Self Checking Register File Using Berger Code

A. H. ABDULHADI ALI H. MAAMAR
Higher Institute of Electronics
Beni- Waled, LIBYA

Abstract: In recent years the complexity of digital systems has increased dramatically. Although semiconductor manufacturers try to ensure that their products are reliable, it is almost impossible not to have faults somewhere in a system at any given time. As a result, reliability has become a topic of major concern to the designers and to the users of these systems. Unfortunately as the scale of integration has increased so also has the occurrence of intermittent faults. The characteristics of these types of faults render them undetectable by standard test strategies. This is particularly problematic with the wide use of complex circuits in safety-critical applications. Ensuring the reliability of these systems is a major testing challenge. The detection of intermittent faults requires the use of Concurrent Error Detection (CED) techniques, which continually monitor the operation of the circuit. One method of implementing CED in VLSI/ULSI circuits is through the use of information Redundancy or coding techniques. This paper investigates the use of Berger code as a means of incorporating CED into a self checking register file.

Key-Words: Self checking, information redundancy, Berger code, register file, concurrent error detection.

1 Introduction

The advances in VLSI/ULSI technology have made possible many changes not only in processor architecture but also in the amount of hardware that can be integrated into a die permitting the implementation of single chip processor. Today, many of the applications which VLSI/ULSI technology is used are deemed to be 'safety' critical, eg., life-support machines, aerospace and petro-chemical industries, nuclear reactor control, railway signaling, commerce and banking etc. Although increased scales of integration offers many advantages, these complex circuits are more susceptible to transient and intermittent faults, these faults are the most frequently occurring faults in digital systems, it has been reported [1] that 90% of VLSI system failures are due to intermittent faults. With the extensive use of these types of circuits in safety critical applications, a major challenge which must be addressed is the development of test techniques to detect transient and intermittent faults. Unfortunately the characteristics of this type of fault, namely random occurrence and short duration, render standard test strategies ineffective. The detection of these types of faults necessitates the use of a test strategy which continuously monitors the operation of the circuit and compares it with some known reference. This approach is usually referred to as Concurrent Error Detection (CED).

Concurrent Error Detection is the process of detecting errors at the same time as the system is performing its normal operation. CED can be achieved through the use of 'Redundancy'. Redundancy is the use of extra recourses time, hardware, or data, beyond the requirements of the unchecked system. There are three types of Redundancy, namely, Hardware Redundancy, Time Redundancy, and Information Redundancy[2][3][4].

Information redundancy (coding techniques) has been identified as a viable mechanism for implementing concurrent error detection (CED) in VLSI circuits; several RISC processors incorporating information redundancy schemes have been designed and fabricated[5][6]. Invariably, the incorporation of CED schemes incur penalties on a design in terms of area overheads resulting from the additional hardware and routing space necessary to implement the scheme, the area overhead incurred is a function of the number of the checkbits (extra bits added to information bits) used in the coding scheme. Amongst all of the separable codes used in CED schemes, Berger code [7] is the least redundant separable code capable of detecting all unidirectional errors. The construction of the code, and its error detection capabilities are outlined below together with a design of a self checking register file using the code.

Register files are generally fast RAMS with multiple read and write ports. The Register File is used to minimize off-chip communications caused by Load/Store instructions, reading an operand from a register file inside the chip is much faster then reading the same operand from off-chip memory. Large register file increases the number of operands that can be stored on-chip, which in turn will increase the performance of the processor. Two independent register files are used, Data Register File (DRF) for storing information bits, and Check Symbol Register File (CSRF) to hold the check symbols of the information bits. Since the code used for CED is a separable code then it is possible to separate the information part and the check symbol part of each code word. Each file has its own address decoding hardware. Thus addressing errors can be detected by a mismatch between the new check symbol generated for the data word read from the DRF and its check symbol stored in the CSRF, the probability of both address decoders being in an error simultaneously with the same fault is very low.

2 Berger code

Berger code[7] is an optimal separable code which can detect all unidirectional errors. A Berger code word of length n bits has I information bits and k check bits, where $[k=\log_2(I+1)]$, $n=I+k$.

Berger codes are useful for encoding the information bits in digital systems because:

1-They are separable codes. No extra decoders are required to extract the information bits, when needed for processing, from the code word.

2- They detect all unidirectional errors; these are most likely to occur in digital systems.

3- They are optimal, in terms of the number of check bits required for I information bits, among all the separable codes that detect unidirectional errors[8].

A code word is constructed by forming a binary number corresponding to the number of ones in the I information bits, and appending the bit-by-bit complement of the binary number as check bits to the information bits [9]. For example, if I = 1100101, k = $[\log_2 (7+1)]$ =3 so the Berger code must have a length of 10 (7+3), k check bits are derived as follows: Number of 1s in the information bits = 4, Binary equivalent of 4 =100. The bit-by-bit complement of 100 is 011, which are the k check bits. Thus, the code word : 1100101 011. It should be clear from the above discussion that the k check bits may be the binary number representing the number of 0's in information bits. Thus, the check bits for Berger codes can be generated by using two different schemes. The scheme that uses the bit-by-bit complement of the binary number corresponding to the number of 1's in the information bits, and the other scheme, which uses the binary number corresponding to the number of 0's in the information bits as check bits. If the number of information bits in a Berger code is $I=2^k-1$, $k\geq1$, then it is called a maximal length Berger code; otherwise it is known as the non-maximal length Berger code. For example, the Berger code 1100101011 is maximal length because k=3 and I =7= (2^k-1), whereas 110100011 is non-maximal length because k=3 and $I=6\neq(2^k-1)$. [2.

3 Register File

Register File is used to minimize off-chip communications caused by Load/Store instructions, reading an operand from a register file inside the chip is much faster then reading the same operand from off-chip memory. Large register files increase the number of operands that can be stored on-chip, which in turn will increase the performance of the processor. The size of the register file is trade off between chip area and the storage space, increasing the size horizontally (increasing the number of bits in each register) means more chip area needed to implement each extra bit, the number of bits in each register depends on the size of operand (data word) that the ALU can process. When a coding technique is used for error detection, not only the data word has to be stored in the Register File, but also its check symbol, consequently the horizontal size of the register file not only depends on the size of the data word, but also on the number of bits in the check symbol, which also depends on the code used. Increasing the register file vertically (adding more registers to the register file), will increase the size of the address decoders, requiring an increase in chip area to implement the additional registers and the extra hardware for the address decoders. Two independent register files will be designed. The Data Register File (DRF) for storing information bits, and Check Symbol Register File (CSRF) to hold the check symbols of the information bits. Since the code used for concurrent error detection is a separable code then it is possible to separate the information part and the check symbol part of each code word. Each file has its own address decoding hardware. Thus addressing errors can be detected by a mismatch between the new check symbol generated for the data word read from DRF and its check symbol stored in the CSRF, since the probability of both address decoders being in an error simultaneously with the same fault is very low. The DRF consists of 32 general purpose registers available to the user, each register is 32 bits wide. The basic cell used to build data register file comprises a 1-bit register cell which has three ports, ports (A,B) used as Read ports only, and port C which can be used as a Write or a Read port. The Data Register File communicates with other blocks in the chip over three buses, each bus has its own address decoder to select one register at a time, two registers can be selected at a time for read operations via two buses, also the contents of any register can be read via two buses at a time, but it is not possible to select one register for read and write operation at the same time. Only one write operation can be performed at a time to place word into one of the registers of the DRF. Each register has five input lines to control its functions, these are: Register Read via bus A (RDA), Register Read via bus B (RDB), Register Read via bus C (RDC), Register Write via bus C (RWC), and Clear Register (CLR), the CLR control line is used to clear the contents of the selected register to zeros.

3.1 Bus Checkers

When data is to be moved from the register file to the ALU via the bus, the data should immediately be checked for any detectable errors. This is carried out by the checker, which consists of a check symbol generator for Berger Code, and a Totally Self Checking (TSC) Two-Rail Checker (TRC). The check symbol generator is a zero counter as presented in [7], it counts the number of zeros in the information bits of any information word, and gives the number of zeros which represents the check symbol. When the check symbol becomes available it is then compared with the stored check symbol for that particular word for the checker used for either Bus A or Bus B having been previously extracted from the CSRF.

When designing a circuit which incorporates a concurrent error detection capability, the question immediately arises regarding the number and placement of the checkers; this is trade-off between area and error latency time, that is the delay between the error occurring and its detection. The number of checkers is usually equal to the number of the major buses used to connect the main blocks together, for example if two Buses (Bus A and Bus B) are used to move data from DRF to the ALU, then two checkers are needed, one checker for each bus. However, if only one checker is used to check the two buses, then only one operand can be checked and moved from DRF to ALU at a time, consequently the ALU cannot receive two operands in parallel and the system works as if it has only a single bus between DRF and the ALU. In this register file only two checkers are used, one checker for each bus, simultaneously checking the operands moving from DRF to ALU via bus A and Bus B.

.

Once the number of the checkers has been decided upon, their placement in the circuit must be considered. Figure 1 shows the checkers for buses A and B being located between ports A and B of the DRF and input latches A and B of the ALU. This means that to move any operand from the DRF to the ALU, the operand is moved first from the DRF to the checker on the bus, if it is error free then operand will be moved to the ALU input latch connected to the same bus; however, if an error is detected the operand will not be transferred to the ALU input latch and an error signal will be sent to the control unit to handle the error. The advantage of this method is that faulty data cannot be processed in the ALU. However, it has the major disadvantage of introducing a delay in the data transfer between DRF and the ALU. Since the data will not be available to the ALU until it has been processed by the checker, the total process time will be checking time plus ALU processing time. There is no requirement, however, to directly check the operands to the ALU, because if the ALU processes faulty operands the result cannot affect any other blocks in the circuit, as a data transfer from ALU cannot be initiated unless the operands and output result are error free; further more, since the ALU is a combinational circuit any previous faulty computation cannot affect the next operation.

An alternative checker placement for buses A and B is shown in figure 2, in which the checkers and the ALU are connected in parallel, hence overcoming the above delay penalty. To move the operands from the DRF to the ALU, the operands can be transferred directly into the input latch of the ALU and at the same time a copy of the operands is moved to the checkers, the ALU can start processing the operands at the same time as the checkers start to check the operands, if no errors in the operands are detected, then the result obtained from the ALU can be taken as result of error free operands. The output result must be checked against any error generated by ALU circuit during processing, if the checkers detect an error in the operands, an error signal will be activated to stop the processor from passing the result. Again the result of processing faulty data cannot be propagated to other blocks in the circuit since data transfer cannot be initiated unless operands are fault free. The advantage of this method is that the total processing time is only equal to ALU Processing time, as the checking and processing of the operands is done in parallel

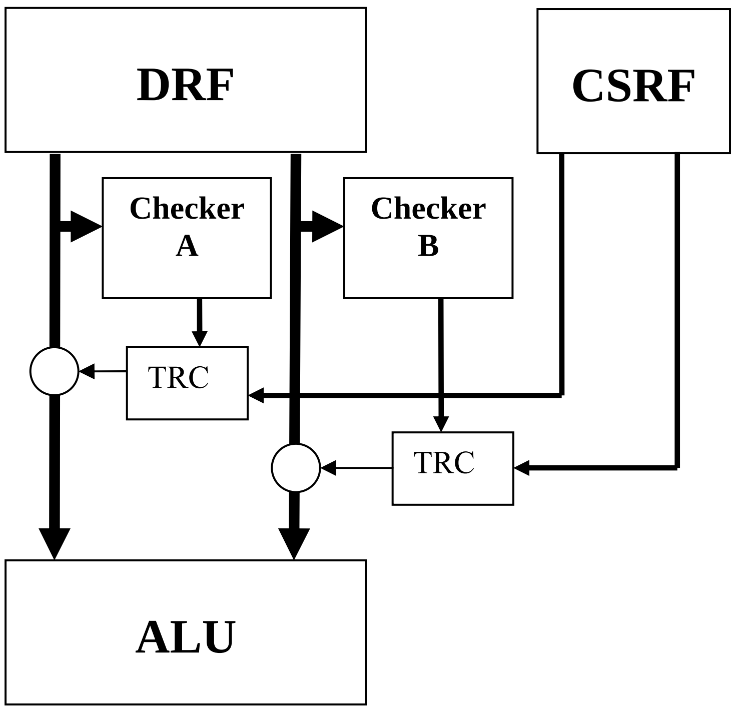

Figure 1 Bus Checkers located between the ALU and the Register File

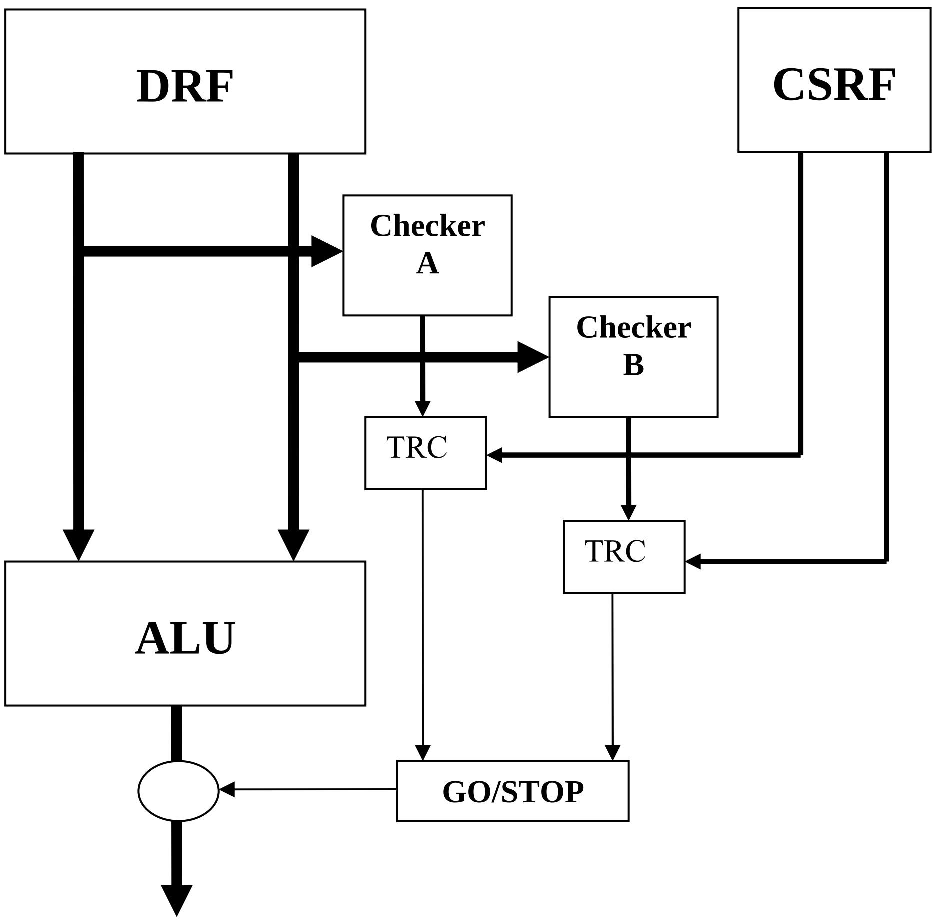

Figure 2 Bus checkers and ALU connected in parallel

3.2 Controlling the Register File

The Register File which comprises the DRF and CSRF, is controlled by set of signal lines, which come from controller. The control lines comprise: First a set of register select lines which come from the address decoders; second, a set of operation lines supplied by the Control Signal Generator. Since the RF comprises two sub-register files, both needs to receive, simultaneously, the same control signals to perform similar read operations in parallel in order to extract of codeword (i.e. data plus check symbol) from the RF. Furthermore in order to permit simultaneous read or write operations in both sub- register files, the DRF and CSRF require their own set of three address decoders. Operations on the DRF and CSRF are defined by a set of 'operation lines'. These operations are:- Read Selected Register via bus A (RDA), Read Selected Register via bus B (RDB), Write Selected Register via bus C (WRC), and Clear Selected Register (CLR). These operations can be performed on any register in the RF. The contents of any register can be read via bus A or B, however a word can only be written into any register in the RF via bus C. It is possible to read the contents of any register via two buses at the same time, although read and write operations cannot be performed simultaneously on the same register location. The contents of any register can be forced to zero by the control signal CLR. To read the contents of a register, the control unit selected the register and the bus to be used (A or B), then the Control Unit sends an DRA signal to the RF, and the contents of the selected register in DRF are gated to the selected bus and becomes available to the destination block, at the same time the check symbol of the data word is gated from the selected register in CSRF to the corresponding Check Symbol Bus.To write a word to a register in the RF the address of the register is sent to bus C address decoder (Bus C in the DRF, and Bus C in the CSRF); consequently a write operation can be performed via bus C only. After the DR is selected, the data to be written into the selected register is placed on bus C, the Control Unit sends a WRC to the RF, after some delay the data on the bus C is copied into the selected register. Clear operation is used to clear the contents of any register in the RF, bus C address decoder is used to select the register to be cleared, then the Control Unit sends the CLR signal to the RF, after short delay the contents of the selected register is forced to zero.

3.3 Implementation of the Register Files

Two independent Register Files are implemented. The Data Register File (DRF) which holds the data words, and the Check symbol Register File (CSRF) which stores the Check Symbols of the data words, these are shown in figures 3, 4 respectively. The two files could have been implemented as a combined 32×38 bit file, requiring 3 decoders. However, with this configuration faults in the address decoder could remain undetected, since any incorrectly selected data would be extracted with its matching check symbol. To avoid this situation, at the expense of using more area, two separate files, 32x32 bits, and 32x6 bits, were used, requiring 6 address decoders (three for each file). With this configuration addressing errors can be detected since these will be a miss match between the extracted data and the check symbol. The possibility of both decoders being in error simultaneously is very low. The same basic cell is used to build the DRF and the CSRF, that is, a 1-bit three port register cell. In the DRF the cell is used first to build a 4-bit register cell, which is then cascaded eight times to built the 32-bit Register. The CSRF simply comprises six single bit cells. The control lines (RDA, RDB, RDC, WRC, and CLR) are distributed to each register in the file as shown in figure 5. The signals are buffered using inverters, first the signal coming from the address decoder is fed to two inverters, and each inverter is then used to drive 4 other inverters, each of which is used to buffer the control signal to 4-bits of the 32-bit register. This control signal buffering scheme ensures that each bit of the register receives a sufficiently strong control signal, in other words the Least and the Most significant bits receiver the same strong control signal from the address decoder associated with a given bus. As the length of the register used to build the CSRF is only 6-bits, then only two cascaded inverters are used to buffer the control signal and the output of the last inverter is used to drive the 6-bit register. The two register files receive control signals from their respective controllers, that is the DRF receives the control signals from the Information Block Controller, and the CSRF receives the control signals from the Checker Block Controller. Both controllers perform the same operations (Read, Write or Clear) at the same time, any mismatch in the two control signals coming from the controllers, produce two different operations, which can be detected.

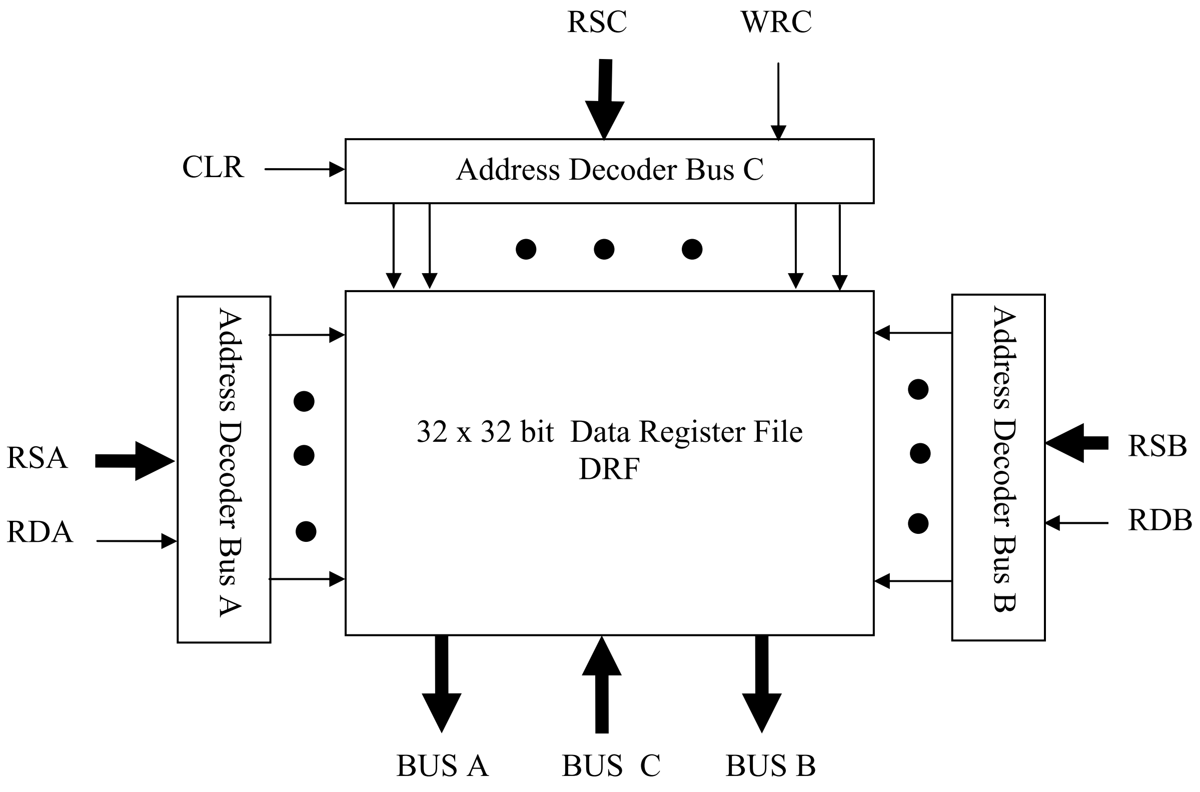

Figure 3 Data Register File (DRF)

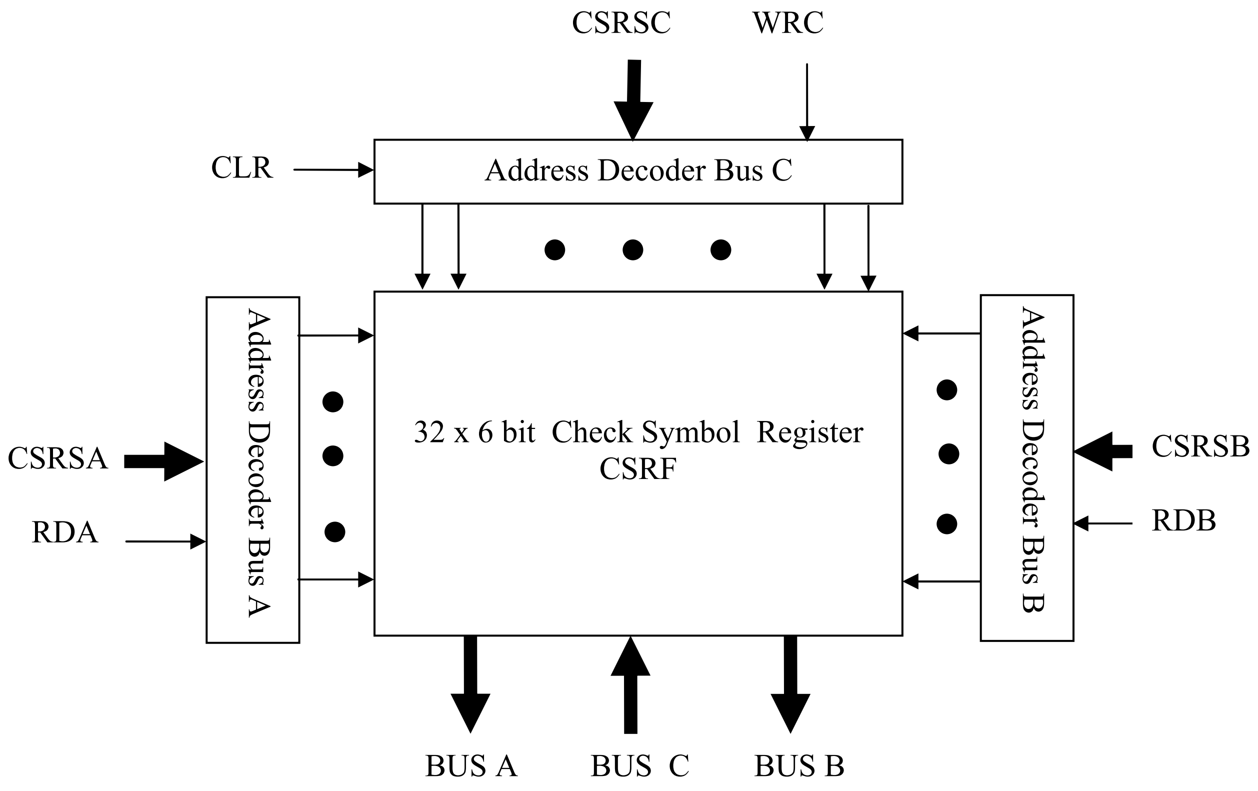

Figure 4 Check Symbol Register File (CSRF)

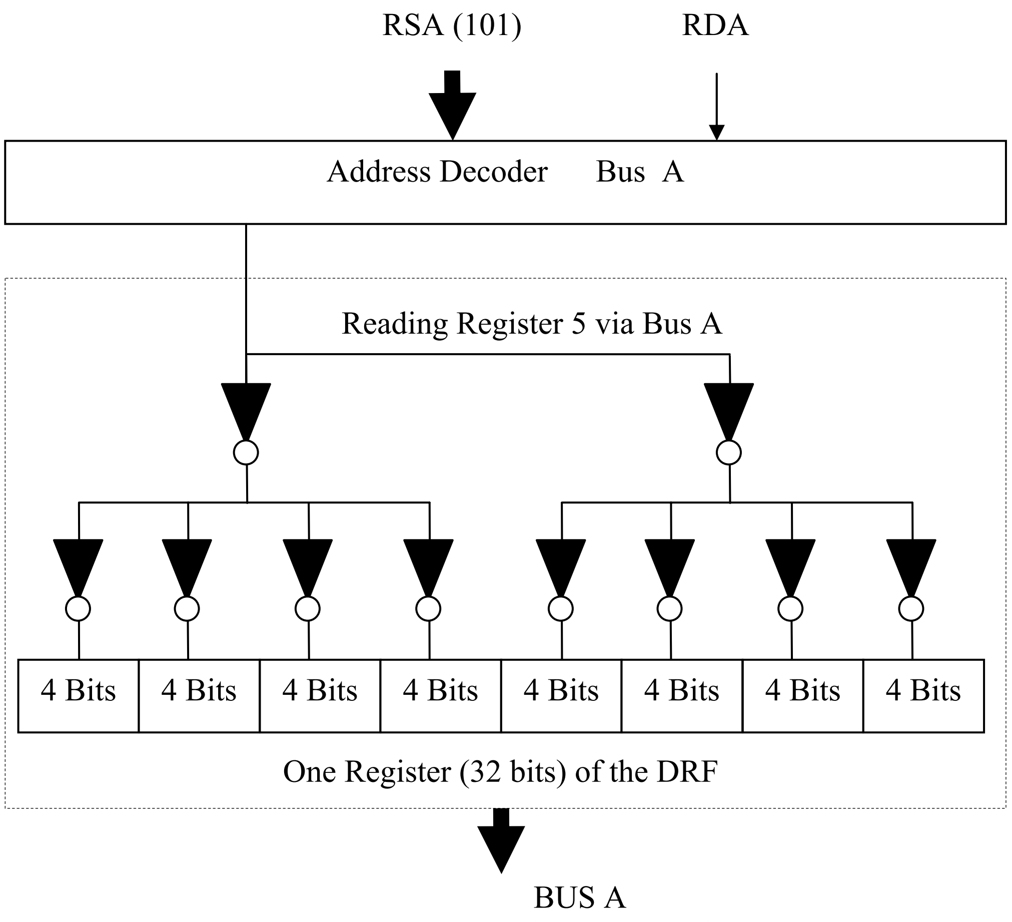

Figure 5 32- bit register in the DRF

4 Conclusion

The work in this paper is concerned with the investigation of Berger code as a means of integrating a Concurrent Error Detection (CED) scheme into a VLSI circuit. Berger code has the advantage that it can detect all unidirectional errors The design of a Self-Checking Register File using Berger code have been presented, the file is self checking against errors affecting the information bits and the checkbits. Since the code used is a separable code then two seperate files were designed (DRF for storing the infromation bits, CSRF to hold the checkbits), each file has its own address decoding hardware. Thus addressing errors can be detected by a mismatch between the stored checkbits and the generated checkbits for the word read from the DRF. The penalty of using two seperate files is the area overhead used for the extra address decoders.

References

[1] J. Clary and R. Sacane," Self-Testing Computers", IEEE Comp., vol.12,No.10, October 1979, pp. 49-59.
[2] Parag K. Lala "Self-Checking and Fault-Tolerance Digital Design", Morgan Kaufmann Publisher, 2001.
[3] Subhasish Mitra, "Diversity Techniques for Concurrent Error Detection", Technical Report, Center for reliable computing, May 2000.
[4] Teijo Lehtonen, Juha Plosila, Jouni Isoaho, "On Fault Tolerance Techniques towards Nanoscale Circuits and Systems" Turku Center for Computer Science, TUCS Technical Report, No 708, August 2005, Finland.
[5] Russell, G. and Elliot, I.D., "Design of Highly Reliable VLSI Processors Incorporating Concurrent Error Detection and Correction " , Proceedings EURO ASIC91, May 1991 Paris .
[6] A. Maamar, and G. Russell, "A 32-bit RISC processor with concurrent error detection", Proc. 24th Euromicro Conference, August 1998, Sweden, pp 461-467.
[7] J.M. Berger, " A note on error detection codes for Asymmetric Channels " , Information and Control, vol. 4, March 1961, pp68 – 73 .
[8] M. Oma'a, O. Losco, C. Metra, A. Pagni, " On the Selection of Unidirectional Error Detecting Codes for Self-Checking Circuits Area Overhead and Performance Optimization", Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS'05), 2005 IEEE
[9] M. A. Marouf and A. D. Friedman, "Design of Self-Checking checker for Berger codes", IEEE transactions on computers, Volume 42, Issue 8, August 1993.