A New Concurrent Detection of Control Flow Errors Based on DCT Technique*

Hung-Chuan Lai[1], Shi-Jinn Horng[1,2,4,5], Yong-Yuan Chen[3], Pingzhi Fan[4], and Yi Pan[5]

[1]Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, 106, Taipei, Taiwan
[2] Department of Electronic Engineering, National United University, 360, Miao-Li, Taiwan
[3] Department of Computer Science and Information Engineering, Chung-Hua University, 300, Hsinchu, Taiwan
[4]Institute of Mobile Communications, Southwest Jiaotong University, 610031, Chengdu
[5]Department of Computer Science Georgia State University Atlanta, GA 30302-4110
E-mail: horngsj@yahoo.com.tw, d9115002@mail.ntust.edu.tw

Abstract

In this paper, a program is first divided into several data computing blocks (DCBs) by the branch instruction; each DCB can then be recognized as an image. We then use the one dimension discrete cosine transform (1-D DCT) to compute each DCB to generate several signatures including 5-bits relay DCT signature (R-DCT-S) and 32-bits final DCT signature (F-DCT-S). These generated signatures are embedded into the instruction memory and then used to do the run time error checking. The watchdog should not reduce the processor performance, not increase the fault detection latency and not increase the memory overhead to store the signatures; in this paper, the processor degradation can be improved by doing the whole block error checking after the branch instruction, the fault detection latency is improved by doing the intermediate error checking at the R-type instruction, and the memory overhead is reduced by storing the R-DCT-S to the R-type instruction. The experimental results show that the proposed watchdog has very high error detection coverage and shortest error detection latency to detect either single fault or multi-faults, no matter what the fault is transient or intermittent.

Index Terms- *control flow check, embedded signature monitoring technique, error detection coverage, error detection latency, fault injection, watchdog processor*

1. Introduction

Recently the very large scale integration (VLSI) is got into the nano meter processing technique era. The complexity of a chip design is progressively. Therefore, the chip will produce transient faults when it is used in the adverse environment. For instance, alpha particles and atmospheric neutrons often cause the sensitive electronic device in unsettled state [15]. In general, transient faults are going to disappear by the passage of time. However, electron migration will make the chip that was worked for a long time in the unsettled situation. The faults will change transient into intermittent. Because the technique of nano meter processing is very difficult, some intermittent faults are hidden in the chip originally. Therefore, the yield degradation is distinctly. For this reason, the fault tolerance IC design is valued year by year. The target of fault tolerance IC design is to improve the yield and system reliability by fault detection and recovery.

This paper is mainly aimed at control flow error checking in 32-bit RISC processor. The subject of control flow error checking technique is separated into software detection scheme and hardware detection scheme presently. The control flow checking technique is demanded to block the program instructions no matter what it is from software or hardware detection scheme and there is no branch instruction in the block. In other words, instructions are executed sequentially by the

* This work was supported in part by the Southwest Jiaotong University Visiting Professor Fellowship and the University Doctorial Research Foundation under grant No.20020613020, Ministry of Education. It was also partially supported by the National Science Council under contract number NSC 96-2918-I-011-002.

0-7695-3054-0/07 $25.00 © 2007 IEEE
DOI 10.1109/PRDC.2007.61

processor. The instruction block here is called a data computing block (DCB).

Most of the software detection schemes use the re-computing concept to reach the target of error detection [2], [5], [7], [17]. Some software detection schemes are to copy instructions to another memory then to run both instructions to detect the errors [2], [5]. The way of this scheme not only increases the processor performance degradation but also wastes the memory space. The extra hardware cost is the lowest in the software detection scheme relatively.

Hardware detection schemes usually use embedded signature monitoring (ESM) technique [1], [3], [18], [19], [23], [25], [26], [27], [29] to detect the errors. In [3], the ESM performance can be measured by:

1. Error detection coverage.
2. Error detection latency.
3. Instruction memory space overhead.
4. Watchdog processor complexity.
5. Main processor performance degradation.

In the ESM technique, the number of embedded signatures is affected by the number of DCBs. The error detection coverage, error detection latency, instruction memory space overhead and main processor performance are affected by the number of embedded signatures. In addition, the DCB is partitioned under the constraint of some watchdog algorithm even there is no branch instruction in the DCB [3], [29] therefore the ESM technique will be affected.

In this paper, it is based on the hardware detection scheme and the one dimension discrete cosine transform (DCT) methodology to generate the signatures (DCT signatures). We name it as DCT watchdog scheme. There is one 32-bit final DCT signature (F-DCT-S) and several 5-bits relay DCT signatures (R-DCT-S) in a data computing block. The F-DCT-S and R-DCT-S are fixed skillfully to reduce the processor performance degradation and the instruction memory overhead. The DCT watchdog scheme detects control flow errors concurrently. The scheme not only detects the single fault but also multiple faults, no matter what the faults are either transient or intermittent. We also use Weibull distribution to inject faults [3], [4], [6], [22], [29] into the circuit of the system. In the part of hardware implementation, we choose thirty eight instructions from MIPS-R2K instruction set [3], [29]. We use VHDL to implement an experiment processor and DCT watchdog processor that could be simulated the fault injection and error detection analysis by Model Sim5.5A simulator. Our test bench and fault injection tools are the same as those used in [29]. The DCT watchdog scheme will get high error detection coverage and shortest error detection latency.

This paper is organized as follows: In Section 2, the DCT watchdog scheme is presented. In Section 3, we will introduce the system hardware architecture implementation. Then, the simulation environment setting and fault injection experiment results are proposed in the following section. Finally, the conclusions are included in Section 5.

2. DCT Watchdog Scheme

In this paper, the DCT watchdog scheme is completed by ESM technique and DCT math formula. The kernel idea of ESM technique is to use the finite signatures to express the DCB information. A good signature is created by the watchdog algorithm which must be unique and one to one mapping. A signature is generally created by hashing function, data encode or data compress technique. But, the instruction memory space overhead is an importance factor in the watchdog algorithm, so the signature length should be fixed. Therefore, signatures are going to take an impact for the ESM technique. The watchdog algorithm must reduce the impact efficiently.

Owing to the good filter capacity, the DCT has been applied to image processing territory, such as image scaling, watermark, human face detection, image compression, and etc. The input image is filtered by eight one dimension DCT (1-D DCT) filters. See Fig. 1, for details. After applying the DCT to an input image, we can get a transferred DCT matrix that can be used to analyze the texture of the original image easily. For instance, the texture of the original image maybe has low frequency, high frequency, horizontal, vertical and oblique texture. The 1-D DCT formula shown in Eq. (1) can be inferred by 1-D DCT filters shown in Fig. 1.

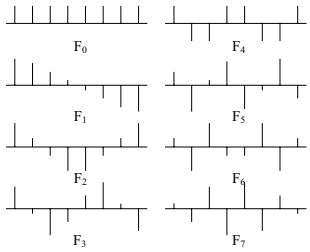


Fig 1.DCT filters

The DCT watchdog scheme is using 1-D DCT filters to calculate the signatures. The instructions of a program can be thought as a program instruction image (PII). The size of a program instruction image is N*IL (IL:

Instructions Length = 32-bits; N: the number of instructions of a program). In Fig. 1, an instruction can be thought as consisting of several pixels, where the resolution of a pixel is M-bits. For example, suppose M is 8-bits, it means that there are 256 gray-level of this pixel. As we can see, there are 6 pixels in Fig. 1, the resolution of a pixel is either 6-bits or 5-bits. Although the PII constructed in Fig. 1 seems meaningless for human eyes, the DCB information is really clearly expressed. Based on the PII, we can extract the signatures (R-DCT-S and F-DCT-S will be discussed in Sec. 2.2.1) from it using 1-D DCT filters as shown in Eq. (1).

$$T(u) = \frac{c(u)}{2}\sum_{i=0}^{7} t(x)\ \cos\frac{(2x+1)u\pi}{16}$$

$$c(u) = \begin{cases} 1/\sqrt{2}, & for\ \ x = 0 \\ 1, & otherwise \end{cases} \qquad (1)$$

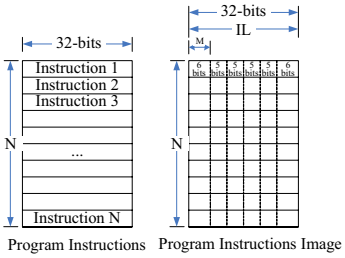


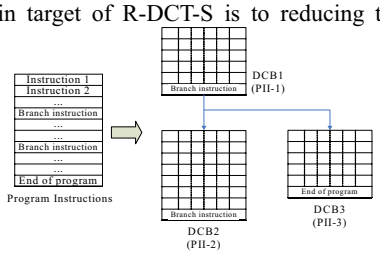
Fig. 1: Program Instruction Image (PII)

2.1 Data Computing Block (DCB)

The ESM technique is used in this paper. The program instructions will be divided into many instruction blocks via branch instructions. Two neighboring instructions are put into the same block if the leading instruction is not a branch instruction. We name an instruction block as a Data Computing Block (DCB). As we can see, the last instruction of each DCB is a branch instruction except the last block. Because, the performance of a processor is affected by the number of DCBs (described in Introduction section), therefore the DCB is not further partitioned under the constraint in this paper [3], [29]. After the block partition, each DCB has a unique PII as shown in Fig. 3. The F-DCT-S and R-DCT-S can be calculated for each PII by the DCT watchdog scheme.

2.2 Relay DCT Signature (R-DCT-S) & Final DCT Signature (F-DCT-S)

In this section, we will introduce how to create Relay DCT Signature (R-DCT-S) and Final DCT Signature (F-DCT-S) by the DCT formula for each PII. A F-DCT-S (32-bits) and several R-DCT-Ss (5-bits) will be created by the DCT watchdog scheme for each PII. The R-DCT-Ss will be used in the concurrent detection of control flow errors. The main target of R-DCT-S is to reducing the error



detection latency. We use F-DCT-S to checking the control flow errors finally.

Fig. 2: Data Computing Block (DCB).

2.2.1 DCT Watchdog Scheme Algorithm The input of Algorithm 1 is a PII. In the field of image processing, the image must be normalized (subject 128) before the DCT transform. The resolution of a pixel PII is either 6-bits or 5-bits. Therefore, each pixel must be subjected 16 to normalize the PII in algorithm 1 (Step 14). In Step 17, each row image data of a PII including last-T(0) and last-sum-of-(T(1)…T(7)) is injected into one dimension DCT (1_D_DCT) function as an input. The outputs of 1_D_DCT function are T(0), T(1), T(2) T(3), T(4), T(5), T(6), and T(7), respectively. Let the values of T(0) mod 25 and (T(1)+T(2)+…+T(7)) mod 25 for the last row instruction be defined as last-T(0) and last-sum-of-(T(1)…T(7)), respectively. The initial values of last-T(0) and last-sum-of-(T(1)…T(7)) are both zero. Next, we define twenty instructions to be special instructions including add, addu, and, break, dadd, daddu, dsub, dsubu, mfhi, mflo, nor, or, sllv, slt, slru, srav, srlv, sub, subu and xor, where each instruction is an R-type instruction selected from MIPS R-2K instruction set. The special instructions provide 5-bits free space (the shmt field) to embed the R-DCT-S (the shmt field is useless in R-type instruction, it will be described in Section 2.3).

When a row image is input to the 1_D_DCT function, we must check if the current instruction is a special instruction or not. If it is true, the fifth input value of 1_D_DCT function must be set to zero (PII[4][i]=0). After 1_D_DCT function, we set last-T(0)=T(0) mod 25and last-sum-of-(T(1)…T(7))= T(1)+T(2)+…+T(7) mod 25. The values of last-T(0) and last-sum-of-(T(1)…T(7)) are feed back to

1_D_DCT function as the last two inputs for the next instruction and they are used to improve the error detection coverage rate. The faults will be accumulated to the next checking point. We use R-DCT-S for the intermediate error checking. Suppose the current instruction is a special instruction, in Step 21, the R-DCT-S will be stored into the fifth field of the current instruction for error checking, where R-DCT-S= (T(0)+T(1)+…+T(7)) mod 25, the reason for mod 25 is that there are only 5-bits free spaces in a special instruction. Let heap-DC and heap-AC denote the accumulated DC value and AC value, respectively. Since the maximum output of 1_D_DCT function is probably twice as big as the input values, to disperse the DC and AC values, in Steps 22, 23, we first shift both heap-DC and heap-AC values left one position and then sum it up to DC value (T(0)) and AC value ((T(1) +T(2) +… +T(7)), respectively, for each instruction of a PII. We repeat the above operations for each instruction of a PII. Finally, we combine the heap-DC and heap-AC in Step 25 to form the F-DCT-S. The flow chart of DCT watchdog scheme algorithm is shown in Fig.

***Algorithm 1**: DCT watchdog scheme algorithm*

*1. input PII[m][n] //PII of DCB (size=m*n); initial m=0…5*
2. function 1_D_DCT( ); //one dimension DCT
3. one dimension DCT output: {T(0), T(1) T(2), T(3) T(4), T(5) T(6), T(7)};
4. variable last-T(0)=0; //5-bits
5. variable last-sum-of-(T(1)…T(7))=0; //5-bits
6. variable R-DCT-S=0; //Relay DCT Signature, 5-bits
7. variable F-DCT-S=0; //Final DCT Signature, 32-bits
8. variable heap-DC=0; //To heap all of the DC value, 16-bits
9. variable heap-AC=0; //To heap all of the AC value, 16-bits
10. special instructions: add, addu, and, break, dadd, daddu, dsub, dsubu, mfhi, mflo, nor, or, sllv, slt, slru, srav, srlv, sub, subu, xor
11. start
12. for(int i=0; i<n; i++)
13. begin
14. PII[m][i]=PII[m][i] – 16;
15. if PII[m][i] is a special instruction;
16. then PII[4][i]=0;
17. {T(0), T(1) T(2), T(3) T(4), T(5) T(6), T(7)}=1_D_DCT(PII[0][i], PII[1][i], PII[2][i], PII[3][i], PII[4][i], PII[5][i], last-T(0), last-sum-of-(T(1)…T(7)));
18. last-T(0)=T(0) mod 25;
19.last-sum-of-(T(1)…T(7))= (T(1)+T(2)+…+T(7)) mod 25;
20. R-DCT-S= (T(0)+T(1)+…+T(7)) mod 25;
21. if PII[m][i] is a special instruction, the R-DCT-S is embedded in PII[4][i];
22. heap-DC=(heap-DC<<1)+T(0);
23.heap-AC=(heap-AC<<1)+(T(1)+T(2)+…+T(7));
24. end
25. F-DCT-S={heap-DC, heap-AC};
26. The F-DCT-S is embedded at the end of block;
27. finish;

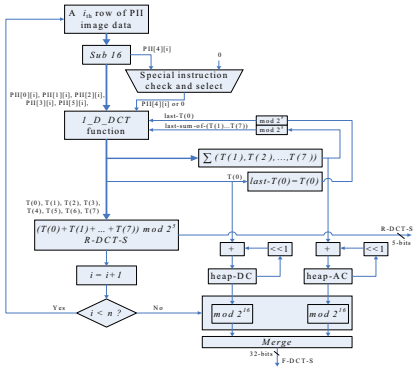


Fig4. The flow chart of DCT watchdog algorithm

2.2.2 Signature Word Structure. In this paper, we used the MIPS R-2K processor to do the simulation. MIPS R-2k is a 32-bit fixed length processor. There are three instruction types including R-type, I-type and J-type in the instruction set architectures (ISA). The ISA is described in Fig. 3. We used the twenty R-type instructions (add, addu, and, break, dadd, daddu, dsub, dsubu, mfhi, mflo, nor, or, sllv, slt, slru, srav, srlv, sub, subu, xor) as special instructions to reduce the fault detection latency. The field of "shamt" (5-bits) in the special instruction is a redundancy space [18]. The DCT watchdog scheme uses this space to record the R-DCT-S. In this paper, we used R-type instruction to segment any types of instructions of a PII. Since there is a large probability, a special instruction can appear in a DCB. We can store the cumulated R-DCT-S to the shmt field of the current special instruction. During the run time of a program, we can do the fault detection when the current instruction is a special instruction. This can reduce the fault detection latency quite a few.

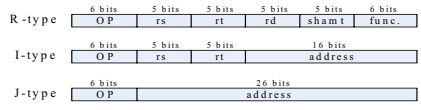


Fig. 3 ISA of MIPS R-2K processor.

In Algorithm 1, for a PII, it will create several R-DCT-Ss and store the cumulated R-DCT-S to the shamt field of a special instruction and the F-DCT-S is stored after the branch instruction of a PII. The formats of the signature R-DCT-S and F-DCT-S are shown in Fig. 4.

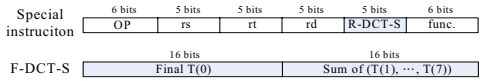


Fig. 4 Word structures of R-DCT-S and F-DCT-S.

The signatures must be embedded into an instruction memory to process the control flow error checking during the run time by the embedded signature monitoring technology. Some watchdog techniques embedded a signature to the beginning instruction of a DCB [3], [29]. As we know the signature cannot be executed by the processor at the first instruction of a DCB. Therefore, the processor will waste one clock cycle. The performance degradation will be occurred in this situation. Because the branch prediction is not processed at branch instruction for some embedded processors including MIPS R-2k, the F-DCT-S will be inserted after the branch instruction in this paper. When a branch instruction was executed, the processor must be hold 1 clock cycle. The F-DCT-S will be used to detect the control flow error in this time by the DCT watchdog processor. So the performance degradation will not be happened in this situation. The simulation and analysis in Section 4 will discuss the relationship between the branch prediction and performance degradation. The relationship between R-DCT-S, F-DCT-S of a DCB is shown in Fig. 5. The 4th, 6th and 7th instructions in Fig. 5 are special instructions. The original value in the shamt field is “00000”. The R-DCT-S will be cumulated and embedded into the shamt field by the DCT watchdog algorithm. Finally, the F-DCT-S will be inserted into the 9th instruction of a DCB as the 8th instruction is a branch instruction of a DCB.

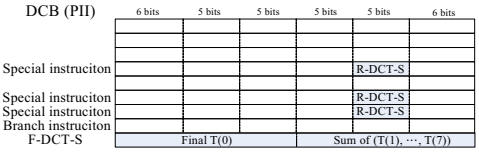


Fig. 5 R-DCT-S and F-DCT-S in DCB.

2.3 Error Detection Flow of DCT Watchdog Scheme

We use ESM technique to check the control flow error. For the first step, the original program (or machine code) is transferred into the watchdog machine code by Algorithm 1. For the sake of speeding up encoding the machine code, the DCT watchdog algorithm(Algorithm 1)can be embedded into an assembler. In this paper, we design an external watchdog signature inserting program to create watchdog machine code in Fig. 6.

Fig. 6 signature injection.

In order to detect the control flow error and execute the watchdog machine code, the control circuit of MIPS R-2K should be modified and the processor should be extended and added a DCT watchdog processor (the hardware architecture will be described in Section 3). The error detection flow of the DCT watchdog scheme is described in Algorithm 2. The DCT watchdog processor reads instructions from the instruction bus of MIPS-R2K. During the execution time, if the current instruction is a special instruction or the last instruction of a DCB, the corresponding R-DCT-Ss and F-DCT-S in the watchdog machine code will be extracted and stored in the DCT watchdog registers as the golden signatures, see Steps 14 and 31. The run time F-DCT-S and R-DCT-Ss will be computed by the DCT watchdog processor in Steps 30 and 19, for each instruction of a DCB. The DCT watchdog processor will compare the run time F-DCT-S and R-DCT-Ss with the golden signatures during the execution of a special instruction or the last instruction of a DCB, respectively. If the result is the same, there is no run time error; otherwise, a run time error occurs and the DCT watchdog processor will inform the MIPS R-2K to enter the recovery state. This is shown from Step 22 to Step 35. The recovery latency will be reduced as it immediately gets into the recovery state when there was an error detected. If an error is detected at the last instruction, it must execute the DCB from the beginning instruction again. The error detection flow of the DCT watchdog scheme is described in Algorithm 2.

Algorithm 2: *Error detection flow of DCT watchdog scheme*

*1 input WPII[m][n] //WPII: watchdog machine code block(size=m*n); initial m=0…5, n=0..n-1*
2 function 1_D_DCT ( ); //one dimension DCT
3 one dimension DCT output: (T(0), T(1) T(2), T(3) T(4), T(5) T(6), T(7));
4 variable last-T(0)=0; //5-bits
5 variable last-sum-of-(T(1)…T(7))=0; //5-bits
6 variable heap-DC=0; //To heap all of the DC value, 16-bits
7 variable heap-AC=0; //To heap all of the AC value, 16-bits
8 variable gold-F-DCT-S=0;

9 variable gold-R-DCT-S=0;
10 start
11 for (int i=0; i<n; i++)
12 begin
13 if WPII[m][i] is a special instruction
14 {gold-R-DCT-S=WPII [4][i];
WPII[4][i]=0; }
15 if (i<n-1)
16 {(T(0), T(1) T(2), T(3) T(4), T(5) T(6), T(7))= 1_D_DCT(WPII[0][i], WPII[1][i], WPII[2][i], WPII[3][i], WPII[4][i], WPII[5][i], last-T(0), last-sum-of-(T(1)…T(7)));
17 last-T(0)=T(0) mod 2^5 *;*
18 last-sum-of-(T(1)…T(7))= (T(1)+T(2)+…+T(7)) mod 2^5 *;*
19 R-DCT-S= (T(0)+T(1)+…+T(7)) mod 2^5 *;*
20 heap-DC=(heap-DC<<1)+T(0);
21heap-AC=(heap-AC<<1)+(T(1)+T(2)+…+T(7));
22 if WPII[m][i] is a special instruction
23if R-DCT-S = gold-R-DCT-S // Compare R-DCT-S with gold-R-DCT-S;
25 {The fault is not happened or not be aroused ;}
26 else
27 An error arises and notifies the processor to do error recovery ;}
28 }
29 if i=n-1 //the last instruction of WPII
30 { F-DCT-S=(heap-DC, heap-AC);
31. gold-F-DCT-S=(WPII[5][i], WPII[4][i], WPII[3][i], WPII[2][i], WPII[1][i], WPII[0][i]);
32 if F-DCT-S = gold-F-DCT-S // Compare F-DCT-S with gold-F-DCT-S;
33 The fault is not happened or not be aroused;
34 else
35 An error arises; the processor is going to do the re-computation ;}
36 }
37 end

3. Hardware Implementation

3.1 DCT Watchdog Processor Hardware Architecture

According to Eq. (1), it is very hard to implement the hardware as there are some cosine functions and multiplications. The hardware implementation of DCT was discussed in other papers including Fast DCT [12], Butterfly [24], [28], and etc [9], [11], [20]. Some of them are very fast, but consume a lot of hardware resources. In order to synchronize with MIPS R-2K, the DCT watchdog processor should be fast enough and cannot waste too many hardware resources. Based on these criteria, Eq. (1) can be transformed into Eq. (2) and then into Eq. (3). In Eqs. (2) and (3), the values of cosine function for 7 different angles are represented by 7 constants A, B, C, … and G, respectively. As we can see, these 7 constants are multiplied with t(i), and almost appeared in each T(i), $0 \leqq i \leqq 7$ in Eq. (3). We can also use the two's complement to do the subtraction. Instead of using the multiplication hardware for implementation, we use a ROM to store the values of 7 constants multiplied by a t(i) at first; then the stored values can be used further by using t(i) as an index. Since there are two types of resolutions of each row (instruction) of a PII, we should use two ROMs with different input sizes. See Fig. 1for the implementation. It then can reduce the hardware cost and chip area quite a few. T(i), $0 \leqq i \leqq 7$ in Eq. (3) can be easily obtained by summing up the values retrieved from ROMs. The hardware implementation is shown in Fig. 8.

$$\begin{bmatrix} T(0) \\ T(2) \\ T(4) \\ T(6) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{bmatrix} \times \begin{bmatrix} t(0)+t(7) \\ t(1)+t(6) \\ t(2)+t(5) \\ t(3)+t(4) \end{bmatrix}$$

$$\begin{bmatrix} T(1) \\ T(3) \\ T(5) \\ T(7) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & D \end{bmatrix} \times \begin{bmatrix} t(0)-t(7) \\ t(1)-t(6) \\ t(2)-t(5) \\ t(3)-t(4) \end{bmatrix}$$

$$A = \cos\frac{\pi}{4},\ B = \cos\frac{\pi}{8},\ C = \sin\frac{\pi}{8},\ D = \cos\frac{\pi}{16}$$

$$E = \cos\frac{3\pi}{16},\ F = \sin\frac{3\pi}{16},\ G = \sin\frac{\pi}{16} \qquad (2)$$

$$\begin{aligned}
T(0) &= \tfrac{1}{2}*(A*t(0)+A*t(1)+A*t(2)+A*t(3)+A*t(4)+A*t(5)+A*t(6)+A*t(7)) \\
T(1) &= \tfrac{1}{2}*(D*t(0)+E*t(1)+F*t(2)+G*t(3)-D*t(4)-E*t(5)-F*t(6)-G*t(7)) \\
T(2) &= \tfrac{1}{2}*(B*t(0)+C*t(1)-C*t(2)-B*t(3)-B*t(4)-C*t(5)+C*t(6)+B*t(7)) \\
T(3) &= \tfrac{1}{2}*(E*t(0)-G*t(1)-D*t(2)-F*t(3)+F*t(4)+D*t(5)+G*t(6)-E*t(7)) \\
T(4) &= \tfrac{1}{2}*(A*t(0)-A*t(1)-A*t(2)+A*t(3)+A*t(4)-A*t(5)-A*t(6)+A*t(7)) \\
T(5) &= \tfrac{1}{2}*(F*t(0)-D*t(1)+G*t(2)+E*t(3)-E*t(4)-G*t(5)+D*t(6)-F*t(7)) \\
T(6) &= \tfrac{1}{2}*(C*t(0)-B*t(1)+B*t(2)-C*t(3)-C*t(4)+B*t(5)-B*t(6)+C*t(7)) \\
T(7) &= \tfrac{1}{2}*(G*t(0)-F*t(1)+E*t(2)+D*t(3)-D*t(4)-E*t(5)+F*t(6)-G*t(7))
\end{aligned} \qquad (3)$$

Symmetric

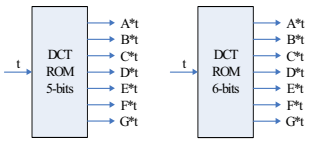


Fig. 7 1-D DCT watchdog ROMs

The hardware architecture of the DCT watchdog processor is shown in Fig. 9. The inputs of the DCT watchdog processor are 32-bit instruction bus and 2-bit CPU control signals. The instruction obtained from the instruction bus is segmented by R-type instruction format and enters into the 1-D DCT block. The opcode (6-bits) and func (6-bits) fields of the instruction are then fed into the watchdog control unit. The control unit can determine if the current retrieved instruction is a special instruction or not. If it is, the R-DCT-S

(stored in the shmt field) of the current instruction will be extracted and stored in the gold-R-DCT-S register and then compared to the value stored in the R-DCT-S register for the run time error checking. Also while the last instruction is executed, the F-DCT-S is retrieved from it and stored into the gold-F-DCT-S register and then compared to the currently value stored in the F-DCT-S register for the error checking for the whole block. In addition, the MIPS R-2K processor can control the DCT watchdog processor through the 2-bit CPU control signals; “00” means to let the DCT watchdog sleep, “01” means to reset the DCT watchdog, “10” means to run the DCT watchdog and “11” means to let the DCT watchdog be idle. Fig. 9 shows the hardware architecture of a pipeline processor with the DCT watchdog processor. The latter concurrently extracts and executes the instruction from the instruction bus connected to the former. The former can control the latter through the 2-bit CPU control bus. In this architecture, the control flow errors of the instruction bus, instruction memory, program control, address generator and control unit will be detected by the DCT watchdog processor.

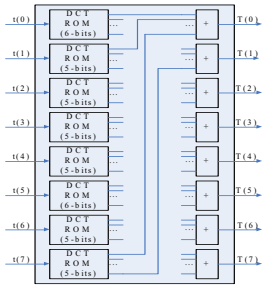


Fig. 8 1-D DCT hardware architecture

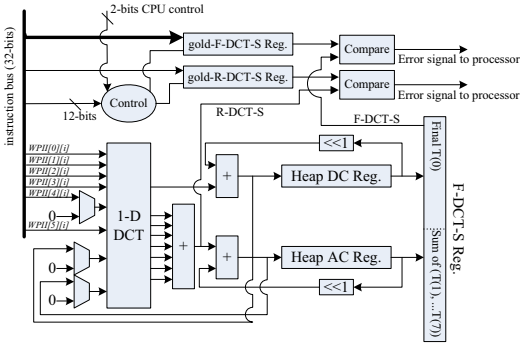


Fig. 9 The Hardware of the DCT Watchdog Processor

4. Experiment and Results

The DCT watchdog and a pipeline MIPS R-2K processor were implemented by VHDL. We use Model Sim5.5A (company Mentor Graphics) to simulate the system functions and synthesize the system circuits by Synopsis TSMC 0.35um technology library.

4.1 Fault Module and Simulation Fault Injection Experiment

In order to get the exact simulation results as [29], the fault injection tool and test bench programs are the same as those used in [29]. The fault injection tool use Weibull distribution to simulate the life cycle of a chip based on two useful variables α=1.0 (useful-life period) and λ=0.001. There are four test bench programs for the experiment that were partitioning into Workload 1 and Workload 2. The branch instructions are distributed extensively in Workload 1; the length of a DCB of the test bench programs of Workload 1 is relatively short. We mixed “n! (n=10)”, ”5×5matrix multiplication”, and “quick-sort (sorting 10 element)” to make test bench programs for Workload 1. On the contrary, the length of a DCB of Workload 2 is relatively long; the IDCT (Inverse DCT) was classified into Workload 2.

The fault module in this paper is listed in the following:

1. The stuck-at faults are injected into the simulation system. The value of a fault is selected randomly from stuck-at-zero and stuck-at-one.
2. System faults can be permanent or transient. The fault duration time was created by a random function for transient faults. The minimum duration time is one clock cycle. The total time of faults was happened until the simulation was terminated which will be defined to be the fault duration for permanent faults.
3. The situation of system faults can be either single fault or multiple faults.

We use the fault injection tool to inject the faults into Fig. 10 simulation system. The fault injection target was separated into Target 1 and Target 2. Target 1 contains the instruction memory and instruction bus, representing the sources of any instruction bit errors. Target 2 includes the address generating circuit, program counter, address bus to the instruction memory, representing the sources of control flow errors. The probability of a fault injected into a specific component is based on the hardware complexity among the selected components for fault injection targets.

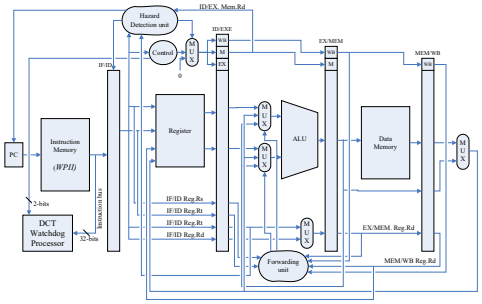


Fig. 10 A processor with the DCT watchdog processor

4.2 Hardware Overhead

The simulation pipeline processor is MIPS R-2K in this paper. We modified the VHDL pipeline processor code as used in [29]. Only the decoder of control circuit is different to that used in [29]. In order to keep results correctly, we used the same technology to synthesize the pipeline processor.

		Area (gate count)	Max. Frequency (MHz)	System Clock
Pipeline Processor*	This paper	25,549	85	85
	[29]	25,549	85	85
Watchdog	This paper	2,512	85	85
	[29]	2,452	86.4	85

*Note: the area does not include the instruction memory and data memory.

Table. 1 Performance of the system architecture.

Test Bench	Signature memory space overhead	
	[29]	This paper
Matrix-Multiplication	7.44%	6.06%
Quick-Sort	18.26%	18.26%
Factorial (*n!*)	26.32%	26.32%
Eight-Queens	32.98%	32.98%
IDCT(8×8)	3.9%	0.82%

Table. 2 Signature space overhead for various test bench.

4.3 Coverage and Latency

In order to guarantee the data of simulation in accordance to that used in [29], we were carrying on 1,500 times in the following fault simulation experiment. The single fault was injected into the system for each experiment to get the fault detection latency and coverage. We used the same fault injection tool as used in [29] to do the experiment. For each experiment, we injected a fault to the test bench by Weibull distribution. A fault is selected randomly from either stuck-at-zero or stuck-at-one. The detailed simulation results are shown in Tables 3 and 4.

6. Conclusions

In this paper, we use the characteristic of DCT to catch the texture of a DCB and apply the DCT to the IC control flow check field. The redundancy instruction space for the RISC instruction set architecture will be applied to store R-DCT-S ingeniously to reduce the fault detection latency. In order to avoid the performance degradation, we embed the F-DCT-S into the end of a DCB. Through a large amount of simulations and analyses, we can get high error detection coverage, low error detection latency, low instruction memory space overhead, low watchdog processor complexity and a little main processor performance degradation.

		Workload 1		Workload 2	
Distribution of DCB length	[29]	BL≦8: 73% BL>8: 27%		BL≦8: 11% BL>8: 89%	
	[This paper]	BL≦8: 73% BL>8: 27%		BL≦8: 1% BL>8: 99%	
Memory space overhead	[29]	13.4%		3.9%	
	[This paper]	13.2%		0.82%	
Performance degradation	[29]	11.25%		6%	
	[This paper]	Min. 0%	Max. 9.28%	Min. 0%	Max. 0.67%
Coverage[29](hybrid)	Latency[29] (hybrid)	99.6%	2.4 clocks	99.47%	4.1 clocks
Coverage [This paper](mix)	Latency [This paper](mix)	99.72%	2.9 clocks	99.9%	0.72 clocks
Coverage[29](vertical)	Latency[29] (vertical)	98.8%	9 clocks	98.9%	14 clocks
Coverage [This paper](F-DCT-S)	Latency [This paper](F-DCT-S)	99.72%	8.79 clocks	99.9%	72.5 clocks

Table 3. Effects of various workloads

			Block-based (Workload 1 and Workload 2)
Coverage	Good state	[29] (vertical)	97.2-99.1 %
		[This paper] (F-DCT-S)	100 %
	Poor State	[29] (vertical)	91.6-98.3 %
		[This paper] (F-DCT-S)	100 %
Coverage	[29] (hybrid)		99.62-99.99 %
	[This paper](Mix)		100 %
Latency	[29] (hybrid)		0.65-0.8
	[This paper](Mix)		0.72-0.83

Table 4. Coverage and latency data derived from block-based simulation

6. Reference

[1] Amir Rajabzadeh, M.M., Ghassem Miremad, Error Detection Enhancement in COTS Superscalar Processors with Event Monitoring Features. Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004.

[2] B. Nicolescu, Y.S., R. Velazco, Software Detection Mechanisms Providing Full Coverage Against Single Bit-Flip Faults. IEEE TRANSACTIONS ON NUCLEAR SCIENCE, 2004. VOL. 51, NO. 6.

[3] Chen, Y.Y. and K.L. Leu, Signature-monitoring technique based on instruction-bit grouping. Computers and Digital Techniques, IEE Proceedings -, 2005. vol. 152(no. 4): p. 527-536.

[4] Delong, T.A., B.W. Johnson, and J.A. Profeta, III, A fault injection technique for VHDL behavioral-level models. Design & Test of Computers, IEEE, 1996. vol. 13(no. 4): p. 24-33.

[5] Fazeli, M., R. Farivar, and S.G. Miremadi, A software-based concurrent error detection technique for power PC processor-based embedded systems. Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on, 2005: p. 266-274.

[6] Gil, D., et al., A study of the effects of transient fault injection into the VHDL model of a fault-tolerant microcomputer system. On-Line Testing Workshop, 2000. Proceedings. 6th IEEE International, 2000: p. 73-79.

[7] Goloubeva, O., et al., Soft-error detection using control flow assertions. Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on, 2003: p. 581-588.

[8] Hennessy, J. and D. Patterson, Computer Architecture: A Quantitative Approach. Third ed. 2002: Morgan Kaufmann.

[9] Hsiao, S.F., et al., Efficient VLSI Implementations of Fast Multiplierless Approximated DCT Using Parameterized Hardware Modules for Silicon Intellectual Property Design. Circuits and Systems I: Regular Papers, IEEE Transactions on, 2005. vol. 52(no. 8): p. 1568-1579.

[10] Jenn, E., Fault injection into VHDL models: the MEFISTO tool. Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on, 1994: p. 66-75.

[11] Junqiang, L. and Z. Xinhua, Embedded image compression using DCT based subband decomposition and SLCCA data organization. Multimedia Signal Processing, 2002 IEEE Workshop on, 2002: p. 81-84.

[12] Kwak, J. and J. You, One- and two-dimensional constant geometry fast cosine transform algorithms and architectures. Signal Processing, IEEE Transactions on, 1999. vol. 47 no. 7, p. 2023-2034.

[13] Larus, J., SPIM S20: A MIPS R2000 Simulator. 1997. p. 1-25.

[14] Leveugle, R., T. Michel, and G. Saucier, Design of microprocessors with built-in on-line test. Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium, 1990: p. 450-456.

[15] Ma, T.P. and P.V. Dressendorfer, Ionizing Radiation Effects in MOS Devices and Circuits. 2001, New York: Wiley-Interscience.

[16] Patterson, D. and J. Hennessy, Computer Organization & Design: The Hardware/Software Interface. 1998: Morgan Kaufmann.

[17] Reis, G.A., et al., SWIFT: software implemented fault tolerance. Code Generation and Optimization, 2005. CGO 2005. International Symposium on, 2005: p. 243-254.

[18] Rodriguez, F., J.C. Campelo, and J.J. Serrano, Improving the interleaved signature instruction stream technique. Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on, 2003. vol. 1: p. 93-96.

[19] Schuette, M.A. and J.P. Shen, Processor Control Flow Monitoring Using Signatured Instruction Streams. Transactions on Computers, 1987. vol. C-36(no. 3): p. 264-276.

[20] Shen-Fu, H. and S. Wei-Ren, A new hardware-efficient algorithm and architecture for computation of 2-D DCTs on a linear array. Circuits and Systems for Video Technology, IEEE Transactions on, 2001. vol. 11(no. 11): p. 1149-1159.

[21] Technology, M., ModelSim SE/PLUS 5.7b. 2003.

[22] Tsai, T.K., et al., Stress-based and path-based fault injection. Transactions on Computers, 1999. vol. 48(no. 11): p. 1183-1201.

[23] Upadhyaya, S.J. and B. Ramamurthy, Concurrent process monitoring with no reference signatures. Transactions on Computers, 1994. vol. 43(no. 4): p. 475-480.

[24] Venkatesh, S. and S. Srinivasan, Modified butterfly structure for efficient implementation of pruned fast cosine transform. Electronics Letters, 1998. vol. 34 no. 14, p. 1383-1385.

[25] Wilken, K. and J.P. Shen, Continuous signature monitoring: efficient concurrent-detection of processor control errors. Test Conference, 1988. Proceedings. 'New Frontiers in Testing'., International, 1988: p. 914-925.

[26] Wilken, K. and J.P. Shen, Continuous signature monitoring: low-cost concurrent detection of processor control errors. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 1990. vol. 9(no. 6): p. 629-641.

[27] Wilken, K.D. and T. Kong, Concurrent detection of software and hardware data-access faults. Transactions on Computers, 1997. vol. 46(no. 4): p. 412-424.

[28] Yuh-Ming, H., W. Ja-Ling, and H. Chiou-Ting, A refined fast 2-D discrete cosine transform algorithm with regular butterfly structure. Consumer Electronics, IEEE Transactions on, 1998. vol. 44(no. 2): p. 376-383.

[29] Yung-Yuan, C., Concurrent detection of control flow errors by hybrid signature monitoring. Transactions on Computers, 2005. vol. 54(no. 10): p. 1298-1313.