A cost-effective concurrent control flow checking scheme using loop detection and prediction

Gunbae Kim
Computer Systems & Reliable SOC Lab.
School of Electrical and Electronic Engineering
Yonsei University, Seoul, korea
Kgb9572@soc.yonsei.ac.kr

Sungho Kang
Computer Systems & Reliable SOC Lab.
School of Electrical and Electronic Engineering
Yonsei University, Seoul, korea.
shkang@yonsei.ac.kr

Abstract *- As the modern processor design is changing rapidly, the system is more susceptible to transient fault and concurrent error detection for the processor is considered as a very important issue. But it imposes too much overhead to adopt concurrent error detection capability on the system. In this paper, a new approach to resolve the problems of concurrent error detection is proposed. A loop detection scheme is introduced to reduce the repetitive loop iteration and memory access. To reduce the memory overhead, an offset to calculate the target address of branching node is proposed. Performance evaluation shows that the new architecture has lower memory overhead and frequency of memory access than previous works. In addition, it provides the same error coverage and requires nearly constant memory size regardless of the size of the application program. Consequently, the proposed architecture is a cost effective method for the commercial off the shelf products.*

Keywords: watchdog processor, transient fault, loop detection, control flow error detection

1 Introduction

As device geometries decrease, several design constraints make power surge and coupling problems very serious one. Therefore, the requirements for employing concurrent error detection in microprocessors are increasing. Also, as the deeper pipeline depth is used wide, many latches and flip-flops are used in pipeline. Because the memory element is vulnerable to the soft error or single event upset, the more memory elements are used, the more frequent single event upsets (SEU) occur [1]. Furthermore, as computers are more common in automobile, transportation and industry, processors are susceptible to the noisy environment. After all, these problems make transient fault occur more frequently.

There are a lot of approaches to detect transient faults making the performance of the processor unreliable at instant time [4]. The watchdog processor is one of the approaches to detect the control flow errors [2, 3, 4, 5]. It is a simple coprocessor and operates independently and the concurrent control flow error detection is examined simultaneously. There are several approaches to study the architecture of watchdog processor, such as CERBERUS-16, RMP [2], WDP [4]. WDP proposes the efficient hardware architecture using modularity and introduces address break property of branch to detect the control flow error. However, for the iterative operations, such as loop, WDP modules access all nodes in the loop during the iteration, occupying the unnecessary bandwidth of the system bus. Also, WDP requires significant memory overhead.

To resolve these problems, we propose a new advanced control flow checking scheme adopting the loop detection method, ***watchdog processor with loop detection and prediction (WLDP)***.

In section 2, terms and operations of the WLDP will be defined and explained. In section 3, the architecture of the WLDP and the operations of the internal modules are described. Finally, in section 4, simulation results are discussed

2 The operations of control flow checking and loop detection

To adopt the control flow graph (CFG) checking and the loop detection scheme, we must define and categorize the terms and the operations used in the WLDP. There are eight types of nodes as the direction and the required condition of the branch. Each type of node has its unique operation and opcode. With this definition, we develop the watchdog instruction to describe and mimic the operations of the application program.

2.1 Node types and its operation

2.1.1 Starting node (000)

It is defined as an entry point of the CFG. When this node is encountered, i.e. when the application program is started, the reference address is compared with the retiring address. If they are equal, the PC is incremented and the next node is fetched by using the PC. If not, error signal is set.

2.1.2 Proceeding node (001)

This node carries out proceeding to the next node. It is encountered when the branching node is taken or there is no address break in the control flow. When it is reached and the retiring address is equal to the reference address, PC is incremented and the next node is fetched by using the PC. If not, error signal is set.

2.1.3 Unconditional, backward branch node (010)

When the node fetched and analyzed with the previous PC indicates this type and the retiring address is equal to the reference address, it is regarded as the loop tail and the target node address is calculated with the offset. For the prediction of the loop tail, the previous node address is calculated only at the first loop iteration and then it is saved. After these activities, the target node calculated before is fetched for the next operation.

2.1.4 Unconditional, forward branch node (011)

For this node, the address of target node is calculated with the offset when this node is indicated by the previously fetched node and the retiring address is equal to the reference address. After this operation, the target node calculated before is fetched for the next operation.

2.1.5 Conditional, backward branch node (100)

The operations of this type are similar to the case of unconditional, backward branching node except that the next node address of watchdog is calculated and saved. The conditional branch is taken in the main processor if the condition of it is agreed. Therefore, if the branch is not taken in the main processor, then, for the watchdog processor, the misprediction is happened. To prevent the misprediction in the watchdog processor, the next node address must be saved and then the target node calculated before is fetched.

2.1.6 Conditional, forward branch node (101)

The operations of this type are similar to the case of unconditional, forward branching node. The only difference is that the undesired error at the next stage occurs by the condition check. When the undesired error occurs, the next node of the previous branching node is fetched instead of triggering the error signal.

2.1.7 Subroutine call node (110)

For the subroutine call node, the target address of the subroutine node is calculated with the offset. Next, the present PC is incremented and saved in the stack to return from the subroutine and then the target node which is calculated before is fetched for the next operation.

2.1.8 Return from subroutine node (111)

For this node, the return address of subroutine node is pulled out from the stack when the retiring address and the reference address are identical. After this activity, the target node of the address, which is pulled out previously, is fetched.

Whenever backward branches are taken, new loop iterations start and continue until the loop termination is approved [6, 7]. Additionally, the loop head and tail occur repeatedly at the watchdog processor during the loop iteration. If loops are used frequently in some program, loop activities will require more bus occupation times to fetch the data from the watchdog memory. Hence, the number of watchdog memory accesses depends on the number of loop iterations. In order to reduce the memory accesses during iteration, the loop detection and prediction scheme is introduced. Loop detection and prediction enables us to predict the loop behavior whenever the previous node of loop tail is encountered. Therefore memory accesses to fetch the loop head and tail are never happened during loop iteration.

3 The architecture of the WLDP

The architecture of the WLDP is shown in Figure 9. The WLDP is composed of four modules; central processing module (CPM), loop detection module (LDM), retiring address processing module (RPM) and the memory for the reference program of the WLDP.

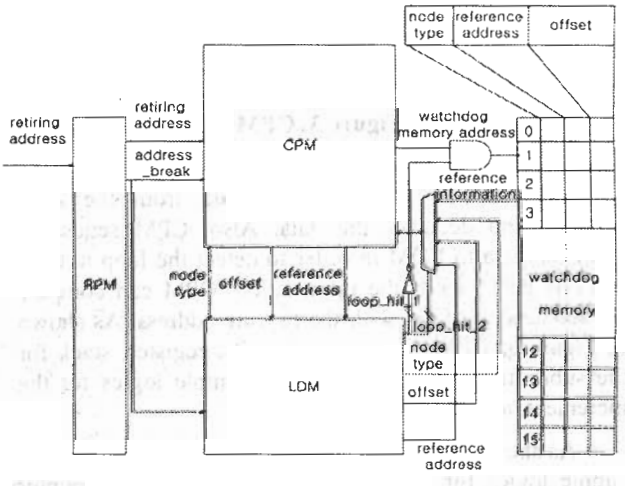

Figure 1. Proposed architecture of the WLDP

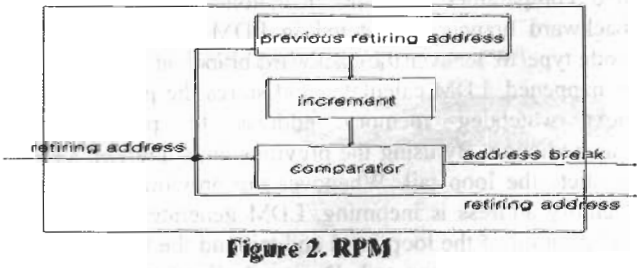

Figure 2. RPM

As shown in Figure 1, reference program of the WLDP uses three instruction bit fields; node type, associated reference node address and the offset between

the retiring instruction address and the target address. The component of each module and the operation sequence of the WLDP are described as follows.

As shown in Figure 2, when a retire hit indicates that the main processor retires the current instruction, RPM receives the retiring instruction address from the address bus. Subsequently, the internal logic increments the previous retiring addresses. If they are equal, an address break does not occur. Unless, it is assumed that the address break is happened. After these operations, RPM sends the retiring address and address break signal.

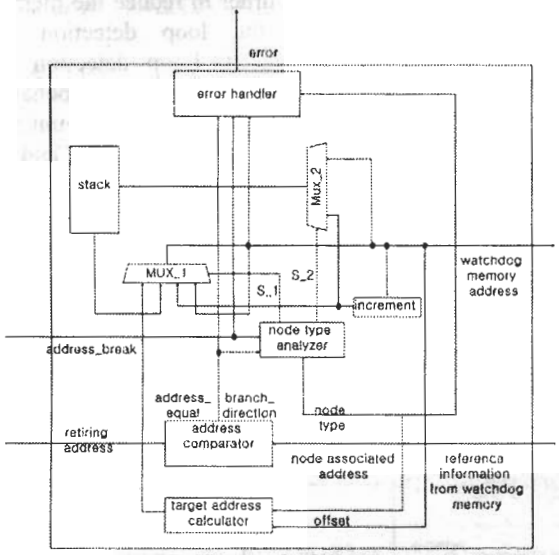

Figure 3. CPM

CPM fetches the reference data from the local memory and detaches the data. Also, CPM sends the detached data to LDM in order to detect the loop tail. As soon as RPM sends the information, CPM can compare the reference address with the retiring address. As shown in Figure 3, CPM is composed of PC register, stack for the subroutine call and return, the simple logics for the increment, and comparators.

As shown in Figure 4, LDM is composed of registers, simple logics for the decrement and increment, counter for the serial operation from loop tail to the loop head, and comparators. For the detection of the loop, the backward branch is detected in LDM by checking the node type. Whenever the backward branch at the loop tail is happened, LDM calculates and stores the previous and next watchdog memory address to prevent the misprediction. By using the previous node address, LDM predicts the loop tail. Whenever the previous watchdog memory address is incoming, LDM generates the stored information of the loop head and tail and the next node is predicted as the loop tail. During the loop iteration, the memory accesses of the loop head and tail are never happened. Therefore, two memory accesses per loop iteration are saved. As the number of the loop iterations increases, the reduction rate of memory accesses increases in double. The whole memory accesses of CPM are controlled by the loop hit signal of LDM. LDM receives the detached reference information, and transmits the loop hit signal, the previous node address and the next node address. If the loop hit signal is set, CPM receives the reference information from LDM. Otherwise, it receives the reference information from the local memory. If the loop is not terminated, LDM sends the data for the loop tail, the loop head, and the loop hit signal, whenever the previous node of the loop tail is encountered. Otherwise, the next node data saved in LDM is sent to CPM, and the loop hit signal is reset.

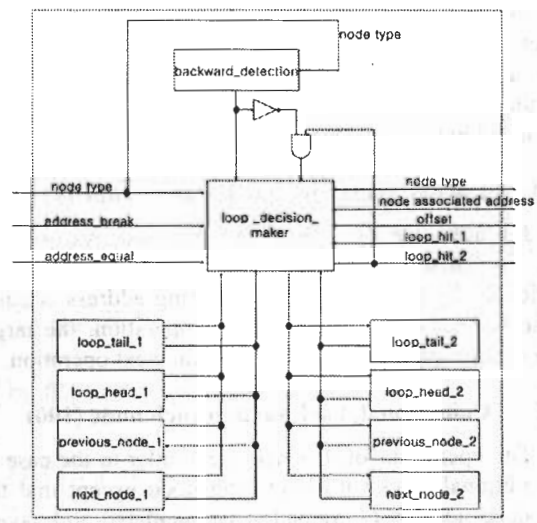

Figure 4. LDM

4 Simulation results

Table 1 shows the types of error used in the simulation. Each types of error are related to the several node types. The effect of error is explained as follow.

Table 1. Error types and description

Error type	Related node type	Description
Type 1	000	Address error at non-branching node
	001	
Type 2	010, 011, 100, 101, 110, 111	Address error making illegal loop
		Address error making illegal forward branch
		Address error making illegal subroutine call
		Address error making illegal return from subroutine
		Address error making illegal branch from the inside of t he loop to the outside
Type 3	All types	Address error making illegal branch form the inside of n ode to the inside or the outside

Table 2. Error coverage

Error type	Related node type	Number of injected errors	Error coverage
Type 1	000	165	100%
	001	1374	100%
	010	465	100%
Type 2	011	1293	100%
	100	1234	100%
	101	154	100%
Type 3	110	221	100%
	111	233	100%
	All nodes	1044	100%

As shown in Table 2, the error coverage is 100% for the each type of errors and the WLDP has no loss of the error coverage.

Table 3. Loop detection coverage

Loop type	Number of loops	Loop detection	Loop detection coverage	Error coverage of each error type		
				Type 1	Type 2	Type 3
Single	123	Yes	100%	100%	100%	100%
Overlapped	21	Yes	100%	100%	100%	100%
Doubly nested	65	Yes	100%	100%	100%	100%
Triple nested	34	Only two	99%	100%	100%	100%
Total	243		99.75%	100%	100%	100%

Table 3 shows the loop detection accuracy of the WLDP. The depth of loop detected by the WLDP is limited to the doubly nested case. If the depth of loop is more than two, the WLDP does not predict the other loops inside the doubly nested loop, but only check the control flow integrity. As shown in Table 3, loop detection coverage is 100% for the single loop and doubly nested loop. In case of triple nested loop, loop detection coverage is 99%, but the error coverage is not affected. Therefore, the results show that loop detection and prediction scheme does not degrade the performance of the control flow checking operation.

Table 4. Analysis for memory access reduction

Loop type		Entire loop			First iteration			Second-final iteration			
		iteration	tail	head	Initial access for prediction			Reduced access			
					tail	head	iteration	tail	head	iteration	%
Single		100	100	100	4	4	4	96+4	96	96	98
overlapped		27	27	27	6	6	6	21+6	21	21	92.3
Doubly nested	Outer	15	15	15	3	3	3	12+3	12	12	90
	Inner	60	60	60	5	5	5	55+5	55	55	95.8
Tripple nested	Outer most	13	13	13	5	5	5	8+5	8	8	80.8
	Inner	46	46	46	12	12	12	34+12	34	34	86.9
	Inner most	308	308	308	36	36	36	272+36	272	272	94.2
Total		569	569	569	71	71	71	569	498	498	91.1

Table 5. Comparison about memory overhead of the WDP & WLDP

Number of nodes(M)	Offset (bit)	Third field (bit)	
		WLDP	WDP
100	6.64	2564	3500
200	7.64	2664	7000
300	8.23	2723	10500
400	8.64	2764	14000
500	8.97	2797	17500
600	9.23	2823	21000
700	9.45	2845	24500
800	9.64	2864	28000
900	9.81	2881	31500
1000	9.97	2897	35000
2000	10.96	2996	70000
4000	11.96	3096	140000
6000	12.55	3155	210000
8000	12.96	3196	280000
10000	13.29	3229	350000

Table 4 shows memory access reduction results. The WLDP only accesses the reference memory at the first loop iteration. So, the loop head and tail saved at the first iteration are used for the prediction of the iteration form the first to the last. Also, at the last loop iteration, the next address of loop tail is predicted. Therefore, the WLDP reduces additional memory access. Table 4 shows the results of the memory size reduction. As the number of nodes increase, the size of memory of WDP increases linearly, but that of the WLDP is nearly steady. So, only $\log_2 M$ bits are required for the offset instead of the whole data bits. Also, the memory overhead reduction is enhanced as the number of nodes in the program increase.

5 Conclusion

We develop the advanced concurrent control flow error detection scheme using the loop detection and prediction. The main idea of the WLDP is minimization of unnecessarily iterative activities of the watchdog processor. The hardware scheme is introduced for the WLDP. To reduce the hardware and memory overhead, the offset is used to calculate the target addresses. The loop detection module enables us to reduce the memory access and bus occupation by the watchdog processor. Consequently, the WLDP is an efficient and cost effective method to detect control flow errors.

References

[1] A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. Lie, D. D. Mannaru, A. Riska, D. Milojicic, "Susceptibility of Modern Systems and Software to Soft Errors," Technical Report, Hewlett Packard Lab, 2001.

[2] A. Mahmood and E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors - A Survey," IEEE Transactions on Computers, Vol. 37, pp.160-174, 1988.

[3] N. Oh, P. P. Shirvani, E. J. McCluskey, "Control-Flow Checking by Software Signatures," IEEE Transactions on Reliability, Vol. 51, 111-122, 2002.

[4] T. Michel, R. Leveugle, G. Saucier, "A New Approach to Control Flow Checking without Program Modification," Proceedings of 21st International Symposium on Fault-Tolerant Computing, pp.334-341, 1991.

[5] M. Namjoo, E. J. McCluskey, "Watchdog Processor and Capability Checking," International Symposium On Fault -Tolerant Computing, pp. 94-97, 1995.

[6] M. Kobayashi, "Dynamic Characteristics of Loops," IEEE Transactions on Computers, Vol 32, pp.125-132, 1984.

[7] T. Sherwood and B. Calder, "Loop Termination Prediction," Proceedings of the 3rd International Symposium on High Performance Computing, 2000.