approach\***

### **I. Summary**

Asynchronous Cellular Arrays assume a timing model according to which the state transitions of each cell take place at random times independently of other cells. An asynchronous mode of timing may benefit physical implementations of cellular arrays in time same way it benefits asynchronous circuits. Because the absence of a central clock eliminates the need to distribute its signals to all elements, circuits consume less power and dissipate less heat as well. These two advantages may be imperative with alternative technologies such as molecular electronics, quantum dot cellular automata, Rapid Single Flux Quantum (RSFQ) superconducting, which is a digital electronics technology that relies on quantum effects in superconducting materials to switch signals, instead of transistors, etc.

Nanocomputing has the important design consideration of robustness, which affects to the reliability of nano-scale device. There are two types of robustness, which consists of fault-tolerance and defect tolerance. The ability to recover the data from transient errors during computing of the system is fault tolerance. Another type is defect tolerance, which is the ability to operate flawless in the presence of permanent hardware errors that emerged in the manufacturing process.

Asynchronous Cellular Arrays is arrays in two or three dimension of identical cells. For storing the state of a cell, each side of cell (four sides) has a memory of a few bits attached to it. We can use the combination of four sides of memory to configure that cell. When the configuration is occurred, the state of that cell has been changed meaning that the cell has a transition to new state. However, there are transition rules to follow in the left hand side to the right one. If there is no transition rule matched in left hand side, the state will not be changed.

A delay insensitive circuit is a type of asynchronous circuit which performs a logic operation often within a computing processor chip. Instead of using clock signals or other

---

\* Peper, F. et al, Fault-Tolerance in Nanocomputers: A Cellular Array Approach, *IEEE Transactions on Nanotechnology*, **3**, No.1 March 2004

global control signals, the sequencing of computation in delay insensitive circuit is determined by the data flow. Typically handshake signals are used to indicate the readiness of such a circuit to accept new data (the previous computation is complete) and the delivery of such data by the requesting function. Similarly there may be output handshake signals indicating the readiness of the result and the safe delivery of the result to the next stage in a computational chain or pipeline. In a delay insensitive circuit, there is therefore no need to provide a clock signal to determine a starting time for a computation. Instead, the arrival of data to the input of a sub-circuit triggers the computation to start. Consequently, the next computation can be initiated immediately when the result of the first computation is completed.

The main advantage of such circuits is their ability to optimize processing of activities that can take arbitrary periods of time depending on the data or requested function. An example of a process with a variable time for completion would be mathematical division or recovery of data where such data might be in a cache. The cell configurations behave like the primitives according to the set of six transition rules (See in paper: table II). The fork primitive looks like a fan-out element, which produces one signal on each of its two output paths for every signal it receives from its input path. The merge primitive will generate one output with two input paths. However, the signals which are arriving from the input paths are redirected to the output path. Therefore, the input signals are generating the two consecutive output signals in sequence. The resettable modulo 2 counter (R-counter) primitive is generating two output paths (a, k) with two input paths (r, b). Two input signals on a give rise to one output signal on b. However, one input path on a and r will give rise to one output signal on k. If there two inputs on a and one input on r, two cases above can be considered. If there is only one input on a or r, output will not be come out until one more signal in input to a. The signal primitive is a propagating signal on a path. The nature of the signal primitive is that other signals on the same path may delay the signal, but never change it.

If cellular Arrays cannot work, it means they cannot make the recovery from the mistake in a transition or some bits of a memory flipping due to noise. To ensure more reliable computation, cellular arrays with fault-tolerance are implemented by using techniques from error correcting coding theory. In other words, we try to recover data from transient errors during computation. There are 3-tuples (n, m, d): 'n' is word length, 'm' is the number of code words, and 'd' is the minimum distance between any two code words. They use 3-tuple to design an efficient error correcting code for 2 bits memory. If a code has the minimum distance d, thus we can correct them up to  $(d-1)/2$  errors. In addition, if d is an even number, it can also detect  $d/2$  errors. In order to design the efficient system for fault-tolerant, if n is fixed, the value of both d and m should be as large as possible.

However, there are rule, called Plotkin bound, to follow. If  $n < 2d$  then, these should be hold for binary codes:

$$m \leq 2(d/(2d-n)) - \text{if } d \text{ is even}$$

$$m \leq 2((d+1)/(2d+1-n)) - \text{if } d \text{ is odd}$$

There is a method for making the correction of errors in the bits which is called majority voting. It is done by counting the occurrence of each of the four messages 00, 01, 10, 11 among the bit pairs, and setting all bits according to the bit pair with the highest count. Therefore, (5, 4, 3) code allows the correction of one arbitrary bit error. Likewise, (14, 4, 7) repetition code allows to correct up to three arbitrary bit errors, as well as (14, 4, 9) code can allow to correct up to four arbitrary bit errors. According to resolving by the majority voting for four, five, and six errors, they reveal that about 65% of the cases with four erroneous bits, about 52% of the cases with five bits of error, and about 29% of the cases with six bits of error can be resolved.

Another method to correct the errors is called distance comparison. This method calculates the distances to the table entries, and selects the code word with the smallest distance as the most likely. This method has reduced the complexity because of the small number of code words, and it comes with the bonus that in many cases five and some cases even six errors can be corrected. They've found that about 75% of the cases with five erroneous bits and about 20% of the cases with six erroneous bits can be resolved. However, the choice for a particular word length is determined, apart from how big a memory may be, by the probability of errors occurring and their type.

This paper leaves open the challenge of configuring the inherently homogeneous cellular hardware into particular delay insensitive circuit. In order to implement configurable asynchronous cellular array, they need more bits of memories to carry configuration information. Likewise, to implement error correcting codes, we need more code words than four which is assumed in this paper.

## **II. Discussion**

Nanotechnology is a vastly growing field which is fundamentally different from what we have been working on. In nanoelectronics, there is a drastic shift from working with CMOS and a top-down approach to setting a blue print to molecular and chemical elements that have to be self-assembled. The emergence of new devices and circuits has led to the need to have tools and architectures to handle the new concepts. One such attempt that caters to many of the characteristics of nano-components is the asynchronous cellular array approach. The key element is that it can handle fault tolerance and also has

a regular structure of locally connected elements that will make it easier to integrate our architecture to devices that are self-organized.

Even though the discussion was limited due to time constraints, here are some of the issues that we raised in the discussion of asynchronous cellular arrays. The first point raised was a basic question of how a circuit would work with a clock. How can you make an asynchronous circuit work? An asynchronous circuit is a circuit in which the parts are largely autonomous. They are not governed by a clock circuit, but instead only wait for the signals that indicate completion of instructions and operations. This is different from a synchronous circuit which operates according to clock timing signals. ♦

Some of the benefits due to non-existence of a clock that were discussed include lower power consumption due to the fact that no logic element ever transitions unless it is performing useful computation, the elimination of the need to route clock signal everywhere there is a logic element, and some speedup in operation due to restriction of speed from the clock period. The class agreed that it is beneficial to have a design where we would not have to deal with the issues of clock routing and power issues in the nano-regime as these requirements pose a considerable problem to the development of nano circuits which would need a lot more clock routing and deal with power due to an increase in the number of logic elements.

On the flip side, the class agreed that even though the paper mentions a couple of nano technologies (molecular electronics, quantum dot cellular automata, RSFQ) it was not quite obvious how these technologies would be able to adopt the asynchronous cellular array architecture. The closest nano application that would benefit from this architecture would be the QCA. QCA seems to have the idea of a regular structure that can represent a few states which when cascaded would lead to a circuit. At first, it would seem that this approach would be a perfect fit. However, it was pointed out that QCA is highly dependent on the concept of timing. Timing is crucial for determining the directionality of the circuit. Thus, it would seem to need major changes to the formulation to come up with an asynchronous QCA that we can then apply the architecture in question.

Another interesting aspect of the approach that is proposed is the idea that it is able to handle transient faults (fault tolerance). The actual asynchronous cellular array is not inherently fault tolerant, but they used error correction codes to make the architecture fault tolerant. The basic idea they use it to encode each bit of all the memories associated

---

♦ “Asynchronous Circuits”, *Wikipedia*, [www.wikipedia.com](http://www.wikipedia.com).

with a cell with a codeword. This would be able to handle transient faults that might occur in the cellular array.

In the consideration of using this architecture for nanotechnology, a question was raised into how worried we need to be about transient faults when it seems that defects are more abundant. It was pointed out that with increasingly smaller sizes (nano components) and the corresponding increase in the number of elements that would make up a nano circuit, transient faults are a major design problem that we need to deal with. Furthermore, these nano devices are going to be running at voltage levels that are a lot smaller than we are dealing with right now. We can see clearly that in nanotechnology transient faults would not only be more likely, but would also have an easier time to cause faults in our computation. This is something that needs to be researched into as most people seem to be focusing only on the tolerance of manufacturing defects.

In general, even though it is a bit hard to start thinking in terms of an asynchronous logic and it was not clear how some of the nanotechnologies would adopt to such an architecture, it was enlightening to see the possibilities for research in advancing nanoelectronics by developing not just nano devices, but also architecture and fault tolerance methods. We also learnt that fault tolerance is not something that has to be independently implemented, but can be meshed in with an appropriate architecture.