Testability of VLSI

Lecture 14
Fault Tolerant VLSI Design

By Dr. Sanjay Vidhyadharan

Error Detection

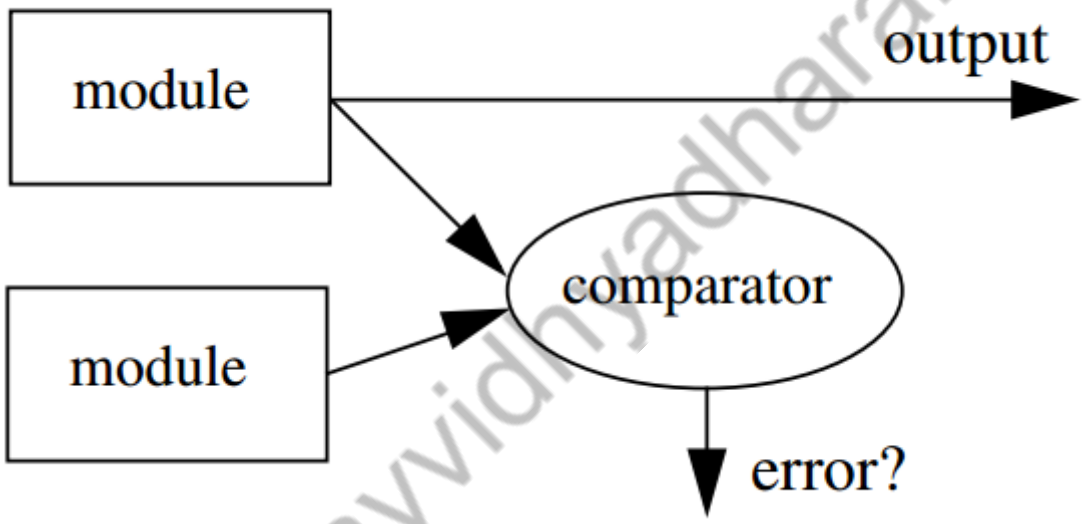
1. Error needs to be detected first to rectify (tolerate) the error.

2. Even if the detected error is not rectified warning is to be generated, as a measure of *safety*.

3. The key to error detection is redundancy. The three classes of redundancy
   - Physical (sometimes referred to as "spatial")
   - Temporal, and
   - Information

Physical Redundancy

Dual Modular redundancy (DMR) with a comparator

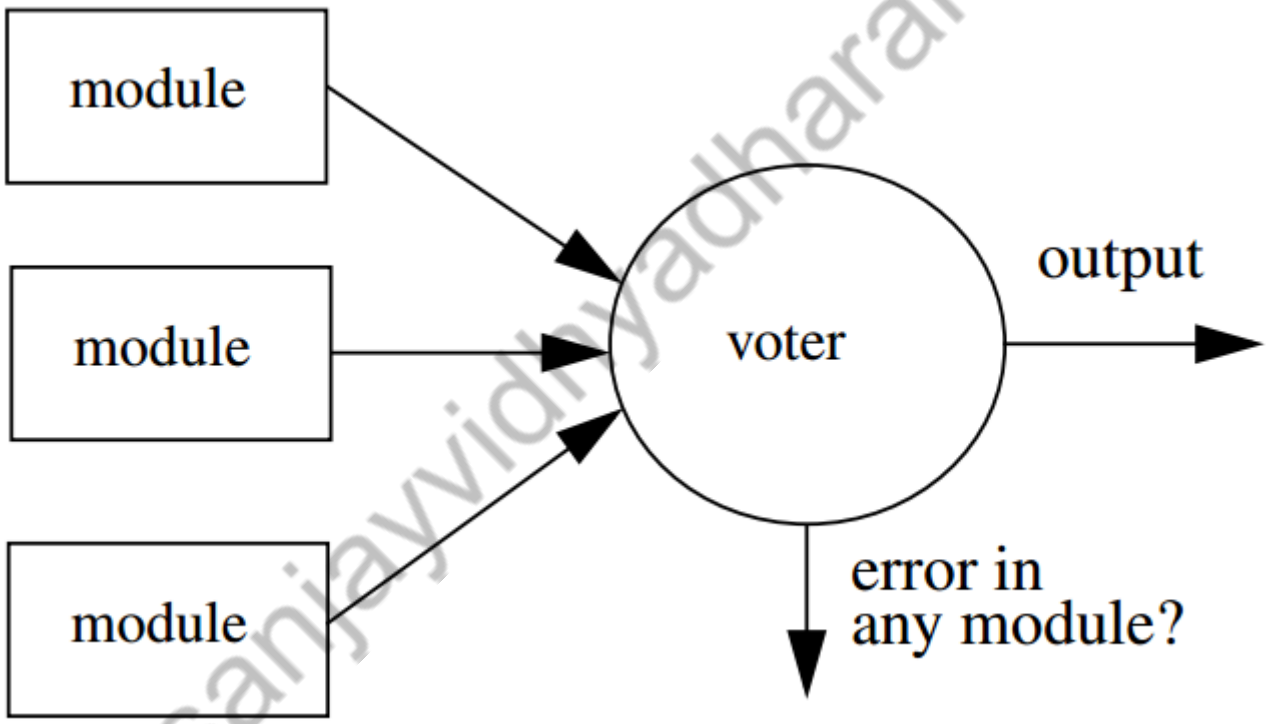


Excellent error detection

All errors except for errors due to design bugs, errors in the comparator, and unlikely combinations of simultaneous

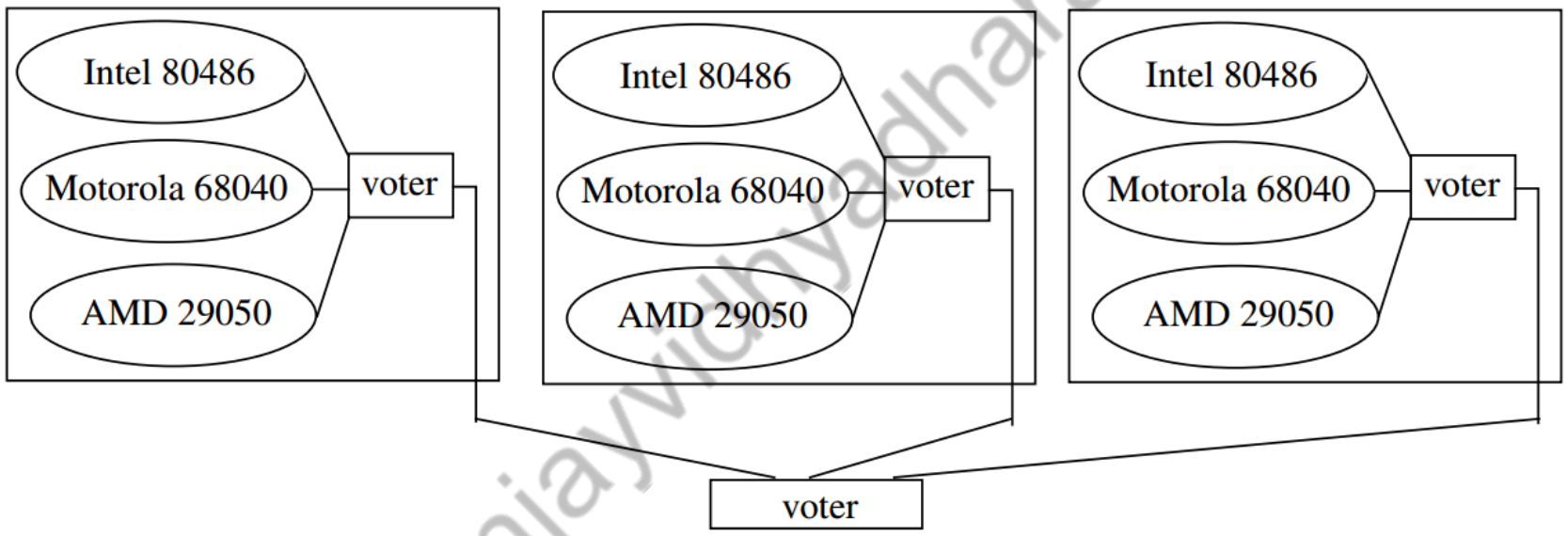
Physical Redundancy

Triple modular redundancy (TMR)



Physical Redundancy

Triple modular redundancy (TMR)



Boeing 777's triple TMR

[1] Fault Tolerant Computer Architecture by Daniel J Sorin

Temporal Redundancy

Temporal redundancy requires a unit to perform an operation twice one after the other, and then compare the results.

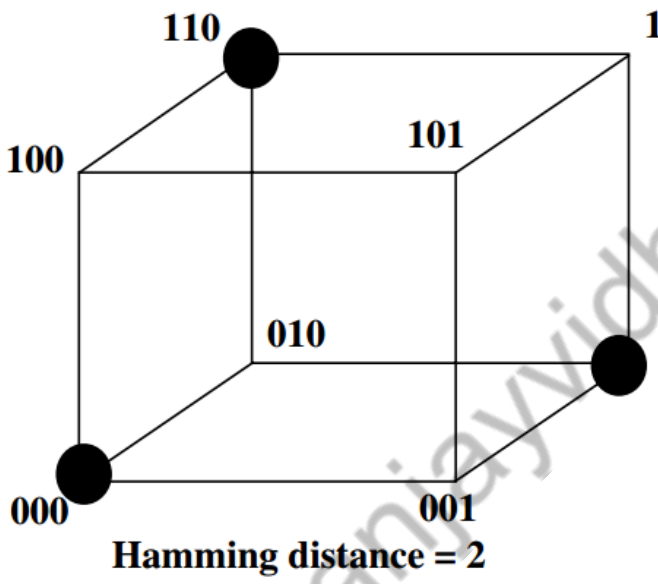
Total time is doubled unless pipe-lining structure

Unlike with physical redundancy, there is no extra hardware or power cost (once again ignoring the comparator).

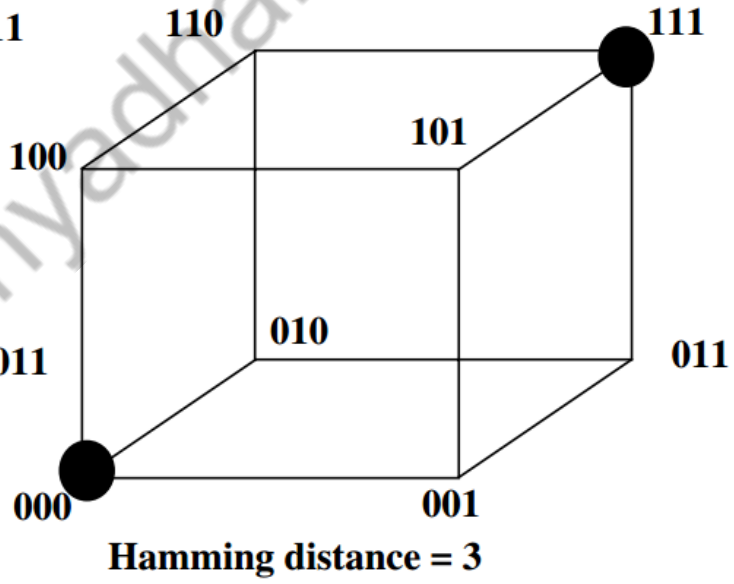
Information Redundancy

Add redundant bits to a datum to detect when it has been affected by an error.

Odd or Even Parity Check



Hamming distance = 2



Hamming distance = 3

Single bit error detected but double error is not detected

Single bit error and double error is also detected

Information Redundancy

Classification of Code:
Example “Single-error correcting (SEC) and double-error detecting (DED)” SECDED : HD4

HD3: Can either correct single errors ***or*** detect single or double errors, but it cannot do both.

Information Redundancy

$2^r >= m+r+1$

m is number of bits in original data

r is number of redundant bits to be added

n=m+r final number of bits in the coded data string

Example m = 4

r=1 : 2 >= 4 + 1 +1

r=2 : 4 >= 4 + 2 +1

r=3 : 8 >= 4 + 3 +1

			2^2		2^1	2^0
m_4	m_3	m_2	P_3	m_1	P_2	P_1

[1] Fault Tolerant Computer Architecture by Daniel J Sorin

Information Redundancy

Example: 0011 (Even Parity)

7	6	5	4	3	2	1
0	0	1	P_3	1	P_2	P_1
0	0	1	P_3	1	P_2	P_1
0	0	1	1	1	1	0

2^3	2^1	2^0	$P_1(2^0)$	$P_2(2^1)$	$P_3(2^3)$
0	0	1	1		
0	1	0		2	
0	1	1	3	3	
1	0	0			4
1	0	1	5		5
1	1	0		6	6
1	1	1	7	7	7

Even Parity

P_1 : **0**,1,1,0

P_2 : **1**,1,0,0

P_3 : **1**,1,0,0

Assuming an erroneous data

1	0	1	1	1	1	0

Error Code (P_3,P_2,P_1)

(1,1,1) -> 7

Assuming an erroneous data

0	0	1	1	1	1	1

Error Code (P_3,P_2,P_1)

(0,0,1) -> 1

Example

Write and read from memory

Error Detection in Microprocessor Cores

Functional Units

General Techniques. To detect errors in a functional unit, we could simply treat the unit as a black box and use physical or temporal redundancy

Another general approach to functional unit error detection is to use *arithmetic codes*.

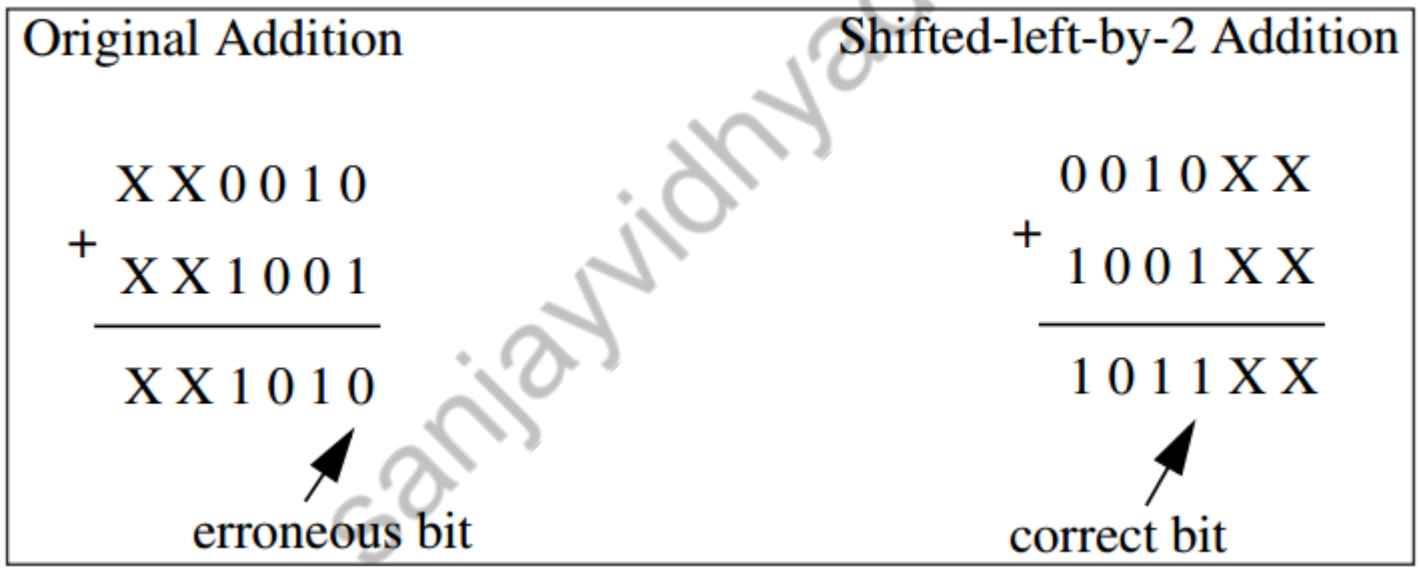
Example
A+ *B*=*C*
10A+ 10*B*= 10*C*, If we get 10C it is error free
Error causes the adder to produce a result that is not a multiple of 10

Error Detection in Microprocessor Cores

Functional Units

Re-execution with Shifted Operands (RESO) : *Adder*

Input operands are shifted before the redundant computation.



Error Detection in Microprocessor Cores

Functional Units

Multipliers

$A \times B = C \rightarrow [(A \bmod M) \times (B \bmod M)] \bmod M = C \bmod M.$

With an appropriate choice of *M*, the modulus operation can be performed with little hardware

6 X 12 = 72
6 Mod5 X 12 Mod5 = 72 Mod5
1 X 2 = 2

Error Detection in Microprocessor Cores

Tightly Lockstepped Redundant Cores

Physical redundancy to replicate a core (DMR or TMR).
Results compared after every instruction or perhaps less frequently.
The frequency of comparison determines the maximum error detection latency

IBM System/390 is a discontinued mainframe

Error Detection in Microprocessor Cores

Redundant Multithreading Without Lockstepping

Simultaneously multithreaded (SMT) cores such as the Intel Pentium 4 provided an opportunity for low-cost redundancy. An SMT core with *T* thread contexts can execute *T* software threads at the same time. If an SMT core has fewer than *T* useful threads to run, then using otherwise idle thread contexts to run redundant threads provides cheap error detection.

Error Detection in Microprocessor Cores

Dynamic Verifcation of Invariants

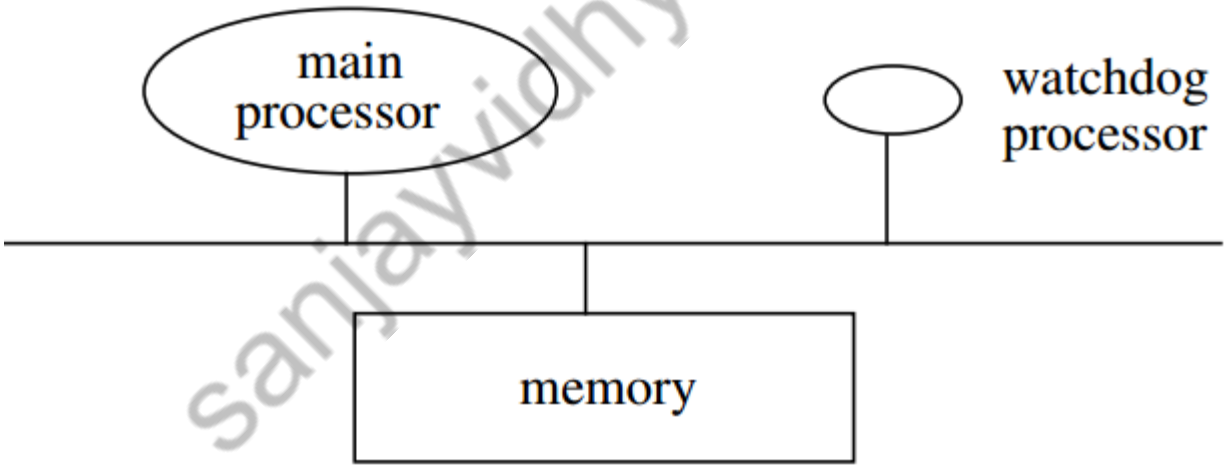
Rather than replicate a piece of hardware or a piece of software, another approach to error detection is dynamic verification. At runtime, added hardware checks whether certain invariants are being satisfied. These invariants are true for all error-free executions and thus dynamically verifying them detects errors. The key to dynamic verification is identifying the invariants to check.

Control Logic Checking: For a given instruction, some of the control signals are always the same. To detect errors in these control signals, the authors add logic to compute a fixed-length signature of these control signals, and the core compares this signature to a prestored signature for that instruction.

Data Flow Checking:

Error Detection in Microprocessor Cores

Watchdog Processors. Most of the invariant checkers we have discussed so far have been tightly integrated into the core. A watchdog processor is a simple coprocessor that watches the behavior of the main processor and detects violations of invariants. A typical watchdog shares the memory bus with the main processor. The invariants checked by the watchdog.



Error Detection in Microprocessor Cores

Using Software to Detect Hardware Errors.

SWAT used mostly simple hardware checks with a little additional software

Original Code

```
add r1, r2, r3        // r1 = r2 + r3
xor r4, r1, r5        // r4 = r1 XOR r5
store r4, 0($r6)      // Mem[$r6] = r4
```

Code with EDDI-like Redundancy

```
add r1, r2, r3        // r1 = r2 + r3
add r11, r12, r13     // r11 = r12 + r13
xor r4, r1, r5        // r4 = r1 XOR r5
xor r14, r11, r15     // r14 = r11 XOR r15
bne r4, r14, error    // if r4 !=r14, goto error
store r4, 0($r6)      // Mem[$r6] = r4
```

Error Detection in Caches and Memory

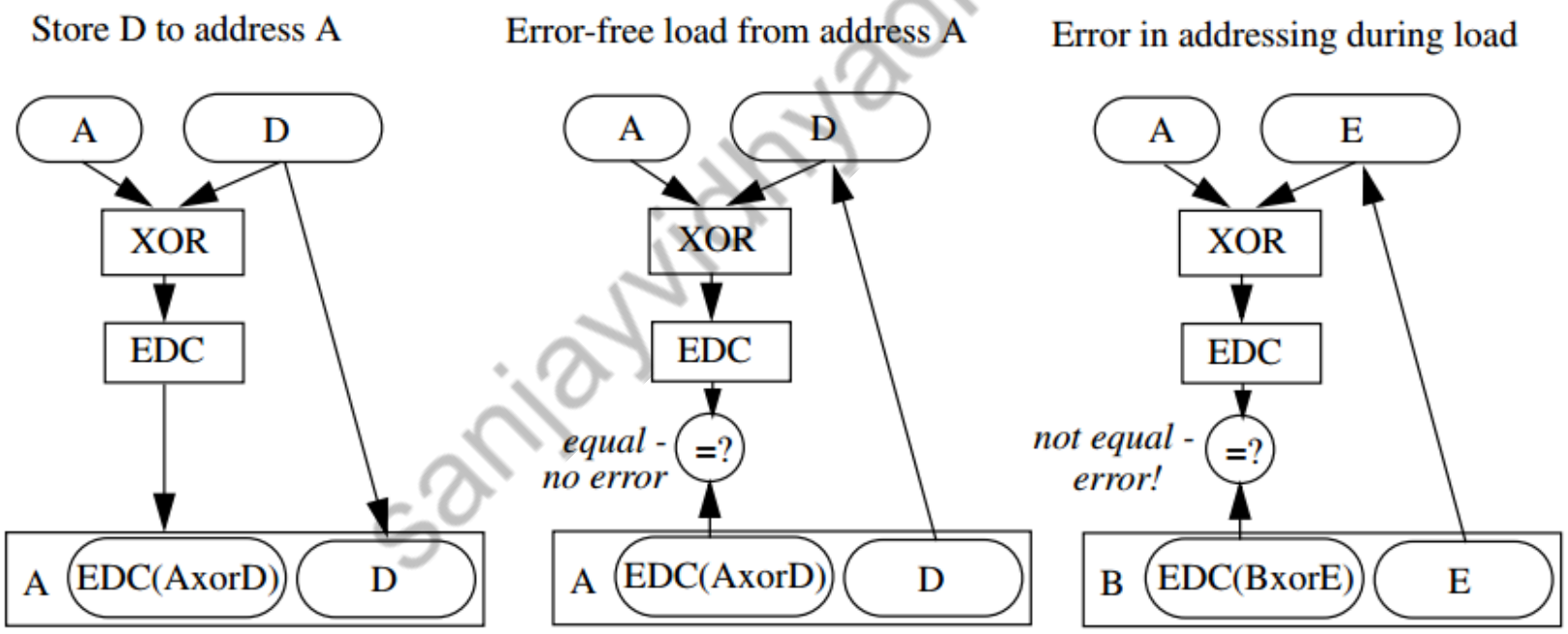
In most computers, the levels of the memory hierarchy below the L1 caches, including the L2 cache and memory, are protected with ECC. The L1 cache is either protected with EDC as in the Pentium 4, UltraSPARC IV , and Power4 or with ECC (as in the AMD K8 and Alpha 21264.

The choice of error codes represents an engineering tradeoff. Using EDC on an L1 cache, instead of ECC, leads to a smaller and faster L1 cache. However, with only EDC on the L1, the L1 must be write-through so that the L2 has a valid copy of the data if the L1 detects an error. The writethrough L1 consumes more L2 bandwidth and power compared to a write-back L1

Error Detection in Caches and Memory

Detecting Errors in Addressing

Consider the case where a core accesses a memory with address B, and the memory erroneously provides it with the correct data value *at address C*. Even with EDC, this error will go undetected because the data value at address C is error-free.

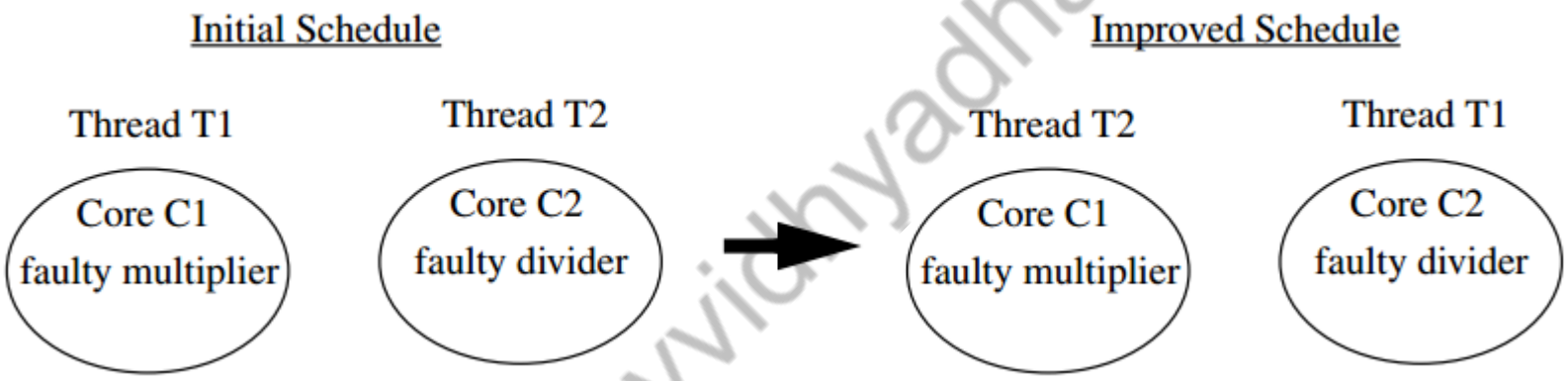


Self-Repair

MULTIPROCESSORS

1. **Core Replacement : Faulty Cores replaced with redundant core**

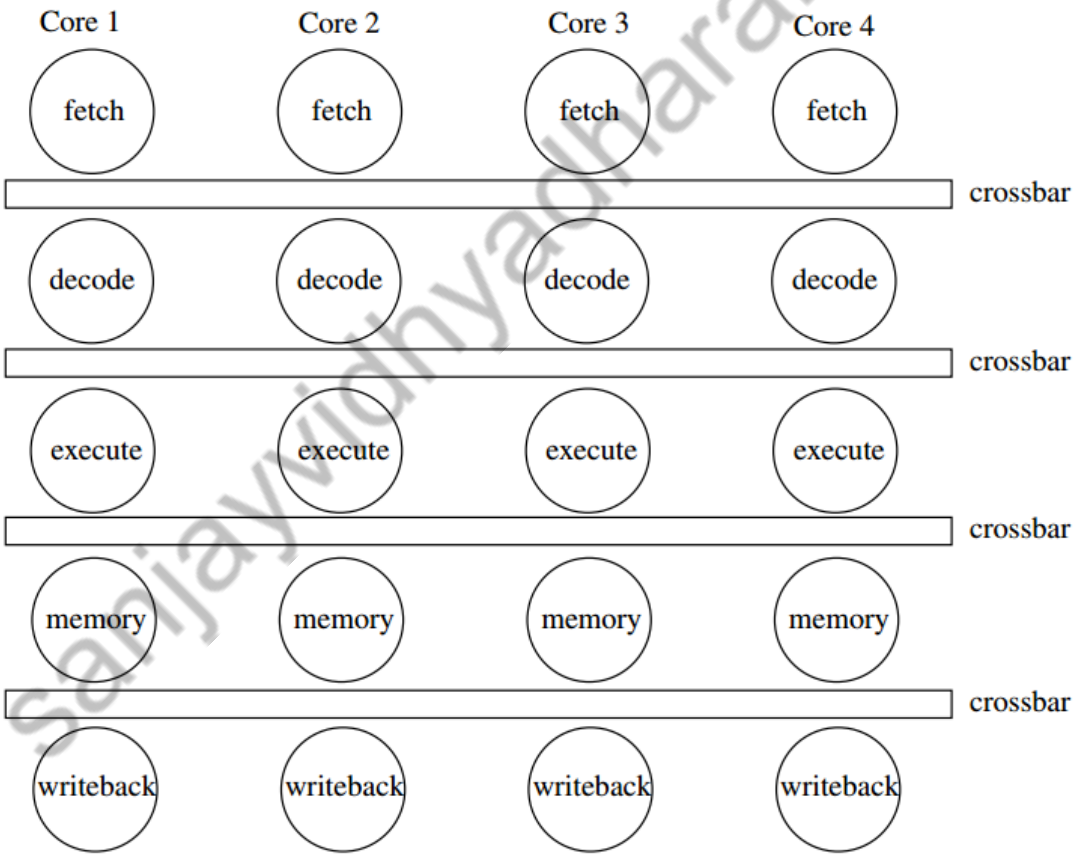
2. **Using scheduler to hide faulty functional units**



Self-Repair

MULTIPROCESSORS

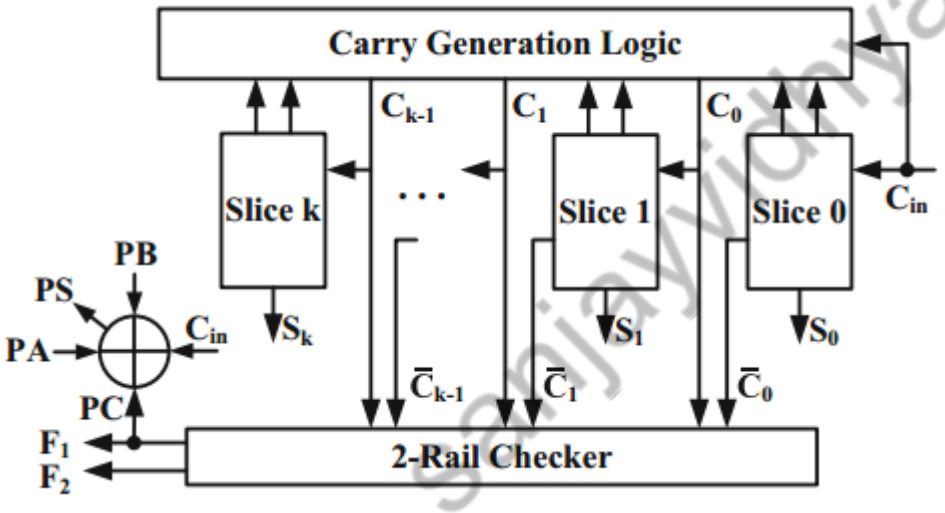
3. Sharing resources across cores

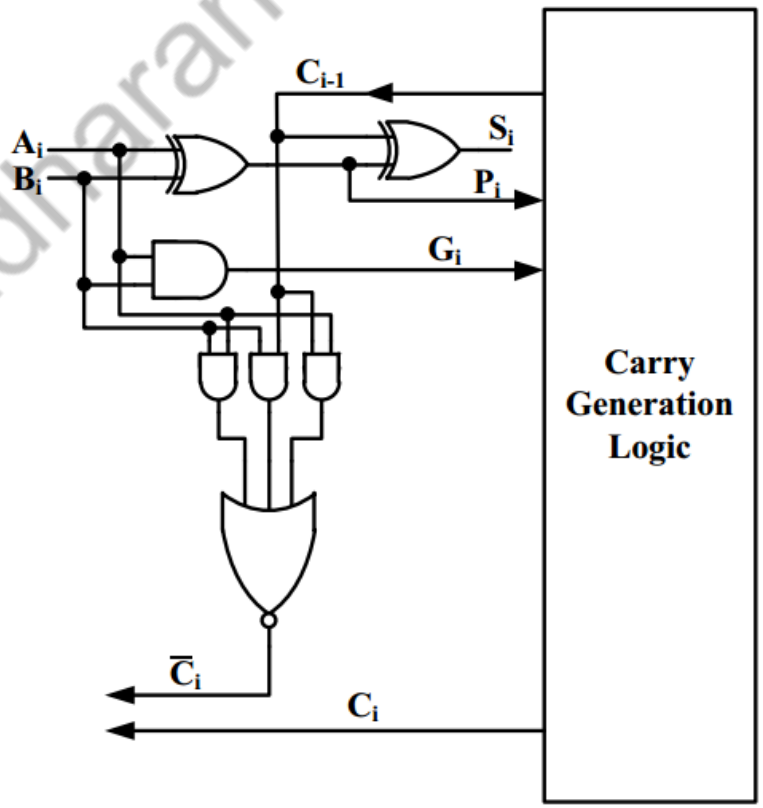


Fault Tolerant Adders

$$PS = \sum_{i=0}^{n-1} S_i = \sum_{i=0}^{n-1} A_i \oplus B_i \oplus C_{i-1}$$

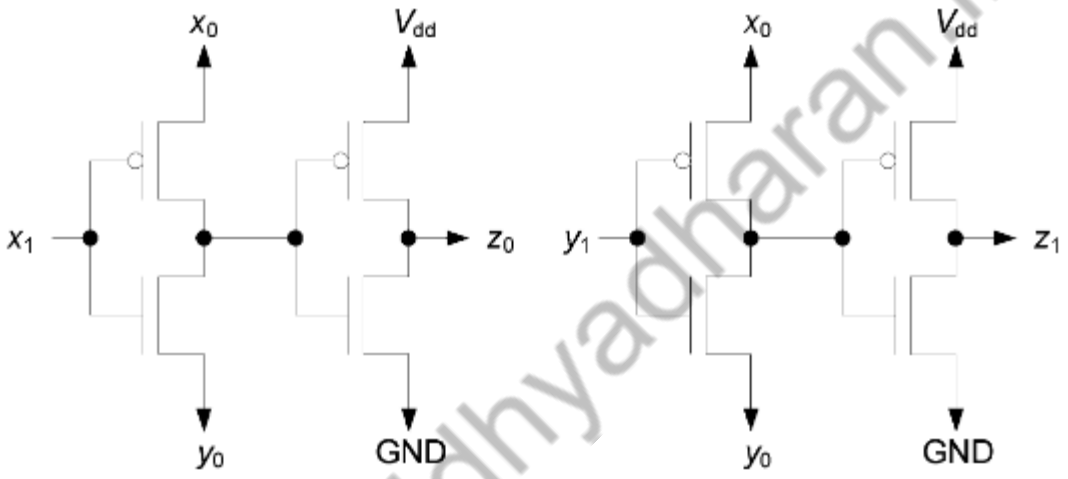
$$= \sum_{i=0}^{n-1} A_i \oplus \sum_{i=0}^{n-1} B_i \oplus \sum_{i=0}^{n-1} C_{i-1} = PA \oplus PB \oplus PC$$





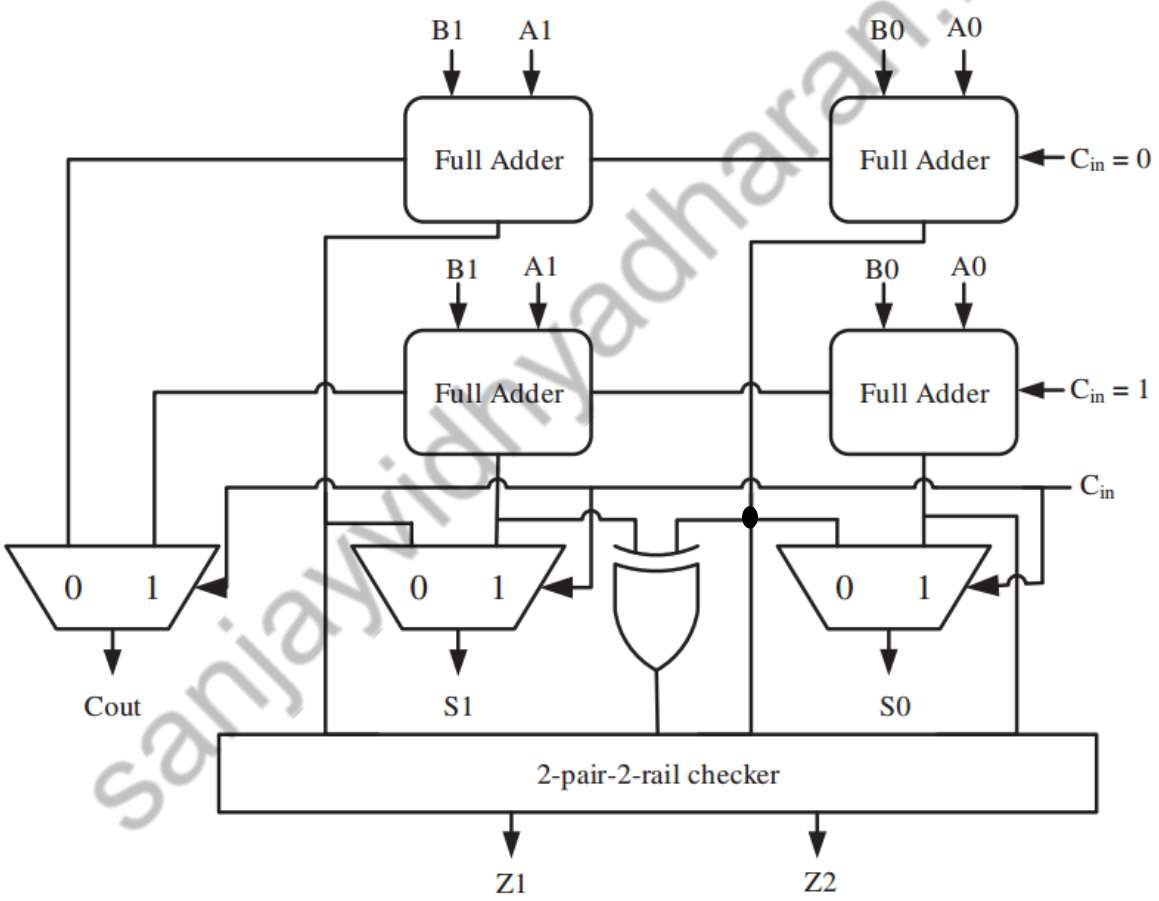
Design of Fault Tolerant Adders: A Review Ghashmi H. Bin Talib1 · Aiman H. El-Maleh1 · Sadiq M. Sait

Fault Tolerant Adders

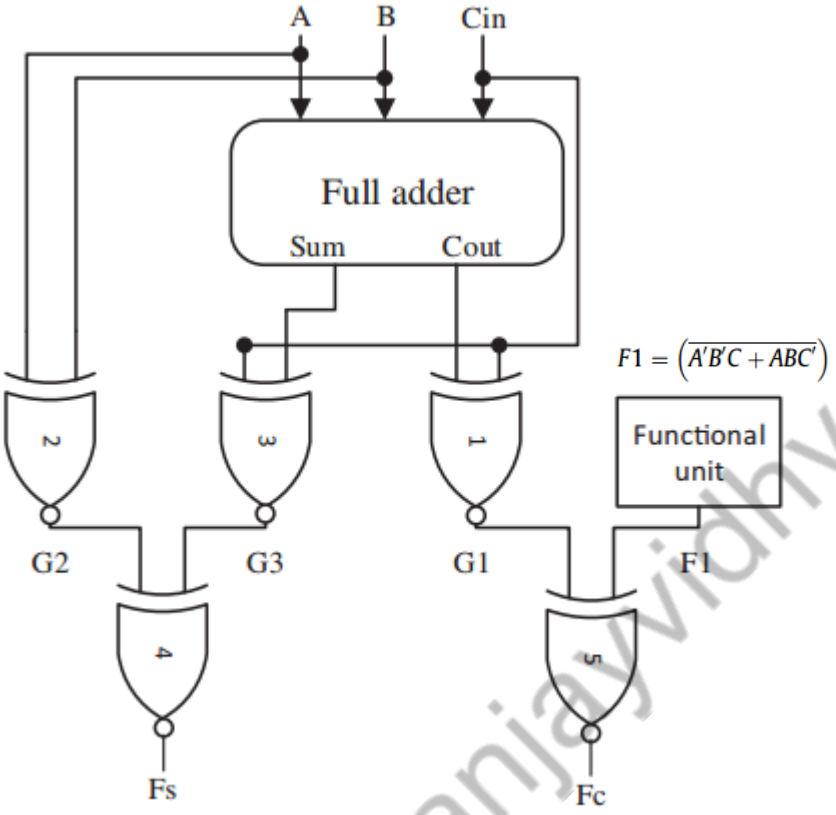


The two-pair two-rail checker receives pairs of inputs $(x0, y0)$ and $(x1, y1)$ where $x0 = y0'$ and $x1 = y1'$. The outputs of the checker are also in two-rail form $z0 = z1'$. It

Fault Tolerant Adders



Fault Tolerant Adders

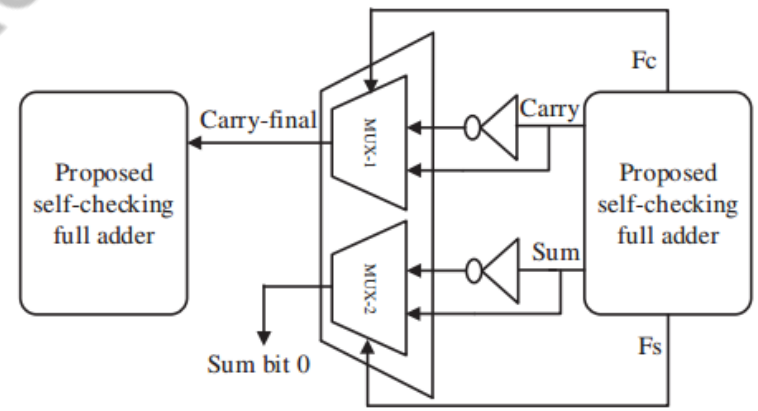


$$F1 = \left(\overline{A'B'C + ABC'}\right)$$

$$G2 = \overline{(A \oplus B)}$$

$$G3 = \left(\overline{Sum \oplus C_{in}}\right)$$

$$Fs = \left(\overline{G2 \oplus G3}\right)$$



P. Kumar, R.K. Sharma / Engineering Science and Technology, an International Journal 19 (2016) 1465–1472

Fault Tolerant Adders with Reversible Gates

Reversible Gates

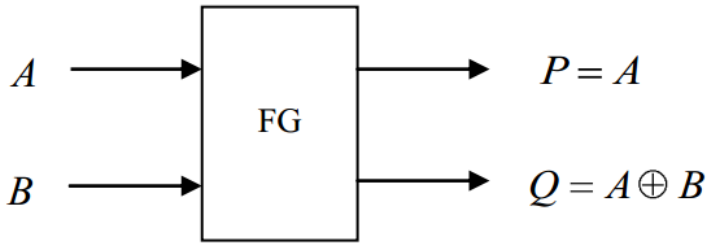


Figure 1: Feynman Gate

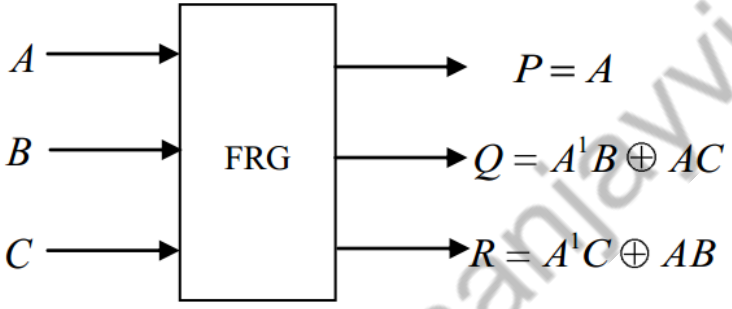
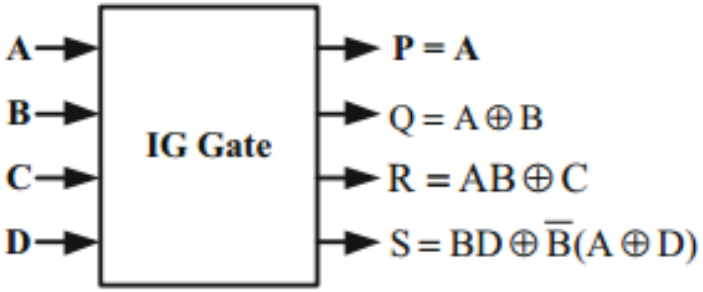
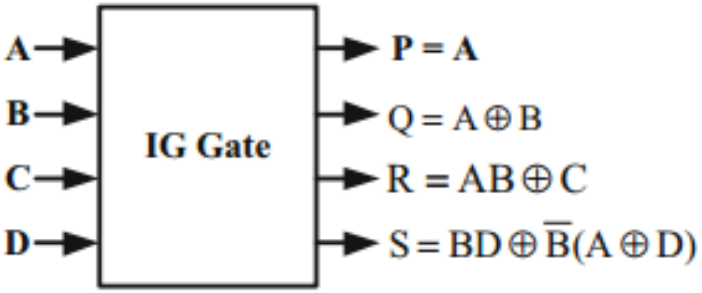


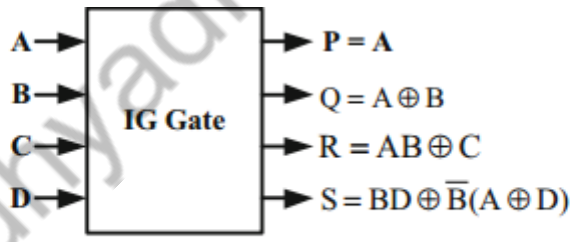
Figure 2: Fredkin Gate

When B is zero, the gate acts as a copying gate or a buffer where both the output lines contain the input A. When B is one, the complement of A is obtained at the output Q.

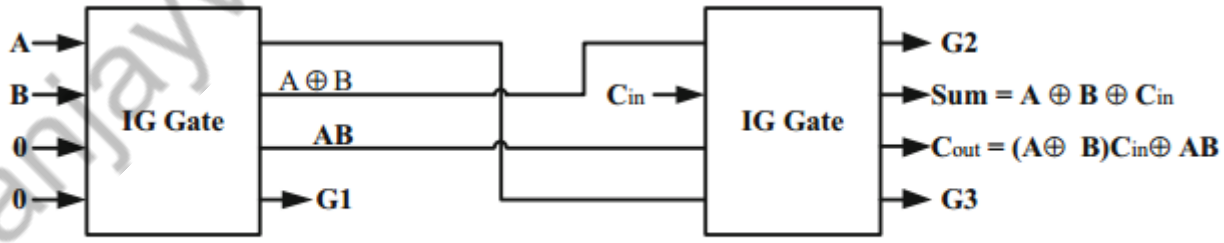


Fault Tolerant Adders with Reversible Gates





(a) IG Gate



(b) FTFA

Design of Fault Tolerant Adders: A Review Ghashmi H. Bin Talib1 · Aiman H. El-Maleh1 · Sadiq M. Sait
Arabian Journal for Science and Engineering https://doi.org/10.1007/s13369-018-3556-9

References

1. Fault Tolerant Computer Architecture by Daniel J Sorin

Thankyou