KUNGL TEKNISKA HÖGSKOLAN

International Master Program in System-on-Chip Design

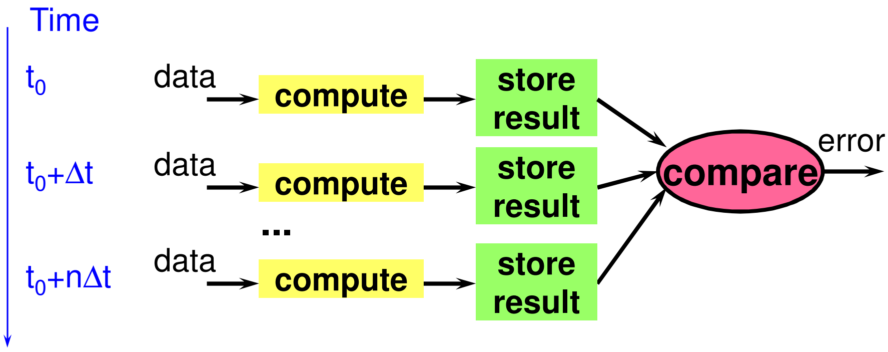
Time redundancy

Time redundancy

- Both hardware and information redundancy can require large amount of extra hardware
- time redundancy attempt to reduce the amount of extra hardware at the expense of additional time
- in many applications time is less important than hardware

Transient fault detection

- Transient faults can be detected by repeating computation several times



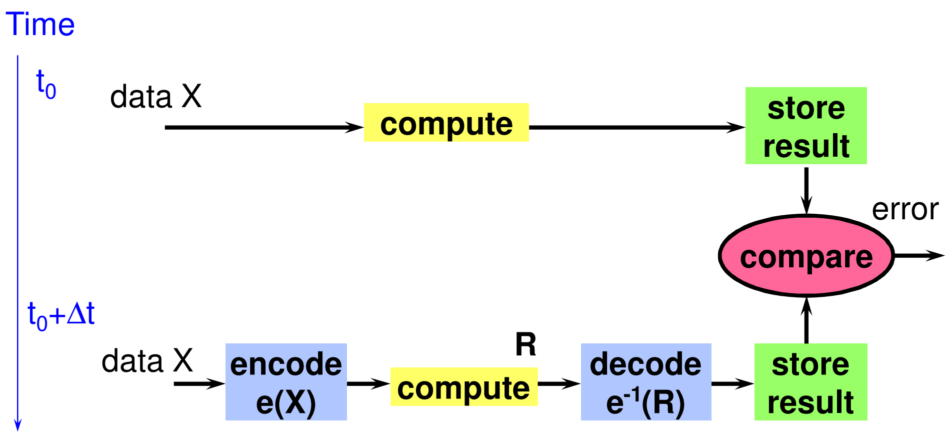
Distinguishing b/w transient and permanent faults

- Time redundancy can be used to distinguish between transient and permanent faults:
  - re-compute after the detection of the first fault
  - if fault disappears, then it was transient
  - if not, the fault is permanent, remove the faulty part of the system
- Saves resources

Permanent fault detection

- Permanent faults can be detected by repeating computation several times using different coding schemes
  - (+) minimum extra hardware is used

Permanent fault detection scheme



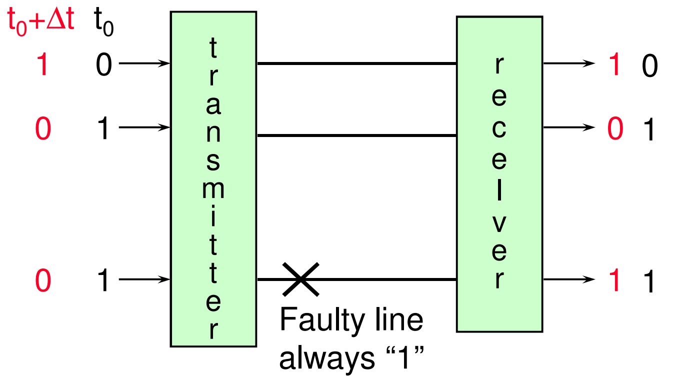
Coding schemes

- Four different approaches to coding:
  - alternating logic
  - recomputing with shifted operands
  - recomputing with swapped operands
  - recomputing with duplication and comparison

Alternating logic

- At time t_0, use the original data
- At time $t_0+\Delta t$, use the complement of the data
- 2 applications:
  - transmition of digital data over wire media
  - fault detection in digital circuits
- Suitable for stuck-at type fault detection

Alternating logic time redundancy



Faulty line causes both, data and complement, to become 1.

Duality

- Alternating logic concept can be used for detecting fault in logic circuits which implement self-dual functions
- A dual of a function f is defined as

$$f_d(x_1,x_2,\ldots,x_n) = f'(x'_1,x'_2,\ldots,x'_n)$$

- If the input $(x_1,x_2,\ldots,x_n)$ is applied to the circuit computing f and $(x'_1,x'_2,\ldots,x'_n)$ is applied to the circuit computing f_d, then the outputs will be complementary

Computing dual function (1)

- Dual of f can be obtained as follows:
  - replace AND with OR, and OR with AND
  - replace 0 with 1, and 1 with 0

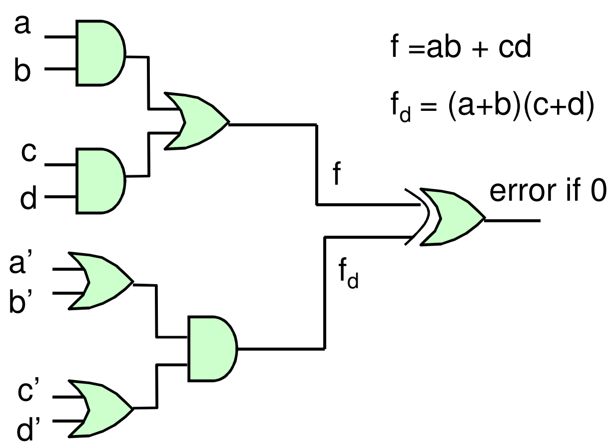
$$f = x_1x'_2 + x_3 \quad \rightarrow \quad f_d = (x_1 + x'_2)\, x_3$$

Computing dual function (2)

- We can also compute a dual of f by
  - complementing f
  - replacing each variable by its complement

$$f = x_1x'_2 + x_3$$
$$f' = (x'_1 + x_2).\, x'_3$$
$$f_d = (x_1 + x'_2).\, x_3$$

Example



$f = ab + cd$

$f_d = (a+b)(c+d)$

Self- duality

- A function is self-dual if $f_d = f$
- For example, sum and carry are self-dual functions

x_1	x_2	x_3	f_{sum}	f_{carry_out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Self- duality

- For a circuit implementing a self-dual function, if the application of an input assignment $(x_1,x_2,\ldots,x_n)$ followed by the input assignment $(x'_1,x'_2,\ldots,x'_n)$ produces output values which are equal, then the circuit has a fault

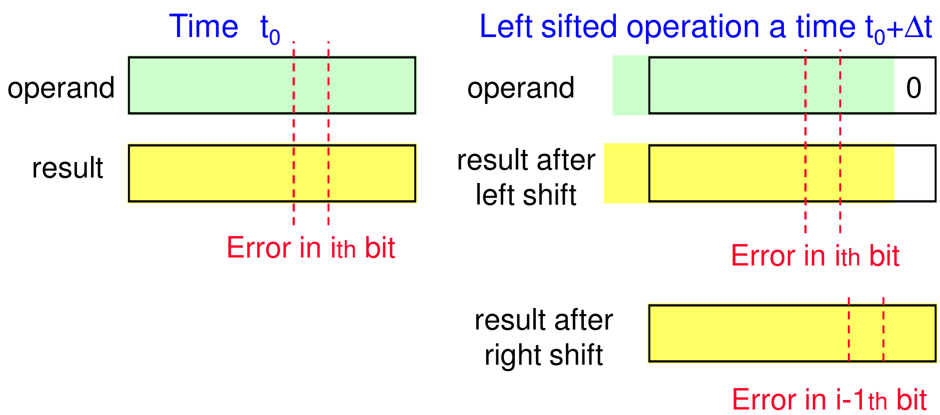
Recomputing with shifted operands

- At time t_0, use the original data
- At time $t_0+\Delta t$, encode using left shift and decode using right shift
- Suitable for fault detection in ALUs with bit-sliced organization

Recomputing with shifted operands

- Basic principle
  - during first computation, ith bit is erroneus
  - during second computation, (i-1)th bit will be affected because of the left shift
  - after the right shift, the results will disagree in both, ith and (i-1)th bits
- An extra bit is required for left shift

Recomputing with shifted operands

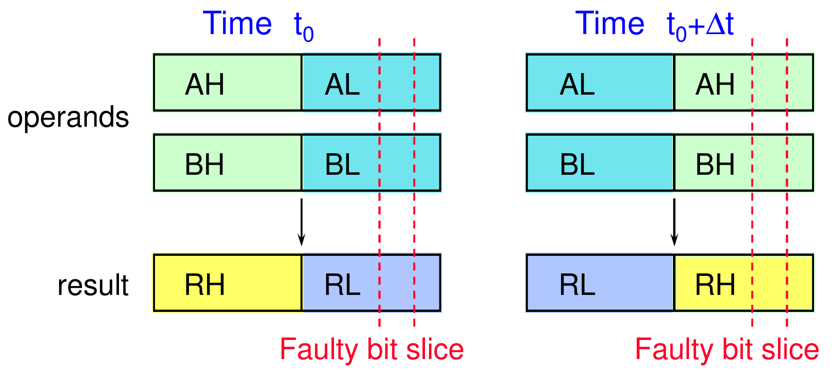


By comparing both results, the error will be detected

Recomputing with swapped operands

- At time t_0, use the original data
- At time $t_0+\Delta t$, swap the upper and lower halves of the operands
- Suitable for fault detection in ALUs with bit-sliced organization

Recomputing with swapped operands

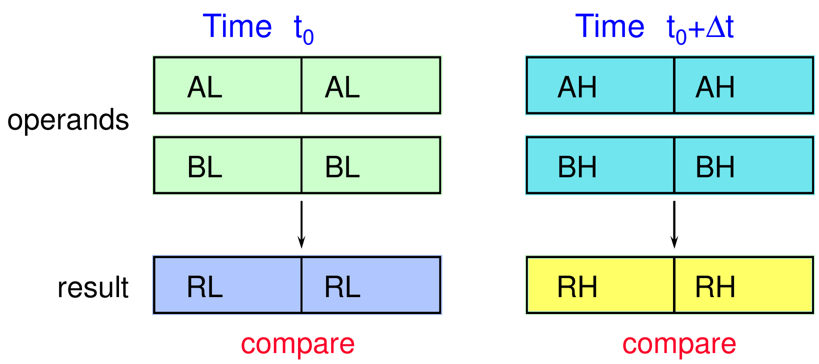


By comparing upper and lower halves of both results, the error will be detected

Recomputing with duplication and comparison

- At time t_0, perform operation on duplicated lower halves of the operands
  - compare results and, if agree, store one to represent the lower half of the final output
- At time $t_0+\Delta t$, perform operation on duplicated upper halves of the operands
  - compare results and, if agree, store one to represent the upper half of the final output

Recomputing with dublication and comparison



By comparing two halves of the result, the error will be detected

Time redundancy for error correction

- Time redundancy can provide error correction if the computations are repeated 3 or more times

Next lecture

- Software redundancy

Read chapter 7

of the text book