Fault-tolerant systems based on coding technique

Tatjana Nikolic
Faculty of Electronic Engineering, University of Nis, Serbia

Outline

- Fault-tolerant systems
- Fault-tolerant design
- Redundancy
- Coding
- Fault-tolerant communication

Motivation

- No matter how robust the design is, **100% fault free** design is **impossible**
  - there is not a single large piece of software or hardware that is free of bugs
  - space shuttles have flown with potentially serious bugs
- The challenge of designing complex systems
  - techniques to reduce the number of faults
- Systems
  - **recognize** the existence of **faults**
  - incorporate techniques **to tolerate** these faults
- **Fault-tolerant (FT)** systems
  - to achieve the needed **reliability and availability**
  - to **tolerate faults** by detecting failures
  - to isolate defect modules (the rest of the system can operate correctly)

Performances and reliability

- Design of complex SoC:
  - ↑ IP cores/chips functionality
  - – efficient data transfer with reduced number of wires
  - ↓ power consumption
  - **enhancing reliability with as low as possible area and time overhead**

- SoCs – focused on the computational aspects
  shrinking technology and growing complexity
  - high performance, reliable **interconnection architecture**

- To increase system reliability
  - two aspects of the design

computation-based	communication-based

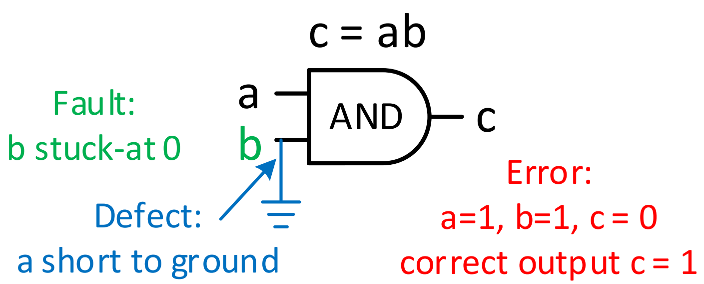
Fault-tolerant systems

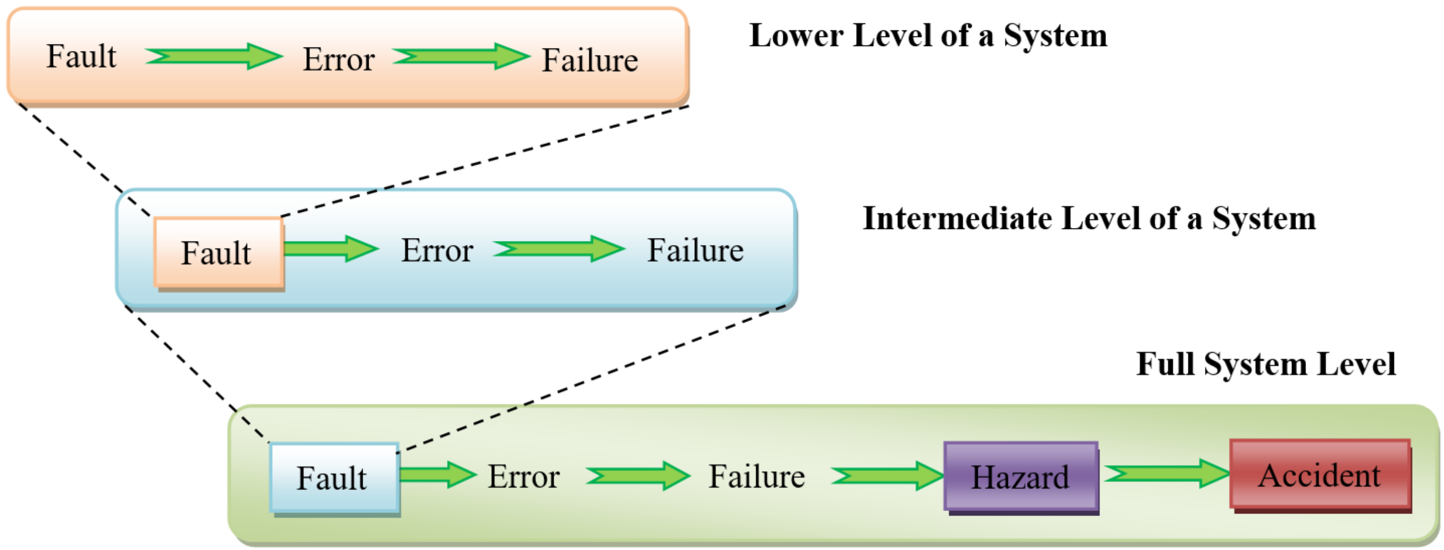
- Applications that require FT:
  - **Critical Application**: Aircraft, Nuclear reactor, Medical equipment
  - **High Computing Systems**: Complex systems with a million devices
  - **Harsh Environment**: Systems open to high vibration, temperature, humidity, electromagnetic disturbances, particle hits

- **Computers in aerospace systems** – a prime example
  - **life-critical** – passengers or astronauts
  - must operate **fault-free for many hours** (space missions)
  - high altitude aircraft – **in harsh environments**
    - The Sun – major and highly variable — source of particles
    - Airplanes – low rate of particle hits – conventional FT
    - Spacecraft – higher levels of radiation – more extensive protection
  - **big-budget items** – considerable costs of FT

Fault, error and failure

- **Fault**: a representation of a "defect" at the function level
  - frozen memory bit
  - stuck-at fault
  - alpha particle hit or cosmic ray ionization
  - uninitialized variable in software
- **Error**: a manifestation of fault; can cause failure
  - an incorrect result of a calculation
  - incorrectly transmitted data
- **Failure**: a system failure (it operates differently from intended)





Fault classification

- **Duration**: Hardware faults
  - Permanent fault – always present after its occurrence
    - burned-out lightbulb, broken wire
  - Transient fault – occurs randomly and only once
    - memory cell with contents that are changed spuriously
  - Intermittent fault – occurs at intervals, irregular; from time to time
    - loose electrical connection

- **When they were introduced**: phases of the system's lifetime
  - design phase
  - system implementation
  - system operation due to hardware degradation or harsh environments
    - high levels of radiation
    - excessive temperatures

Fault–tolerant design

- Technology scaling ⇒ increased sensitivity to faults
  - crosstalk, power supply noise, cosmic rays and alpha particles
- A good FT system design
  - study of design, failures, causes of failures, system response to failures

Dependability

a measure of user's trust into the system

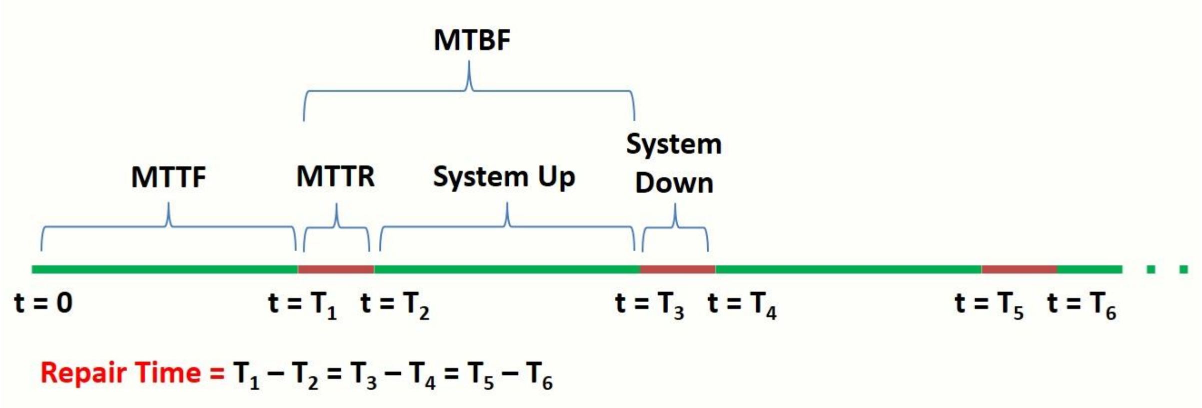
Reliability – R(t)

continuity of correct service **as specified**

Availability – A(t)

readiness for correct service **when requested**

MTTF – Mean Time To Failure
MTBF – Mean Time Between Failures
MTTR – Mean Time To Repair



Redundancy

- Redundancy – the basic principle of FT design
  - it is incorporated ⇒ system can operate correctly in the presence of faults
- Redundancy
  - having **more of a resource** than is minimally necessary
  - **masks** or works around failures
- Forms of redundancy:
  - **hardware**
  - **software**
  - **information**
  - **time** redundancy
- Hardware faults
  - hardware, information, or time redundancy
- Software faults (bugs)
  - software redundancy

Forms of redundancy

- **Hardware redundancy**
  - incorporating extra hardware
    - to detect or override the effects of a failed component
  - drawback:
    - cost of the extra hardware, power consumption
  - hierarchy:
    - system level, multiple modules, individual devices

- **Information redundancy**
  - error detection and correction coding:
    - extra bits (check bits) are added to the data bits

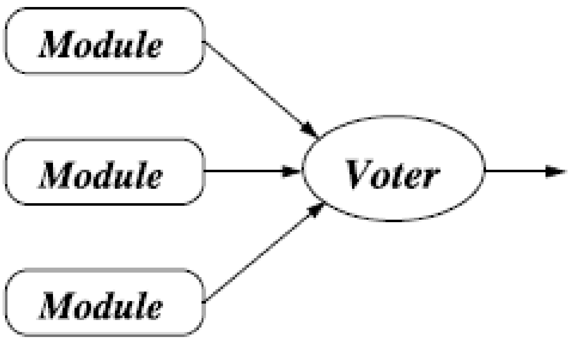
- **Time redundancy**
  - reexecution of the same program on the same hardware

- **Software redundancy**
  - execution of different software modules (performing the same functionality)

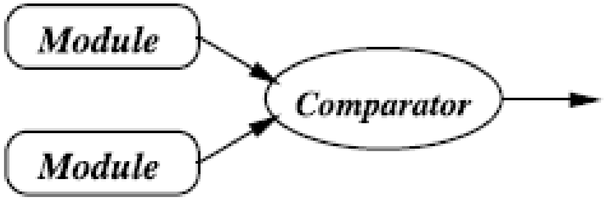
Hardware redundancy

- Resilient structures with redundant components

- **M-of-N system**
  - N modules and a voter
  - at least M of them for proper operation
  - **Triplex – triple modular redundant (TMR)** system
    - three identical modules; outputs are voted on
    - 2-of-3 system: most (2 or 3) modules work correctly



- **Duplex system**
  - two hardware modules and a comparator
  - comparator – module outputs are in agreement
    $\Rightarrow$ the result is assumed to be correct

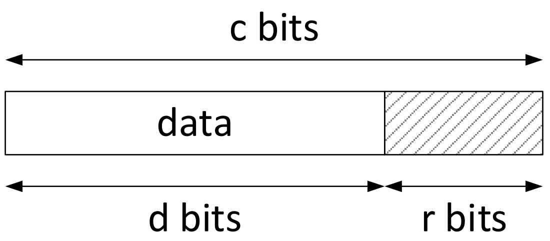


System reliability: $$R_{M\_of\_N}(t) = \sum_{i=M}^{N} \binom{N}{i} R^{i}(t)\,[1 - R(t)]^{N-i}$$

Reliability of module

Information redundancy

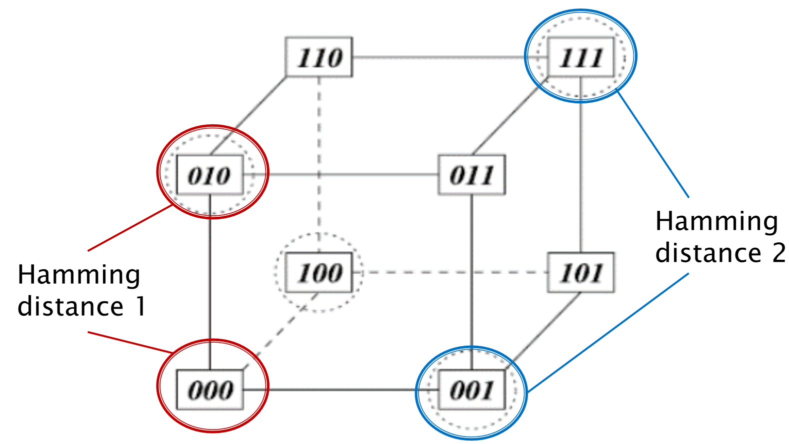
- **Coding** – common form
  - adds check bits to the data
  - verification of correct data
  - correction of erroneous data bits, in some cases
- **Code** – the set of all codewords
  - d-bit data word -> encoded -> c-bit codeword
  - 2^c binary combinations – valid and invalid codewords
  - an invalid codeword indicates an error



- The rate of a code – the fraction of bits that are nonredundant – (d/c)
- A separable code – separate fields for data and check bits
- A nonseparable code – data and check bits integrated together

Hamming distance, code distance

- **Hamming distance** – two codewords
  - the number of different bit positions
- 3-bit word space



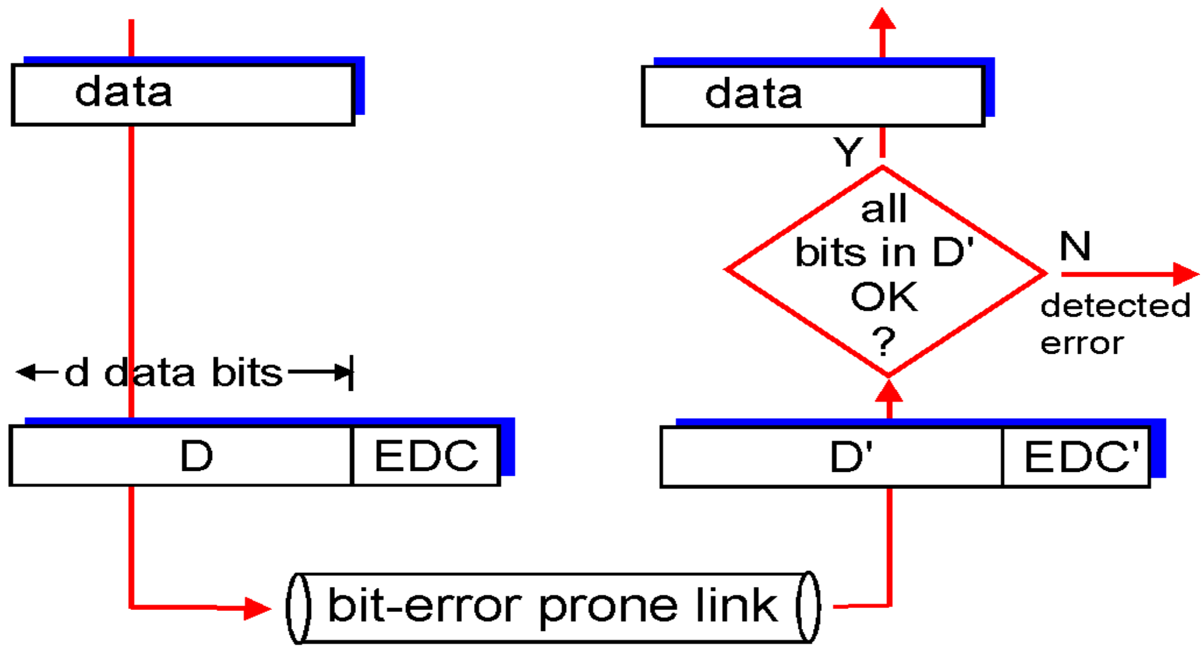
- **Code distance**
  - minimum Hamming distance – any two **valid codewords**

- to detect up to k bit errors
  - code distance – at least **k+1**
- to correct up to k bit errors
  - code distance – at least **2k+1**

- Code – detects any single-bit error
  - four codewords {001, 010, 100, 111} – distance of 2
- Code – detects any single- or double-bit error
  - codewords {000, 111} – distance of 3
  - corrects any single-bit error
    - if double-bit errors are not likely

Error-detecting / error-correcting codes

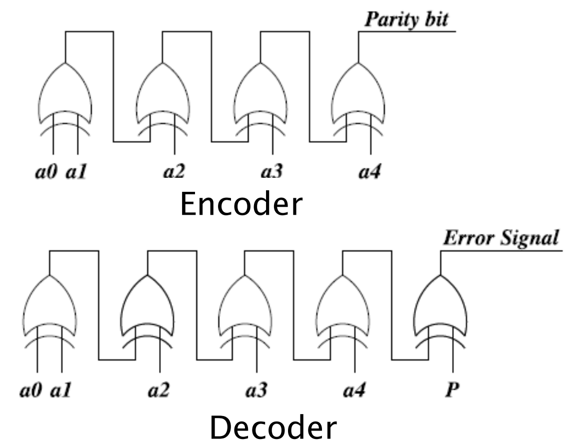
- Parity
- Checksum
- M-of-N
- Berger
- Cyclic codes



D = Data protected by error checking
EDC= Error Detection and Correction bits (redundancy)

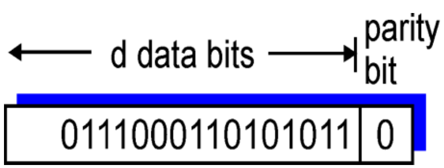
- larger EDC field ⇒ better detection and correction

Parity codes

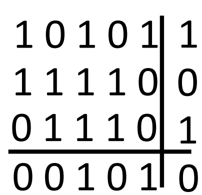


- A parity-coded word – d data bits and an extra bit
  - even or odd parity
- Variations of the basic parity code:
  - byte-interlaced parity code – a parity bit is assigned to every byte
  - overlapping parity – the data is organized in a two-dimensional array

single bit parity:
- detect single bit errors
  - distance – 2

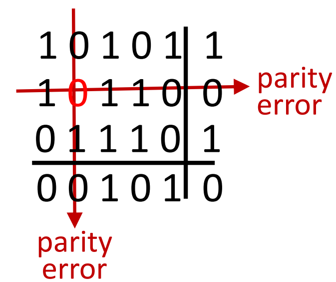


two-dimensional bit parity:
- detect and correct single bit errors

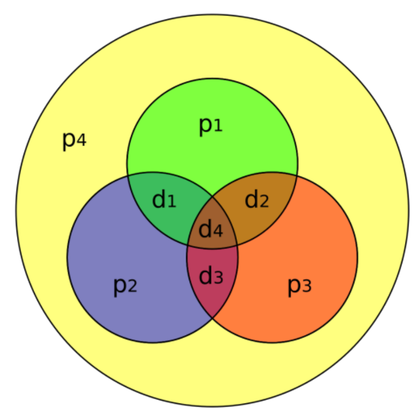


no errors

detected and correctable single-bit error:



- Hamming(7,4) code – adds three parity bits to four data bits
- Hamming(8,4) single-error correcting/double-error detecting (SEC/DED)
  - to improve the error detection capabilities
  - adds an extra check bit



Checksum

- Checksum – the basic idea
  - to add up the blocks of data and to transmit this sum with data
  - the receiver adds up the data and compares this sum with the checksum it received
- Data words – d bits
  - (A) Single precision – modulo-2^d addition
  - (B) Double precision – modulo-2^{2d} addition
  - (C) Residue checksum – takes the carry out of the d-th bit as an end-around carry

0000	0000	0000
0101	0101	0101
1111	1111	1111
0010	0010	0010
0110	00010110	0111
(A)	(B)	(C)

M-of-N codes

- M-of-N code – unidirectional error-detecting code
  - Unidirectional errors – all the affected bits change in the same direction (0 → 1 or 1 → 0)
- M-of-N code
  - N-bit codeword – M bits are 1
- Any single-bit error will be detected
  - changes the number of 1s – to M+1 or M−1

2-of-5 code

Digit	Codeword
0	00011
1	00101
2	00110
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

Berger code

- Berger code – a unidirectional error detecting code
  - separable
- For d data bits – $\log_2(d+1)$ check bits
- Encoding process
  - count the number of 1s in the data word
  - express this count in binary representation
  - complement it
  - append this quantity to the data
- Example: to encode 11101
  - there are four 1s in it,
  - it is 100 in binary
  - complementing results in 011
  - the codeword will be 11101011

Cyclic codes

- **Encoding**
  - multiplying (modulo-2) the data word by a constant number:
  - the coded word is the product
- **Decoding**
  - dividing by the same constant:
  - if the remainder is nonzero, an error has occurred
- Cyclic codes
  - codeword $a_{n-1}, a_{n-2}, \ldots, a_0$ – its cyclic shift $a_0, a_{n-1}, a_{n-2}, \ldots, a_1$ is also a codeword
  - Example: {00000, 00011, 00110, 01100, 11000, 10001, 00101, 01010, 10100, 01001, 10010, 01111, 11110, 11101, 11011, 10111}
- Example: Encoding the data word 10001100101 by multiplying with 11001 and decoding by dividing
  - Codeword: 110000100011101

```
        10001100101
  ×           11001
  -----------------
        10001100101
       00000000000
      00000000000
     10001100101
    10001100101
  -----------------
    110000100011101
```

```
110000100011101 : 11001 = 10001100101
11001
-----
    10100
    11001
    -----
     11010
     11001
     -----
        11111
        11001
        -----
          11001
          11001
          -----
          00000
```

Concurrent error detection

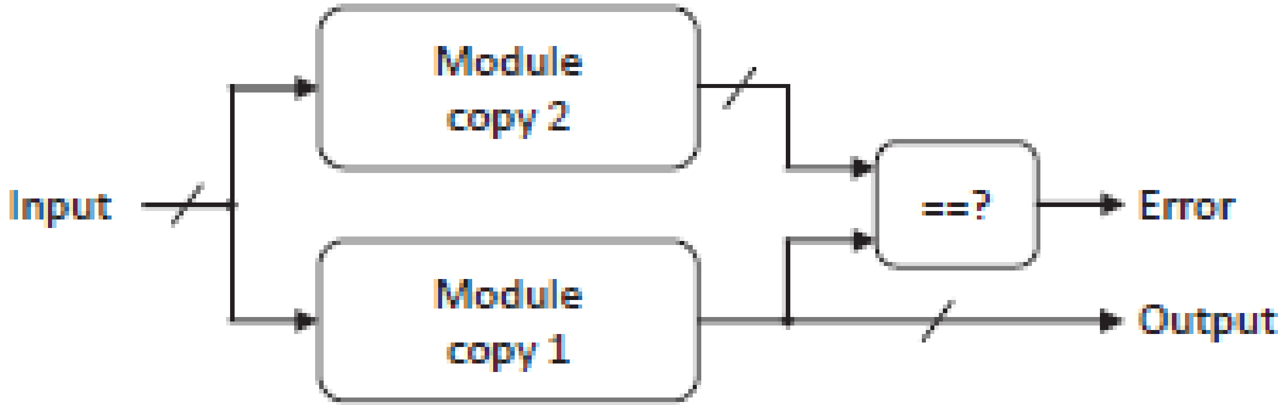
- Detection of an fault – first step in FT systems

- **Concurrent** (on-line, implicit) **error detection, CED**
  - circuit level technique – during system operation
- CED is focused on mission critical systems
  - high levels of reliability
  - the cost is of less importance
- Objective of CED:
  - detection of errors as early as possible

- **Self-checking, SC** – hardware failure detection
  - the ability to verify on-line whether there is any faults
  - allows faults to be detected, preventing data contamination
- **Techniques** for designing **SC** circuits:
  - duplication with comparison
  - use of error detecting codes

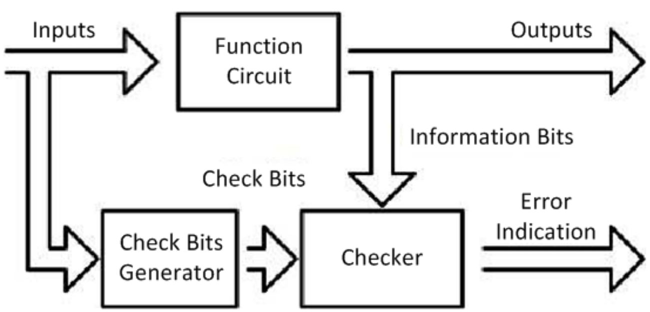
Duplication – self-checking technique

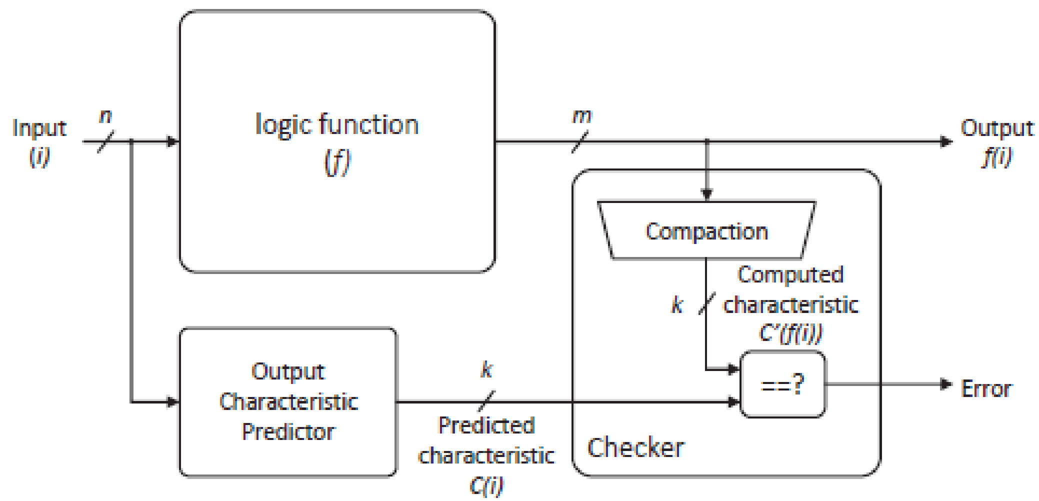
- Duplication with comparison
  - CED based on hardware redundancy
- Design
  - two identical copies of a circuit compute the results
  - the comparator examines the identity property between their outputs and flags error



Self-checking circuits based on EDC

- Self-checking circuit
  - Functional block (Function circuit & Check bit generator) – produces encoded outputs
  - Checker – monitors the output and signals the appearances of a noncode word
- Error detecting codes, EDC
  - introduce redundancy in information representation
  - improve the data integrity of the Function circuit
    - implementing a block which **predicts** some characteristic





Synthesis of SC circuits

- 12 combinational circuits of standard architecture
- the insertion of CED in VHDL RTL description
- a synthesis tool to implement the SC into FPGA

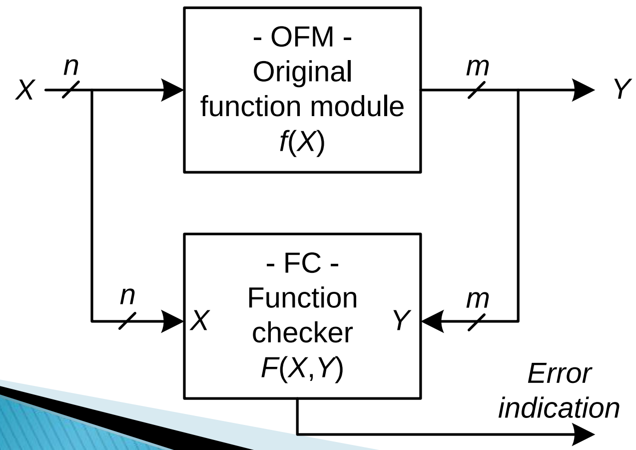
12 circuits		orig	dup	Ber	pg1	pg2	pg4
AVERAGE	area overhead (%)	0	157	251.1	77.3	119.6	90.9
	speed decrease (%)	0	61.9	69.3	19.1	38.6	32.6

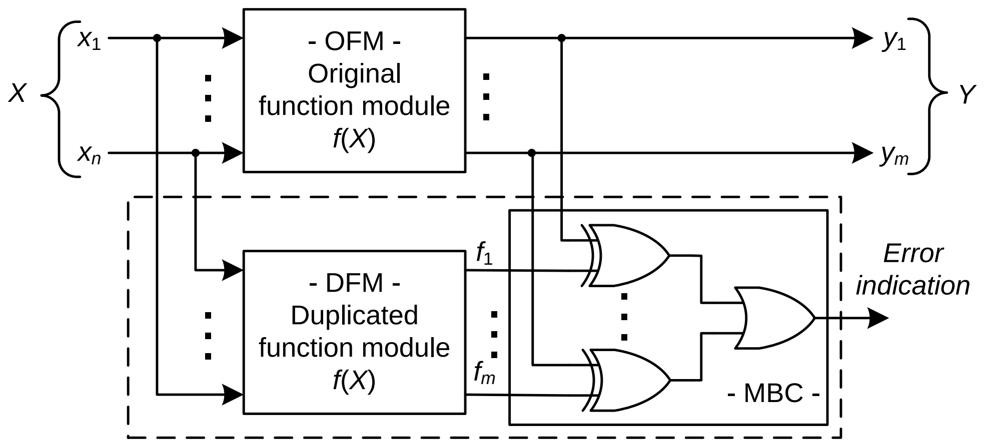
A parity check scheme is superior one
- least amount of **average area overhead and speed decrease**

Partially SC circuits

- Partial function checking
  - compromise: hardware overhead (<100%) and error-detecting (<100%, >90%)
  - duplicated function module & m-bit comparator ⇒ **function checker, FC**
- The **FC** implements characteristic function, F, of the original function f
  - F(X,Y)=0 if Y=f(X), and F(X,Y)=1 if Y≠f(X)

- **Partial function checker (PFC)** implements function F*(X,Y)
  - F* under-approximates F – if F*(X,Y) agrees with F(X,Y) when F(X,Y)=0
  - F*(X,Y) – arbitrary selected when F(X,Y)=1 – to reduce the complexity of F*





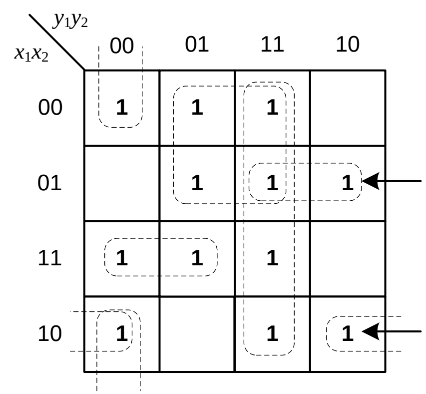
Approximation of characteristic function

- Challenge – algorithms
  - good under-approximations of the F with minimal cost
- Truth table – 2-input 2-output function
  - characteristic function – F
  - two under-approximation functions – F_1^*, F_2^*

x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	0

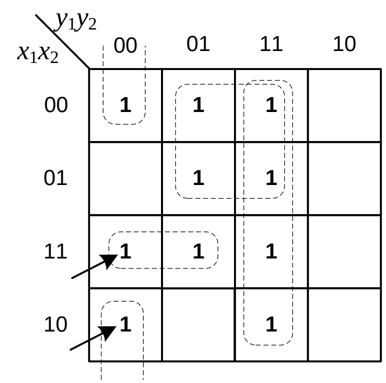
f: $y_1 = x_1'x_2' + x_1x_2$
$y_2 = x_1x_2'$

x_1	x_2	y_1	y_2	F	F_1^*	F_2^*
0	0	0	0	1	1	1
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	1	1	1	1
0	1	1	0	1	0	0
0	1	1	1	1	1	1
1	0	0	0	1	1	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	1	1	1	1



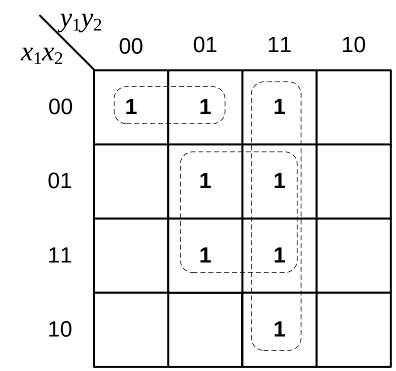
$F = x_1x_2'y_2' + x_1'x_2 + x_1x_2y_1' +$
$+ x_2'y_1'y_2' + y_1y_2 + x_2'y_1y_2$

(cv=1.00, 6 prod. terms/16 literals)



$F_1^* = x_2'y_1'y_2' + x_1'y_2 + x_1x_2y_1' + y_1y_2$

(cv=0.83, 4 prod. terms/10 literals)



$F_2^* = x_1'x_2'y_1' + x_2y_2 + y_1y_2$

(cv=0.75, 3 prod. terms/7 literals)

Synthesis of PSC circuits

- A set of benchmark circuits – to demonstrate the efficiency of the PSC

Circuit (*f*)	Characteristic function - *F*	Approximated characteristic function - F^*				
	cv=1.0	*cv*=0.99	*cv*=0.98	*cv*=0.95	*cv*=0.90	*cv*=0.85
Average overhead (%)	**184**	**102**	**86.8**	**68.1**	**49.6**	**40.2**

The trade-off between the conflicting objectives

low hardware overhead

high error coverage

Fault tolerant communication

- Interconnection architecture – high performance and reliable
  - complexity of the contemporary SoC
- Large number of wires for faster communication:
  - interconnections – dominant source of energy consumption
  - reliability decreases
    - susceptible to noise sources, crosstalk, radiation
- Interconnects – unreliable medium
  - due to faults
- Design of SoC interconnection architectures

Fault tolerance mechanisms for improving communication reliability

Coding schemes in communications

- Coding in interconnections
  - technology independent solution
  - optimization of interconnect design
    - energy efficiency, speed, reliability

- CED interconnect scheme – preserves fault-secure
  - produces correct output
  - indicates erroneous situations

- Interconnect networks – shared bus – TDMA, CDMA

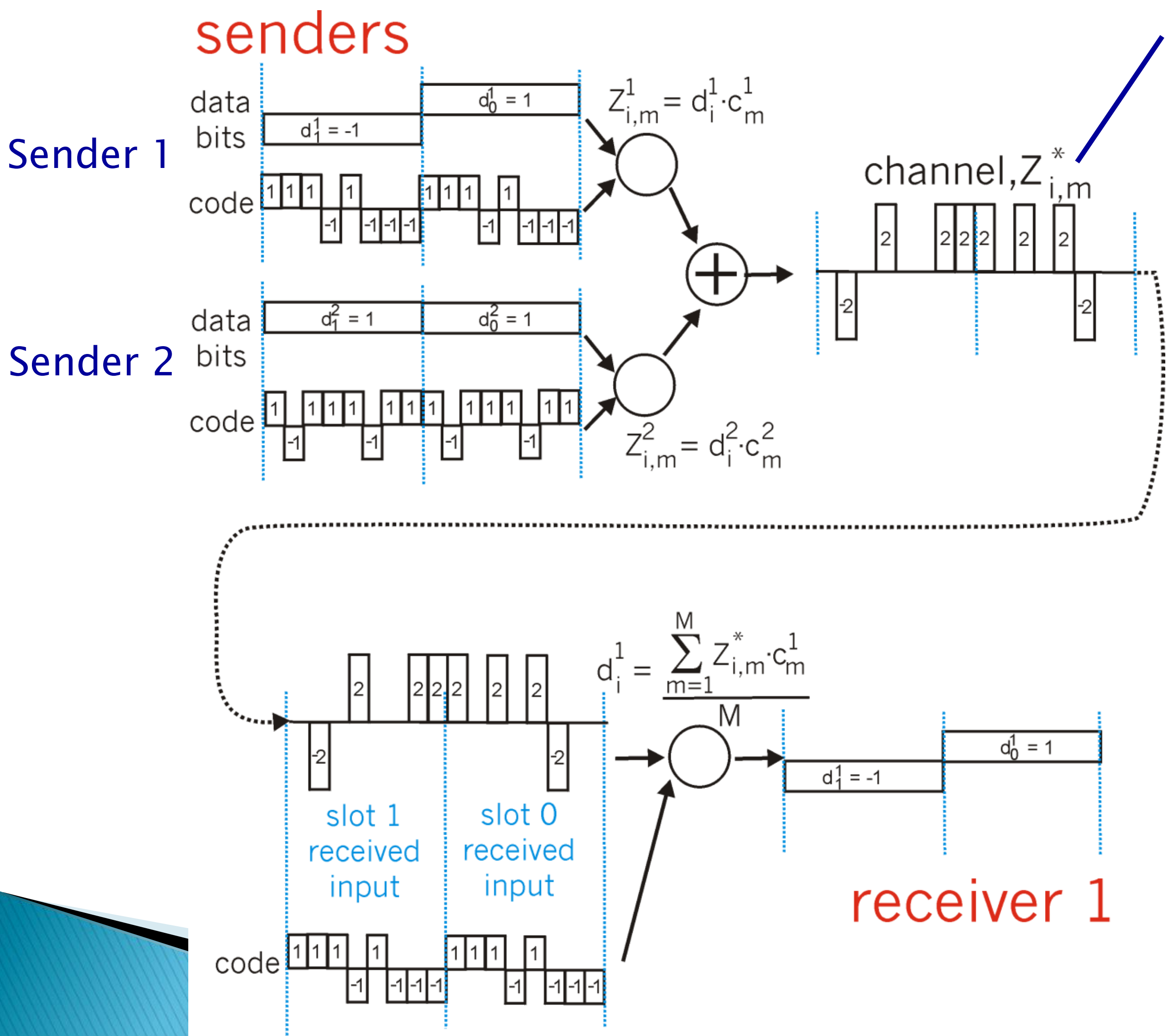
- **CDMA:** Code Division Multiplexed Access
- sharing medium based on the use of orthogonal codes
  - to separate simultaneously transmitting channels

- CDMA technique – SoC
  - efficient (high-bandwidth) communication protocol

CDMA

- Spread spectrum technique
  - unique "code" assigned to each user
    - "chipping" sequence (code) to encode data
  - multiple users "coexist" and transmit simultaneously
    - minimal interference
  - encoded signal = (original data) X (chipping sequence)
  - sum-chips = summed chips of the same weight (encoded signals)
  - decoding = (sum chips) X (chipping sequence)
- Spreading data by CDMA
  - **fault tolerant mechanism** – information redundancy
  - expands data bandwidth
    - allows data **recovery**
    - improves the **reliability** in spite of a few spreading bits loss
      - a bit-error (in the sum-chip), can be masked by the rest, correct sum-chips

CDMA encode/decode



channel sums together transmissions by sender 1 and 2

n senders

$\lceil log_2 n \rceil + 1$ wires

using same code as sender 1, receiver recovers sender 1's original data from summed channel data!

Improving fault-tolerance: LCDMA-duplication-triplex

- LCDMA – Logic CDMA
  - several blocks send data simultaneously over a single wire – efficiently
  - limited error correcting capability
- LCDMA and hardware redundancy (duplication, triplex)
  - efficient and fault-tolerant data transmission
  - to trade off the reliability and cost of interconnect

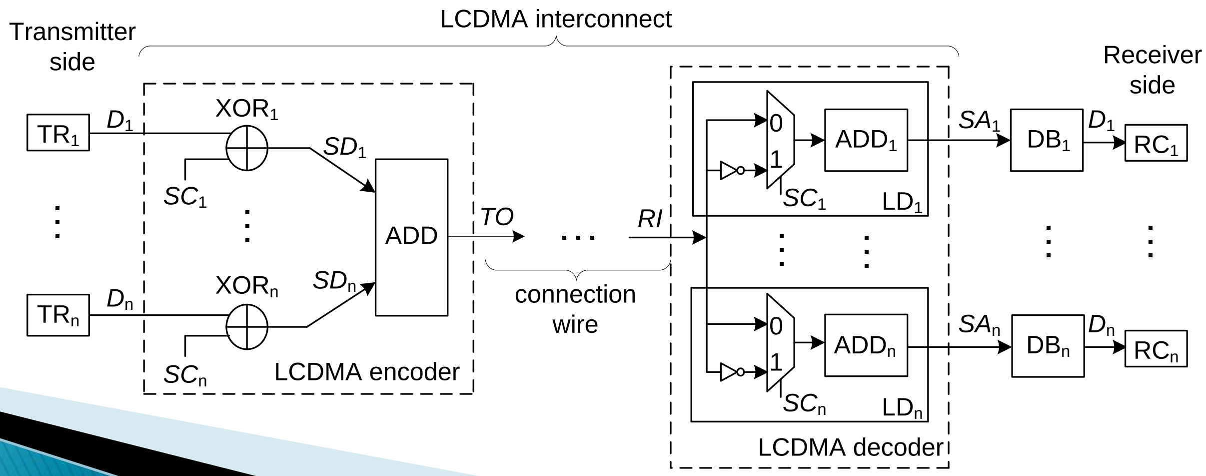
The ADD sums up chips of the same weight

TO(j) – the most significant bit of a sum for each of m generated sum-chips

$$TO(j) = MSB\left(\sum_{i=1}^{n} SD_i(j)\right)$$

Di – output bit, the sign of the value at the adder output

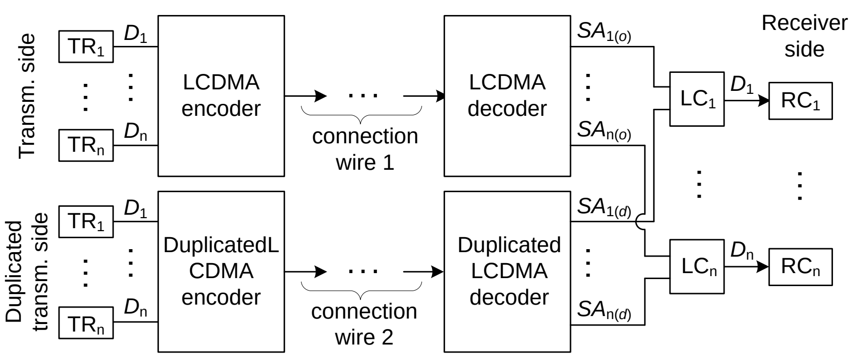
$$D_i = MSB(SA_i)$$



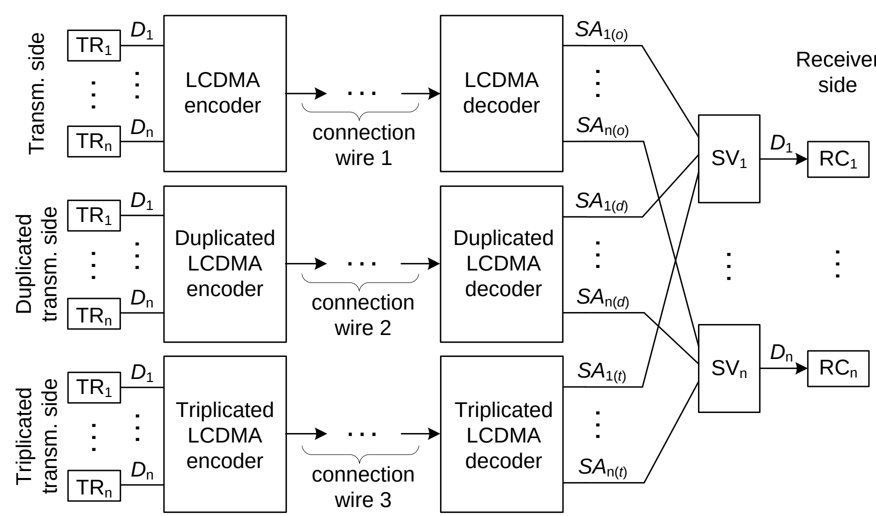
LCDMA–DLC, LCDMA–TSV

- **LCDMA–DLC** – Logic CDMA and Duplication with Logic Comparison
  - further enhances the system ability to tolerate errors
- **LCDMA–TSV** – Logic CDMA and Triplication with Sign Voter

$$D_i = MSB\left(SA_{i(o)} + SA_{i(d)}\right)$$

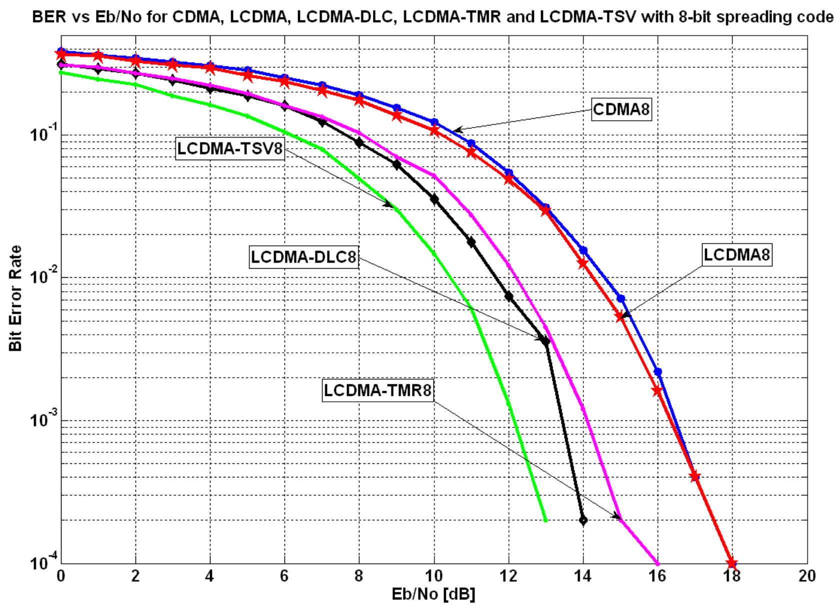


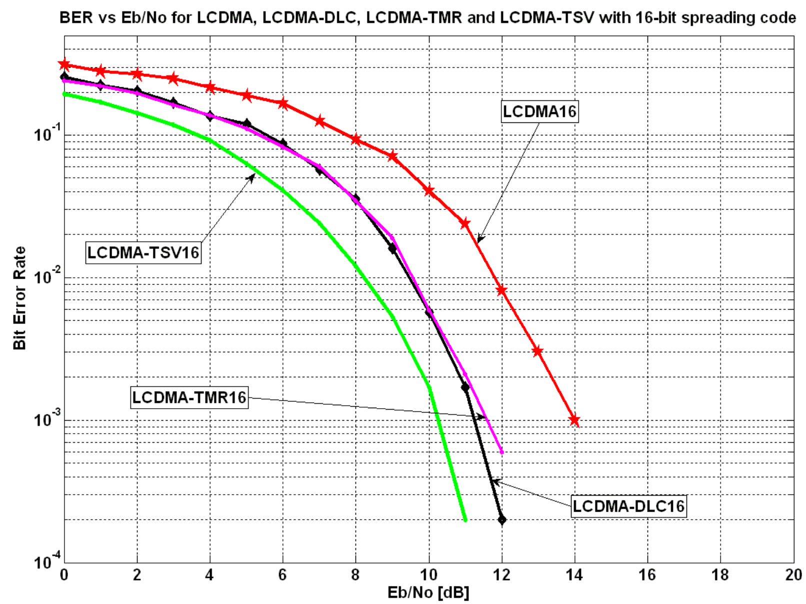
$$D_i = MSB\left(SA_{i(o)} + SA_{i(d)} + SA_{i(t)}\right)$$



BER versus SNR

- MATLAB simulation results for 8- and 16-bit spreading code lengths





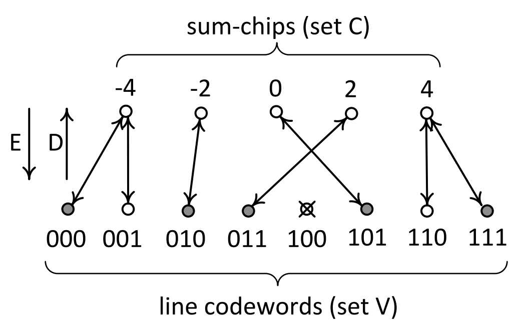
- **BER performance** – Logic CDMA and triplication with sign voter

Improving FT: encoding scheme

- The conventional **binary CDMA** bus
  - moderate fault tolerance – inadequate encoding of the sum-chips
  - signed binary numbers in two's complement representation
- **Weighted binary encoding** – not suited to CDMA
  - a bit-error at the two most significant bits
    - can cause a sign change in the sum of sum-chips
    - cannot be masked by the rest, correct sum-chips
- **Non-weighted encoding** scheme
  - inbuilt information redundancy
    - instead of hardware redundancy
  - the capability of tolerating a single-bit error
    - without extra wires

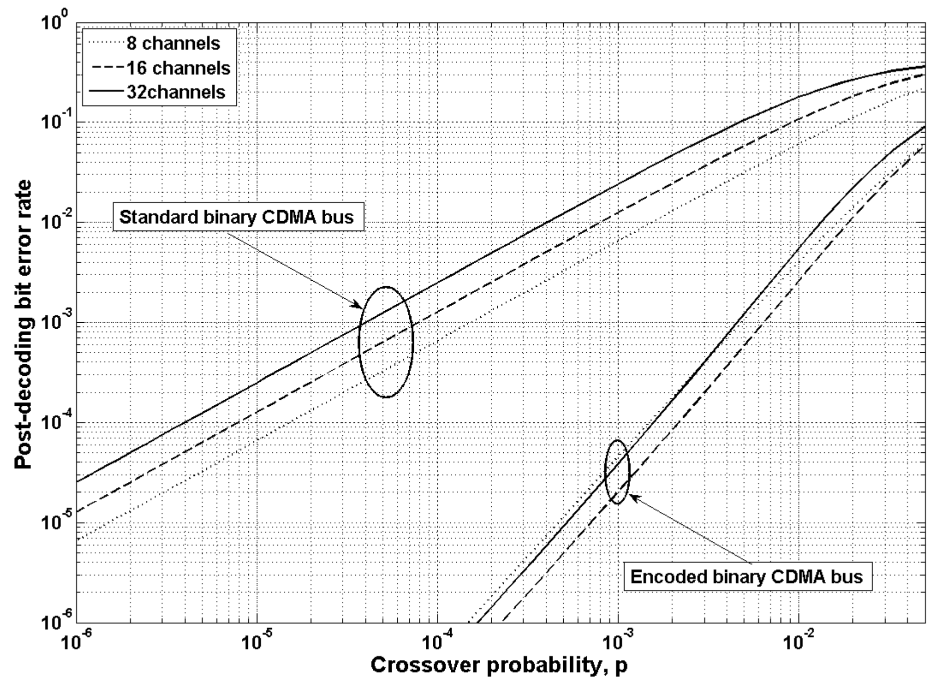
BER versus crossover probability

- Non-weighted encoding – low-cost FT scheme
  - improves bit error rate performance of the binary CDMA bus



If due to single-bit error, codeword vi is changed to vj, then the sum of the sum-chip values will have the same sign

sum-chip	LC_8 primary codeword	LC_8 secondary codeword	LC_{16} primary codeword	LC_{16} secondary codeword
16			01011	01111
14			11011	
12			00011	
10			01001	
8	0101	0111	01010	
6	1101		11010	
4	0001		00010	
2	0100		01000	
0	0000		00000	
-2	0010		00100	
-4	1000		10000	
-6	1110		11100	
-8	1010	1011	10100	
-10			00101	
-12			10001	
-14			11101	
-16			10101	10111



Conclusion

- Computers in **aerospace systems**
  - a prime example of designs that must support fault tolerance
- **Radiation**
  - prominent cause of hardware failure
- Combination: **coding** and **hardware redundancy**
  - effective fault tolerance