Proceedings of the IEEE International Conference on Computer Design
Port Chester, New York, pp. 302-307, October 1984.

REDUCING COMMON MODE FAILURES IN DUPLICATE MODULES

Yuval Tamir and Carlo H. Séquin
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

Abstract — An effective scheme for detecting errors caused by hardware faults is to continuously compare the outputs of a pair of hardware modules performing identical operations on the same inputs. This approach is viable only if there is a very low probability that the two modules will fail in exactly the same way and generate identical *incorrect* outputs. While such *common mode failures* cannot be entirely eliminated, the probability of their occurrence can be reduced. We discuss the implementation of pairs of VLSI modules that perform identical functions but are less susceptible to common mode failures than pairs of identical circuits. Based on examples of NMOS and CMOS circuits, it is shown that the likelyhood of common mode failures can be reduced, at a relatively low cost, by using a combination of techniques carefully tailored to the functional and physical characteristics of the different parts of the circuit.

I. Introduction

An effective error detection scheme is of crucial importance to any fault tolerant system. In the past, different concurrent error detection techniques were used for different parts of the computer system in order to minimize the amount of redundant hardware required. With VLSI technology, hardware is no longer expensive but designing and testing chips is costly and time-consuming, and new, more complex, hardware failure modes are possible.[3] Given these properties of VLSI, *duplication and matching* at the level of large functional modules is an attractive approach to implementing concurrent error detection.[4,7,11] Each functional module, such as a microprocessor, a memory management unit, or even a complete microcomputer, is realized with two physical modules that operate synchronously and perform identical functions on the same inputs. The outputs of the two modules are continuously compared and any mismatch signals an error. We will henceforth use the acronym *SCFM* to denote a **s**elf-**c**hecking **f**unctional **m**odule implemented by duplication and matching. The two physical modules whose outputs are continuously compared will be referred to as *modules*.

If the two modules and the comparator are implemented on a single chip, the self-checking capability of the chip may also be used to simplify its testing throughout its life: from wafer probe testing that is part of the manufacturing process to final acceptance tests by users. The simplification of testing is achieved by eliminating the need to store the correct responses to long test sequences and compare them with the actual responses of the chip during testing. Testing can proceed at the normal system clock rate and only the outputs of the comparator need to be monitored.

This work was supported by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3803, monitored by Naval Electronic System Command under Contract No. N00039-81-K-0251.

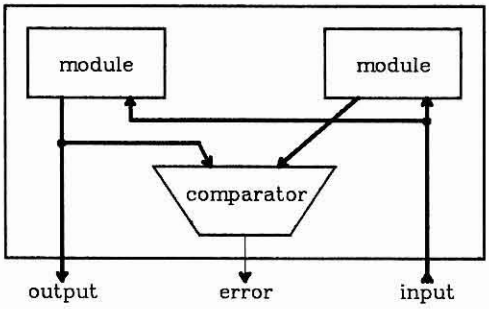


Fig. 1: A Self-Checking Functional Module (SCFM)

With duplication and matching, all errors caused by hardware faults are detected as long as: (1) the comparator is fault-free and (2) the two modules do not fail at the same time in exactly the same way. The simplest implementation of duplication and matching is to use pairs of modules that are identical physical duplicates. While more than half the hardware is used for error detection, relatively little additional design effort is required for the comparator and the interconnections between the comparator and the two modules. As long as the above two assumptions hold, this scheme guarantees the detection of errors caused by all possible hardware faults including multiple faults and other complex faults that are not covered by many of the coding techniques commonly used for concurrent error detection.

Since the comparator may fail and mask possible mismatches between the modules, it is imperative that faults in the comparator be detected soon after they occur so that the rest of the system can be informed that the SCFM has lost its self-checking capabilities.[11] This requirement can be fulfilled by using a *self-testing* comparator that signals its own faults during normal operation.[2,12]

Unfortunately, it is not possible to guarantee that the two modules do not fail simultaneously in exactly the same way and produce identical incorrect results. Such *common mode failures* (henceforth, *CMFs*) may occur as a result of environmental factors, common design weaknesses, as well as unrelated faults that just happen to cause the same incorrect results to be produced.

We provide a formal definition of CMFs and describe their possible causes. Several schemes for reducing the probability of CMFs are discussed. Cells that are commonly used in NMOS and CMOS VLSI circuits are analyzed, and it is shown that no one technique is effective for all possible cells. An optimal balance of cost, performance, and effectiveness in reducing common mode failures, can be achieved with a combination of techniques tailored to the characteristics of specific cells.

II. Common Mode Failures

Common mode failures may be caused by environmental factors such as power supply fluctuations, pulses of electromagnetic fields, or bursts of cosmic radiation, that can affect both modules at the same time, triggering similar design weaknesses and causing simultaneous identical failures of both modules. Simultaneous module failures may also be caused by faults that occur at different times in parts of the modules that suffer from identical design weaknesses and are infrequently exercised.

Modern VLSI technology has made the implementation of two complex modules (such as processors) on the same chip feasible. For systems requiring high reliability, a self-checking microprocessor, implemented with duplication and matching, is an attractive building block. However, when the two modules are fabricated on the same chip, CMFs are more likely to occur due to the tighter electrical and physical coupling between the two modules and as a result of similar weaknesses in the two modules caused by fabrication flaws specific to the wafer containing the chip.

As previously mentioned, the self-checking capability of the chip may also be used to simplify its testing. As a result of fabrication defects in chips that have never been tested (i.e., have not yet gone through wafer probe testing), CMFs of the two modules may be relatively common if the modules are physical duplicates. Hence, if wafer probe testing relies on the self-checking capability of the chip, different physical implementations of the two modules must be used.

For pairs of modules that are identical physical duplicates, the meaning of the term "common mode failures" appears obvious. However, if the two modules perform identical functions but are physically different, there is no direct correspondence between physical faults in the two modules and the meaning of the term is unclear. Hence, there is a need for a definition of CMFs that is applicable to modules that are physically different.

In the rest of this section, F will denote the set of all *single faults*, where a single fault is a fault caused by a single physical defect.[12] In discussing the failure of the two modules in a SCFM, we consider "double faults" (f_1, f_2) where $f_1 \in F$ affects one of the modules while $f_2 \in F$ affects the other module.

The two modules are denoted by A and B. When both modules are fault-free, both are implementations of some function Z. The implementation of Z by module A is denoted by Z_A. For every input I, $Z_A(I) = Z_B(I) = Z(I)$. When the module A is affected by a fault $f \in F$, it performs the function Z_A^f. The two modules may produce identical incorrect results due to unrelated faults that just happen to affect the outputs in the same way. In this situation, f_1 affects A, f_2 affects B $(f_1, f_2 \in F)$, and there is an input I such that $Z_A^{f_1}(I) = Z_B^{f_2}(I)$ even though $Z_A^{f_1}(I) \neq Z(I)$. Hence, there is a non-zero probability that a supposedly self-checking SCFM will fail to flag erroneous output. Thus, the SCFM is <u>not</u> *fault-secure*[13] with respect to certain "double faults" that affect both modules.

In the worst case, the new functions, $Z_A^{f_1}$ and $Z_B^{f_2}$, performed by the faulty modules are identical, and the fault is *never* detected since for *every* input I, $Z_A^{f_1}(I) = Z_B^{f_2}(I)$. In this case, the SCFM is not even *self-testing*[13] with respect to the "double fault" (f_1, f_2).

While it is clearly impossible to ensure that the SCFM will be fault-secure with respect to every double fault $(f_1, f_2) \in F \times F$, one might hope that appropriate implementation of the modules can ensure that module functions, as modified by the faults, are not identical, so that the fault is detectable. If this is done, the SCFM is *partially self-checking*[13] with respect to all double faults $(f_1, f_2) \in F \times F$. We thus make the following definitions:

Def. 1: The two modules in a SCFM are said to be affected by *common mode failures*, if and only if, there exists at least one input vector for which both modules produce incorrect outputs, *and* for every input, the outputs from the two modules are identical.

Def. 2: Two modules are said to have *independent failure modes* with respect to a fault set F, if and only if, for every double fault $(f_1, f_2) \in F \times F$, such that f_1 affects one of the modules and f_2 affects the other, there exists at least one input that results in different outputs from the two modules.

In the definition above, F does not include faults on the input and output lines of the modules since it is clearly impossible for the two modules to have independent failure modes with respect to such faults.

III. Implementing Modules with Independent Failure Modes

Unfortunately, implementing modules that perform identical functions but have independent failure modes with respect to a reasonable fault set (e.g., single stuck faults) appears to be very difficult (perhaps impossible). We have considered several simple combinational modules, but for no function were we able to discover two implementations with independent failure modes with respect to single stuck faults. We believe that such implementations do not exist. As noted in Section I, one of the benefits of using duplication and matching for self-checking subsystems is that relatively little extra design effort is required in order to implement the self-checking property. Even if it is possible to design very simple modules that have independent failure modes with respect to single stuck faults, it is unlikely to be practical and economically feasible, especially if we take into account more realistic fault models[3,12] and consider complex functional modules (such as microprocessors).

Assuming that there is no practical way of implementing modules that have independent failure modes with respect to all double faults, we concentrate our efforts on reducing the probability of those double faults that are more likely to occur than random double faults. The technology and circuits used to implement the modules in an SCFM determine which double faults are more likely to occur and whether they are detectable. Hence, we consider a particular implementation technology and "representative" example circuits rather than attempt to apply uniform analysis to all possible circuits. In this paper only NMOS and CMOS VLSI implementations are considered. As a "representative" circuit we consider the Berkeley RISC microprocessor[8] for which there is an NMOS VLSI implementation[8] as well as a nearly complete CMOS layout[9]

Many months (or years) are devoted to the design of

VLSI chips in order to achieve maximum functionality, performance, and reliability with the given technology. In most cases it is unacceptable to double the design time and development cost of a VLSI chip simply to achieve more reliable error detection by reducing the probability of CMFs. Completely independent implementations of the two modules in the SCFM are therefore not practical. The use of duplicate physical modules in the SCFM is the lowest cost alternative. However, given the time and resources spent on designing a VLSI chip, it is clearly worthwhile to spend a few additional weeks on the implementation of both modules and minimize some of the performance and yield costs of using duplication and matching. A practical approach to implementing modules with independent failure modes involves spending most of the effort designing and optimizing one module and then "designing" the second module by modifying the first one. In the following sections we discuss how this overall approach can be applied for the specific "typical" circuit mentioned in the previous paragraph.

IV. Dual Implementations

For every combinational Boolean function $f(x) = f(x_1, x_2, \cdots, x_n)$ there is a corresponding *dual* function g such that $g(x) = \overline{f}(\overline{x})$ for every x. In the circuits C_f and C_g that implement the functions f and g, respectively, voltage levels represent the logic values. If both C_f and C_g are implemented using *positive logic* (i.e., the "high" voltage level representing a logic 1 and the "low" level representing a logic 0), C_g (C_f) is a *negative logic* implementation of the function f (g). The circuits C_f and C_g are said to be *dual implementations* of the function f. C_f and C_g are said to be *dual circuits*.

Dual implementations of arbitrarily complex sequential logic circuits are also possible. If the inputs to the negative logic implementation are complements of the inputs to the positive logic implementation, the corresponding outputs from the two implementations are complements of each other.

Sedmak and Liebergot[7] have suggested that the probability of CMFs in a SCFM can be reduced by using dual modules rather than pairs of identical modules. The inputs to the SCFM are passed unmodified to the positive logic module (henceforth called the *p-module*), and are complemented for the negative logic module (*n-module*). If the two modules are operating correctly, their outputs are complements of each other and can be "compared" using a two-rail code checker[2, 12] (Fig. 2).

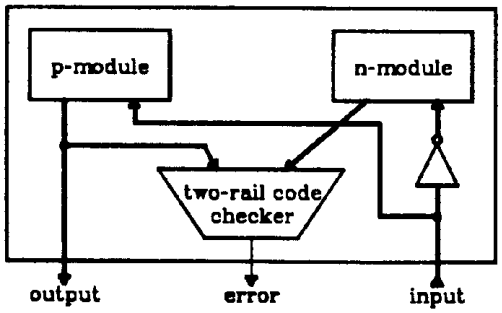


Fig. 2: An SCFM Based on Dual Implementations

There are several advantages to the use of the above scheme over the use of two modules that are physical duplicates: (1) If the modules are VLSI chips and the same masks are used in fabricating both modules, circuit design faults and faults in the masks result in identical incorrect results. With the dual implementations, different masks must be used since the circuits are different.[7] (2) Some pattern sensitive faults, such as those caused by electromagnetic coupling between lines or marginal design of the circuit timing, may be more likely to cause errors during voltage transitions in one direction. With dual circuits, the voltage transitions on corresponding lines in the two modules are in opposite directions so the probability of identical pattern sensitive faults occurring in the two modules simultaneously is reduced. (3) If the two modules are physical duplicates, all lines in both modules change value in the same direction at the same time. As a result there may be "spikes" in the power supply lines to the SCFM which can trigger intermittent faults. With dual circuits the problem is alleviated since values in the two modules change in opposite directions.

If SSI technology is used, dual logic implementation is relatively straightforward — the positive logic module can be designed first and then converted into a functionally equivalent negative logic module by a simple one-to-one replacement of gates and flip-flops with their negative logic equivalents. Both modules have the same structure and the logic values on corresponding lines of the two modules are identical. However, since a logic 1 (logic 0) in the n-module is represented by the same voltage as a logic 0 (logic 1) in the p-module, the voltages on corresponding wires of the two modules are complements of each other. Following De Morgan's theorem, and "labeling" gates with their positive logic functionality, for every OR (AND) gate in one of the modules there is an AND (OR) gate in the other. Similarly, for every positive-edge-triggered flip-flop there is a negative-edge-triggered flip-flop, and vice versa.[7] In this environment the structure of the module and the performance of the corresponding "building blocks" is identical (or very similar) so the extra design time for the negative logic module is small and there is no performance penalty.

If VLSI technology is used, dual implementations is more problematic since it is not possible to convert an existing positive logic chip to negative logic by a simple replacement of standard building blocks. Even in the conversion of NOR gates and NAND gates to negative logic (*i.e.*, replacing NOR with NAND and vice versa) may be quite difficult due to two main factors: (1) The different gates have different topologies so the layout of the entire chip may have to be modified in order to provide room for the new gates. (2) The fan-in capability of different gates may be different — for example, in NMOS, it is possible to implement a NOR gate with a large number of inputs while a NAND gate with more than three or four inputs is not practical. Furthermore, the circuit is not simply a collection of standard logic gates and may contain transmission gates, precharged buses, register files, PLAs, decoders, dynamic logic subcircuits, etc. In a given technology, converting some of these types of circuits to negative logic may require significantly more area and/or result in lower performance.

We evaluate the dual implementations approach to reducing CMFs by considering the conversion of a positive logic VLSI module to negative logic. This conversion does

not necessarily involve converting the entire module at the lowest level (i.e., individual FETs) to negative logic. It may be preferable to design the n-module so that some of the subcircuits in the p-module have direct negative logic equivalents in the n-module while other subcircuits are used unmodified in the n-module. The only critical requirement is that the n-module "behave" as the negative logic equivalent of the p-module at the interface between the n-module and the rest of the SCFM. We discuss the possible choices of subcircuits to be converted and the consequences of those choices in terms of design effort and the types of CMFs that can thus be eliminated.

A. NMOS Implementation

Standard NMOS circuits are fundamentally asymmetrical. The available devices are enhancement mode FETs (EFETs) and depletion mode FETs (DFETs). The EFETs are turned on by the "high" gate voltage and turned off by the "low" gate voltage. The DFETs are always "on" but have a higher conductivity when their gate voltage is high. There is no device that can perform the dual function of the EFET, i.e., be turned on by a low gate voltage and off by a high gate voltage. There are important consequences to this asymmetry:

(1) One of the useful building blocks of NMOS circuits is the transmission gate that can be implemented using only one EFET without power or ground connections (Fig. 3-A). The dual implementation of this function requires three FETs as well as a power and ground connection since the control signal must be inverted (Fig. 3-B).

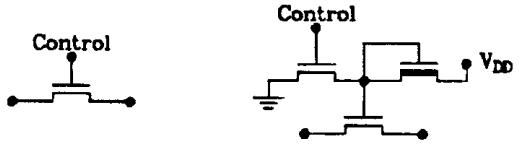


Fig. 3: An NMOS Transmission Gate

(2) Static logic gates use passive pull-up devices (DFETs). These gates are able to drive capacitive loads from high to low much faster than from low to high.

(3) As mentioned earlier, positive logic static NOR gates with a large number of inputs can be implemented. However, a correspondingly simple and fast NAND gate cannot be implemented since the delay of an NMOS ratioed logic NAND circuit increases in direct proportion to the number of inputs[5]

(4) Precharged buses are often used in VLSI chips as a space-efficient method of allowing a large number of data sources to the bus. Since EFETs are the only devices that can be completely turned off, both the pull-up and the pull-downs must be EFETs. Since EFETs make better pull-downs than pull-ups, it is much more efficient to precharge the buses to high and drive data on the bus with pull-downs than the other way around.

The above constraints on NMOS circuits prevent the simple conversion of many of the common subcircuits in an NMOS VLSI chip to negative logic. One of the difficulties is that many of the control lines in such a chip are connected to pass transistors that are selectively turned on depending on the clock phase and the operation performed: buses are precharged and discharged through EFETs selected by control lines, the inputs to the ALU are selected with a multiplexer implemented with pass transistors, data is "loaded" to latches through pass transistors, etc. Due to the large number of these pass transistors, it is not feasible to replace them with their negative logic equivalents that require much more area and power (Fig. 3). Given that it is impossible to convert the entire chip, including all the control circuitry, to negative logic, the question is which subcircuits is it practical to convert and would such conversion help reduce CMFs.

In terms of design effort, the most efficient way to implement the n-module is to use the original p-module and complement all its inputs and all its outputs. Unfortunately, this approach has no benefits in term of reducing the probability of CMFs and results in a performance penalty due to the delays of the inverters.

In order to reduce the probability of CMFs, more differences in the implementations of the two modules must be introduced. The next "step up" in this direction is to implement an n-module in which all data is stored and transferred in negative logic but positive logic subcircuits from the p-module are used for data processing and for control. The input data to the n-module is already in negative logic (Fig. 2) and is transferred through internal buses and stored in internal registers without modification. The registers and buses require no circuit modification in order to store and transfer negative logic data. Since the data on internal buses is negative logic while the data processing subcircuits are designed for positive logic inputs, the inputs and outputs of subcircuits such as the ALU must be complemented at their interface with the rest of the chip.

This approach avoides the problems with control circuits described earlier: buses, multiplexers, and latches are not modified and the transmission gate EFETs or pull-down EFETs they contain are controlled by signals with the same polarity in both modules. Since the instructions, as well as the data, are complemented before the n-module, some modifications to the various decoders are necessary. Fortunately, decoders often require that each input will be available in both complemented and uncomplemented form. In the NMOS RISC chip, the opcode decoder, the register file decoder, the shift amount decoder, and the jump condition code decoder, all already use inverters in order to generate the complemented form of their inputs. Due to the regular structure of the decoders, modifying them for the n-module is a trivial task: the connections made to the complemented and uncomplemented versions of each input are interchanged.

In RISC, the main "data processing" subcircuits are the ALU, the shifter, and the program counter incrementer. As previously indicated, it is possible to use the p-module implementation of these subcircuits in the n-module if they are preceded and followed by inverters. In order to make room for the additional inverters, major parts of the circuit must be moved. With appropriate design tools, making such a modification is not difficult. However, these inverters require additional area and increase the power consumption. Furthermore, the identical data processing circuits in both modules may be a source of CMFs which originate from both hardware defects and design weaknesses.

Converting the ALU to negative logic is suprisingly

simple. The sum and carry circuits of a full adder is its self-dual.[10] Thus, *no* modification is required for that part of the circuit. In addition to the arithmetic sum, the RISC ALU also generates the logical AND, OR, and XOR (exclusive OR) of its inputs. The actual output of the ALU is determined by a 4-to-1 multiplexer. By interchanging two of the control lines to that multiplexer, the postive logic OR can be selected by the AND instruction and the postive logic AND can be selected by the OR instruction. The only function that requires modification is the XOR. For this particular case, the simplest solution is to connect an inverter to the output of the postive logic XOR function. Since the performance of the ALU is determined by the worst-case addition time, the delay of the extra inverter in the XOR circuit does not affect system peformance.

One of the necessary modifications to the shifter is the conversion of the shift amount decoder to accept negative logic inputs. As discussed earlier, this modification is very simple. The only other problem is with logical shifts that shift in logic 0's to replace bits that are shifted out. In the n-module the "high" voltage level must be shifted in instead of the "low" voltage as in the p-module. This change can be done with a small modification to the control circuitry that drives the shifter.

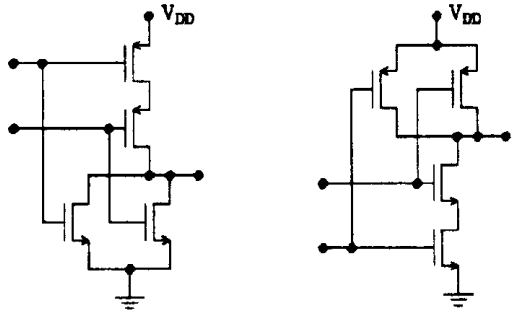
There is no simple modification to the program counter incrementer. However, the basic cell of this circuit is so small that a complete negative logic replacement can be developed very quickly.

There are numerous ways in which the circuit modifications described above can help reduce CMFs. For example: (1) Shorts between data lines carrying complementary values usually result in both lines at the low voltage. Thus, both lines in the p-module change to logic 0 while the corresponding lines in the n-module that are similarly shorted change to logic 1. (2) Buses that fail to precharge in both modules will be interpreted as all zeroes in the p-module and all ones in the n-module. (3) If timing is not properly designed and there is insufficient time to drive the bus from one of its sources, different lines on the bus will be effected (the ones that must be discharged) and the failure will be detected. (4) The worst case delay for the ALU is determined by the carry propagation. If the ALU is modified as described above, the worst-case propagation for the two modules occurs for different inputs since the sum and carry circuits are identical while the ALU inputs in the p-module are always complements of the ALU inputs in the n-module. Hence, ALU failure, due to careless design of the timing or a particular fabrication run that yields especially slow devices, is unlikely to occur in both modules simultaneously.

Since most of the control circuits used in the n-module are identical to those used in the p-module, one might assume that there are many CMFs possible due to identical defects in those circuits in the two modules. This situation can be improved if the various decoders in the chip are modified as described in Section V. Furthermore, many identical defects in the control circuitry lead to different effects on the data in the two modules. For example, if several bus sources (pull-downs) are selected at the same time (e.g., due to a fault in the opcode decoder), the resulting value on the bus will be the AND function of all the sources in the p-module and the OR of all the sources in the n-module.

B. CMOS Implementation

The *p*-channel FETs (PFETs), available in CMOS circuits, are turned on by the "low" voltage and turned completely off by the "high" voltage thereby providing the dual function of the *n*-channel FETs (NFETs). As a result, at first glance, it appears that with CMOS technology it is relatively simple to convert the positive logic module to negative logic. Specifically, it can be shown that a positive logic, ratioless CMOS circuit can be converted to a negative logic circuit by replacing all NFETs with PFETs, replacing all PFETs with NFETs, connecting all V_{DD} lines to ground, and connecting all ground lines to V_{DD}.



A. Positive Logic B. Negative Logic

Fig. 4: A CMOS NOR Gate

Unfortunately, due to the different mobilities of the majority carriers in NFETs and PFETs, these devices are not completely symmetrical. The W/L ratio of a PFET has to be approximately twice the W/L ratio of an NFET in order to achieve similar drive capability. Thus, in order to optimize performance when similar high-low and low-high propagation times are required, the PFETs used must be approximately twice the size of the corresponding NFETs. Since the gate capacitance is proportional to the size of the device, the delay caused by the PFETs due to their gate capacitance is larger than the delay caused by NFETs with equal drive capability.

Due to the advantages of NFETs, even in the CMOS RISC layout many more NFETs than PFETs are used. For example, NFETs are used in the shifter, which is basically an array of pass transistors. In the register file, the word lines, that select the register whose value drives the bus, do so by turning on a column of NFET pass transistors. In both these cases, PFET pass transistors and buses that are "precharged" low could be used. However, a design based on PFETs would be significantly larger and/or slower, as discussed above. Due to similar reasoning, NFETs are also used in the pull-down arrays of PLAs and decoders, while large PFETs are used for precharging lines to the V_{DD}. Even with static gates used for random logic, the PFET pull-ups are approximately twice the size of the corresponding NFET pull-downs.

In order to maintain similar performance and module area, the p-module cannot be converted to an n-module by the simple procedure outlined earlier. Instead, the conversion is more similar to the conversion of NMOS circuits and similar considerations apply. On the other hand, the availability of PFETs can, at times, simplify the

conversion. For example, in RISC, a large 32-input NOR gate is used to generate the Z flag, which is set when the result of an operation is zero. This gate is dynamic, with a single pull-up and a column of NFET pull-downs connected to a latch holding the result of the operation. In NMOS there is no simple way to convert this zero-detect circuit to negative logic: a column of 32 inverters must be used to invert the output of the latch and drive the pull-downs of the large NOR. With CMOS, a large negative logic NOR gate can be implemented using a single NFET pull-down and a column of PFET pull-ups connected to the output of the latch. If the performance of the circuit is critical, the PFETs will have to be larger than the corresponding NFET pull-downs in the p-module. However, the PFETs do not increase the power consumption, and the extra area of the larger PFETs is much smaller than the area required by a column of inverters.

V. Other Implementation Techniques for Reducing CMFs

As indicated in Section IV, not all the subcircuits in a VLSI chip are amenable to dual implementations. In those cases where dual implementations lead to unacceptable costs in terms of area and performance, other techniques for reducing CMFs are needed. The general "rule of thumb" is that the probability of CMFs can be reduced by increasing the "differences" between the modules. These differences may be not only in the low-level circuits but also in high-level module structure and in the fabrication process.

Modules that are likely to fail in different ways may be developed by two independent teams working from the same specifications.[1] The main problem with this approach is, of course, increased design cost, which makes it impractical for most applications.

If the two modules are *not* on the same chip, chips fabricated by different companies may be used. Platteter[6] utilized this idea in constructing a fault-tolerant processor from three functionally identical microprocessors manufactured by different companies. Obviously, this can be done only with modules that are "popular" chips for which there are "second sources."

Even when it is not possible to convert a subcircuit to negative logic, it may still be possible to modify its structure without changing its function. We have previously discussed the modification of decoders for use with negative logic inputs. Another simple modification to the decoder is to change the order of output lines in the layout so that shorts between adjacent lines will affect logically different lines in the two modules. Similar restructuring can also be done in a PLA where the order of both the product term lines and the output lines may be changed.

If the register file decoder is restructured as suggested above, this also implies a "restructuring" of the register file itself. Different registers are next to each other and different registers are at the periphery of the register file where they may interact with other subcircuits and cause a module failure.

VI. Summary and Conclusions

One of the few ways in which an error detection scheme based on duplication and matching can fail is if the two modules whose outputs are compared fail simultaneously in exactly the same way. We have discussed the possible causes of such common mode failures and presented a formal definitions of CMFs that is applicable even when the two modules are not identical circuits. In order to reduce the probability of CMFs in the two modules, the circuits and fabrication process used in their implementations must be as different as possible. We have discussed techniques for developing such different implementations using NMOS or CMOS VLSI chips and evaluated their costs, in terms of resources and design time, and their effectiveness in reducing CMFs.

With both NMOS and CMOS VLSI chips the probability of CMFs can be reduced by using dual implementations and simple topological modifications of a carefully selected group of subcircuits. Converting a positive logic module to negative logic is a critical step in developing modules with independent failure modes. While the problems encountered in converting NMOS and CMOS circuits are similar, converting a CMOS circuit is somewhat easier due to the availability of both *p*-channel and *n*-channel FETs. With both technologies, the conversion can be done at virtually no performance penalty and with extra design effort that is negligible when compared to the effort required to design and implement a VLSI chip. The ability to reduce CMFs at a relatively low cost is yet another reason to use duplication and matching, at the level of powerful functional modules, as the general approach for implementing error detection in VLSI systems requiring high reliability.

References

1. A. Avizienis, "Design Diversity - The Challenge of the Eighties," *FTCS-12*, Santa Monica, CA, pp. 44-45 (June 1982).
2. W. C. Carter and P. R. Schneider, "Design of Dynamically Checked Computers," *IFIPS Proceedings*, Edinburgh, Scotland, pp. 878-883 (August 1968).
3. J. Galiay, Y. Crouzet, and M. Vergniault, "Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability," *IEEE Transactions on Computers* **C-29**(6) pp. 527-531 (June 1980).
4. P. S. Kastner, "A Fault-Tolerant Transaction Processing Environment," *Database Engineering* **6**(2) pp. 20-28 (June 1983).
5. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley (1980).
6. D. G. Platteter, "Transparent Protection of Untestable LSI Microprocessors," *10th Fault-Tolerant Computing Symposium*, Kyoto, Japan, pp. 345-347 (October 1980).
7. R. M. Sedmak and H. L. Liebergot, "Fault Tolerance of a General Purpose Computer Implemented by Very Large Scale Integration," *IEEE Transactions on Computers* **C-29**(6) pp. 492-500 (June 1980).
8. R. W. Sherburne, M. G. H. Katevenis, D. A. Patterson, and C. H. Séquin, "A 32-Bit NMOS Microprocessor with a Large Register File," *IEEE Journal of Solid-State Circuits*, (October 1984).
9. M. Takada, "Two-Phase CMOS RISC Design," Unpublished Report, CS Division, University of California, Berkeley, CA (October 1983).
10. K. Takeda and Y. Tohma, "Logic Design of Fault-Tolerant Arithmetic Units Based on the Data Complementation Strategy," *FTCS-10*, Kyoto, Japan, pp. 348-350 (Oct. 1980).
11. Y. Tamir and C. H. Séquin, "Self-Checking VLSI Building Blocks for Fault-Tolerant Multicomputers," *International Conference on Computer Design*, Port Chester, NY, pp. 561-564 (November 1983).
12. Y. Tamir and C. H. Séquin, "Design and Application of Self-Testing Comparators Implemented with MOS PLAs," *IEEE Transactions on Computers* **C-33**(6) pp. 493-506 (June 1984).
13. J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*, Elsevier North-Holland (1978).