

SYSTEM AND METHOD FOR TESTING MEMORY AT FULL BANDWIDTH

BACKGROUND

[0001] From time to time it is important to test computer memory to be sure that a bit, such as a 1, reads as a 1 when that location is read from. It is also important that when a 0 is stored at that same location at a later time, a 0 is subsequently read therefrom. This test is important in order to detect and correct stuck bits and thus requires a cycle from a "1" to an "0" and back to an "1" to be certain that a bit is not stuck.

[0002] In some situations, the system can not directly access certain memory locations so detecting stuck bits is difficult. For example, the bits used to store a tag for a cache line are often not directly readable. Also, error correcting codes (ECC) are also invisible to the user process because the hardware generates those bits to make sure the data is correct when it is read from memory. Thus, ECC bits are self-generated dependent upon the bits stored at a particular location and are not stored under user control.

[0003] Currently, systems use a software loop which walks through every single address in memory, writing a pattern, reading it back, then writing the inverse of that pattern and reading it back. However, because of the issues with the tag and ECC bits, four loops through memory have typically been necessary for testing purposes.

[0004] In systems having large memories the existing bit testing process is time consuming and as memory is added the amount of initialization time increases.

SUMMARY

[0005] Embodiments provide systems and methods for testing a memory where at least one bit field at certain address locations cannot be directly accessed comprising populating a data field within one of the address locations with random data bits, copying certain of the populated data bits into the non-directly accessible data field, replacing the certain copied known data field bits with bits resulting from X/ORing the copied data bits with bits read from the non-directly accessible field, and checking all the data field bits at the address location for mismatched data bits. In one embodiment, known bits are populated into a data field at one of

the certain address locations, and at least some of the known data bits that are copied into non-directly accessible data field. The bits which were copied from the data field are replaced with bits resulting from X/ORing the copied data bits with bits read from the non-directly accessible field, and all the data field bits as the address locations are checked for mismatched data bits.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] FIGURE 1 shows a representative diagram of several memory lines depicted during a write operation;

[0007] FIGURES 2A and 2B show a diagram of the memory lines of FIGURE 1 during a read operation according to one embodiment; and

[0008] FIGURE 3 shows a flow chart of one embodiment.

DETAILED DESCRIPTION

[0009] FIGURE 1 shows several lines 0-N of memory, each with one or more data fields and one ECC field. Some memory lines, for example, lines 0 and 3, also contain a tag field, which is shown for generalization. As will be detailed hereinafter, when the system is initializing the memory, some data bits from one of the data fields are copied into the ECC field and, if appropriate, some data bits from a data field, e.g., a second data field, are copied into the tag field. If the ECC field contains 12 bits, for example, the first 12 bits from data field 0 may be copied to the ECC field (bits 132-143). The tag field is shown having 26 bits, thus the first 26 bits from data field 1 may be copied to the tag field. Of course, portions of data fields or combinations of data fields may be utilized in providing the appropriate number of bits for a particular field, such as the aforementioned ECC field and tag field, if desired. As discussed above, the ECC and tag fields are fields not easily accessed by testing routines and usually cannot be directly read. Any other fields in the data line can be similarly handled, if desired.

[0010] FIGURE 2A shows line 0 during the read operation where the 12 bits from the ECC field are exclusive OR (X/OR) with the original 12 bits from data block 0 and the results (which should be all zero's) are read back into block 0. The reason the results of the X/OR should be zeros is because the bits in the ECC field are the same as the bits in block 0,

because it was those bits that were stored in the ECC at the previous write, as discussed with respect to FIGURE 1. If the bits are not all zeros, then an error is detected.

[0011] FIGURE 2B illustrates that this same operation occurs with respect to the 26 bits from the tag block, if tag block is present.

[0012] The operation of the system and method will now be discussed with respect to flow chart 30, shown in FIGURE 3. Memory initialization begins in process 301 by setting the memory data path (MDP) block in a mode such that it will take a copy of the data fields and store them into the tag and the ECC fields, as discussed above. In this example, the ECC field is 12 bits and the tag field is 26 bits, so the system uses a specified 12 bits of data from field 0 (FIGURE 1) and puts those bits into the ECC field (assuming a tag field is present), takes a copy of 26 bits from data field 1 (FIGURE 1) and puts those bits in the tag field.

[0013] Process 302 sets up the mode to take the ECC field (and the tag field, if present) and exclusive OR those bits with the copied bits from data fields 0 and 1. At this point, the system is ready to loop through every address in memory to test the memory. Processes 303, 304, and 305 control the application of a randomly generated pattern to all the addresses in memory. At point 306, all the memory locations have the random pattern stored therein. The ECC and the tag fields of each memory line have a copy of the data bits from fields 0 and 1, as discussed above. Process 306 resets the address pointer to the first address. Process 307 performs a read to the address pointer and checks that the data is equal to the random pattern and that the fields 0 and 1 which were copied to the ECC (and tag) fields have been replaced by 0's. Thus, the first 12 bits of data field 0 will have 0's therein, regardless of what random pattern was sent to that field. The remaining data field bits would be the random pattern that was originally written. When process 307 reads the data fields, the 0s should be detected in the first 12 data bit positions (and in data bit positions 0-25 of field 2, if tag bits are present). If this is what is detected, then the memory is clean. Process 308 writes the opposite of the random pattern that was written during the previous write operation (this is the bitwise complement of the random pattern which causes all the bits to flip). Process 309 checks if the last address has been reached yet. If not, process 310 increments the address and the loop through memory continues.

[0014] After this loop, process 311 resets the address pointer back to the starting address. Process 312 is part of a final loop through memory, this time checking that data equals

the opposite of random pattern and that fields 0 and 1 are still 0's. Process 313 writes all 0's to the address so that each memory location has been initialized to zero. Processes 314 and 315 loop through the addresses until the last address is processed. When complete the memory will have all zeros written to it and every bit has been flipped and tested for stuck faults in memory. At this point every location in memory has been tested for a stuck-at fault and also has been initialized to all zeros. Note that the system and method could test the memory line by line, as discussed or could load the entire memory (or portions thereof) and test for errors on larger portions at one time.

[0015] The final phase before the machine is put into the normal operation according to one embodiment, is to put the MDP back into the normal mode of operation, via process 316, which means it is taken out of the mode, which causes it to X/OR the tag and ECC fields on reads. The MDP is also put back to the mode where it calculates ECC normally and stores the ECC and tag bits as it normally would. The memory is initialized and is ready to start being used for executing normal code.

[0016] One implementation to support these modes of operation is by using programmable control status register (CSR) settings. The CSRs are set to take certain bits and pass them to MUX (not shown) to choose whether those bits should also be stored in the ECC or tag fields. The MUX would take as its control signal the CSR value for the mode bit and choose whether to store the normal generated ECC or the copy of data 0, as discussed, with respect to FIGURE 1. There is a separate CSR control bit that acts as a MUX select for the tag fields. In order to control the read from memory, a mode bit is used to select whether it is to store zeros (in anticipation the X/OR operation code to be used for matching purposes).

[0017] The result of the system and method just discussed is a memory initialization algorithm used to pattern test all memory with only three loops. Using this arrangement, it is possible to check stuck faults for every single bit location in memory. The system and method discussed above causes the direct checking for errors as it moves through all the addresses, so it knows whenever a field is non-zero or that the pattern does not match the pattern that was written. This is in contrast to systems where the software writes patterns but does not actually check the data. In some systems, there may only be a check that the chip set did not log any ECC errors. In these systems, the ECC mode remains on during initialization. The hardware will log an ECC error and the specific address that had the error. The problem

with such systems is that the memory system reorders all accesses. Thus, if the software is written to read address zero, one, two and three, the memory system can actually do it, for example, in the order of three, one, zero, two. Accordingly, these systems do not know which address was the first one that with an error. For example, assume address three logged an ECC error. Addresses zero, one, and two cannot be assumed to be good since the memory system accessed address three first. With the system and method discussed above, the software actually checks each bit for each address and knows exactly what the first address is that has an error. This operation is carried out by the processors at full bandwidth.

[0018] An alternate system which appears to reduce the number of passes through memory for testing purposes, but which actually does not test every bit is to write a random pattern to all memory in the first pass. In the second pass, the system reads back all the data that was written and then writes the opposite data, thereby, flipping every bit that was previously written. On the third pass, the system reads back the flipped bits and checks those bits and then writes zeros. The problem with such a system is that it does not have a way to directly access the ECC locations. By flipping all the data bits the assumption is that the ECC bits are error correction bits, and thus would automatically flip when the data bits changed. However, such may not be the situation. Since such a system cannot directly modify the ECC bits, nor, can it actually read these bits, there is no assurance that the ECC (or the tag) bits are correct. For absolute assurance, it is necessary to add another loop to figure out which bits were not modified and then change some of the bits in the data so that those corresponding ECC (or tag) bits would get flipped.

[0019] Another problem with the just described solution is that upon detection of a miscompare (an error) there is not a good method for determining where that error was. A simple example of such a problem is that the system cannot determine if the error is in the actual data bit, or in the ECC bits which change depending upon the data bits. The problem is compounded by memory reordering during the write cycle. Because such systems rely on the error detection chip to spot errors, which, in turn, relies on a proper ECC code, errors are hard to pinpoint.

[0020] As discussed above, one of the improvements of the instant system and method is that the underlying process has visibility to every address, so as soon as it gets a miscompare, then it knows exactly which address has a problem.

CLAIMS

What is claimed is:

1. A method of testing a memory where at least one bit field (ECC, TAG) at certain address locations cannot be directly accessed, said method comprising:

populating a data field (DATA) within one of said address locations with random data bits (303);

copying certain of said populated data bits into said non-directly accessible data field (301, 303);

replacing said certain copied known data field bits with bits resulting from X/ORing said copied data bits with bits read from said non-directly accessible field (302, 307); and

checking all said data field bits at said address location for mismatched data bits (307).

2. The method of claim 1 further comprising:

at the completion of said checking step, repeating said populating, copying, replacing and checking steps, wherein within said populating step each bit that was originally a '1' becomes a '0' and each bit that was originally a '0' becomes a '1' (308).

3. The method of claim 1 further comprising:

at the completion of said checking step, repeating said populating, copying replacing steps for each address location, one at a time.

4. The method of claim 3 further comprising:

at the completion of said last mentioned step, repeating said populating, copying, replacing and checking steps for each address location, wherein within said populating step each bit that was originally a '1' becomes a '0' and each bit that was originally a '0' becomes a '1' (308).

5. The method of claim 4 further comprising:

at the completion of said last-mentioned steps setting all address locations to '0' (303).

6. A method for initializing a memory, said method comprising:
setting an initialization mode with respect to said memory (301), reading data bits into said memory at a particular address location (303);
copying certain of said data bits into at least one other field at said address locations (301, 303);
comparing said copied bits in said field with said certain bits and replacing said certain bits with bits from said comparison (307); and
reading data bits from said address location, thereby checking every bit at said address location (307).

7. The method of claim 6 further comprising:
reading the inverse of said data bits into said memory at said address location (308);
copying certain of said data bits into at least one field of said address locations (301, 308);
comparing said copied bits in said field with said certain bits and replacing said certain bits with bits from said comparison (312); and
reading data bits from said address location, thereby checking every bit at said address location (312).

8. A memory test system in which at least one bit field (ECC, TAG) at certain address locations within said memory cannot be directly accessed, said system comprising:
at least one data field (DATA) at one of said address locations having with it known populated data bits;
copying circuitry for copying certain of said known data bits into at least one of said non-directly accessible data fields (301, 303);
bit comparison circuitry for replacing said certain copied known data field bits with bits resulting from comparing copied ones of said data bits with bits read from said non-directly accessible field (302, 307); and
checking circuitry for comparing all data field bits at said address location against data bits expected to be at said address location (307).

9. The system of claim 8 wherein certain of said data field bits include:
bits which should be "0", regardless of whether said bits were populated as a "0" or a "1".

10. The system of claim 8 wherein said bit comparison circuitry includes:
an X/OR circuit.