Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

CONCURRENT ERROR DETECTION IN SELF-TIMED VLSI

D. A. Rennels
H. Kim

April 1993
CSD-930013

Concurrent Error Detection in Self-Timed VLSI *

David A. Rennels and Hyeongil Kim
Computer Science Department
6291 Boelter Hall
University of California
Los Angeles, CA 90024
Phone:(310)825-4033 FAX:(310)825-2273
E-mail:rennels@cs.ucla.edu

Abstract

This paper examines architectural techniques for providing concurrent error detection in self-timed VLSI pipelines. Signal pairs from DCVSL self-timed logic are compared with a checker that is composed of a tree of dual-rail (morphic) comparators to detect errors and signal completion. An efficient implementation is shown that compares favorably in speed and area with conventional completion signal generators. The control of the pipeline is then examined, and techniques are described that detect errors in the C-gates and other circuits that provide handshaking control. Based on these studies we have concluded that self-timed logic offers considerable fault-tolerance potential due to its built-in redundancy that can be effectively exploited for error checking.

Keywords: self-checking, self-exercising, self-timed, concurrent error detection, VLSI circuits.

*This research was sponsored by the Office of Naval Research under grant (N00014-91-J-1009).

1 Introduction

There has been a great deal of interest lately in asynchronous logic design in the VLSI community [Mart89, Jaco90]. Self-timed logic is typically more complex than synchronous logic with similar functionality, but it offers potential advantages. These include higher speed, the avoidance of clock-skew in large chips, better layout topology, and the ability to function correctly with slow components. In addition self-timed logic contains redundancy for signalling completion that can be used for testing and error detection. One of the reasons Muller developed early self-timed designs was to provide error detection because high error rates were expected in the early hardware of the ILLIAC machine.

A great deal of research has been conducted into the testability of asynchronous Differential Cascode Voltage Switch logic. Fault modeling and simulation of DCVS circuits is discussed in [Barz85] and the DCVS circuits provide on-line testability with their complementary outputs [Mont85]. In [Jha89] testability of DCVS EX-OR gate and DCVS parity trees is analyzed. Methods for testing DCVS one-count generators are given in [Jha90] and also discussed the testing of DCVS full and half-adders. In [Taka90] easily testable DCVS multipliers are presented in which all detectable stuck-at, stuck-on and stuck-open faults are detected with 6 test vectors. It is shown that how concurrent testing of DCVS circuits can be performed under a single transistor fault and the impact of multiple faults on DCVS circuits is examined [Kano90]. A technique for designing self-checking circuits using DCVS logic was presented in [Kano92].

We began this study to explore the feasibility of using asynchronous logic in the interface/controller chip of the self-checking self-exercising memory system previously described at FTCS [Renn91]. The synchronous nature of that design caused what we felt were unnecessary delays, so we decided to examine an asynchronous design as an alternative. A key requirement of the design was concurrent error detection. Thus our approach has been from an architectural viewpoint.

The Starting Point – Synchronous Circuits with Concurrent Error Detection

A common way to make synchronous hardware systems with concurrent error detection is to duplicate the circuit modules, run them with the same clock, and compare their outputs. Duplication and comparison has become widely accepted and is supported in chip sets by commercial manufacturers (e.g. Intel) and DoD-supported projects such as the GVSC and RH-32 processors. One way to do this in a self-checking fashion is to invert the outputs of one module to obtain morphic (1,0 or 0,1) pairs and compare its outputs with the corresponding outputs of the other module using a tree of self-checking comparators of the type introduced by Carter et. al. many years ago [Cart68, Sedm80]. These checkers are self-checking with respect to stuck-at faults, and will, in most cases detect transient errors that cause the

coding to be incorrect at clock transitions. They may fail under Byzantine conditions where noise or marginal circuit conditions cause the checker to see correct coding while the circuit receiving the output data sees an incorrect value.

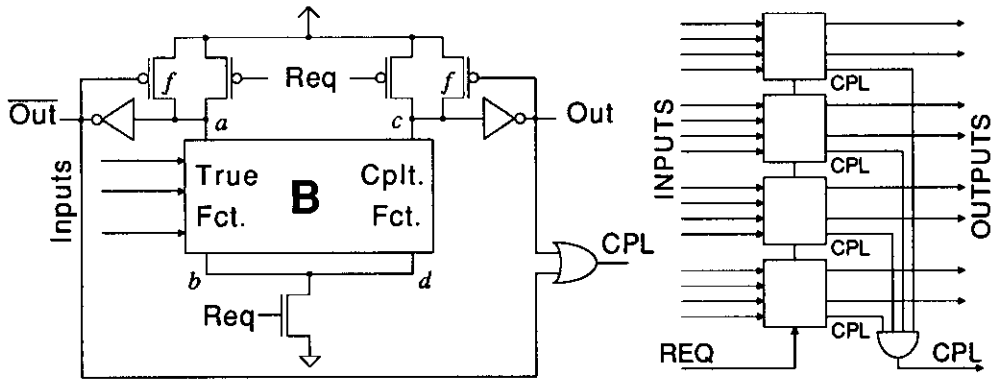
When one examines conventional duplicated synchronous systems, the cost of concurrent error detection is a doubling of active circuits plus the addition of comparators. The overhead of error detection in asynchronous designs should be similar to the synchronous case and it is already an integral part of the design. This is discussed below.

The Analogous Asynchronous Design Style – Differential Cascode Voltage Switch Logic (DCVSL)

An asynchronous design requires redundant encoding that can provide completion information as part of the logic signals. A logic module is held in an initial condition until a completion signal arrives from a previous module indicating that its inputs are ready. Then it is started, and when its outputs indicate completion, other modules may be started in turn. In general this requires a form of encoding that allows the receiver to verify that the data is ready. Also, the checker that it uses to determine completion much be glitch-free, i.e., it can have no intermediate states during transitions that raise a false completion signal before the inputs are set up.

One way to do this in self-timed designs is to use a form of 1-out-of-2 coding [Cart72]. Output signals from various logic modules are sent as a set of two-wire pairs, taking on the values 0,0 before starting, and 0,1 or 1,0 after completion. Such logic can be implemented using differential cascode voltage switch logic (DCVSL). A typical DCVSL gate is shown in Figure 1(a). It is a differential precharged form of logic. When the *Req* (request) signal is low, the PMOS pull-up transistors precharge points a and c to Vdd. At this time the circuit is in an initial state, and its outputs are 0,0. The circuit block **B** contains two complementary functions. One pulls down, and the other is an open circuit. When the input signals are ready, *Req* is raised, the pull ups are turned off, the NMOS pull down transistor connects points b and d to ground, and the circuit computes an output value. The side that forms a closed circuit forms a zero and the side that remains an open circuit remains precharged. The outputs, driven by inverters, go from 0,0 to either 1,0 or 0,1 and the completion signal *CPL* is generated from either a logical OR or the exclusive OR of the outputs. As in the case of duplex self-checking synchronous circuits, the functions are duplicated in true and complement form, but here the state 0,0 on a signal pair is a valid setup signal, and the arrival of complementary values signal completion.

It is intuitively obvious that DCVSL can provide a degree of error detection. Consider a single DCVSL circuit (Figure 1(a)). The circuit block **B** can be designed so that a fault or error will only affect one of the two sides (true or complement) and therefore only affect one output [Barz85]. However, the effects of faults and errors can be quite complex due to timing



(a) A Single Circuit (b) A Multi-Module Circuit

Figure 1: Differential Cascode Voltage Switch Logic

considerations. For example, a transient error could cause multiple completion signals, or a data line could change to an incorrect state in the time delay while a completion signal is being generated. Since completion signals that start the next computation step are generated as a combinational function of a large number of data signals the probability of transient and Byzantine errors is probably greater than synchronous circuits where special low-impedance clock drivers are used.

In order to develop a fault-tolerance strategy it is useful to move to a higher level and examine how a group of DCVSL functional blocks (i.e., combinational circuits) are combined into a pipeline.

2 Fault-Tolerance in Cascaded Self-Timed Circuits

As shown in Figure 2, several self-timed blocks can be cascaded to form a pipelined structure, in which self-timed combinational circuits, designated functional blocks (FB) are denoted as $A, B, C...$ and Interconnection/Synchronization Circuits(ISC) are $1, 2, 3...$ Each ISC contains a register for holding data between stages. For example, the output of FB A is latched at the register of ISC 1 when its completion signal Rin_1 goes high and it can be used by FB B as an input when $Rout_1$ goes high. When $Rout_1$ goes high, the FB B starts evaluation with the input and generates a completion signal and an output. The completion signal is used as Rin_2 of ISC 2. $Aout_2$ of ISC 2 is the same signal as Ain_1 of ISC 1.

The transition graph of signals in Figure 2 is shown in Figure 3 which implements a full-handshake between function blocks as explained in [Meng91]. This is the synchronizing function performed by the ISC. Both the positive and negative values of the control signals

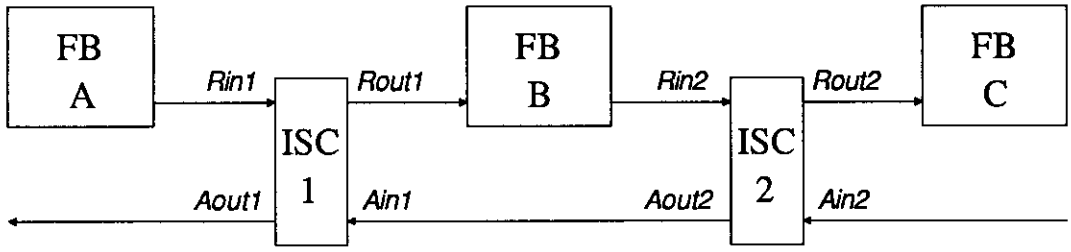


Figure 2: Pipelined Blocks

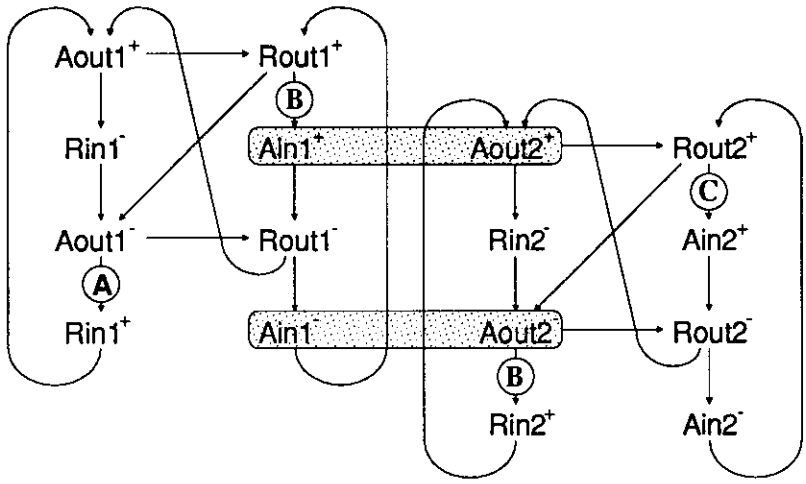


Figure 3: Transition Graph Derived from Full-Handshake

are shown by the superscript + and −. Arrows show signal conditions that must be true before the following transition is allowed to proceed. A careful examination of the graph shows that this provides the appropriate interlocking so that a module on the right has to complete before the module on the left is allowed to take the next computational step. The $Rout^-$ to $Rout^+$ step then provides the reset to pull up the DCVSL functional block before the next computation is started.

Figure 4 shows a typical timing diagram of signals associated with an ISC, (e.g., ISC 1). Arrows in the diagram indicate causes that must proceed events.

3 Implementing Concurrent Error Detection

In order to achieve concurrent error detection, three conditions are needed (1) the errors in the functional blocks (FB) must map to uncoded 1-out-of-2 pairs, (2) checking must be provided for detecting these errors, and (3) control signal errors must be detected either by

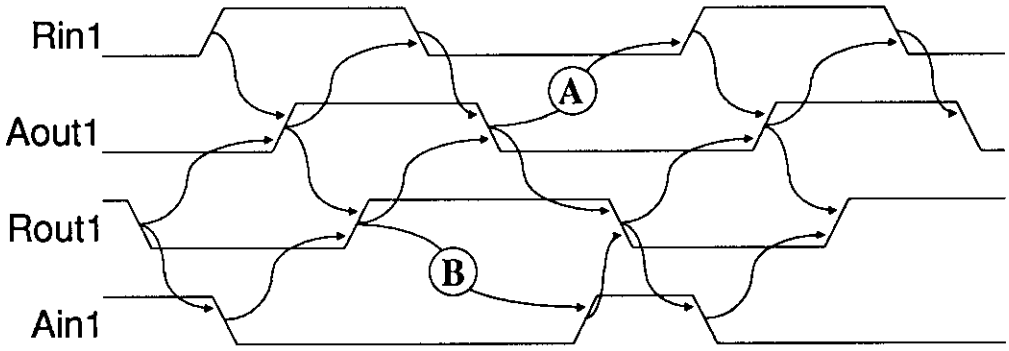


Figure 4: Timing Diagram of Full-Handshake

their causing detectable errors in the data and/or by the use of monitoring circuits (e.g. time-out counter) in the ISCs. We will first look at error detection in the ISCs.

3.1 Error Detection in Synchronization and Control

In the circuits presented by Meng, the register in the ISC (that holds data between functional blocks) is positive edge-triggered, and only true DCVSL outputs are transmitted from the previous block. Since the DCVSL needs true and complement data lines at the input, the register provides true and complement outputs. Latching completion signal can be generated by comparing input and output signals of the register, i.e. if output signals are the same as the input signals, then latch completion is assumed. This completion signal is used as *Aout* and the request signal enables the following functional block.

There are a number of ways that a transient error or permanent fault can go undetected with this circuit. As examples, a clock to a flip flop can fail, leaving old data in a latch, or wrong data may be loaded when a transient occurs on the load signal in Figure 5(a). These errors can not be detected, since the register has true and complement outputs. Similarly, errors in the C gates and their associated logic can produce undetectable errors.

3.1.1 A Modified Control Circuit

To improve the testability and fault tolerance of the interconnection/synchronization circuit we modified the circuit in Figure 5(a) to that in Figure 5(b). The register in Figure 5(b) has two gated latches for each DVCSL output pair, and thus it accepts morphic inputs instead of single line inputs as in Figure 5(a). Both latches are reset after its contents have been used to give it a better chance to detect errors if it is clocked when data is changing or if some of the latch pairs fail to be reloaded. The dual gated latches are simpler than the single

positive edge triggered flip-flops used in Figure 5(a).

Since the register is reset after computation is done and the input to the register is a morphic pair, we can generate the latch complete signal, *Aout*, using an OR-AND circuit rather than the more complex COMPARE-AND circuit used in the original design. The *completion* signal from the Functional Block is generated by a 1-out-of-2 checker that waits until all signal pairs have at least one "1" value before signalling completion. It can also explicitly detect (1,1 and 0,0) error outputs. This circuit is partially self-checking and it will be described later.

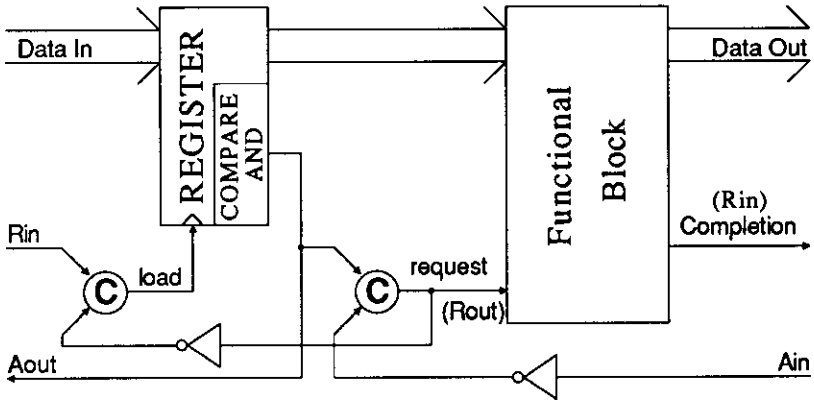
3.1.2 Stuck-at Faults

The interlocking nature of the feedback control signals causes the circuit to "hang up" and stop if one of the signals *Ain*, *Aout*, *Rin*, *Completion*, .. sticks at a one or zero value (see Figure 3). A time out counter is employed to detect the stopped condition.

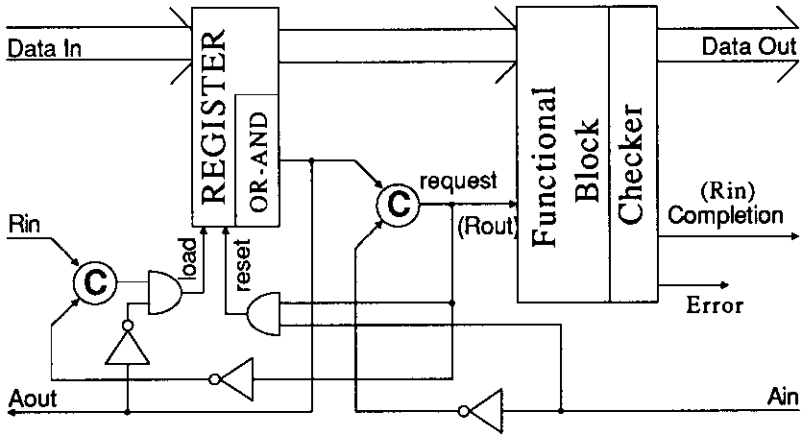
In nearly all cases, stuck-at values in a register or functional block will cause a detectable value of 0,0 or 1,1 to appear at the checker. This occurs because the dual-rail DCVSL logic block circuits pass on an uncoded (0,0 or 1,1) outputs when an uncoded input (0,0 or 1,1) occurs. When input signals occur that would normally cause a stuck circuit to go to the other value, its complementary circuit takes on the same value, generating an uncoded signal that passes through the Functional Block to the checker.

The reset signal sets all register pairs to 0,0 to enable detection of faults caused by the inability to clock one or more sets of latches. It is redundant so that if it sticks at zero, a second fault must occur before an error is generated. If it sticks at one, the register will be permanently reset to 0,0 pairs, *Aout* will never go high, and the circuit will stop.

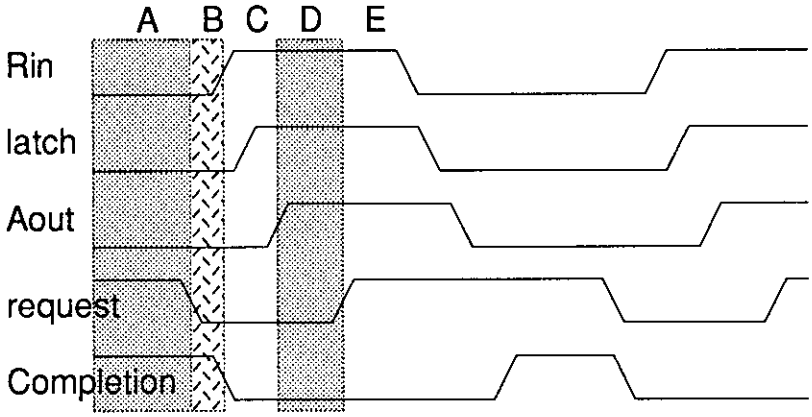
As soon as the latch complete signal goes high, the *load* signal to the register goes low in order that the latched data are not disturbed by changing data from the preceding stage. If the *load* sticks at zero, the registers will be permanently reset and *Aout* will not be generated, halting the circuit. A stuck at one *load* signals will cause the register not to be held constant while the functional block is working. The C-gate preceeding a register normally prevents the register from being reloaded while the outputs of the circuit that sent it inputs is being to reset to 0,0. The stuck at one *load* will allow the register to change while the following functional block is using its data. The results, though difficult to predict, are likely to produce a detectable coding error in the following stage.



(a) Original ISC Circuit (Meng)



(b) Modified Interconnection/Synchronous Circuit



(c) Timing Diagram for Transient Effects on the ISC

Figure 5: Interconnection/Synchronous Circuit

3.1.3 Transient Errors

It is not possible to exhaustively analyze transient error responses. Extensive simulation is probably the only way to characterize the coverage of these circuits under various transient conditions. However, we will examine two examples of transient errors that are most likely to occur.

A Muller-C circuit has the property that the output signal will change to the input value when both inputs are of the same values; otherwise the output stays unchanged. This hysteresis can mask out transient errors under some circumstances. A typical timing diagram of the ISC signals of Figures 5(a) and (b) is shown in Figure 5(c). Consider the different effects of transients on control lines during these times.

Rin **Transient Errors:**

Rin is more likely to experience transient errors or permanent faults than most other signals because it is generated as a combinational function of all the data outputs from the previous stage's function block.

Interval A – Any transients in the *Rin* signal during the timing zone, denoted as 'A', can not change the latch signal because the other C-gate input is low. In this period both of the ISC circuits in Figure 5 have the same responses.

Interval B – Zone 'B' is the time the circuit is ready for another computing step. Here, the C-gate inputs are different so any transient in the *Rin* signal sets the *load* signal. A transient in the original ISC with single flip-flops in Figure 5(a) clocks in wrong data and drives incorrect but re-encoded morphic values to the functional block – resulting in an undetectable error. But the ISC in Figure 5(b) waits for morphic data from the previous functional block and provides correct values to the functional block despite of incorrect timing caused by the glitch in the *Rin* signal. (Remember that the latch pairs are all set to zero before the computation starts. Due to the hysteresis of the C-gate, the *load* signal will remain at one until all of the data stabilizes correctly to (1,0 or 0,1) in the dual latches. Then the *Aout* signal resets the *load* signal. Similarly transient faults in the *load* signal can be tolerated in the ISC of Figure 5(b), but not in that of Figure 5(a).

Intervals C,D,E – During the rest of the cycle, transients on *Rin* will be masked by the C-gate because it has output of one and the other input is one.

The modified circuit (Figure 5(b)) has only one period (B) when *Rin* is sensitive to transient errors. A transient that holds the value of *Rin* to one or zero for an extended period of time only halts the circuit for the transient duration. The interlocking sequence does not allow processing to proceed to the next step.

Errors in the latch completion signal *Aout*

As discussed above, the data latch completion signal *Aout* is generated by an OR-AND check of the register's contents, i.e., all of the output pairs must have one line go to logic one. When it occurs it means that the previous stage can precharge its functional block to prepare for the next data while the current stage can start computation with the latched data.

If there is a falsely generated latch completion signal (i.e., false *Aout* signal) in the original circuit of Figure 5(a), the previous stage is forced to abruptly finish computation that should be continued in normal condition and it prepares for next computation by precharging the circuit. The flip flops of the register take the wrong data and convert it into properly coded 1-out-of-2 data. The Functional Block of the current stage starts computation using the wrong data and this error is not detected.

Now consider what happens in the modified circuit of Figure 5(b) when a transient occurs in *Aout*. If *Aout* takes on value one when it should be zero, this means that the data in the register is uncoded, but its availability is signalled. The functional block will be started with uncoded data. (This will be detectable by checkers in the following block.)

A more difficult case occurs when *Aout* takes on a correct value of one, but a transient causes it to transition to zero prematurely. This does not affect the *request* signal to the functional block of the current stage if it has started, but it affects that of the previous stage. *Rin* may be prematurely dropped and a new step computed in the previous stage. It will not be loaded until the functional block completes so no error will occur. If the *request* to the functional block has not yet been generated, the circuit will stop because the register cannot be loaded.

3.2 Summary of Control Error Detection

From the above discussion we have concluded that the modified control circuit has good, but certainly not perfect, error detection properties. It is a distinct improvement on the original design. The analysis of error effects is certainly far from complete, and the circuit is far too complex to work them out analytically. Our next step will be to conduct switch-level simulation experiments to identify conditions that are covered and not covered, and possibly further improve the design.

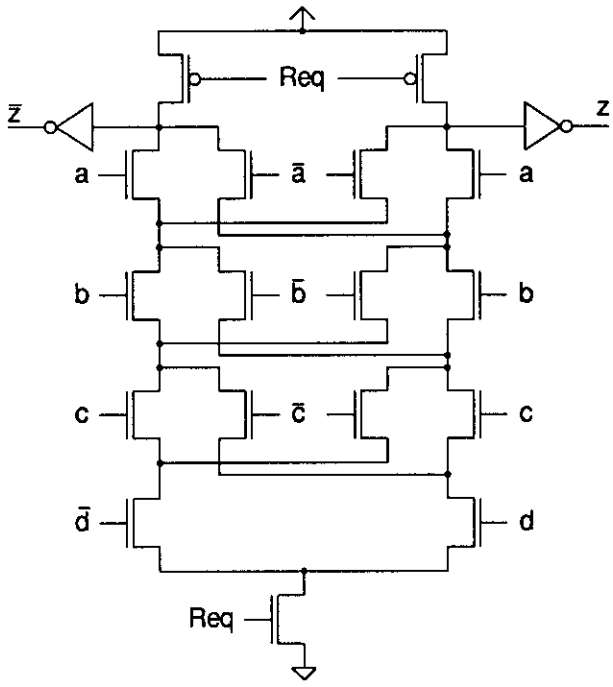


Figure 6: A 4-Input Self-Timed Checker

4 Error Detection in the Registers and Functional Blocks

As discussed above, data is output from the functional blocks, and a checker is used to determine if each pair has complementary values. The checking circuit, implemented as a tree of four input-pair dual rail checkers shown in Figure 6. This DCVSL circuit has the logic function: $z = \overline{a \oplus b \oplus c \oplus d}$ and $\overline{z} = a \oplus b \oplus c \oplus d$. Since the PMOS and NMOS transistors are separated in this self-timed circuit, the circuit area can be relatively small. By inspection one can see that any input pair of 0,0 causes a 0,0 output, and any input pair of 1,1 causes a 1,1 output.

An example of five four-input checkers combined into a tree to check a 16-pair functional block is shown in Figure 7. If any input data pair is incorrect (remains at 0,0 or goes to 1,1), the outputs $Z, \overline{Z}$ take on values 0,0 (detected by a simple time-out counter) or 1,1 (detected by the AND function). The checking circuit is equivalent to the self-checking checker (morphic AND) tree developed by Carter et. al.

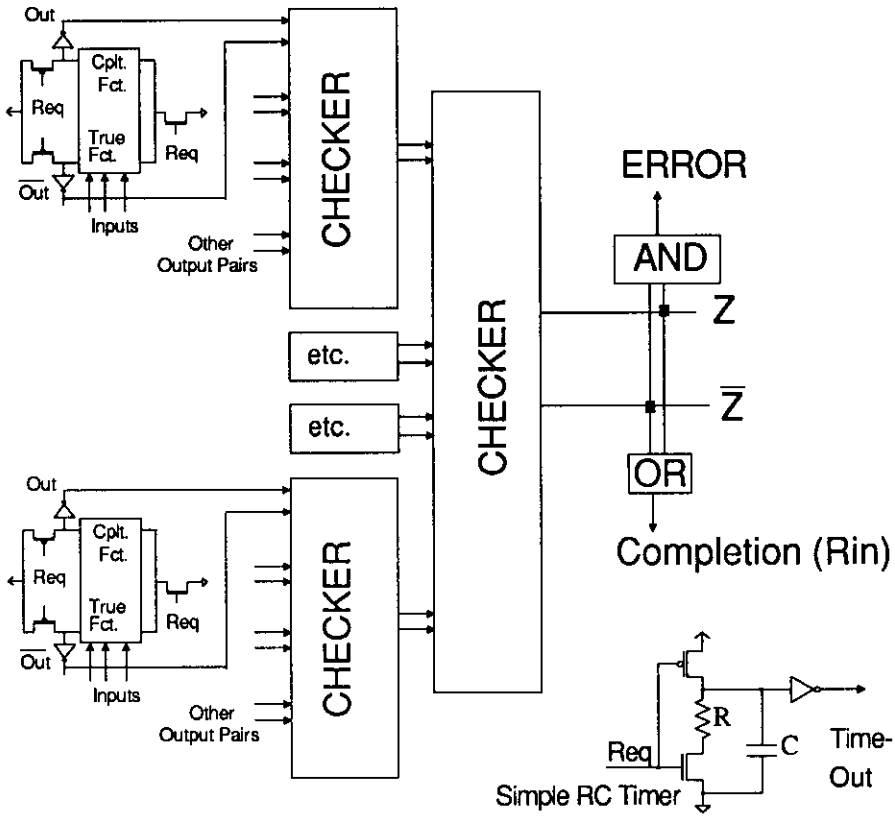


Figure 7: The Function Block Output Checker

4.1 The Checker Circuit

In a synchronous system this type of dual-rail checker is self-checking, but in an asynchronous system the behavior is somewhat different from the synchronous case for the reasons discussed above. The cases are enumerated below:

Case 1 Behavior when the circuit being checked and the checker are fault-free.

All pairs equal 1,0 or 0,1 – For error free cases the signal pairs of the circuit being checked will start at all zeros and the pairs will transition to 0,1 or 1,0. Under these conditions, the checker provides a "glitch-free" logic, and when the signals have stabilized, the final checker output will transition to 1,0 or 0,1. This is detected by the OR function in Figure 7 that generates the completion signal (*Rin* to the next stage).

Case 2 Behavior when the circuit being checked (CBC) has a single error and the checker is fault-free.

- One pair has values 0,0 – For those errors in the CBC that produce a 0,0 in one of its output pairs, a tree of checkers will continue to output 0,0, and no completion signal will be generated. This is detected using an independent time-out circuit.
- One pair has values 1,1 – For CBC errors that produce a 1,1 in one of its output pairs, the checker tree will eventually produce a 1,1 output also. However, for a short period of time, the erroneous signal pair will transition from 0,0 through a correctly coded value 0,1 to an incorrect value 1,1. This will, in effect, cause the signal pairs down one path of the tree to rapidly ripple from a legal value 0,1 or 1,0 to 1,1, and the final output pair $Z, \overline{Z}$ will go through a sequence of 0,0, to (0,1, or 1,0), to 1,1. The first transition to 0,1 or 1,0 signals completion through the OR function in Figure 7. The detection of the second transition to 1,1 is done with the AND function in Figure 7. Note that after the completion signal is generated there must be sufficient delay in resetting the CBC in order to assure detection of the 1,1 condition.

Case 3 Behavior when the CBC is good but the checker tree has stuck-at variables.

- Cases that lead to time-out detectable output values of 0,0 are stuck-open transistors in the pull down tree or a stuck-at-zero inverter. For some inputs the error will not be detected because the stuck at conditions are the same that the inputs would have generated. So the circuit pairs remain complementary causing a correct $Z, \overline{Z}$ output. So long as no double error occurs in the data, no computational error occurs and no harm is done.

  For other input patterns the effect is to generate an output pattern of 0,0 at one checker circuit because the complementary circuits take on identical (off or zero) values. Since these checker circuits are combined into a tree, the 0,0 output propagates through other checkers and holds the final output at 0,0. The final complete *Rin* is not generated, and this will be detected by the time-out counter. Analogous to self-checking circuits, the right input pattern has to appear to flush out this fault condition and eventually cause an error signal.
- The remaining faults are stuck-short transistors in the pull down tree and stuck-at-one values in the inverters. These may result in a premature completion signal and could lead to subtle undetected error conditions. Some input data patterns will lead to a $Z, \overline{Z}$ output of 1,1 (i.e. the complementary transistor or inverter gets a correct state identical to the failed one). In this case, as in case 2 above, an error will be detected by the AND gate if the circuit is not reset too quickly.

  Other input patterns will lead to a $Z, \overline{Z}$ output pattern of 0,1 or 1,0. Here the stuck circuit is stuck at the value the data would cause it to take anyway so the

data drives the complementary circuit to the opposite value of the stuck circuit (The other inverter is "1" or the other transistor is open.) The error will not be detected, but an early completion signal may be generated. In effect, the tree decides immediately that some of the circuits are complete, and data may be prematurely loaded as one or more 0,0 pairs into the next pipeline stage. These errors will go undetected in the current stage, but the next stage should stop because it received an 0,0 input and cannot complete.

5 Issues in Implementing a Self-Timed Memory Interface

In a paper in FTCS21, we described a self-checking self-exercising memory system under development at UCLA that can be composed of two chip types, a special RAM with internal parity checking and a Memory Interface Building Block (MIBB) that implemented SEC-DED codes and had extensive concurrent error detection features [Renn91]. In the MIBB design we found that speed was limited by the synchronous clocking scheme that typically required five constant length high speed clocks to control and synchronize read and write operations. Also it was determined that higher performance could be obtained if data was read from the RAM in blocks, and pipelining was used in the MIBB. Thus we have begun to explore the use of a self-timed design for a higher performance MIBB chip. The architecture is preliminary, and is expected to change as the various function blocks are further mapped out.

Figure 8 shows the data path of the MIBB presented at FTCS21.

In the original synchronous design, the critical timing path is from the Spare Plane Switching Circuit (SPSC) to the Error Detection circuits for the Hamming Code Tree (HCT). This path contains a Spare Plane Switching Circuit (SPSC), Inverter, bus with large capacitance, HCT (checking and generating SEC-DED code), and a self-checking comparator for the tree.

This data path can be implemented in self-timed circuits as shown in Figure 9. The functional blocks of Figure 8 are replaced by self-timed functional blocks with attached registers to form a pipeline. The registers attached to the self-timed blocks store data during precharge and computation of the blocks. The SPSC and Inverter are merged to one functional block. Since data from outside of the chip are only in true form, pad driving circuits and registers receiving data from outside are designed to handle two-to-one and one-to-two interfaces. Since the RAM chips do not provide morphic outputs, their completion signal must be artificially generated. The single bidirectional bus was replaced with separate

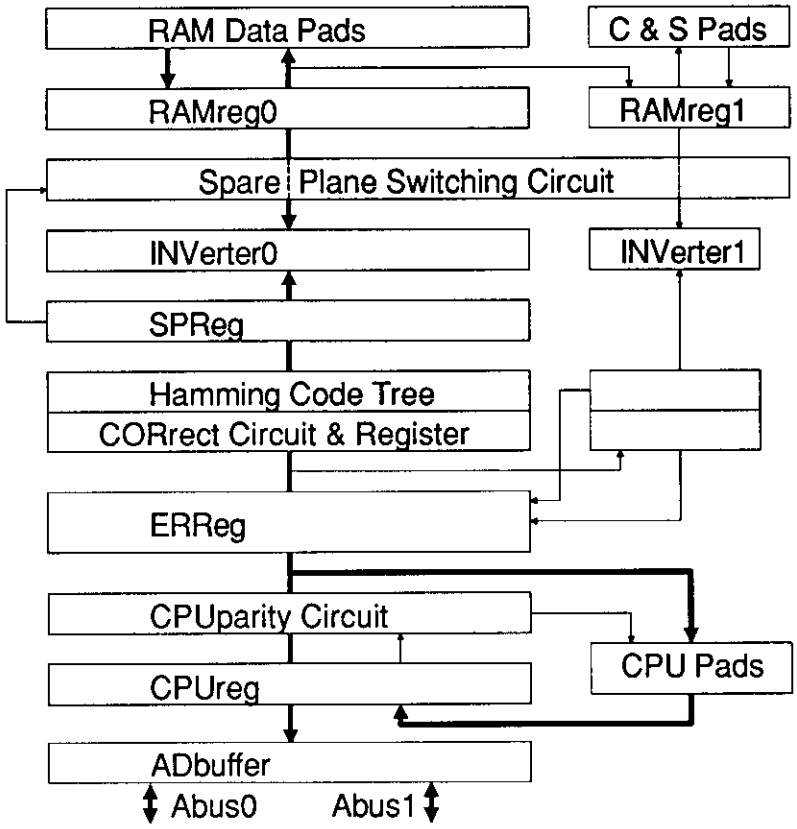


Figure 8: The Synchronous MIBB Data Section

READ and WRITE paths. The READ path consists of three (read operation without error) or four (read operation with data error) pipelined stages, while the WRITE path has two pipelined stages.

There are potential advantages and disadvantages associated with the asynchronous data path implementation. Among the potential advantages are:

1. There is more comprehensive error detection, since the individual signal pairs can be checked in addition to the Hamming code and bus parity codes across the data words. This makes it easier to differentiate between data errors from the RAMs (code errors only) and errors in the MIBB chip (code errors and errors in the morphic signal pairs).

2. If the data path can be set up as a micropipeline, control signals are localized and require less routing.

3. Faster operation can be achieved using the pipeline for burst data transfer.

Potential disadvantages of the asynchronous data path include:

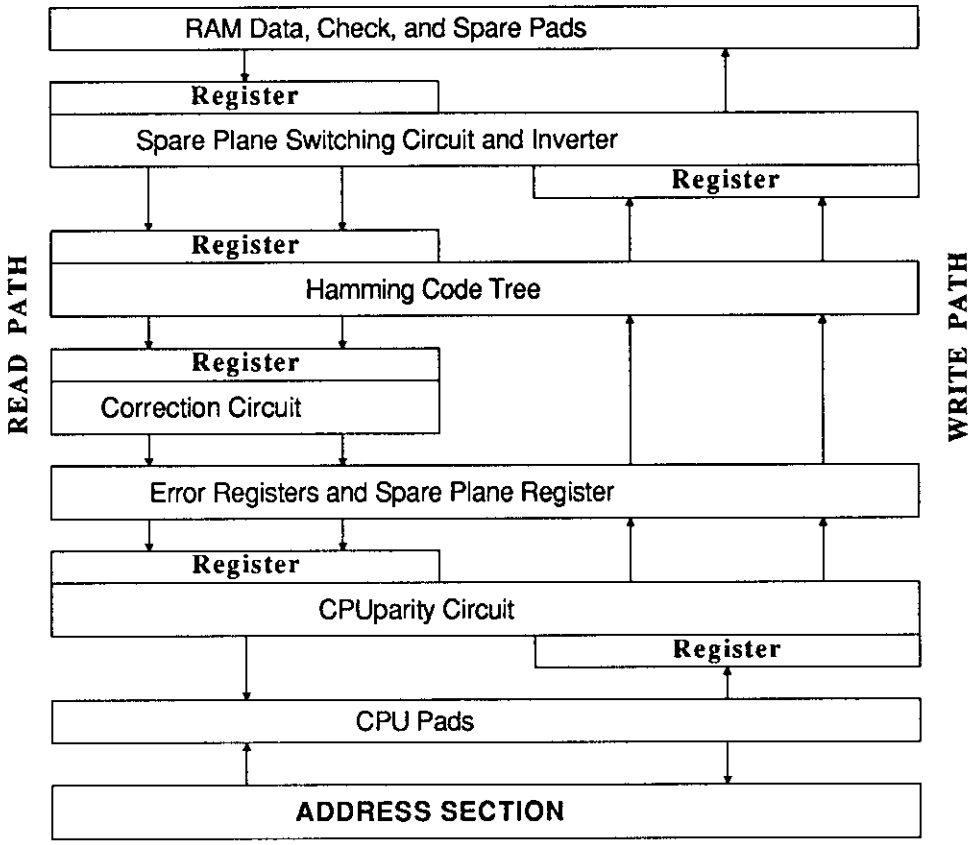


Figure 9: Data Section of the Asynchronous MIBB

1. More routing is required for data signals.
2. The number of storage cells for registers is doubled (for true and complement values).
3. Increased logic. This is deceptive however, because self-timed logic is often more compact than conventional CMOS. Since the PMOS pull ups are limited in number and better isolated from the NMOS logic trees, the area cost of separating NMOS and P-wells is considerably reduced. Some self-timed circuits take less space than synchronous circuits (as will be seen in Section 6 below).
4. Overhead for handshaking introduces circuit and time delays.

6 A MIBB Circuit Example

For an implementation example, we choose the slowest circuits of data path, the Hamming Checking Tree (HCT) and the self-checking comparator to compare synchronous and asyn-

chronous designs. The HCT uses seven trees of XOR gates to generate 7 syndromes. The individual checkers average 14 two-input gates each and the checkers together require a total of 96 gates. In the synchronous design, we generate both a true and complement version of each parity check using one tree of XOR gates and another tree of XNOR gates. The results of these checks are compared with a self-checking checker to distinguish between data errors and checker errors.

The basic cells for the synchronous HCT are the 2-input XOR gate and 2-input XNOR gate which generate true and complement outputs (shown in Figure 10(a) and (b)). A 4-input self-timed XOR gate is used to implement the self-timed (asynchronous) HCT, and is shown in Figure 10(c). Since the self-timed tree carries both true and complement signals it has additional error checking properties and does not need to be duplicated.

If an error occurs in the data path that causes a signal pair of 0,0 or 1,1 it can be detected. If a 0,0 appears on any input pair, the output will also be 0,0 and propagate through the tree. Similarly, if an input pair is 1,1, and output pair of 1,1 will be generated and propagate through the tree. An open transistor will cause either a correct output or the value of 0,0 for some error patterns. Similarly, a shorted transistor will cause either a correct result or an output of 1,1, depending upon the input pattern.

Table 1 presents comparison of the synchronous and asynchronous designs of the HCT in terms of transistor count, input load, and time delay using SPICE-simulations with 2-um design rules. Transistor sizes of the gates are not optimized, but are fixed $4um$ and $8um$ to NMOS and PMOS respectively.

Each of 96 CMOS-XOR gates and each of 96 CMOS-XNOR gates in the synchronous design have 10 transistors, for a total of 1920. The self-timed XOR gate is a four-input device, and thus there are only 35 of them required. The DCVSL logic is especially efficient in implementing multi-input gates. A total of 805 transistors are used. From the summary we see that the self-timed parity tree is several times smaller and is faster than the synchronous CMOS design.

7 Conclusions

This preliminary study indicates that self-timed design techniques can be adapted to fault-tolerant systems, and that they offer considerable potential in the implementation of modules that have concurrent error detection. An asynchronous pipeline controller was presented that was enhanced to provide improved concurrent error detection, and the error detection techniques for combinational functions were discussed. Preliminary studies were presented comparing a portion of a Memory Interface Building Block implemented in synchronous

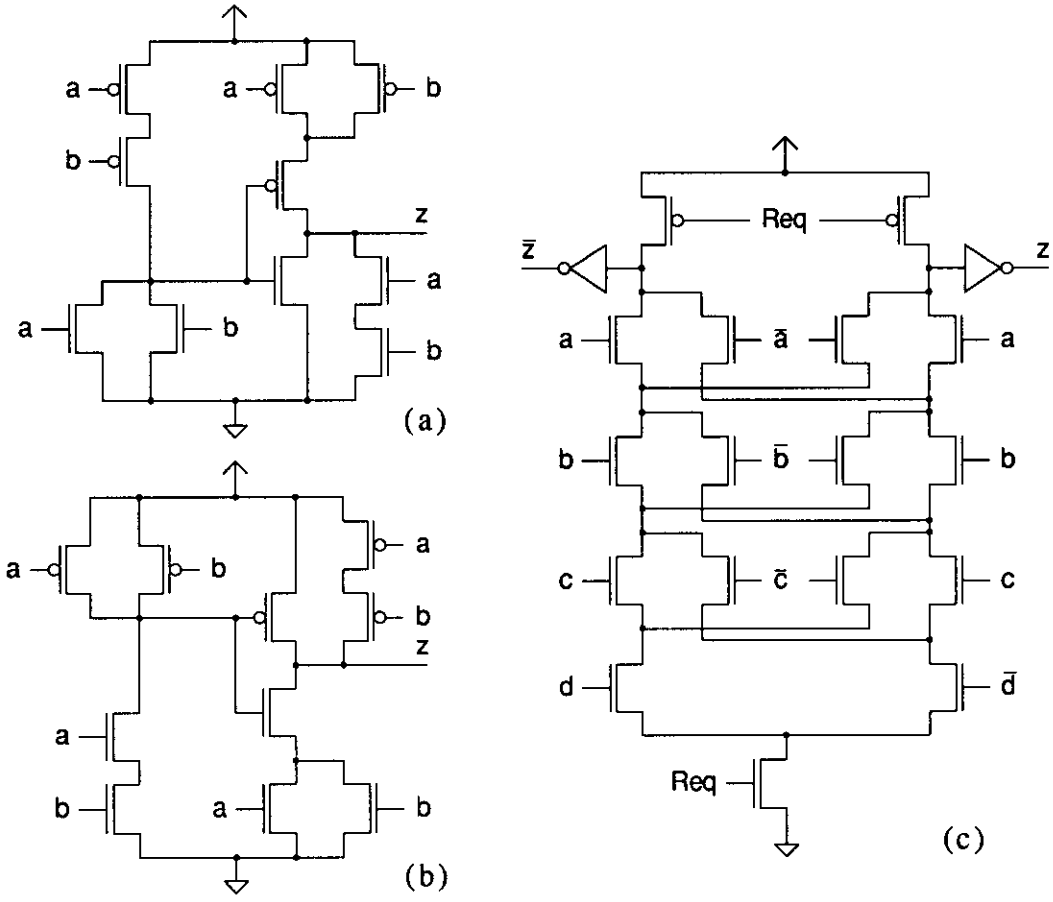


Figure 10: (a) XOR, (b) XNOR, and (c) Self-Timed XOR Gates

versus asynchronous logic.

Although the error detection coverage of these circuits appears to be high, it is hard to get an accurate measure by analysis, and we only presented a qualitative discussion of what we viewed as the most likely errors and faults. We are beginning a series of simulation experiments to experimentally evaluate their response to various error conditions to better understand their fault and error coverage.

Of course, self-timed logic is a matter of religion to many, but it is not clear to what degree it will ever displace conventional clocked CMOS designs. We make no projections here, but only note that asynchronous design is very interesting, and its fault-tolerance properties need to be explored from an architecture prospective.

The cost of this approach is reasonable, and we are optimistic that this design style will become more important as fault tolerant systems made are made from larger chips with smaller feature sizes.

		Synchronous				Asynchronous		
		XOR	XNOR	Checker	Total	XOR	Checker	Total
Number of Gates		96	96	36	228	35	3	38
Number	PMOS	480	480	72	1032	210	18	228
of	NMOS	480	480	72	1032	595	39	634
TRs	Total	960	960	144	2064	805	57	862
Input Loading		4 tr	4tr	4tr		2tr	2tr	
Time Delay		5.2ns	5.3ns	3.5ns	8.8ns	3.9ns	3.2ns	7.1ns

Table 1: Comparison of Two Designs in Implementing The HCT

References

[Barz85] Z. Barzilai, V. S. Iyengar, B. K. Rosen, and G. M. Silberman. Accurate Fault Modeling and Efficient Simulation of Differential CVS Circuits. In *International Test Conference*, pages 722–729, Philadelphia, PA, Nov 1985.

[Cart68] W. C. Carter and P. R. Schneider. Design of Dynamically Checked Computers. In *Proc. IFIP Congress 68*, pages 878–883, Edinburgh, Scotland, Aug 1968.

[Cart72] W. C. Carter, A. B. Wadia, and D. C. Jessep Jr. Computer Error Control by Testable Morphic Boolean Functions - A Way of Removing Hardcore. In *1972 Int. Symp. Fault-Tolerant Computing*, pages 154–159, Newton, Massachusetts, June 1972.

[Jaco90] Gordon M. Jacobs and Robert W. Broderson. A Fully Asynchronous Digital Signal Processor Using Self-timed Circuits. *IEEE Journal of Solid-State Circuits*, 25(6):1526–1537, Dec 1990.

[Jha89] Niraj K. Jha. Fault Detection in CVS Parity Trees: Application to SSC CVS Parity and Two-Rail Checkers. In *Proc. 19th Int. Symp. Fault-Tolerant Computing*, pages 407–414, Chicago, IL, June 1989.

[Jha90] Niraj K. Jha. Testing of Differential Cascode Voltage Switch One-Count Generators. *IEEE Journal of Solid-State Circuits*, 25(1):246–253, Feb 1990.

[Kano92] N. Kanopoulos, Dimitris Pantzartzis, and Frederick R. Bartram. Design of Self-Checking Circuits Using DCVS Logic: A Case Study. *IEEE Transactions on Computer*, 41(7):891–896, July 1992.

[Kano90] N. Kanopoulos and N. Vasanthavada. Testing of Differntial Cascode Voltage Switch (DCVS) Circuits. *IEEE Journal of Solid-State Circuits*, 25(3):806–813, June 1990.

[Mart89] Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus. The Design of an Asynchronous Microprocessor. Technical Report Caltech-CS-TR-89-2, CSD, Caltech, 1989.

[Meng91] Teresa H. Meng. *Synchronization Design for Digital Systems.* Kluwer Academic Publishers, 1991.

[Mont85] R. K. Montoye. Testing Scheme for Differntial Cascode Voltage Switch Circuits. *IBM Technical Disclosure Bulletin*, 27(10B):6148–6152, Mar 1985.

[Renn91] David A. Rennels and Hyeongil Kim. VLSI Implementation of A Self-Checking Self-Exercising Memory System. In *Proc. 21th Int. Symp. Fault-Tolerant Computing*, pages 170–177, Montreal, Canada, June 1991.

[Sedm80] Richard M. Sedmak and Harris L. Liebergot. Fault Tolerance of a General Purpose Computer Implemented by Very Large Scale Integration. *IEEE Transactions on Computer*, 29(6):492–500, June 1980.

[Taka90] Andres R. Takach and Niraj K. Jha. Easily Testable DCVS Multiplier. In *IEEE International Symposium on Circuits and Systems*, pages 2732–2735, New Orleans, LA., June 1990.