KUNGL TEKNISKA HÖGSKOLAN

International Master Program in System-on-Chip Design

Fault Tolerance in VLSI Systems

Overview

- Opportunities presented by VLSI
- Problems presented by VLSI
- Redundancy techniques in VLSI design environment
  - Duplication with complementary logic
  - Self-checking logic
  - Reconfigurable array structures

Opportunities presented by VLSI

- VLSI allows us to put more circuitry in a smaller and more reliable package
- This implies that many FT approaches that were previously not cost effective can now be used
  - duplicated processors can now be placed on a single chip (on multiple boards before)
  - triple modular redundancy becomes less costly

Opportunities presented by VLSI

- Fault detection and fault location can now be provided within the IC itself (only on board or the system level before)
- Detecting faults closer to the site of their origin minimizes the propagation of errors throughout the system

Opportunities presented by VLSI

- VLSI gives the possibility for improving the design process of fault tolerant systems by using standard library of building blocks
  - for example, we can have in the library a processor with built-in fault detection capabilities or a memory with error-correcting code

Opportunities presented by VLSI

- It is possible to use redundancy to improve the yield of VLSI circuits
  - often the yield of complex ICs is less than 10%
  - low yield implies high cost of circuit
- IC can be made usable if additional circuitry is included to replace some, or all, defective modules with spares

Problems presented by VLSI

- As the level of integration increases, the common faults are moved from the pins and the package to the semiconductor material
- The increased compexity of design increases the probability of design errors
- Lower operational voltages decrease the noise margins and increases the frequency of transient faults

Problems presented by VLSI

- Common-mode faults occur when two or more identical modules are affected by faults in exactly the same time
  - if two modules in a triple modular redundancy system experience a common-mode faults, the majority voting will produce the erroneous result
  - common-mode faults in duplication with comparison scheme would go undetected

Redundancy techniques in VLSI

- Duplication with complementary logic
- Monotonic logic
- Self-checking circuits
- Reconfigurable arrays

I. Dublication with complementary logic

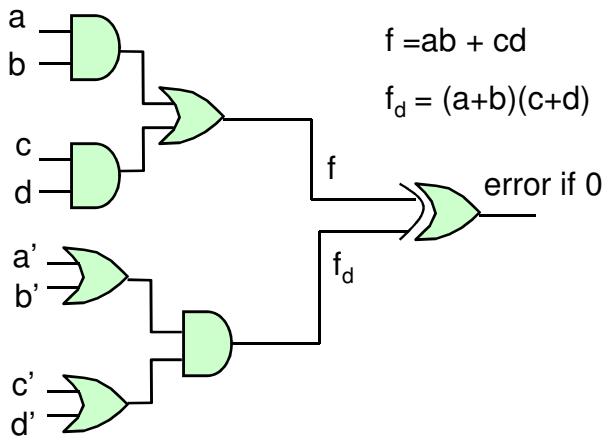
- Complementary logic to combat common-mode faults
- In complementary logic one module is designed using positive logic while the other one – using negative logic
  - In positive logic, higher voltage represents logic 1 and lower voltage represents logic 0
  - In negative logic, lower voltage represents logic 1 and higher voltage represents logic 0

Duality

- If we know function f realized in positive logic, than we can determine function realized in negative logic by computing the dual of f
- Dual of f can be obtained as follows (1):
  - replace AND with OR, and OR with AND
  - replace 0 with 1, and 1 with 0

$f = x_1x'_2 + x_3 \quad \rightarrow \quad f_d = (x_1 + x'_2).\ x_3$

Example



Advantages of using complementary logic in VLSI

- Use of dual complementation forces the use of separate masks for two modules
  - decrease the probalility of common-mode faults
- Corresponding lines in two modules are always at different voltage levels
  - a short between two such line results in one line having error, and another – not, i.e. fault will be detected

Duality in nanotechnology: CAEN

- CAEN = Chemically Assembled Electronic Nanotechnology
- Dense regular two-dimensional architecture: *nanoFabric* composed of *nanoBlocks*
  - size: a few nanometers
  - construction: self-alignment and self-assembly
  - power consumption: much less than CMOS

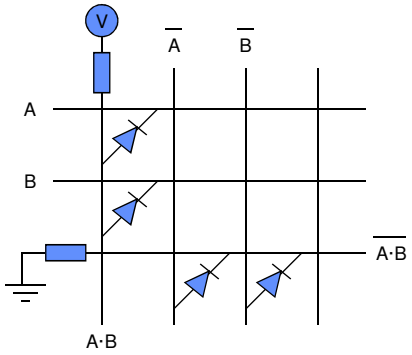
S. Goldstein and M. Budiu, ``NanoFabrics: Spatial computing using molecular electronics,'' *Proceedings of the 28th Annual International Symposium on Computer Architecture*, June 2001.

CAEN restrictions

- Due to difficulties with precise collocation of nanowires two-terminal devices are used
  - no inverters can be built
  - all logic signals have to be available both complemented and non-complemented

NanoBlock

- *nanoBlock* is a molecular logic array that can be programmed to implement a two-input Boolean function and its dual
- AND and OR are duals
- $f_d(X')$ is a complement of $f(X)$
- $A' + B'$ is a complement of $A \cdot B$

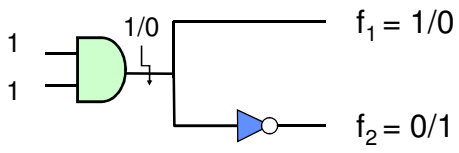


II. Monotonic logic

- A circuit is monotonic if it implements a monotonic function
  - e.g. a monotonically increasing function increases of stays unchanged when the input value increases
- Any circuit composed of AND and OR gates is monotonic
- Any single stuck-at fault will cause only unidirectional errors on the output

Example

- It is possible that a single stuck-at fault causes a bi-directional error on the output

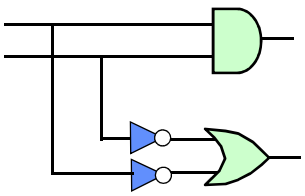


Internally monotonic circuit

- If a circuit contains inverters on primary inputs only, the internal part of the circuit is monotonic
- Any single stuck-at faults will cause unidirectional errors only
- Is the output of the circuit is encoded in Berger of m-of-n code, all such errors will be detected

Example, re-implemented

- Previous example can be re-implemented as



Internally monotonic implementations

- Circuit implementing two level sum-of-products (PLA style)
- Circuits obtained by replacing each node of a Binary Decision Diagram by a sub-circuit $x \cdot f_0 + x \cdot f_1$
  - x is the variable representing the node
  - f_0 and f_1 and the co-factors

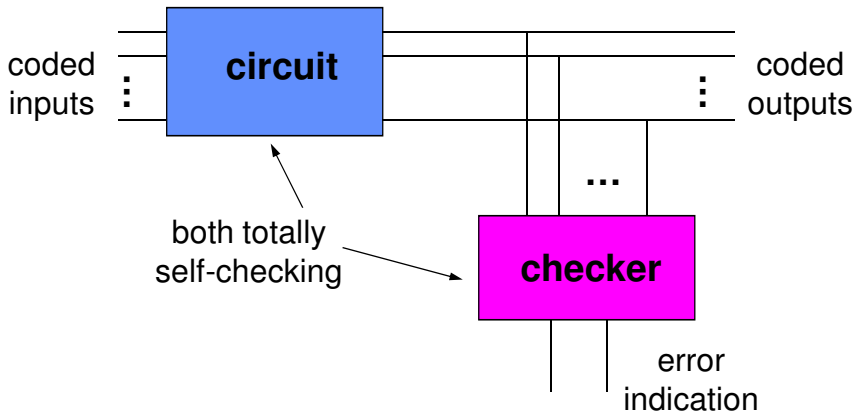
III. Self-checking circuits

- A self-checking circuit automatically detect a fault during normal operation, without applying any extra tests
- The basic idea is to code the outpus and/or inputs so that only fault-free circuit will produce a valid code word on the output
- In presence of fault the output is an invalid code word

Totally self-checking circuits

- A circuit is totally self-checking if:
  - For any valid input code word, any single fault either produce an invalid code word on the output, or doesn't produce the error on the output (fault secure property)
  - Any single fault is detectable by some valid input code word (self-testing property)

Basic structure of a totally self-checking circuit

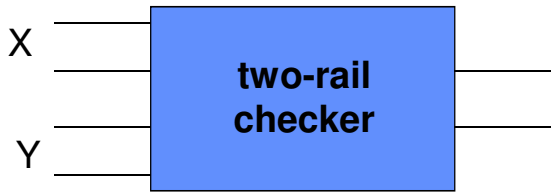


Checker

- Checker determines whether the output of the circuit is a valid code word
- Checker also determines whether a fault occur within itself

Design of two-rail checker

- Compare two input words $X=(x_1,x_2,..,x_n)$ and $Y=(y_1,y_2,\ldots, y_n)$ which should normally be complementary: $Y = X'$
- Outputs are dual functions



f	f_d	conclusion
0	0	error
0	1	OK
1	0	OK
1	1	error

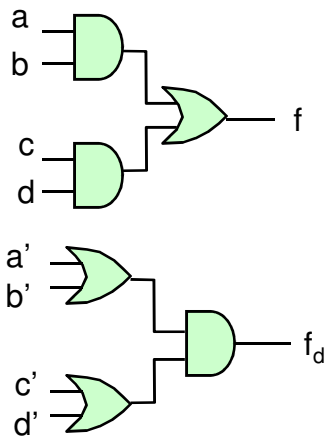
Is dual-rail checker totally self-checking?

- It is fault secure:
  - Any single fault on primary input will result in a non-valid code word and produce non-complementary outputs (will be detected)
  - Any single internal fault will affect only one output (dubicated complemented circuits are physically separated) and produce non-complementary outputs (will be detected)
- It is self-testing:
  - because it is nonredundant

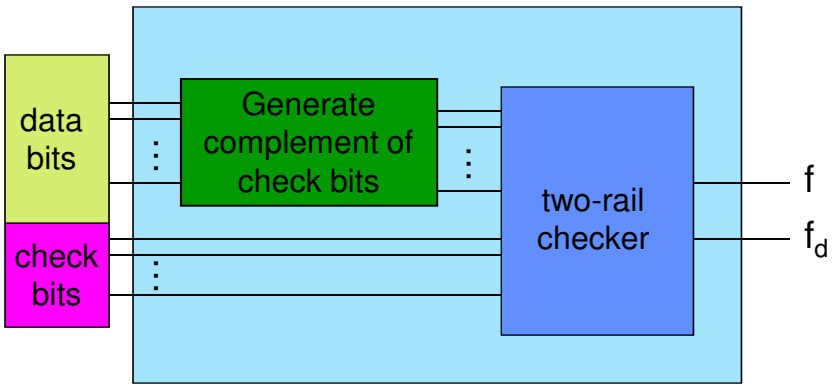
Nonredundant circuit

- A circuit is nonredundant if, for every line k within the circuit, the output of the circuit is sensitive to the change in the value on line k for at least one input combination

Totally self-checking checker for a 2-of-4 code



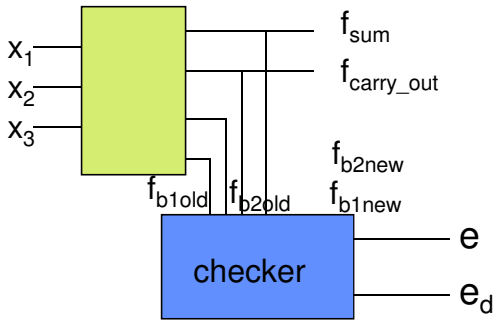
General structure of a totally self-checking checker for separable codes



Example of circuit output encoding: full adder with Berger code

x_1	x_2	x_3	f_{sum}	f_{carry_out}	$f_{berger1}$	$f_{berger2}$
0	0	0	0	0	1	1
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
1	0	0	1	0	1	0
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	0	1

Self-checking adder using Berger code



IV. Reconfigurable arrays

- VLSI allows efficient implementation of array structures
- 100s or 1000s of processing elements can be connected in a near-neighbor structure
- 2 problems:
  - A chip with 1000s of processing elements will likely contain faulty elements after manufacturing
  - Faults occuring during the operation should be handled through reconfiguration

Three types of reconfiguration

- Fabrication-time
  - Performed immediately after manufacturing
- Compile-time
  - Performed after each use of the array, but not during the normal operation
- Real-time
  - Performed during the normal operation (without interraption)

Fabrication-time reconfiguration

- Primary goal is to increase the yield
  - in VLSI yield can be 10% or less
- External tests are used to detect and locate the faults off-line
- Reconfiguration algorithms are used to find an interconnection pattern to create a functional array
- The reconfiguration is usually irreversible

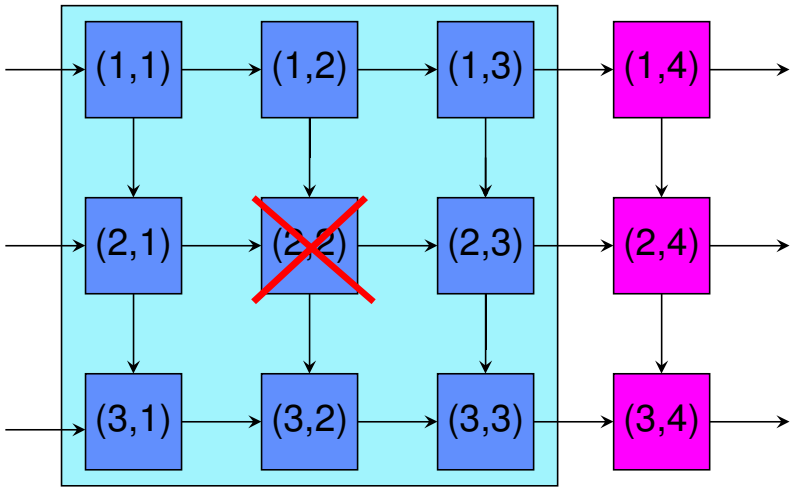
Compile-time reconfiguration

- Detection algorithm detects faults on-line
- The array is shut down
- The faults are located off-line
- Reconfiguration algorithms is apply to remove the faulty ellements
- No time-constraints are placed of the repair time

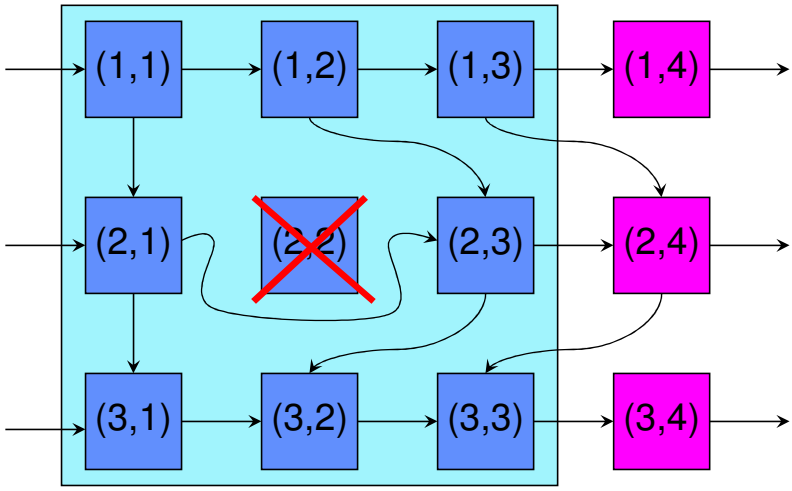
Techinques for compile-time reconfiguration

- Use of a single spare row or column an the rippling replacement
- Used both a spare row and a spare column and the fault-stealing replacement
- Used multiple spare rows and a spare columns and the repair-most replacement

Rippling replacement



Rippling replacement



Run-time reconfiguration

- The faulty element is either masked or detected, located and removed immediatelly
  - In some applications (real-time control systems) errors are allowed for a short period, if they can be repaired quickly
- Often both masking an reconfiguration is performed

Techinques for run-time reconfiguration

- Successive elimination of rows and/or columns where the faulty element is detected
  - Set switches so that complete row/column is bypassed
- Algorithm-based reconfiguration
  - Use techiniques specific to the particular algorithm, e.g. matrix multiplication

Next lecture

- Case study (not covered in the text book)