A Clock-fault Tolerant Architecture and Circuit for Reliable Nanoelectronics System

Woon Tiong Ang, Hui Fei Rao, Changhong Yu*, Jilin Liu*, I-Chyn Wey*, An-You (Andy) Wu*, Hong Zhao, and Jie Chen

Electrical and Computer Engineering Department, University of Alberta, Edmonton, Canada
*Electrical Engineering Department, Zhejiang University, China
+Graduate Institute of Electronics Engineering, and Department of Electrical Engineering, National Taiwan University, Taiwan

Corresponding Author: <woon@ualberta.ca>

Abstract

Due to discrepancies in manufacturing process and the probabilistic nature of quantum mechanical phenomenon, nanoelectronic devices cannot be made as reliable as current microelectronic devices. As a result, fault-tolerant architectures are a prerequisite to building reliable electronic systems from these unreliable nanoelectronic devices. One important design aspect of nanoelectronic architecture that demands attentive consideration is clock generation and distribution. Various defects and interference such as doping discrepancies, supply noise and cross-talks could lead to clock irregularity and malformed clock signals, thus resulting in faulty operations of sequential circuits. Generally, these errors are not readily amenable to efficient correction using error-correcting codes known to date.

In this paper, we propose a novel fault-tolerant architecture for a parallel computation structure. The fault-tolerance capabilities built into this architecture allow for effective remedy against the deleterious effects of random clock abnormality and reduce the probability of computational errors. Central to the operation of the proposed fault-tolerant architecture is a novel clock-fault detection circuitry. In order to illustrate the fault-tolerance capability rendered by the detection circuitry, an error probability analysis is performed. Finally, a prototype CMOS design of the circuitry is demonstrated.

Section 3 features the proposed clock-fault detection circuit. A CMOS prototype design of the circuitry is demonstrated. Our simulation shows that with only a two-fold increase in hardware counts, the proposed architecture can gain significant fault-tolerance capability.

Keywords: reliability, reliable electronics, nanoelectronics, fault tolerance

1. Introduction

Hardware defect is an increasingly significant issue in the design and manufacture of modern integrated circuits. As the number of transistors on a chip increases to the giga scale, the economic cost required to maintain a defect-free hardware becomes astronomically prohibitive. Furthermore, the future of integrated circuits sees the use of novel technologies including single-electron tunneling (SET) [1] device, resonant tunneling transistors (RTT) [2], carbon nanotubes, and quantum cellular automata (QCA) [3]. Nano-electronics devices such as these, due to their quantum mechanical nature, cannot be expected to operate with perfect predictability at all times. Moreover, the environment in which the circuits are operated may further contribute to the unreliability of the circuit via the addition of noise and various forms of interference. Therefore, future nanoelectronic circuits should be designed to operate reliably in the presence of permanent hardware defects and transient signal faults.

Various fault-tolerance techniques have been studied by physicists and computer scientists, the most notable of which includes Von Neumann [4]. Techniques such as N-tuple modular redundancy (NMR) [4], Triple Modular Redundancy (TMR) [5][6], NAND multiplexing [4] and others based on reconfigurable hardware [7][9] have been proposed in the literature. Nevertheless, little in-depth analysis has been devoted to the analysis or design of clock-fault handling structure suitable for nanoelectronics system. Error detection or correction for clock signal can be achieved using clock generators based on “quadricorrelator” [10], phase-locked loop (PLL) [11] and Costas Loop[12] architectures but these implementations are generally too hardware-demanding or area-consuming for liberal employment on an intrinsically error-prone nanoscale chip.

This paper proposes a fault-tolerant architecture for a computation unit suitable for employment in a reconfigurable, massively-parallel, and high-throughput computation system. Also proposed is a hardware-efficient clock-fault detection circuit which is central to the operation of the suggested architecture. The paper is organized as follows. In section 2, the proposed architecture of a clock-fault tolerant computation unit is presented. Section 3 features the proposed clock-fault detection circuit. A CMOS prototype design of the circuitry is demonstrated.

2. Architecture of a Clock-fault Tolerant Computation Unit

2.1 Proposed architecture of fault-tolerant computation unit

![Figure 1: Proposed architecture of a fault-tolerant computation unit employed in a 3 by 3 reconfigurable array.](image)

A generic reconfigurable system is shown in figure 1. This system comprises an arbitrary $m$ by $n$ array of our proposed fault-tolerant computation units but for illustration, we only display a 3 by 3...
array. A common clock signal (not shown in the figure) is delivered to the computation units for the purpose of synchronization. The computation units are densely populated on the nanoelectronic fabric to enable fast data transfer between the computation cells. Close proximity between the cells also reduces clock skews that would otherwise occur due to long interconnects. Here we assume, for simplicity, that the reconfigurable system is sufficiently compact so that no excursion to distributed system analysis is necessary. This assumption necessarily limits the clock frequency at which the system may operate with. However, given the size expected of nanoscale circuits, the architecture should work at sufficiently high frequencies well into the gigahertz domain. In the reconfigurable array presented above, all the computation units derive their clock signals from a common source. Hence, a glitch in the clock signal would affect all the computation units concurrently.

2.2 Data Processing Mechanisms
Data for computation are read in from the memory or the register module as shown in Figure 1. Depending on the size of the computational grid, multiple bytes of data could be read and processed concurrently by multiple Arithmetic Logic Units (ALUs) for high-throughput parallel operations. The routing switches can be programmed in advance or during computation to select the optimum ensemble of computation units for the current task. A dedicated controller unit (not shown in the figure) can be used for this purpose.

2.3 Mechanisms for Hardware Defect-tolerance
In anticipation of potential hardware defects, a built-in self-test (BIST) routine can be executed every time the processor module is powered up to single out defective units (computation units or routing switch). Then, routing can be done by the suggested controller unit such that defective units are precluded from the ensemble of units actively participating in the computation process. This idea of defect avoidance via reconfiguration of the hardware has been proposed in the literature [8][9]. In addition to routing around defective units, the controller unit can also route dynamically for increased computational speed or decreased power consumption. The former requires more units to be switched on to maximize parallelism. The latter would switch off some of the units to reduce power consumption.

2.4 Mechanisms for Transient Fault-tolerance
Due to the probabilistic nature of quantum mechanics and the ubiquity of noise and interference, transient errors may occur anytime during the operation of the system. In a sequential logic, clock fault is a dominant issue affecting the speed at which a computation is executed. In a cascaded configuration in which the result of a stage is passed on to the next stage in a pipeline, a clock fault could invalidate the final computation result. The computation would have to be repeated, thus reducing the speed of the computation significantly.

In an effort to combat the problem of clock fault, a straightforward method is to employ redundancies-based method such as the TMR [5][6], which is typically used in fault-tolerant combinational circuits. This, however, is not an optimized solution for clock delivery due to the thickness of interconnects, which precludes them from area-efficient duplication. Note that the size of interconnect is expected to be much larger (for nanometer CMOS at least) than the size of future nanoelectronic devices. This is because wire cannot be shrunk at the rate at which devices can due to the interconnects’ resistive property - thin wire has a high electrical resistance and hence dissipates more power in the form of heat. This could lead to thermal heating in a densely packed nanoelectronic fabric.

For the reasons above, we consider a single-wire (no redundancy) clock delivery system. In order to improve the clock’s reliability, a clock-fault detection circuitry is built into each computational unit to detect transient clock errors. For illustration, an ensemble of three computation units is shown in Figure 2. Note that the computation unit receives its input data from two ‘upstream’ computation units (UCUs). When no clock error occurs, the data from the UCUs can be processed immediately by the downstream computation unit (DCU). In the event of a clock error being detected by any or both of the UCUs, however, a signal is sent to the DCU before the arrival of the next clock cycle. Legitimate data from the UCUs, if there is any, will be stored in a buffer built into the DCU. Invalid data will be discarded as soon as it reaches the DCU, which in turn will stall until it receives a bit of legitimate data from each of the two UCUs. Thus, a pipelined, highly-parallel computation can be performed in a reliable manner.

2.5 Analysis of Clock Signals
For the clock-fault detection circuitry to detect a faulty clock signal, it must first be able to distinguish between a faulty, and hence unacceptable signal, and an acceptable signal. Nevertheless, clock signals can take on a continuum of shapes, making the distinction difficult. In order to circumvent this problem, we define a faulty clock signal as one that fails to trigger a digital logic block. Faulty clock signal may appears in the form of overly thin pulse, missing pulse, two pulses being merged, insufficient pulse voltage etc. For the purpose of detection, we strengthen (digitize to ‘0’ and ‘1’) all pulses to eliminate the case of insufficient pulse voltage. Then we look out for pulses whose widths deviate from the nominal value by a predetermined value. For instance, an ideal clock waveform with pulse width \( w \), and period \( 2w \), as shown in Figure 3, can be labeled as being incorrect if its pulse width \( w < 0.5w \) or \( w > 1.5w \). Otherwise, if the pulse width falls within a certain limit from the nominal value, e.g. \( 0.5w < w < 1.5w \), the clock is labeled ‘correct’. This definition allows us to detect presumably abnormal clock waveforms and hence make informed decision about the odds of invalid computation. Note that the range \( 0.5w < w < 1.5w \) was arbitrarily set and can be resized as desired to suit a specific circuit implementation. Of course, this methodology does not allow us to detect faulty clock signal with absolute accuracy. In any case, it...
provides us a means for evaluating clock integrity with possibly very high level of confidence.

Figure 3: Clock waveforms showing normal, short and long pulse.

Figure 3 also provides a graphical aid for the visualization of the fault-detection mechanism. A contaminated received clock is contrasted against an ideal clock for comparison. When the received clock pulse width is shorter than a predetermined acceptable minimum (termed ‘short pulse’), it will be detected and a warning signal will be sent. In our proposed design, a decision will be made between time $t_1$ and $t_2$. If the received clock pulse’s width is longer than a predetermined acceptable maximum (termed ‘long pulse’), it will be detected between time $t_3$ and $t_4$ and a warning signal is to be sent.

2.6 Evaluation of Clock-fault Probability

We construct our fault model using a clock with stable frequency but with a pulse width $W$ that varies with a normal distribution with mean $W_s$ and standard deviation $\sigma$. The PDF of the distribution is hence given by

$$f(w) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(\frac{(w-W_s)^2}{2\sigma^2}\right).$$  (1)

Thus, the CDF can be derived to be

$$P(W \leq w) = \frac{1}{2} \left[1 + \text{erf}\left(\frac{w-W_s}{\sigma\sqrt{2}}\right)\right].$$  (2)

where $W$ is a random variable for the pulse width $w$.

The probability of a clock error $\alpha$ can then be calculated using the following formulae

$$P_{\text{correct}} = P(0.5W_s < w < 1.5W_s) = \frac{1}{2} \left[\text{erf}\left(\frac{1.5W_s-W_s}{\sigma\sqrt{2}}\right) - \text{erf}\left(\frac{0.5W_s-W_s}{\sigma\sqrt{2}}\right)\right]$$  (3)

and

$$\alpha = 1 - P_{\text{correct}}$$  (4)

Having established a methodology to distinguish between faulty and non-faulty clock signals, and to map clock distribution to clock error probabilities, we set out to evaluate the performance of our proposed architecture.

2.7 Quantitative Evaluation of the Proposed Design

Before simulating the fault-tolerance capability of the proposed architecture, we need to recognize that any of the modules used in the construction of the system could be in error itself, including the fault-detection circuitry. We then begin with a few assumptions.

We assume perfect memory/register and DEMUX modules as these do not form the core of our proposed architecture. We assume that each core component in our proposed architecture fails with the following probability:

<table>
<thead>
<tr>
<th>Component</th>
<th>Error probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>Data Bus</td>
<td>$\beta$</td>
</tr>
<tr>
<td>ALU</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>Fault-detector circuitry</td>
<td>$\gamma$</td>
</tr>
</tbody>
</table>

Figure 4: Error probabilities of core system components.

In what follows, we begin to derive a closed-form formula to evaluate the error probability of an ensemble of interconnected computation units $P_{\text{ENSEMBLE}}$ as a function of $\alpha$, $\beta$, $\epsilon$ and $\gamma$. To this end, we map all the possible fault scenarios in a truth table as shown in Figure 5. First of all, an explanation of the symbols used in the table is in order. The symbols and their corresponding explanations are tabularized in Figure 6.

<table>
<thead>
<tr>
<th>Clock State</th>
<th>Data Bus State</th>
<th>ALU State</th>
<th>Fault-Detector State</th>
<th>Cell Error?</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Figure 5: Error analysis table. The symbols used here are explained in Figure 6.

From the truth table in Figure 5, we can compute:

1. $P_0$: The probability that a computation unit produces a correct result during the current clock cycle.
2. $P_1$: The probability that a computation unit fails to yield a correct result during the current clock cycle.
3. $P_U$: The probability that a computation is to be extended to upcoming clock cycle(s) as a result of the clock-fault detection circuitry issuing a warning.

From the truth table, we can see that

$$P_0 = (1-\alpha)(1-\beta)(1-\epsilon)(1-\gamma)$$  (5)

which follows from the case where all the components operate without faults. $P_U$ can be obtained by observing that a ‘U’ occurs when either one (and only one) of the clock or the clock-fault detection circuitry is faulty. Hence we have

$$P_U = (1-\alpha)\gamma + (1-\gamma)\alpha$$  (6)

Thence it is simple to see that

$$P_1 = 1 - P_U - P_0$$  (7)

We can then compute the error probability of a computation unit
over an arbitrary number of clock cycles. This is given by
\[ P_{CU} = P_1 \sum_{i=0}^{\infty} P_1^i = \frac{P_1}{(1-P_U)} \]  
which can also be written as
\[ P_{CU} = \alpha \frac{P_1}{(1-P_U)} + (1-\alpha) \frac{P_1}{(1-P_U)} \]  
where the first term in (9) corresponds to error probability that stems from clock error \( P_{CU_{-CLK}} \) while the second term represents the error probability due to non-clock error \( P_{CU_{-NONCLK}} \). Note that for simplicity, we assume the error probability due to clock \( P_{CU_{-CLK}} \) is perfectly correlated between all computation units because they share the same clock signals. The error probability of an ensemble of \( N \) computation units can therefore be derived to be
\[ P_{ENSEMBLE} = (1-\alpha)^N \sum_{i=1}^{N} \frac{P_{CU{_{-NONCLK}}}}{i} (1-P_{CU_{-NONCLK}})^{N-i} \]  
\[ + \alpha^N \sum_{i=1}^{N} \frac{P_{CU_{-CLK}}}{i} (1-P_{CU_{-CLK}})^{N-i} \]
where \( P_{CU_{-CLK}} = \alpha \frac{P_1}{(1-P_U)} \) and \( P_{CU_{-NONCLK}} = (1-\alpha) \frac{P_1}{(1-P_U)} \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error occurring during the current clock cycle. Computation complete.</td>
</tr>
<tr>
<td>1</td>
<td>An error occurs during the current clock cycle. Computation complete</td>
</tr>
<tr>
<td>U</td>
<td>No error occurs during the current clock cycle. Computation extended to upcoming clock cycle(s).</td>
</tr>
<tr>
<td>A</td>
<td>Fault-detector falsely detects an error. Computation is to be extended to upcoming clock cycle(s), thus preventing an error, if there is one, from occurring during the current cycle. In the event that no error has occurred, an unnecessary delay has been incurred.</td>
</tr>
<tr>
<td>B</td>
<td>Fault-detector correctly detects an error. Computation is extended to upcoming cycle(s), thus preventing an error from occurring.</td>
</tr>
</tbody>
</table>

Figure 6: Explanations of symbols used in Figure 5.

To demonstrate the effectiveness of the clock-fault detection mechanism, we simulate the performance of an ensemble of three computation units, each of which executes an addition operation on two one-bit inputs (illustrated in figure 7). The simulation is conducted for the following cases:
(a) Without clock-fault detection mechanism
(b) With detection mechanism ( \( \gamma = 0.1, 0.2, 0.3, 0.4, \) and \( 0.5 \))

For each of the above cases, we set \( \beta = 0 \) and \( \varepsilon = 0 \).

Our simulation results are shown in Figure 8. It is found that even with a fairly unreliable fault-detection circuitry ( \( \gamma = 0.5 \) ), the ensemble error probability can be significantly reduced for all value of \( \alpha \). When clock error probability is high (around 0.5), a much lower error probability (around 0.1) is attainable by the ensemble.

Figure 8: Simulation results - ensemble error probability versus clock error probability \( \alpha \) for different value of \( \gamma \).

3. Design of the Clock-fault Detecting Circuit
3.1 Proposed Circuitry
The key idea behind our proposed circuit is based on charge accumulation and detection over time (see Figure 9). The periodic charging and discharging of the accumulator keeps its voltage level within a pre-designated threshold under normal circumstances. In the event that the clock signal deviates considerably from its prescribed characteristics, i.e. if the pulse becomes too long or too short, the voltage across the charge accumulator module will cross the threshold voltage of the level detector, thus triggering the transmission of a warning signal. In this way, potentially fault-inducing clock signals can be detected and precautionary measures can be taken to avoid a miscomputation.

Figure 9: Block diagram of the proposed clock-fault detection circuitry.

3.2 CMOS Prototype of the Proposed Circuitry
In order to present a concrete realization of the clock-fault detection circuitry, we map our proposed idea onto silicon in CMOS 0.18um technology. As shown in Figure 10, the design comprises two complementary circuit sections i.e. the top and the bottom sections. Upon receiving a clock input, the clock signal which could take on a continuum of analog shapes (see figure 11 for illustration) is digitized to either a high (supply voltage) or a low (ground) at the front end of the proposed circuit. The digitized signal then charges and discharges the capacitors at nodes \( C1 \) and \( C2 \) in an alternating fashion. The top Level Detector outputs a
‘high’ at $V_A$ when the threshold voltage of the NMOS transistor $V_{th}$ is exceeded at node $C1$, i.e. when $V_{C1} > V_{th}$. This situation occurs after the capacitor at node $C1$ has been charged ‘overtime’, i.e. for a period greater than $1.5w_z$. In our prototype design, a 500MHz clock is used. This translates to a nominal clock pulse width $w_z$ of $In$. The bottom section of the circuit, owing to the inverter after the clock generator, acts to complement the

detect both short and long pulses and hence produces a warning signal $V_{out}$ in the occurrence of such abnormalities.

In our design, a pull-down transistor is used to ensure a quick drop of the capacitor voltage at the end of a pulse to avoid accumulation of charge which will affect the next charging cycle(s). From Figure 10, we can see that the design only requires 28 transistors and 2 capacitors.

![Figure 10: A CMOS design of the clock-fault detection circuit using 28 transistors and 2 capacitors. A clock with a 500MHz frequency is used in our simulation.](image)

Figure 10: A CMOS design of the clock-fault detection circuit using 28 transistors and 2 capacitors. A clock with a 500MHz frequency is used in our simulation.

top section - it outputs a ‘high’ at $V_B$ when a pulse shorter than an acceptable reference is detected. This is because in this case, $V_{C2}$ charges up to a point where $V_{C2} > V_{th}$, thereby triggering a ‘high’ at $V_B$ (see figure 14). During normal operations, both $V_{C1}$ and $V_{C2}$ stay below $V_{th}$. $V_A$ and $V_B$ therefore stay ‘low’. Using an OR logic gate at the final stage, the output $V_{out}$ asserts a ‘high’ whenever $V_A$ or $V_B$ is ‘high’. In this way, the circuit is able to

![Figure 11: Waveform at various nodes of the proposed clock-fault detection circuit. Under normal clock condition (first cycle), VC1 and VC2 charge and discharge periodically without reaching the threshold (approximately 600mV). VA and VB stay low. Vout also stays low as a result. When a long pulse is present (second cycle), VC1 crosses $V_{th}$ after 1.5 $w_z$ has elapsed. Hence, VA asserts a warning pulse. The presence of a short pulse (third cycle) gives rise to a long pulse in the inverted discretized clock. 1.5 $w_z$ after this long pulse has set in, VC2 crosses the threshold $V_{th}$, thus triggering a warning pulse at VB.](image)

Figure 11: Waveform at various nodes of the proposed clock-fault detection circuit. Under normal clock condition (first cycle), VC1 and VC2 charge and discharge periodically without reaching the threshold (approximately 600mV). VA and VB stay low. Vout also stays low as a result. When a long pulse is present (second cycle), VC1 crosses $V_{th}$ after 1.5 $w_z$ has elapsed. Hence, VA asserts a warning pulse. The presence of a short pulse (third cycle) gives rise to a long pulse in the inverted discretized clock. 1.5 $w_z$ after this long pulse has set in, VC2 crosses the threshold $V_{th}$, thus triggering a warning pulse at VB.

This component count is roughly the same as that required to construct a 2-bit Full Adder. In addition, the value of the capacitances C1 and C2 required are very small (a small fraction of a pF) and hence can be easily accommodated by the input capacitance of the succeeding inverter stage. Therefore, the proposed circuitry imposes a redundancy factor of approximately two for the implementation of a reliable sequential adder circuit. When used in conjunction with more complex circuits such as the ALU, as it was intended for in the proposed architecture of Figure 1, a lesser redundancy factor is necessary. The results of our
simulation using different pulse widths are shown in Figure 12, 13 and 14.

4. Conclusion
Due to high fault and defect rates in nanoelectronic devices, reliable clock distribution is an important issue in the design of future sequential logic circuits. In this paper, we propose a fault-tolerant nanoelectronic architecture for a parallel computation system. A clock-fault detection circuit is also proposed which is central to the operation of the proposed architecture. It is found from simulation that the employment of the detection circuitry can significantly reduce computational errors while incurring only a minimal hardware overhead.

5. Acknowledgements
The authors would like to acknowledge support from the Discovery Grant of Natural Sciences and Engineering Research Council of Canada.

6. References