Design Of A Fault-Tolerant RISC Microprocessor Using VHDL

John A. Wicks, Jr.
Harold L. Martin
School of Engineering
North Carolina A&T University
Greensboro, North Carolina

Abstract

In the development of processors, system reliability is of utmost importance. More specifically, it is desired that a processor function correctly even in the presence of faults. This concept is commonly referred to as fault tolerance.

This paper describes a single fault-tolerant microprocessor whose development has been influenced by the more popular multiprocessor architecture. The characteristics of this microprocessor fall within the basic constraints of the RISC architecture, therefore it is considered to be a fault-tolerant RISC microprocessor. This fault-tolerant microprocessor will be modeled using VHDL, the very high speed integrated circuit hardware description language. A discussion of the use of fault-tolerant concepts in the design of a RISC microprocessor will be the primary focus of the paper. A general overview of the characteristics of VHDL and how this language can be used in modeling a processor will also be discussed in this paper.

Introduction

Computing systems today are used in a wide variety of applications, such as defense systems, the control of modern aircraft and spacecraft, roller coaster control, the processing of bank transactions, telephone systems, the monitoring of nuclear power plants, household appliances, and the monitoring of critically ill patients in a hospital. Technological advances in computer systems will induce a continued growth in the list of possible applications.[1]

A computer, however, like any physical system, is subject to failure, with the consequences ranging from inconvenience to catastrophe. An example of an inconvenience is the failure of a telephone-switching computer which may temporarily prevent the completion of a call. Alternately, the failure of an aircraft control computer may cause a fatal crash. The space shuttle is an aircraft which is totally dependent on the proper operation of its computer, and a mission cannot be aborted if the computers fail. In this case, the concept of fault tolerance is a critical issue in system design.

A computer system is considered to be fault-tolerant if it functions correctly even in the presence of faults. A fault can be defined as a condition existing in a hardware or software module that may lead to the failure of the module. Hardware faults are caused by physical factors resulting from wearout, external disturbances, design mistakes, or manufacturing defects. Software faults, on the other hand, result from design or implementation mistakes. A general approach to fault-tolerant design is the use of protective redundancy to permit continued correct operation of a system after the occurrence of specified faults. This protective redundancy is extra hardware, software, information, or time to mask faults or to reconfigure a faulty system.[2] A discussion of concepts in fault tolerance, including redundancy, and concepts of RISC will be presented in this paper followed by a discussion of the proposed fault-tolerant RISC microprocessor.

Concepts of Fault Tolerance

In the design of fault-tolerant systems, the designer must consider the possible

0094-2898/91/0000/0354$01.00 © 1991 IEEE

occurrence of several different kinds of faults such as transient faults, intermittent faults, permanent faults, logical faults, and indeterminate faults. Transient faults, often caused by external disturbances, exist for a finite length of time and are nonrecurring. Intermittent faults occur periodically and typically result from unstable device operation. Permanent faults are perpetual and can be caused by physical damage or design errors. Logical faults occur when inputs or outputs of logic gates are stuck-at-0 or stuck-at-1. Indeterminate faults occur when inputs or outputs of logic gates float between logic 0 and logic 1.[2]

A system can operate correctly in the presence of the aforementioned faults if the appropriate form of redundancy is incorporated into the system. Two major fault-tolerant design approaches are static and dynamic redundancy. Static redundancy is the use of redundant components so that faults may be masked. Dynamic redundancy is the reorganization of a system so that the functions of a faulty unit are transferred to other functional units. Four specific types of redundancy are information redundancy, time redundancy, software redundancy, and hardware redundancy. Information redundancy is the use of error detecting or error correcting codes for information representation. Time redundancy is the repetition of system operations so that transient faults can be masked. Software redundancy is the inclusion of several alternative programs for system operations so that software faults (design mistakes) can be tolerated. Hardware redundancy is the inclusion of multiple copies of critical components so that intermittent and permanent faults can be tolerated.

Hardware redundancy is the concept used in a very popular architecture for fault-tolerant processors. This architecture is referred to as multiprocessors. A multiprocessor system is a computer system that is made up of several CPUs or, more generally, processing elements which share computational tasks. Multiprocessors are different from multicomputer systems which have several processing elements working independently on separate tasks. The processing elements of multiprocessors typically share communication facilities, I/O devices, program libraries, and databases. Additionally, all of these processing elements are controlled by the same operating system.

The two main reasons for including multiple processing elements in a single computer system are to improve performance and to increase reliability. Performance improvement is obtained either by allowing many processing elements to share the computation load associated with a single large task, or by allowing many smaller tasks to be performed in parallel in separate processing elements. A multiprocessor consisting of n identical processors is an example of an n-unit processor that can, in principle, provide n times the performance of a comparable single-unit system or uniprocessor. The fact that the failure of one CPU does not cause the entire system to fail improves the system reliability. The functions of the faulty processor can be taken over by the other processors which means that the system is fault-tolerant.[3] A proposal for a single fault-tolerant processor using many of the redundancy techniques used in multiprocessors will be presented after a brief discussion of RISC.

RISC

In the pursuit of faster and smaller computers, researchers have found that Reduced Instruction Set Computers are very effective. The early RISC researchers at IBM, Stanford, and Berkeley sought to create computers whose execution of well-defined applications would be optimized. In an effort to reach this optimal execution, the researchers designed systems with the minimal features that were necessary to execute the specific applications they had targeted. Some features of these simplified systems are:

- Relatively few instructions and addressing modes
- Fast single-cycle execution of instructions
- Memory access limited to load and store operations
- Fixed instruction format for simplified decoding

These features are common to almost all RISC systems and are critical to the achievement of the goals set for these systems.[3] RISC systems have oftentimes surpassed their previously defined goals and are now on the market of most major computer manufacturers. Additionally, RISC systems are frequently the top sellers in these markets.

A Fault-Tolerant Reduced Instruction Set Microprocessor

The primary function of a processor, such as the central processing unit (CPU) of a computer, is to execute sequences of instructions stored in memory (main memory), which is external to the CPU. The CPU also supervises the other system components, usually via special control lines. For example, the CPU directly or indirectly controls I/O operations such as data transfers between I/O devices and main memory.

The CPU contains several registers which are used for temporary storage of instructions and operands, and an arithmetic-logic unit (ALU) which executes data processing instructions.[3] The processor in Figure 1 is the RISC microprocessor that will be transformed into a fault-tolerant system.

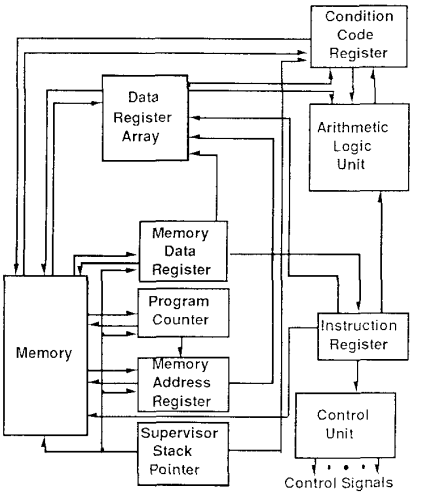

Figure 1.

The processor has the following RISC-like attributes:

- 32 instructions
- 2 addressing modes

- Single cycle execution of all instructions
- 16 32-bit data registers
- Memory access limited to load and store operations

It is proposed that the addition of redundancy to the critical components of this processor may provide a feasible alternative to the multiprocessor architecture. The three techniques of fault tolerance that will be used are triple modular redundancy (TMR), error detection and correction, and arithmetic codes. TMR is the replication of a component into three identical copies where all copies contain the same information and perform the operation at the same time. The output of all three components are then voted upon by a voter module, and the majority output is selected. Therefore, if two of the three copies are functionally correct, the voter will produce that correct output. TMR is arguably the most reliable technique in fault tolerance, but it tends to be bulky when used with large system components. Since the 32-bit memory data register (Fig. 1.) receives critical information from almost all other registers in the system, TMR will be applied at that register.

Error detection and correction codes are useful in masking transient faults. These codes consist of parity bits which are used to detect an error in the parity of a register, and check bits which are used to pinpoint the faulty bit in the register so it can be complimented. These codes are very useful in making memory registers fault-tolerant. Therefore, the memory address register, supervisor stack pointer, instruction register, condition code register, and data register array will be monitored by error detection and correction codes. In the event that uncorrectable errors are detected, each register will cease operation, and a spare register will assume all functions of the faulty register.

An arithmetic error code is a redundant representation of numbers which are included in an arithmetic operation so that errors in that operation can be detected and corrected. One popular class of arithmetic codes is the AN codes. In AN codes, if two registers N1 and N2 are to be added, then the sum of the coded form, AN1 + AN2, is equal to A(N1 + N2), the coded form of the sum, where A is some suitable constant integer. The result of the addition is a codeword, and the sum can be checked for errors. The sum denoted (E represents error)

$$S = AN1 + AN2 + E$$

can be verified by dividing S by A and checking whether the remainder is zero. If the remainder is not zero, some error has occurred in the arithmetic operation. Arithmetic codes can be useful in arithmetic and shift operations.[4] In the proposed processor, the ALU performs arithmetic operations that will be monitored by arithmetic codes. In the event that an uncorrectable error is detected in the ALU, a spare ALU will take over all operations of the faulty ALU. The inclusion of arithmetic codes along with the other concepts of fault tolerance applied to the processor combine to create an arguably efficient system.

Another concern in the design of a fault-tolerant CPU is the design tool that will be used for layout and verification. Regardless of the fault-tolerant approach

that is implemented, VHDL would serve as an excellent language to use in the modeling of the processor.

Characteristics of VHDL

Higher level languages, in general, are really the new system-design approach. Currently, designers are being asked to manage the design of very large ASICs and very large systems. These tasks are far too complex for one person, therefore hardware description languages are being used for managing design complexity.

As stated in the abstract, VHDL is the very high speed integrated circuit hardware description language. In VHDL a given logic circuit is represented as a design entity. The logic circuit represented can be as complicated as a microprocessor or as simple as an AND gate. The design entity consists of two different types of descriptions which are the interface description and one or more architectural bodies. The interface description names the entity and describes its inputs and outputs.The architectural body specifies the behavior of the entity directly or through the structural decomposition of the body in terms of simpler components.[5] Due to these basic characteristics and more complex characteristics not mentioned here, VHDL would serve as an excellent language for the modeling of a processor.

Conclusion

In this paper it was shown that the inclusion of TMR and error detection and correction codes in a system will certainly make the system reasonably fault-tolerant, and any costs incurred due to redundancy are justified by the security that a fault-tolerant system can provide. It was also shown that a single fault-tolerant processor may be a reasonable alternative to a multiprocessor architecture. Furthermore, it was shown that VHDL is an excellent language for the modeling of a processor, regardless of the fault-tolerant approach that is employed.

Finally, it is obvious that there is a great amount of work to be done in the field of fault tolerance using the techniques that are available presently. Furthermore, as fault tolerance applications increase, the demand for even greater degrees of fault tolerance will promote the continued development of new design techniques. Designers of the future will have to work very hard to meet this challenge.

References

1. T. Anderson and P. A. Lee, Fault Tolerance: Principles and Practice, Prentice-Hall International, Inc., Englewood Cliffs, NJ, 1981.

2. V. P. Nelson, "Fault-Tolerant Computing: Fundamental Concepts," Computer, IEEE Computer Society, Los Alamitos, CA, 1990.

3. J. P. Hayes, Computer Architecture And Organization, McGraw-Hill, Inc., New York, NY, 1988.

4. T. R. N. Rao, Error Coding For Arithmetic Processors, Academic Press, New York and London, 1974.

5. J. R. Armstrong, Chip-Level Modeling With VHDL, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.