Defect Tolerance on the Teramac Custom Computer

W. Bruce Culbertson, Rick Amerson, Richard J. Carter, Philip Kuekes, Greg Snider

Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto CA 94304

Abstract

Teramac is a large custom computer which works correctly despite the fact that three quarters of its FPGAs contain defects. This is accomplished through unprecedented use of defect tolerance, which substantially reduces Teramac's cost and permits it to have an unusually complex interconnection network. Teramac tolerates defective resources, like gates and wires, that are introduced during the manufacture of its FPGAs and other components, and during assembly of the system. We have developed methods to precisely locate defects. User designs are mapped onto the system by a completely automated process that avoids the defects and hides the defect tolerance from the user. Defective components are not physically removed from the system.

1 Introduction

The Teramac custom computer [1] makes unprecedented use of defect tolerance, which substantially reduces its cost and enables it to have an interconnection network of unusually high complexity. Teramac is FPGA-based, designed for architectural exploration [2], scalable, and capable of running million-gate user designs at one megahertz.

Defect tolerance is a technique that permits Teramac to perform correctly in spite of large numbers of defects introduced during the manufacture of its components and during assembly of those components into the system. Diagnostic tests locate defective resources (gates, wires, etc.) in the completed system and record them in a defect database. Testing methods have been developed to precisely locate defective user resources within the FPGAs and other system components. Compiling software reads the database and uses only resources which are not defective when mapping user designs onto the system. Defective components are not physically removed from the system.

Teramac may not produce correct results if new failures occur while executing user designs. However, following such failures, it can be returned to reliable operation by rerunning the diagnostic tests and recompiling the user designs.

2 Related work

Most previous proposals for defect tolerant machines use a physical architecture with a very regular and symmetric structure [3-6]. Typically the structure is a two dimensional array. Defect tolerance is achieved by introducing one or more extra rows or columns, plus a routing network which can be programmed to select the working rows or columns, leaving those with bad cells unused.

The European Large SIMD Array (ELSA) [6] is an example of such a machine. ELSA contains wafer-scale devices which are 2D arrays of chips. The chips are logically 12 columns by 6 rows of 1-bit processors. Physically, the chips have seven rows, from which six working rows are selected by making cuts with a laser.

Another wafer scale device, WASP (WSI Associative String Processor) [7], also uses a regular architecture. It makes use of two wafers, one for processing elements and local connections, and the other for long distance connections. The wafers are bumped together. The interconnect wafer includes a partial crossbar to replace a defective processing element with a working spare while keeping the order of all of the bits on a bus logically unchanged.

The defect-tolerance schemes described above modify defective devices, possibly all different physically, to pro-

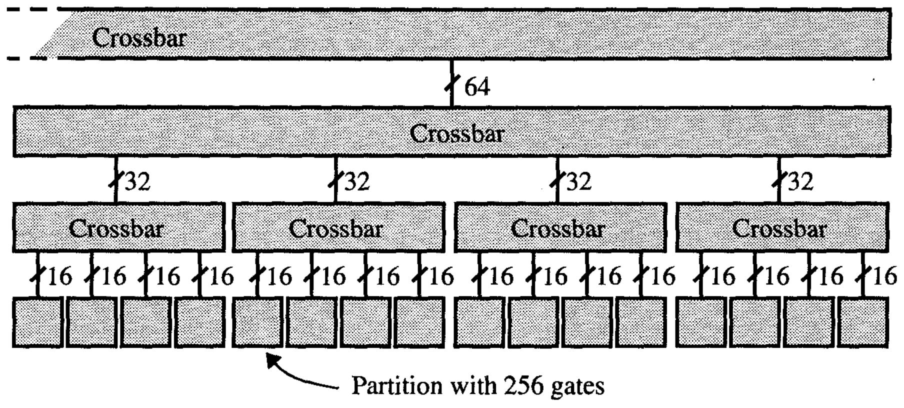


Figure 1. The lower levels of a typical hierarchical interconnect network. Networks which satisfy Rent's rule are sufficient for routing almost all digital designs, regardless of topology. Rent's rule says that the number of wires connecting a subdesign to the rest of the design is proportional to the square root of the number of gates in the subdesign. In this example, the number of wires leaving a partition of the network is exactly the square root of the number of gates in the partition.

duce repaired devices which are all identical logically. This is fundamentally different from the Teramac scheme, in which every specific machine is different. In the case of a reconfigurable machine, the former scheme presents a target for user designs that is stable across machines. The lack of a stable design target across Teramac machines is not a problem because Teramac includes tools to quickly remap designs. In fact, the same tools created to map many designs onto a single Teramac also provide the ability to remap a single design onto many different instances of Teramac.

Schemes using spare rows or columns are inefficient in the sense that a single defective cell causes the entire row or column to become useless. However, swapping whole rows or columns simplifies the repair network. When Teramac has a defective resource, it can usually be avoided without forcing other good resources to be unused. Teramac needs a rich routing network for mapping user designs and the same network is rich enough to route around single defects.

In the machines described above, symmetrical architectures were chosen to simplify repair. Symmetrical architectures are good targets for memories and systolic arrays but most other interesting logic designs are not symmetrical and are difficult to map onto symmetrical targets. Teramac does not favor user designs of any particular topology.

The possibility of using defect tolerance with FPGAs has been previously described [8-13] but we are not aware of systems that have actually been built. One group [9] concludes that the cost of locating defects and recompiling designs for each FPGA is prohibitive. This is probably true in the case of a mass-produced product that uses an FPGA as an ASIC replacement. Teramac has shown that it is not true in the case of FPGAs used in a custom computer.

One effort [14] has explored the possibility of building a wafer-scale FPGA. Defects in user resources are avoided using the same scheme Teramac uses. The difficult problem of power shorts and defective critical signals is solved using laser cutting. A test chip has been fabricated.

3 Why use defect tolerance?

When components like integrated circuits are manufactured, many are rejected because a tiny fraction of the resources they contain are defective. If a system can tolerate such defects and use the remaining resources, component yield increases and cost drops, in exchange for a negligible loss of resource capacity. For this reason, defect tolerance substantially reduces Teramac's cost. Quantitative examples of component cost reductions are given later in this paper.

In addition to reducing cost, defect tolerance is particularly appropriate for Teramac because of two goals we have for the system—goals that are unusual among existing custom computers.

First, we have the objective that Teramac be able run user designs of arbitrary topology—that it would not be limited, for example, to systolic designs. Richard Rent [15] observed that, for typical digital designs, the number of signals crossing the boundary of a logic design partition was roughly proportional to the square root of the number of gates in the partition. So, to accommodate arbitrary user designs, Teramac includes a hierarchical interconnect network satisfying Rent's rule at every level. Figure 1 is an example of such a network.

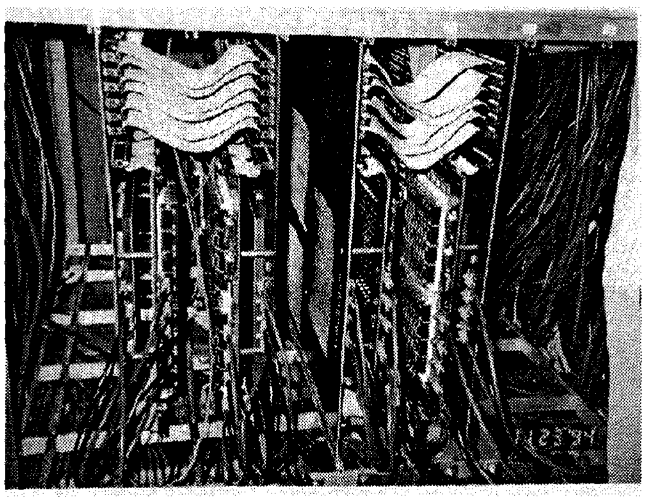


Figure 2. A portion of a Teramac system, showing the PCBs and ribbon cables that implement a part of Teramac's hierarchical interconnect network.

Teramac's interconnect network is implemented in FPGAs, multichip modules (MCMs), and printed circuit boards (PCBs). Rent's rule predicts that we need roughly 38,000 distinct signals running between PCBs in a 16-board Teramac. We are not aware of a practical way to implement that many inter-board signals with acceptable reliability if no defects are allowed. A similar problem exists with the MCMs, which need about 6000 signals. Thus, it is not feasible to build the Teramac interconnect network without defect tolerance. Using defect tolerance, Teramac operates successfully with inexpensive ribbon cables connecting its PCBs, as shown in figure 2. Teramac also successfully uses

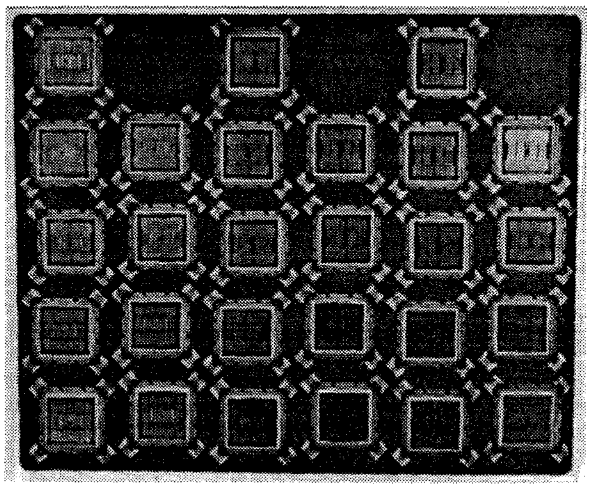


Figure 3. The Teramac MCMs measure 6.1 by 7.4 inches and carry over 6000 signals in 39 layers. This level of complexity would be impractical to manufacture without defect tolerance.

Component	Size	Complexity
FPGA	16.2mm by 16.2mm	0.8 micron CMOS
MCM	15.57cm by 18.80cm	39 layers
PCB	44.45cm by 73.66cm	14 layers

Table 1. Defect tolerance permitted us to design larger and denser components. With more gates per component, signals cross between fewer components and, thus, are faster.

MCMs, shown in figure 3, with twice the number of layers recommended by the MCM manufacturer [16].

Defect tolerance also permits the interconnect components to be larger and denser. Sizes of the FPGAs, MCMs and PCBs are shown in table 1. This helps keep the total number of components and the volume of the complete system within practical limits.

The second goal that makes defect tolerance appropriate is the desire to completely automate the process of mapping user designs onto Teramac. To be fast and have a high likelihood of success, automatic routers need abundant routing resources. So this is another reason, in addition to Rent's rule, for Teramac to have a very ample interconnect network.

Also, automatic design mapping provides a natural way to shield the user from the additional complexity that arises due to defect tolerance. No manual steps are required for the mapping software to read the defect database and map around the defects. Thus, Teramac achieves the benefits of defect tolerance without inconveniencing users and, in fact, the majority of Teramac's users have been unaware of its use of defect tolerance.

4 The Teramac defect tolerance scheme

Defect tolerance is implemented on Teramac by first detecting and precisely locating the defective resources using diagnostic tests. Information about the defects is then stored in a defect database. Finally, when user designs are mapped onto Teramac, the mapping software reads the database and maps the design only onto good resources. This is shown in figure 4.

4.1 Finding defects

Several things complicate the problem of locating the defects. For example, it is not possible to energize and sense arbitrary points in an integrated circuit. Conse-

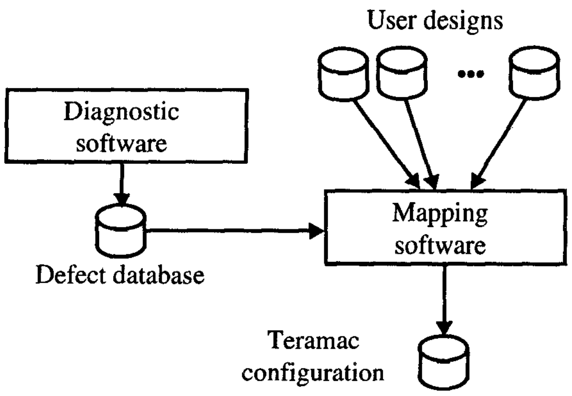


Figure 4. The Teramac defect tolerance scheme. Diagnostic software precisely locates the defects in the system. Mapping software implements users designs in such a way that defective resources are not used.

quently, resources cannot be tested individually. Instead, we combine many resources into test circuits that include some resources that can be probed. A further complication is that no resources are known to be good at the outset of testing. Hence, it is not possible to construct test circuits with many known-good resources and just a single resource-under-test. Thus, when a test fails, there's no easy way to distinguish the bad resources that cause the failure from the good resources which are also involved in the same test.

Most of our diagnostic tests combine numerous resources into signature generators, circuits that produce long, non-repeating sequences of pseudo-random numbers. On each clock cycle, a new number is produced as a function of its predecessor. A correctly operating signature generator, initialized with a specific value and clocked a given number of cycles, will compute a unique final number, called a *signature.* Any error computed by the circuit will be propagated to the end of the sequence, resulting in an incorrect signature. Thus, signature generation is a sensitive test of the resources constituting the generator. Our signature generator circuits store the current value of the sequence on each clock cycle in FPGA registers. The FPGA scan chain permits the registers to be read and written, which in turn allows the sequence to be initialized and the final signature to be checked.

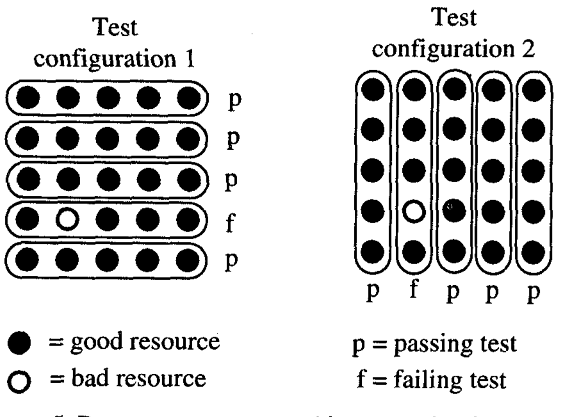


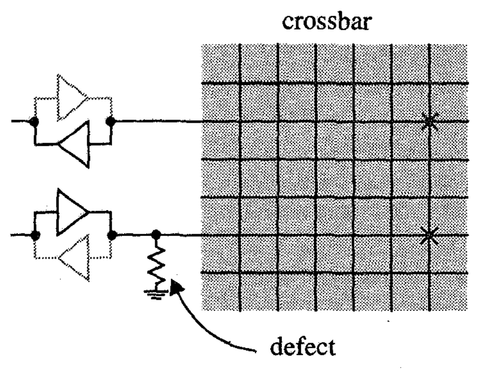
Figure 5. Resources are grouped into test circuits and are declared good after they have participated in a passing test. Thus, after test configuration 1 is run, all the resources except those in the fourth row are declared good; the status of those in row 4 remains unknown. In configuration 2, the resources are grouped differently and tested again. After both configurations are run, all the good resources are identified and, indirectly, the defect is located.

If a test fails (i.e. a signature generator circuit computes the wrong signature), the circuit must contain a defective resource. Unfortunately, a failing circuit usually also contains many good resources. Since we cannot distinguish the bad resources from the good in a failing circuit, we build the defect database entirely from information obtained from passing tests. For the moment, assume that if a test passes, then all the resources used by the test must be good.

In order to find all the good resources despite failing test circuits, we use redundant testing. Specifically, each resource is tested many times, each time grouped with different other resources. This makes it very likely that a good resource will eventually be tested in a group composed entirely of good resources and, hence, will be shown to be good. When all the redundant tests are complete, resources that have not been shown to be good are declared to be bad and are noted in the defect database. Redundant testing is illustrated in figure 5.

Unfortunately, the scheme just described occasionally fails to detect a defect. We observe test failures which are *unexplained*: tests that fail but all of whose resources are shown to be good by other tests. This occurs when there is a resource which is defective in such a way that it works correctly in some test circuits but not in others. Figure 6 is an example of how such a situation can arise when a crossbar wire is shorted to ground through a significant resistance. When the wire is configured to be driven directly by a buffer, the short is too weak to have an effect. However, when the same wire is driven via several crossbar switches, which themselves have significant resistance, the short overpowers the signal. We call such defects *weak*.

We studied the actual unexplained failures that occurred in testing some Teramac FPGAs and noticed an obvious pattern. Typically, if there were any unexplained failures, there would be 50 to 100. The sets of resources used by these tests would have almost no overlap with one conspicuous exception: there would be one resource that was common to



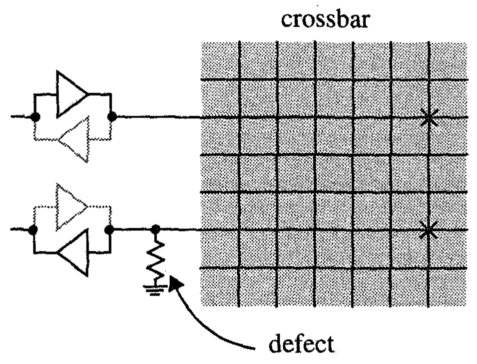


Figure 6. A set of resources is shown in two different configurations. One crossbar line is defective—it is shorted to ground through some resistance. In the top configuration, the defective line is driven directly by a buffer that overpowers the defect, causing the circuit to perform correctly. In the bottom configuration, the defective line is driven via two crossbar switches, which also have significant resistance. Here, the signal is weaker than the defect and the circuit fails. We call defects, like this one, *weak* if they cause some circuits to fail but not others.

every unexplained failure. The common resource was a weak defect.

The pattern suggested a fairly simple way to improve our defect-finding scheme so it would find weak defects. The improved scheme, and in fact any successful scheme, relies on the set of tests being sufficiently rich that every defective resource will fail at least one test. For example, every crossbar wire must be tested in a circuit that drives it via several crossbar switches. Also, the set of tests must be rich enough, and the weak defects rare enough, that every weak defect will be the only defect used in some test.

The improved scheme first performs all the previously described steps: the tests are run, the resources used by passing tests are assumed to be good, and the resources which are used by no passing tests are added to the defect database. The improved scheme then creates a list of all the failed tests which are unexplained. The list is searched to find the resource which is used by the largest number of unexplained failures. If that resource is used by many tests, it is declared bad and added to the defect database. The tests in the list which used that resource are no longer considered unexplained and are deleted from the list. The search is repeated until the list is empty or the search fails to find a resource which is used by many tests. In the later case, which occurs very infrequently, all the resources used by the remaining unexplained tests are added to the defect database. We have found this scheme to be reliable after several years of use.

Some resources produce logically correct results but are slow, with the result that user designs using them fail at their predicted speed. We treat such resources like other defects, i.e. we don't use them. Alternatively, it would be possible to use them if we used their actual delays in our timing analyzer.

We developed a method for measuring the speed of resources. We configure part of Teramac as a frequency counter. We also configure a free-running ring oscillator that includes the resource we want to measure and is connected to the input of the counter. The ring oscillator also includes many other elements to reduce its frequency into the range of our frequency counter. Running this circuit, we measure the period of the ring oscillator. The precision of this measurement is excellent since it is determined by the crystal-controlled system clock that clocks the counter.

Next, we modify the ring oscillator to delete the resource being measured but, otherwise, change the oscillator as little as possible. The period of the new oscillator is measured by running this circuit. Subtracting the two periods, we obtain the delay through the deleted resource. In addition to finding slow resources, we have used this technique to verify the predicted speed of our FPGAs and to measure process variations across the ICs.

4.2 Mapping around defects

The algorithms for mapping user designs onto custom computers are similar to those used for VLSI design. A gate or wire in the user design is assigned a physical gate or wire in the target system. The physical resource is then removed from the pool of resources available for assignment. This continues until all gates and wires in the user design are assigned. Adapting this algorithm for defect tolerance is easy—defective resources are simply removed from the pool of available resources before assignment begins.

Mapping user designs is difficult and, as a result, mapping tools are never able to use 100% of the available resources.

Consequently, custom computer designers always add an extra margin of gates and wires beyond the target capacity. We did not substantially increase the margin to support defect tolerance on Teramac.

5 Design and construction for defect tolerance

Defect tolerance had a large impact on the design and construction of Teramac. Many features were added to the design to minimize the chance that defects would substantially decrease the capacity of a system. Several steps in the assembly of Teramac systems differ from systems without defect tolerance.

The Teramac FPGAs, MCMs, and PCBs each contain critical areas, areas in which a defect would cause a substantial loss of capacity. To maximize yields of these components, a considerable effort was made to minimize their critical areas. The critical areas of components are thoroughly tested before the components are assembled into systems and failing components are rejected. The noncritical areas of components are not comprehensively tested and the testing must detect many noncritical defects for a part to be rejected.

The FPGAs include state machines for reading and writing configurations and scan chains. A defect in a state machine renders the entire chip useless so the state machines are considered critical. Careful design limited the critical area of the FPGA to 7%. The critical area of the FPGAs is tested during wafer testing. The MCMs and PCBs carry many wires which are part of the interconnect hierarchy. These wires are not regarded as critical. Other wires carry clock and control signals to the FPGAs. Defects in these wires make FPGAs unusable; hence, they are considered critical. There are also MCM wires which carry signals from FPGAs to memories; these are also regarded as critical. MCMs contain 6030 wires, of which 4.6% are critical. PCBs contain 8198 wires, of which 7.8% are critical.

Although the critical wires are tested before assembly, some redundancy was nevertheless included in their design. For example, four copies of each FPGA control signal are supplied to each MCM, each copy connecting to a quarter of the chips. If one of these control lines is shorted to ground during assembly of the MCM, for example, only a quarter of the FPGAs would become unusable. This would be unfortunate but not fatal. There is also redundancy in the memory signals. For example, more than one FPGA can drive each memory signal.

The FPGAs on an MCM are connected in a cylindrical mesh network for the purpose of reading and writing configurations and scan chains. This is shown in figure 7. Messages from a controller board to a specific FPGA are first routed to an FPGA at the top end of the cylinder. The messages contain routing headers, with a sequence of "go-down" and "circle" instructions, to direct them to their destination FPGAs. As a result, there are multiple paths between each FPGA and the controller board. If one path to an FPGA is blocked by another FPGA which has failed, an alternate path can be found.

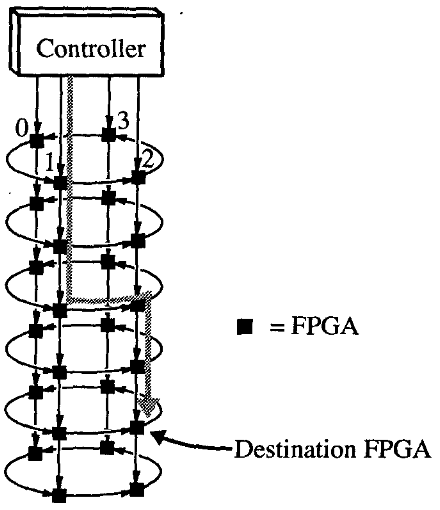


Figure 7. The message network for sending data and configurations between the controller and the FPGAs. To better tolerate defects, the network is redundant. A message following the example path would start with a routing header as follows: "channel 1, down, down, circle, down, down".

The noncritical area of the FPGAs is spot-checked during wafer testing; wafer testers are too limited to permit thorough testing. After considerable trial-and-error, we established a criterion for scoring these tests such that chips that passed had a good likelihood of having high capacity in the system. Testing was complicated by the fact that wafer testers normally reject chips and stop testing after the first test fails, a policy that is incompatible with defect tolerance. It was not easy to make our tester implement our system of scoring multiple tests.

6 Results

Two Teramac systems have been built, one with 8 boards and one with a single board. They have received heavy use and many of the users have been experts in neither defect tolerance nor custom computers. Teramac has met both its performance and capacity goals in spite of thousands of

Resource type	Number	Number defective	Percent defective
Programmable logic cell	221,000	23,000	10.4%
Interchip signal	145,000	13,800	9.5%
Crossbar line	4,880,000	146,000	3.0%
Crossbar buffer	2,420,000	37,000	1.5%
Total	7,670,000	220,000	2.9%

Table 2. Actual defects in the prototype eight-board Teramac system. Interchip defects can be caused by defects in chip I/O pads, chip wire bonds, MCM wires, PCB wires, and/or ribbon cable wires.

defects in the systems. Table 2 lists the quantities of various kinds of defects found in an actual system.

6.1 Cost reduction due to defect tolerance

Defect tolerance significantly reduced the cost of most of the Teramac components. The FPGAs, MCMs, and interboard cables are good examples. The eight-board Teramac system contains 864 FPGAs. Two hundred and seventeen of these are free of defects. Thus, defect tolerance increased the yield (and, hence, decreased the cost) of usable FPGAs by a factor of four. When first approached, the MCM vendor had no idea how to price an MCM with defective wires. A contract was finally negotiated in which Hewlett-Packard paid the standard price for perfect MCMs and, for every perfect MCM, also received, free of charge, another MCM with only critical wires guaranteed to be perfect. This, in effect, cuts the MCM price in half. We verified that all the free MCMs we received did, in fact, have defects. Thus, defect tolerance at least doubles the MCM yield. Defect tolerance allows us to interconnect PCBs with one of the least expensive interconnect technologies: ribbon cables and insulation-displacement connectors.

6.2 Costs and problems due to defect tolerance

In the process of designing and building Teramac, we encountered some extra costs and problems due to our use of defect tolerance. Some of these are intrinsic to defect tolerant systems while others can be avoided with hindsight.

Ideally, a defect affecting a fraction of the physical area of a part will cause at most a loss of a similar fraction of the functional capacity of the part. We experience two kinds situations in which Teramac loses functional capacity out of proportion to the defect area. The first kind is caused by single physical structures that affect many logical structures. For example, the Plasma configuration memory is composed of 128-bit words and the programmable gates have approximately a hundred configuration bits. Unfortunately, there are configuration words that control the programming of 32 different gates. As a result, a defective configuration word, which can be caused by a very small physical defect, can render 32 gates unusable. This could have been avoided by dedicating a single configuration word for the programming of each gate.

Rent's Rule is the second reason we lost disproportionately large amounts of functional capacity. Rent's Rule says that a partition of N gates will be connected to the rest of the circuit by a number of wires that is proportional to the square root of N. The automatic mapping of designs depends on Rent's Rule being satisfied. Hence, if some fraction F of wires in the L'th hierarchy level are defective, then only

$$((1 - F)^2)^L$$

gates may be used in the part of the hierarchy below the defective wires. For example, if 1% of the wires in the third hierarchy level are defective, 5.9% of the gates cannot be used.

The diagnostic tests have costs associated with them. First, it takes time to write them and they are far more complicated than tests that merely detect defects, rather than locate them. Second, development of an effective suite of tests requires experimentation. Early versions of our suite missed certain types of defects so additional tests had to be written. Third, it takes time to run the tests. Currently, it takes a week to run the suite on a single board. There are obvious ways to make our tests run faster but we have not had time to implement them. For example, we currently test one FPGA at a time, although the hardware supports testing hundreds simultaneously.

Although the critical areas on all the FPGAs are tested during wafer test, the critical areas on a few FPGAs nevertheless have failed to operate in the MCMs. Increased attention to static discharge during MCM assembly has improved, but not eliminated, this problem. To date, we have not developed a process to replace these chips, which number about 2% of the total in our eight-board system. Although disappointing, this failure rate has not prevented Teramac from attaining its capacity goal.

7 Conclusion

Teramac is a large, working custom computer that has many users. It makes unprecedented use of defect tolerance, which substantially reduces its cost. For example, the cost of its FPGAs is reduced by a factor of four. Further-more, defect tolerance makes it possible to build a hierarchical interconnect network that is dense enough to route arbitrary user designs. The network uses inexpensive and readily available ribbon cables and MCM technology that would have been prohibitively unreliable without defect tolerance. We have developed new strategies for designing and building defect-tolerant systems and locating their defects. We have also developed design mapping software that automatically maps around defects and shields the user from the complexity that arises from defect tolerance. The result is a robust custom computer that runs applications a hundred times faster than a workstation and is built largely from components that normally would have been discarded.

References

[1] R. Amerson, R. Carter, B. Culbertson, P. Kuekes, G. Snider, *Teramac—Configurable Custom Computing*, Proceedings of the 1995 IEEE Symposium on FPGA's for Custom Computing Machines.

[2] B. Culbertson, R. Amerson, R. Carter, P. Kuekes, G. Snider, *Exploring Architectures for Volume Visualization on the Teramac Custom Computer*, Proceedings of the 1996 IEEE Symposium on FPGA's for Custom Computing Machines.

[3] J. L. Kelly and P. A. Ivey, *Defect Tolerant SRAM based FPGAs*, IEEE International Conference on Computer Design, 1994, pages 479-482.

[4] R. Libeskind-Hadas and C. L. Liu, *Fast Search Algorithms for Reconfiguration Problems*, IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 1991, pages 260-273.

[5] W. Che and I. Koren, *Fault Spectrum Analysis for Fast Spare Allocation in Reconfigurable Arrays*, IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 1993, pages 60-69.

[6] A. Boubekeur, J. L. Patry, G. Saucier, M. Slimane-kadi, and J. Trilhe, *Lessons Learnt from Designing a Wafer Scale 2D Array*, IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 1993, pages 137-146.

[7] M. B. A. Hussaini, H. Bolouri, and R. M. Lea, *Defect and Fault Tolerant Interconnection Strategies for WASP Devices*, IEEE Transactions on Components, Packaging, and Manufacturing Technology - Part B, Vol. 18, No. 3, August 1995, pages 416-423.

[8] K. T. Johnson et al, *General Purpose Systolic Arrays*, IEEE Computer, November 1993, pages 20-31.

[9] Neil J. Howard, Andrew M. Tyrrell, Nigel M. Allinson, *The Yield Enhancement of Field-Programmable Gate Arrays*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol 2, No. 1, March 1994.

[10] Jason L. Kelly, Peter A. Ivey, *Defect Tolerant SRAM based FPGAs*, Proceedings of the IEEE International Conference on Computer Design, 1994, pages 479-482.

[11] Nobuo Tsuda, Tsutomu Ishikawa, Yukihiro Nakamura, *Totally Defect-Tolerant Arrays Capable of Quick Broadcasting*, Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, November 1995, pages 117-125.

[12] Adit D. Singh, *ADTS: An Array Defect-Tolerance Scheme for Wafer Scale Gate Arrays*, Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, November 1995, pages 126-133.

[13] S. Goldberg, S. J. Upadhyaya, *Utilizing Spares in Multichip Modules for the Dual Function of Fault Coverage and Fault Diagnosis*, Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, November 1995, pages 234-242.

[14] G. H. Chapman and B. Dufort, *Making Defect Avoidance Nearly Invisible to the User in Wafer Scale Field Programmable Gate Arrays*, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 1996, pages 11-19.

[15] B. Landman and R. Russo, *On a Pin vs. Block Relationship for Partitions of Logic Graphs*, IEEE Transactions on Computers, December 1971, pages 1469-1479.

[16] R. Amerson, P. Kuekes, *The Design of an Extremely Large MCM-C—A Case Study*, The International Journal of Microcircuits and Electronic Packaging, Volume 17, Number 4, Fourth Quarter 1994, pages 337-382.