Metamorphic Testing and its Application on Hardware Fault-Tolerance

Jie Liu
Dept. of Electrical and Computer Engineering
University of Wisconsin – Madison
Madison, WI 53726
liu9@wisc.edu

Abstract – **In testing, people normally use the "golden model" to judge the correctness of the tested unit. However, in some cases the golden model is difficult to produce, and in some cases the outputs do not need to exactly match the golden model to be considered correct. Metamorphic testing was first introduced to software testing to target such problems in software. In this paper, we will explore the idea of applying metamorphic testing to hardware fault-tolerance. We discuss implementing it using two methods: hardware redundancy and time redundancy. During our case study, we will compare those implementations with a Double Modular Redundancy (DMR) implementation due to their similar functionalities, and discuss the differences in performance, area and fault detection. In addition, we will look at the disadvantages of metamorphic testing in hardware such as single-point of failure, and its inability to give a rational judgment of the correct output if the test fails. Last, we will discuss possible solutions to overcome those disadvantages.**

I. INTRODUCTION

The most common method of verification is to apply a set of tests to a targeted function unit, and verify if the actual output of the function unit matches the expected output of the function that it is supposed to implement. An "oracle" is the methodology to determine whether or not the output of the function unit is correct based on the expected outputs [17]. However, implementing the oracle is expensive and time-consuming. This is also a reason that often times the "oracle" procedure is actually done manually; in this case, the testing/verification process becomes time-consuming and error-prone [17].

The "oracle problem" can be a challenge. Applications in machine learning [14], scientific computing, simulation, optimization, etc. are sometimes "non-testable programs" [18], because the expected outputs are hard to find or do not exist. Computer scientists proposed several methods to address the oracle problem in software verification. A "niche oracle" [18] is a methodology that tests only a subset of all possible test cases where the golden model exists. However, that subset of test cases with a known golden model is often the simplest subset among all test cases. Oftentimes that subset is not sufficient to fulfill the testing purpose, because the more complicated cases are more likely cause errors [18], and the "niche oracle" is unable to test those complicated cases. Another proposed solution to the oracle problem is to use a "pseudo-oracle" [3], where different programming teams are given the same specification of a program, and the teams develop independent versions of the program. During execution, all versions of that program run with same inputs, and their outputs are compared against each other. If one version produces an output different from the rest, then we can decide that version is faulty, and the majority answer is correct. Unfortunately, sometimes there is only one (or only one reasonable) implementation of a program, and multiple versions of the program simply do not exist [13].

To target the issues with "niche oracle" and "pseudo-oracle", metamorphic testing (MT) [4] was introduced as an alternative solution to the oracle problem. Metamorphic testing verifies the correctness of the program using the outputs of multiple executions of the program on different data values, and no oracle is needed. A metamorphic relation (MR) describes the relationship between generated output values based on the relationship between input values. This relation is used to modify the initial test data to create follow-up test cases, and to transform the initial test outputs into a set of expected outputs for the follow-up cases. Metamorphic testing is used with an existing test selection strategy to create follow-up test cases based on initial test cases and the metamorphic relation. When initial test cases do not reveal any failure, MT can be used to further verify the program.

A simple case study in [5] describes a tested program that has a task that computes the sine function. The metamorphic relation of the program uses the property $\sin x = \sin(180 - x)$, to describe a relationship between the output given an input value of x and the output given an input value of $180 - x$. Let $t = 57.3$ be an initial test case; the output of the program is then determined to be $p(t) = 0.8415$. However, assume that this output is not initially known to be correct, because an oracle is not available. The metamorphic test would use a follow-up input value of $t' = 180 - 57.3$. If the execution of this follow-up test produces the output $p(t') = 0.8402$, the test case and its metamorph do not produce outputs with the required relationship (i.e., in this case, they do not exactly match), and thus a fault is detected.

Identity-based verification techniques like *program checkers* [1], which run alongside with a program to check the correctness, also compare the outputs of multiple executions of a program and check if those outputs match a certain identity. These techniques have also long been used in software testing. There are in fact two major differences between MT and the other identity-based verification techniques. First, metamorphic testing is only used as follow-up test cases. Before applying MT, we must already have a test selection strategy and test cases. We use MT when existing test cases are successful and do not reveal any failure. Second, the metamorphic relations can be any expected relations among outputs of multiple executions and the given input, not only identity properties. An example given in [17], a differential equation solver application uses the convergence relation as MR during metamorphic testing.

The MT technique has been used in software fault-tolerance for over ten years, and it is indeed an effective solution to the oracle problem. However, MT has not yet been practiced in hardware fault-tolerance. In this paper, we will extend the concept of MT to hardware fault-tolerance, and explore the possible implementations and usability of MT in hardware.

II. Metamorphic Testing in Hardware

We propose two different implementations of hardware-based metamorphic testing: time redundancy and hardware redundancy. Both implementations preserve the properties of MT as used in software fault-tolerance. MT, whether implemented for software or hardware, requires a test selection strategy. In this case, for hardware-based MT, our test selection strategy is every valid input set that enters the system.

A. *Time-Redundant Metamorphic Testing*

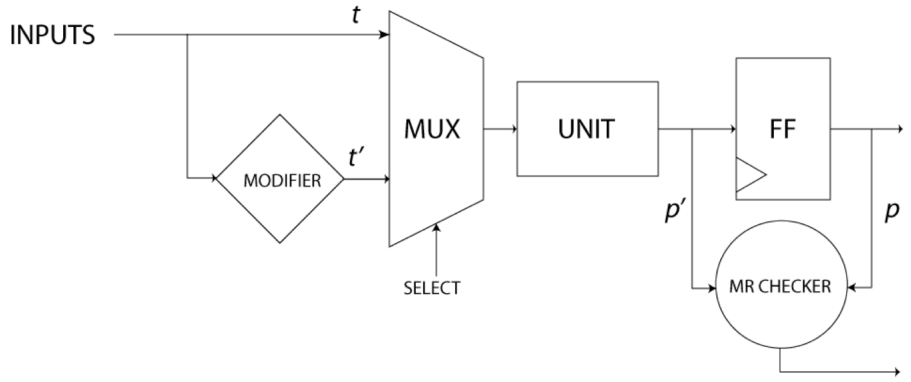

Figure 1. Metamorphic testing with time redundancy

Implementing MT using time redundancy is accomplished in a similar approach as "Recomputing with Shifted Operands" (RESO) [15], where two operands are added by ALU, and during the next execution, those two operands are shifted by one bit, and the two results are checked if the latter one is twice the value as the previous one. In our case, MT can be applied to much smaller modules or function units than ALU. Implementing MT using time redundancy is precisely mapped to the MT procedure in software. As shown in figure 1, the control logic first enables the tested unit to process the original test case *t*, and the result *p* is temporarily stored in a flip-flop. Then the tested unit processes a modified test case *t'*, and the second output of the tested unit is *p'*. The MR checker compares results from both executions and determines if they match the expected metamorphic relation, or else we can conclude a fault is present.

B. *Hardware-Redundant Metamorphic Testing*

The implementation of MT with hardware redundancy has the advantage that executions of the original test case and the follow-up test case can generally be accomplished in parallel, and this parallelism is relatively easier to achieve than it would be in software. The structure of hardware-redundant MT resembles the structure of a duplex modular redundant (DMR) system. A DMR is the simplest technique of the general N-modular redundant (NMR) system [12], where the system detects faults by having two copies of function unit that receive same input, and their outputs are compared by a voter that determines if the outputs are identical; if not, then at least one of the unit is faulty.

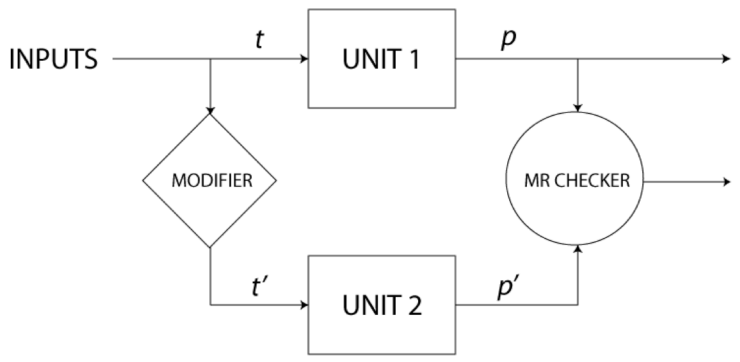

Figure 2. Metamorphic testing with hardware redundancy

As shown in figure 2, there are two duplicate copies of the tested unit. The original test case *t* is goes to the first unit, and a modified test case *t'* goes to the second unit. The outputs of two units *p* and *p'* are then checked for the metamorphic relation. If the tested unit is simple enough, hardware-redundant implementation of MT requires less control logic and storage than the time-redundant implementation.

III. Case Study: CMS Cluster Weighting Module

For the purpose of comparing area overhead, performance, overhead, and effectiveness of MT and different hardware fault-tolerance techniques, in this article we will perform a case study on the Cluster Weighting module, which is a part of the level-1 trigger detector designed for the Compact Muon Solenoid (CMS) project at the Large Hadron Collider (LHC).

A. *Large Hadron Collider (LHC)*

At the European Organization for Nuclear Research (CERN) on the border of France and Switzerland, thousands of scientists and engineers collaborate to contribute to the Large Hadron Collider (LHC) project. The LHC is a large circular tunnel that is 175 meters (574 feet) underground and 27 kilometers (17 miles) in circumference, and it is used towards research in particle physics. The purpose of LHC is to create frequent "big bangs" in a lab environment, so that the particle physicists have the opportunity to discover new particles that are only hypothetical [8]. The Large Hadron Collider collides protons at nearly the speed of light. The energy of collision is about 7 Tera Electron Volts, which is enough to produce many new particles. LHC has four different detectors, and each is an individual project: CMS, ATLAS, ALICE, and LHCb [11]. The four detectors are placed alongside the tunnel, and the velocity of protons is timed so that they collide at the center of each detector.

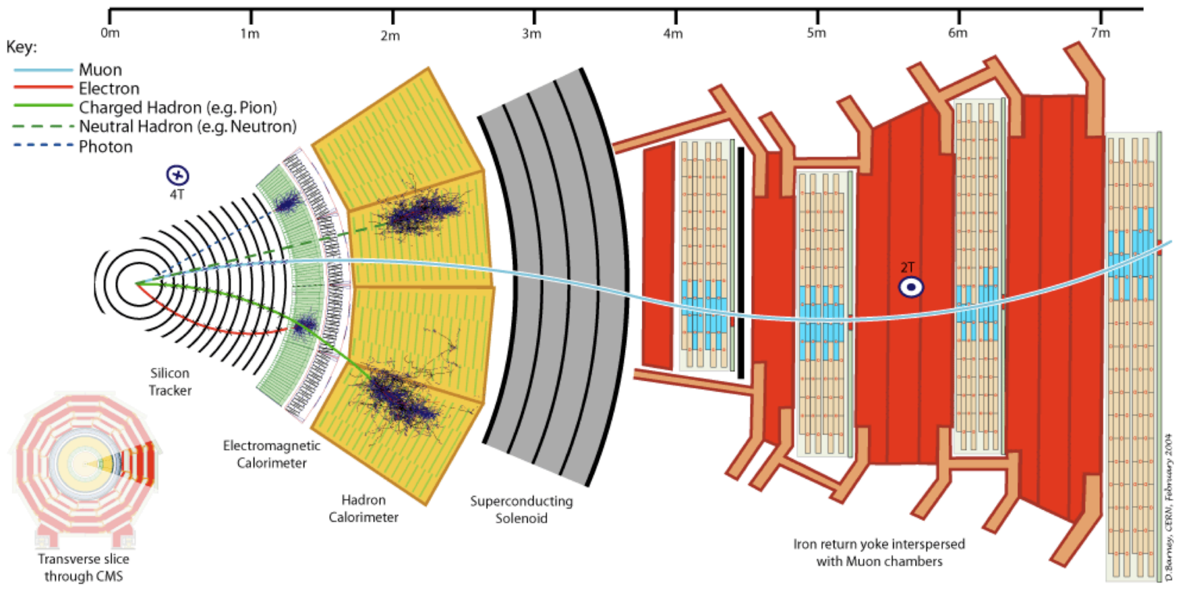
0m
1m
2m
3m
4m
5m
6m
7m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
Transverse slice through CMS
D.Barney, CERN, February 2004

Figure 3. CMS detector slice [11]

B. *Compact Muon Solenoid (CMS)*

The Compact Muon Solenoid (CMS) project is a general-purpose experiment that studies particle physics at high-energy scale. Through the finding of theoretical Higgs Boson particle, we can study the mechanism of electroweak symmetry breaking [6].

Due to the high-speed collision occurrence, it would be too costly to store all the collision data and then analyze it. The CMS detector uses different trigger system to only capture interesting data, and discard non-eventful data. Figure 3 shows a slice of the 4-story-tall CMS detector, where each layer of the detector collects different data from the collision. The level-1 trigger

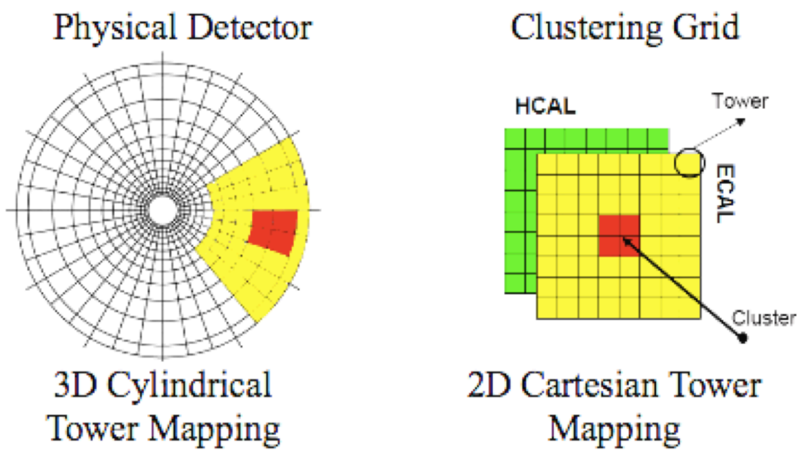

Figure 4. Mapping of ECAL and HCAL [9]

system captures data from Electromagnetic Calorimeter (ECAL) and Hadron Calorimeter (HCAL) [7]. ECAL (the green layer) detects energy of electrons and photons passing through it, and HCAL (the yellow layer) detects the energy of hadrons and neutrons.

C. *Cluster Weighting (CW)*

The Cluster Weighting module is a relatively small module within the level-1 trigger system, and we will conduct a case study of metamorphic testing on this module. As shown at left in figure 4, the calorimeters form a hollow cylinder. The detector transmits data to the processor after mapping the cylindrical coordinates into 2-D Cartesian coordinates, as shown at right in figure 4. The smallest element of the calorimeter is a *tower*, and every 2x2 grid of neighboring towers is a *cluster* [9].

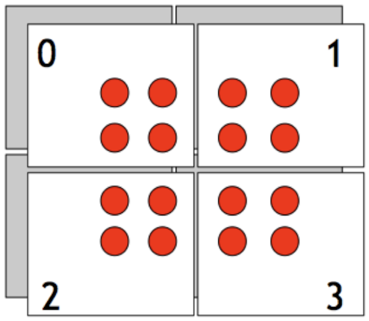

Figure 5. Mapping of 4x4 position points on a cluster

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Table 1. Numerical representation of 4x4 position points on a tower

When a particle hits the calorimeter, the 2x2 grid of towers at that location detects a certain amount of energy. Figure 5 shows a 2x2 grid of neighboring *towers*, which together form a *cluster*. When a particle traverses through the region of *tower 1* and interacts with the calorimeter crystals, *tower 1* would measure a higher energy than *tower 0, 2,* and *3*. The CW module determines particle interaction point on a *cluster* with 4x4 position resolution. As shown in figure 5, each red dot marks a possible interaction point, and table 1 shows the numerical representation of those locations, each representation is a 4-bit binary string. The algorithm to calculate the particle hit location has been designed to optimize the hardware by using only division by powers of two (due to the 16 sub-locations in each 2x2 tower cluster; this is because division by powers of two can be accomplished by shifting the binary value.

Let *w* be CW's output weighting (particle hit location), and let *t0, t1, t2,* and *t3* be CW's input, the energies measured by *tower 0, 1, 2,* and *3*, respectively.

Let d_{eta} = sum of right towers – sum of left towers

$$= (t1 + t3) - (t0 + t2) \qquad (1)$$

Let d_{phi} = sum of bottom towers – sum of top towers

$$= (t2 + t3) - (t0 + t1) \qquad (2)$$

Let *sum* = sum of all towers

$$= t0 + t1 + t2 + t3 \qquad (3)$$

- If $d_{eta} \geq 0$, then *w* is within the right region, else it is within the left region.
- If $| d_{eta} | > sum/2$, then *w* is on the far left/right edge, else it is on the inner layer.
- If $d_{phi} \geq 0$, then *w* is within the bottom region, else it is within the top region.
- if $| d_{phi} | > sum/2$, then *w* is on the far top/bottom edge, else it is on the inner layer.

D. *Metamorphic Relation for Cluster Weighting Module*

Let us design the follow-up test case to be reversed original test case, this way we can explore the symmetrical property of the CW module. For example, if the original test case {*t0, t1, t2, t3*} = {*a, b, c, d*}, and the calculated weight is *w0*; then let us make follow-up test case {*t0, t1, t2, t3*} = {*d, c, b, a*}, which is the previous inputs in reverse order, and the next calculated weight is *w1*.

Let us first look at the simplest scenario: both d_{eta} and d_{phi}, which are calculated from the tower energies, are non-zero. This means that the sum of the right two towers does not equal the sum of the left two towers, and the sum of the bottom two towers does not equal the sum of the top two towers. In other words, the particle interaction point is neither on the vertical nor horizontal center of the cluster. In this scenario, the two weights calculated for the original and morphed cases (*w0* and *w1*, respectively) are mirrored (with respect of each other) about the center point of the cluster. For example, if *w0* is at location 0 (far top and far left), then *w1* is at location 15 (far bottom and far right), as shown in table 1.

The second scenario is when d_{eta} is non-zero but d_{phi} is zero. In this case, the sum of the bottom two towers equals the sum of the top two towers, but the sum of the right two towers does not equal to the sum of the left two towers. In other words, the particle interaction point is on the horizontal center of the cluster. This scenario is special because the algorithm gives priority to the bottom region when d_{phi} is zero. In this scenario, *w0* and *w1* are mirrored about the central vertical axis of the cluster. For example, if *w0* is at location 2 (inner bottom and far left), then *w1* is at location 14 (inner bottom and far right).

The third scenario is when d_{phi} is non-zero but d_{eta} is zero; this means the sum of the right two towers equals the sum of the left two towers, but the sum of the bottom two towers does not equal to the sum of the top two towers. In other words, the particle interaction point is on the vertical center of the cluster. This scenario is special because the algorithm gives priority to the right region when d_{eta} is zero. In this scenario, *w0* and *w1* are mirrored about the central horizontal axis of the cluster. For example, if *w0* is at location 11, then *w1* is at location 8.

The last scenario is when both d_{eta} and d_{phi} are zero. In other words, the particle interaction point is at the center point of the cluster. In this scenario, because our implementation does not have a central point, then both *w0* and *w1* are at location 10, which is to the right and bottom of the center of the grid.

E. *Overhead Comparison*

We have implemented metamorphic testing for CW with both hardware-redundant and time-redundant techniques. We also created a DMR system of CW for comparison. All designs were synthesized on a Vertex-5 FPGA board.

Table 2 shows the area overhead of different implementations, where the first column is a single CW without any fault-tolerant features. We can observe that a simple DMR does not cost much area overhead, because besides a duplicate copy of CW it requires only simple "voter" circuitry to compare the two results.

	Single CW	DMR	HW-redundant MT	Time-redundant MT
# of LUTs	60	73	112	108
# of FFs	17	33	32	57

Table 2. Area overhead of HW fault-tolerance techniques

Both hardware-redundant MT and time-redundant MT require much more logic than DMR, because the process of MR checking is more complicated than a single equality comparator. Meanwhile, time-redundant MT costs more storage area, due to the fact that we need to temporarily store previous executions' results before checking MR.

Table 3 shows the performance overhead comparison among the three fault-tolerance techniques, and the simulated cycle time is 50 ns. We can see that MT with time-redundancy draws a major performance overhead because it takes multiple executions. Due to this nature, MT with time-redundancy will produce the output always one or more cycles later.

	Single CW	DMR	HW-redundant MT	Time-redundant MT
Latency (ns)	7.698	8.791	9.757	58.108
Performance overhead	0%	14.20%	26.75%	655.78%

Table 3. Performance overhead of HW fault-tolerance techniques

F. *Fault Detection*

To test the effectiveness of each fault-tolerance technique, eleven different single stuck-at faults are inserted to each fault-tolerant system, one at a time. For every fault, an exhaustive test is performed, where we tested the system with all possible input sets. However, in the actual operation of the system, the inputs will be the real experimental data sets instead of exhaustive. Therefore a higher amount of inputs are being tested in this project than in the real system. For each fault, the total number of test cases that successfully detect that fault is collected. A higher number indicates that it is more likely to detect that fault in real operation. Due to the fact that the CW module's output the result of comparison on different sub-modules' outputs, each fault is chosen to be a stuck-at fault on the least significant bit along the path of computation so that the fault is hard to detect. Faults are distributed throughout the CW module: close to the primary inputs, close to the primary outputs, and on different computation paths in the middle.

Table 4 shows the overall effectiveness of the different fault-tolerant systems. Each entry is calculated by the equation:

$$\frac{\#\ of\ test\ cases\ detect\ fault\ f}{total\ \#\ of\ exhaustive\ test\ cases} \quad (4)$$

In the above equation, f is an arbitrary fault. This measurement is a statistical calculation of the probability of the system detecting a specific fault for any given input.

These results demonstrate that for the CW circuit, a DMR system and hardware-redundant MT have identical fault detection effectiveness for all tested faults. The reason for that is if a test case is able to excite and propagate the fault to the output of the faulty unit, and that output is checked with the output of the fault-free module (assuming the system only contains a single fault) with a comparator or MR checker, then both DMR and hardware-redundant MT systems would reveal that a fault is present.

In comparison, time-redundant MT is nearly twice as effective as the other two systems in

	DMR	HW-redundant MT	Time-redundant MT
Fault 1	1.73%	1.73%	3.46%
Fault 2	21.82%	21.82%	33.12%
Fault 3	42.78%	42.78%	80.66%
Fault 4	5.70%	5.70%	11.40%
Fault 5	39.45%	39.45%	78.90%
Fault 6	39.45%	39.45%	74.94%
Fault 7	1.58%	1.58%	0%
Fault 8	32.38%	32.38%	0%
Fault 9	16.57%	16.57%	28.84%
Fault 10	13.58%	13.58%	13.58%
Fault 11	14.39%	14.39%	20.77%
Average	20.86%	20.86%	31.41%
Average without f7 and f8	21.72%	21.72%	38.40%

Table 4. Percent of tested inputs where faults were detected using each HW fault-tolerance technique

detecting faults for most cases. It is due to the fact there is only one unit of CW in this system, and if there is a fault and the original input fails to excite and propagate that fault, then the reversed input might have an opportunity to do so in the re-execution. It can be seen as having two attempts to reveal the fault for any given input.

One thing to note is that, despite higher fault detection effectiveness on most faults, the time-redundant MT system fails to detect *fault 7* and *fault 8* on all test cases. This particular incident is caused by the fact that both *fault 7* and *fault 8* modify the result of comparison between *sum/2* and the absolute value of d_{eta} or d_{phi} (equation (1) and (2)). Due to the symmetrical property of CW, d_{eta} (or d_{phi}) is identical for any input and the reversed input. This incident is in fact unavoidable because our MR is built based on the symmetrical property of the CW module. Therefore, even if the fault is excited and propagated, both executions will converge to outputs that satisfy the metamorphic relation regardless.

IV. Issues with Metamorphic Testing in Hardware Fault-Tolerance

After conducting our case study in MT implementation in Cluster Weighting unit, we can conclude several disadvantages with MT in hardware fault-tolerant systems. In this section, we will discuss some of those issues, and our proposed solution to those problems.

A. *Limited Output Information*

Unlike the triple modular redundant (TMR) system, DMR and the proposed MT implementations in hardware are only able to detect the presence of a fault, but they cannot provide an output that is most likely to be correct. In a TMR system [12], three copies of tested unit are compared against each other, and the voter decides the majority result is the correct output. In other words, TMR or other NMR (where N > 2) systems are able to mask of one or more faults.

The proposed solution to this problem is a variation of pair-and-spare [10], where each pair is a MT system, and there are two pairs. As illustrated in figure 6, the MT system on top is the operational pair, and the MT system at the bottom is used as a spare pair. Once there is a fault in

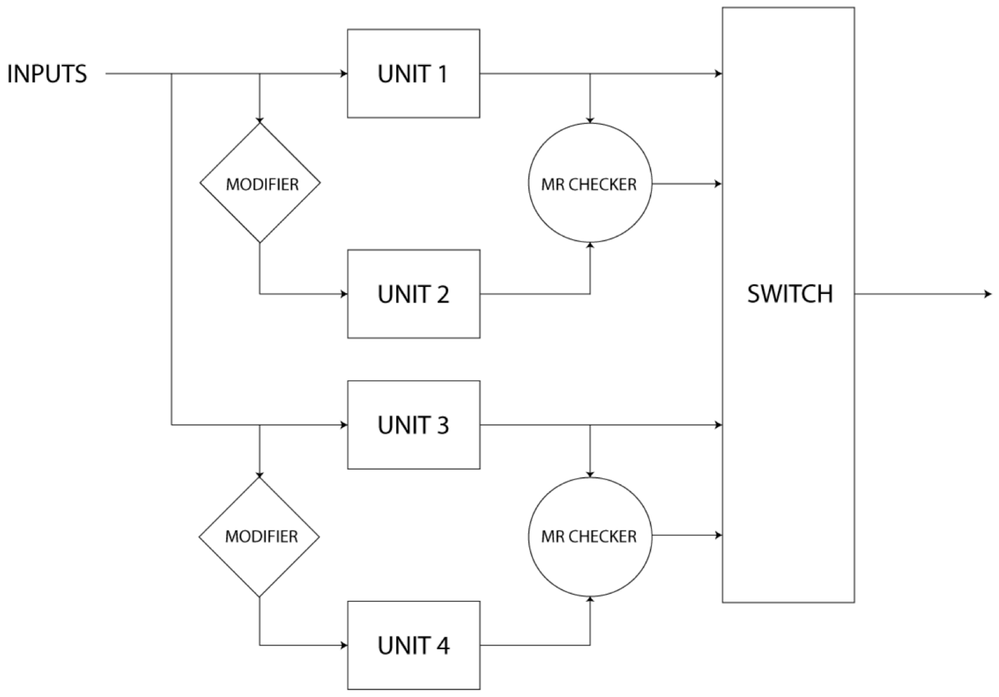

Figure 6. MT pair-and-spare system

the operational pair, the switch determines that the operational pair is faulty and switches the spare pair to be the operational pair, which would produce the correct outputs if the system only tolerates a single fault.

B. *Unable to Detect Certain Faults*

In our case study, we observed that although the overall detection effectiveness is better, there are *fault 7* and *fault 8* who do not affect the metamorphic relation in time-redundant MT, thus those faults cannot be detected in that system. It is to our convenience that those particular faults are likely to be detected by the DMR system, because DMR only compare the outputs of the tested units, and it is not affected by the bottleneck of checking metamorphic relation.

A hybrid MT system in figure 7 is a possible solution to this problem. One part of the system is a time-redundant MT, and the extended part is similar to a DMR system. The MT part of the hybrid system performs re-executions and MR checking as usual, where the DMR part of the hybrid system compares outputs of *Unit 1* and the execution output of original input on *Unit 2*. The switch could determine that a fault is present when either MR checker or DMR comparator fails.

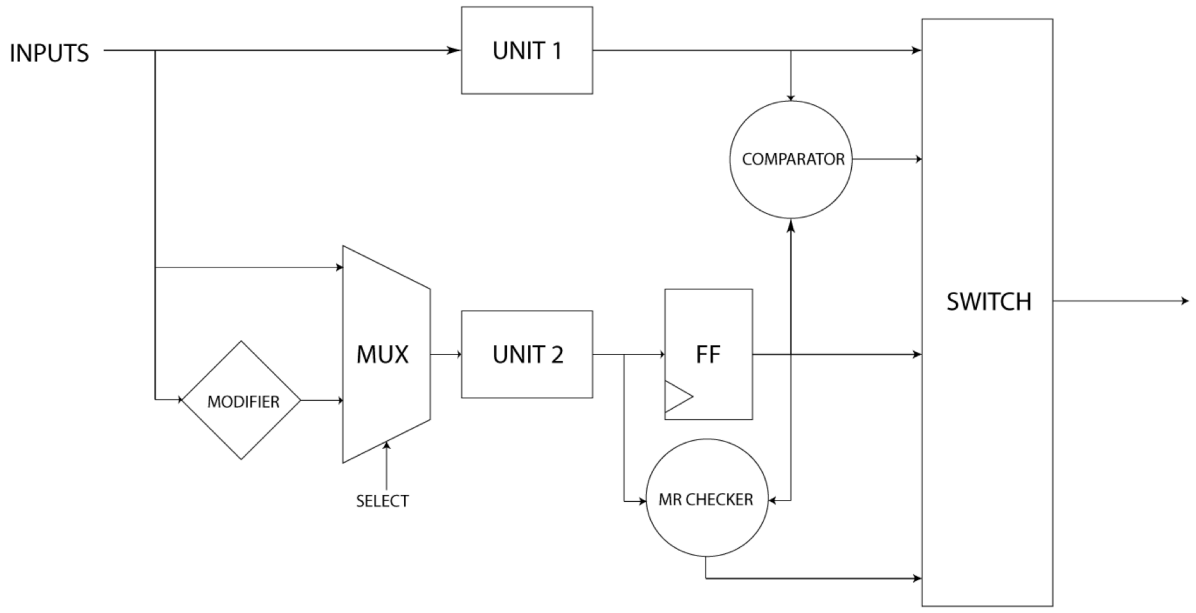

Figure 7. Hybrid MT system

C. *Implementation Issues*

Metamorphic testing has been used in software fault-tolerance for over ten years. However, it has not yet been introduced in hardware fault-tolerance. One of the reasons is that metamorphic relation is hard to find in most hardware units, especially non-computational units (e.g. register files). Another reason is that sometimes it is too expensive to implement MT. As demonstrated in our case study, despite of the simplicity of our MR, metamorphic testing still creates a large area and performance overhead in comparison to a traditional DMR. Needless to say, hardware-redundant MT only demonstrated an identical capability as the DMR system. Although time-redundant MT is more effective in detecting faults in our case study, it suffers a significant performance bottleneck due to re-execution.

To summarize, metamorphic testing should only be conducted when applicable and necessary. MT is most applicable towards the oracle problem, and small computational units where the performance is not critical. As a part of further reading, [5] describes metamorphic relation selection techniques in software, which could be a reference when choosing a MR in hardware.

V. CONCLUSION

In this project, we have presented the basic concept and applications of metamorphic testing in software as an effective solution to the “oracle problem”. We have then explored the idea of applying metamorphic testing in hardware by developing two implementations of MT for hardware fault-tolerance: time-redundant MT and hardware-redundant MT. We conducted a case study on the CMS level-1 trigger system’s Cluster Weighting module. Hardware-redundant MT and time-redundant MT were compared with a traditional DMR, and the cost and performance of these options were evaluated and compared. The results indicate that hardware-redundant MT and DMR are both fast and cost-effective, but that time-redundant MT performs better at revealing faults. Finally, we discussed several limitations of using MT in hardware fault-tolerance and provided possible solutions to them. Future research could develop a more efficient hardware implementation of MT to address remaining performance and cost concerns.

References

[1] M. Blum and S. Kannan, "Designing programs that check their work", Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC '89), p.86-97. ACM Press, New York, 1989. Also Journal of the ACM, 42 (1): p269-291, 1995.

[2] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems", Journal of Computer and System Sciences, 47 (3): p549-595, 1993.

[3] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation", Proceedings of the 8th Symposium on Fault-Tolerance Computing, p.3-9, 1978.

[4] T. Y. Chen, S. C. Cheung, and S. Yiu, "Metamorphic testing: a new approach for generating next test cases", Technical Report HKUST-CS98-01, Dept. of Computer Science, Hong Kong Univ. of Science and Technology, 1998.

[5] T. Y. Chen, F. C. Kuo, T. H. Tse, Z. Q. Zhou, "Metamorphic Testing and Beyond", Proceedings of the International Workshop on Software Technology and Engineering Practice (STEP 2003), p94-100, 2004.

[6] CMS Collaboration, "CMS Physics Technical Design Report, Volume II: Physics Performance", Journal of Physics G: Nuclear and Particle Physics, 34(6), p.955, 2007.

[7] P. Chumney, S. Dasu, J. Lackey, M. Jaworski, P. Robl, W. H. Smith, "Level-1 Resional Calorimeter Trigger System for CMS", Proceedings of Computing in High Energy Physics and Nuclear Physics, 2003.

[8] L. Evans and P. Bryant, "LHC Machine", Journal of Instrumentation, 3 (8), p.S08001, 2008.

[9] A. Gregerson, A. Farmahini-Farahani, B. Buchli, S. Naumov, M. Bachtis, K. Compton, M. Schulte, W. H. Smith, S. Dasu, "FPGA Design Analysis of the Clustering Algorithm for the CERN Large Hadron Collider", Proceedings of the 2009 17th IEEE Symposium on Field Programmable Custom Computer Machines, p.19-26, April 05-07, 2009.

[10] I. Koren and C. Krishna, "Fault Tolerant Systems", Morgan Kauffman Publisher, 2007.

[11] Large Hadron Collider, Available from http://lhc-machine-outreach.web.cern.ch/lhc-machine-outreach/.

[12] F. P. Mathur and A. Avizienis, "Reliability analysis and architecture of a hybrid redundant digital system: Generalized triple modular redundancy with self-repair", in 1970 Spring Joint Computer Conference, Proceedings of the AFIPS Conference, vol. 36, Washington D. C., Thompson, p375-383, 1970.

[13] C. Murphy and G. Kaiser, "Empirical Evaluation of Approaches to Testing Applications without Test Oracles", unpublished.

[14] C. Murphy and G. Kaiser, "Improving the dependability of machine learning applications", Technical Report CUCS-49-08, Dept. of Computer Science, Columbia University, 2008.

[15] J. H. Patel and L. Y. Feng, "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands", IEEE Transactions on Computers, vol. C-31, p589-595, 1982.

[16] G. S. Sohi, M. Franklin, and K. K. Saluja, "A Study of Time-Redundant Fault Tolerance Techniques for High-Performance Pipelined Computers", Digest of Papers, 19th International Symposium on Fault-Tolerant Computing, p.436-443, 1989.

[17] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen, "Metamorphic Testing and its Applications", Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004), Software Engineers Association, Japan, 2004.

[18] E. J. Weyuker, "On Testing Non-testable Programs", Computer Journal, 25(4):465-470, November 1982.