High Reliability Digital Systems - Hardware and Software Support

Dariusz Makowski

Agenda

- System Reliability
- How to Improve Reliability of Digital Systems
- Hardware Redundancy
- Software Implemented Hardware Fault Tolerance

How to Define Reliability

- Probability that System will operate correctly over time t and under stated conditions P[O](t),
- Reliability **R(t) = P[O](t) = 1 - Q(t),**
- A measure of the likelihood of no fault occurring,
- Every system will fail, the question is when and how frequently ?
- We can try to increase the period of time of reliable operation.

System Reliability

- Hardware Reliability,
- Software Reliability,
- Reliability of interaction between hardware and software,
- Reliability of interaction between the system and the operator.

Hardware Reliability

- Component, PCB, interconnection reliability, and failure modes,
- Hard, transient and intermittent failures,
- Random failures - exponentially distributed,

$$R(t)=\exp(-\lambda\cdot t)$$

- Wearout failures - normally distributed,

$$R_w(t)=\frac{1}{\sigma\sqrt{2\pi}}\int \exp-\frac{1}{2}\left(\frac{T-\mu}{\sigma}\right)^2$$

Measures of Hardware Reliability

- MTBF = Mean Time Between Failures,

$$MTBF = \frac{1}{\lambda} \quad => \quad \lambda = \frac{1}{MTBF}$$

- MTTR = Mean Time To Repair,
- Temperature dependency of lambda - failure rates always increase at high operating temperatures,
- Voltage dependency of lambda – failure rates always increase at higher electrical stress levels,
- High stress - high lambda !

Serial Systems

- Failure of single element takes out system,

$$R_s = \prod_{i=1}^{N} R_i = \exp\left(-\sum_{i=1}^{N} \lambda_i \cdot t\right)$$

- Combined reliability of two components in series is always lower than the reliability of individual components,
- More systems → less reliability.

Parallel Systems

- Failure of single element is survivable, but P[S] reduced,

$$R_p = 1 - Q^N$$

- Used in aircraft flight control systems,
- Space Shuttle and critical control applications.

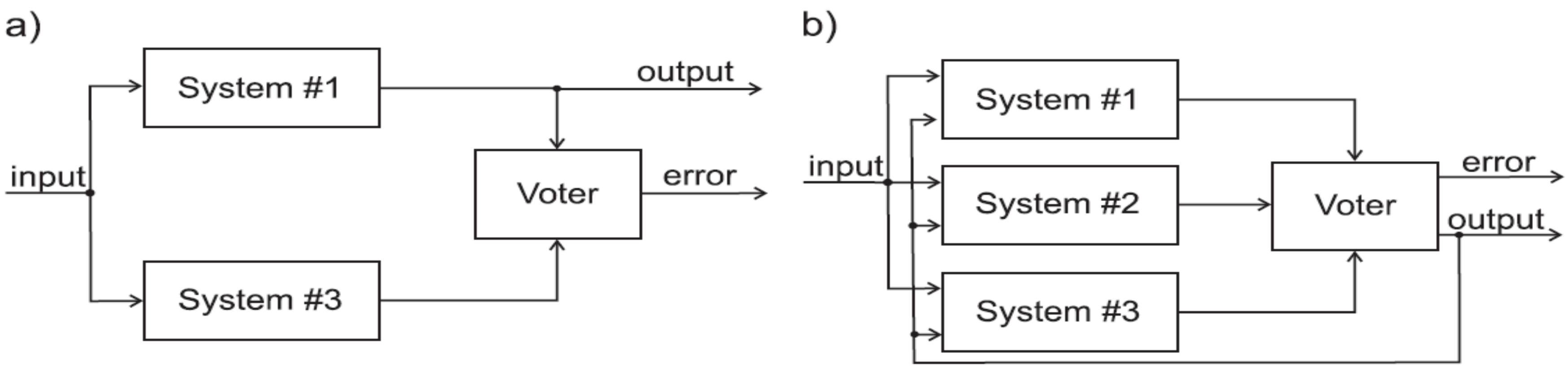
Software vs Hardware Reliability

- Hardware failures can induce software failures,
- Software failures can induce hardware failures,
- Often difficult to separate H/W and S/W failures,
- Cannot apply physical models to software failures,
- Result is system failure.

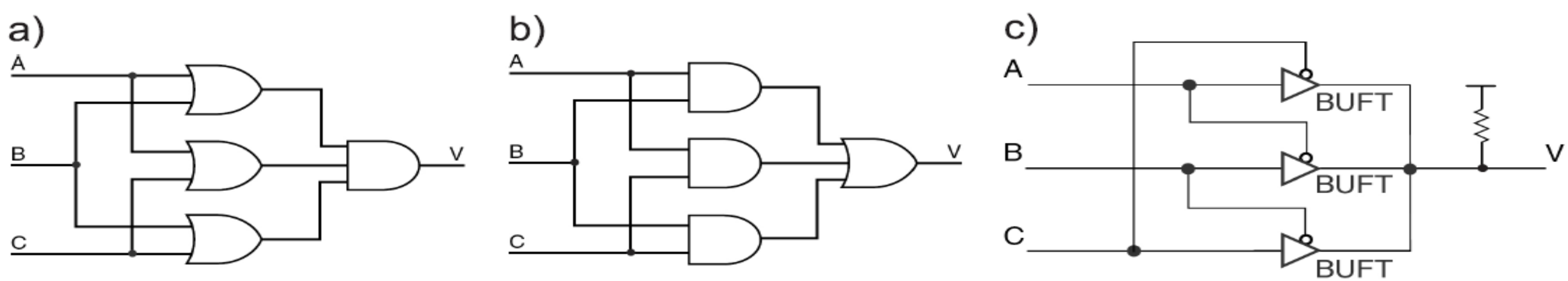
How to Improve Reliability

- Reliability of electronic systems can be enhanced by one of the following methods:
  - Hardware redundancy (DMR or TMR),
  - Time redundancy,
  - Error Detecting and Correcting (EDAC) methods,
  - Software Implemented Hardware Fault Tolerance (SIHFT),
- Redundancy and EDAC methods require additional hardware and do not guarantee error-free operation.

Hardware Redundancy

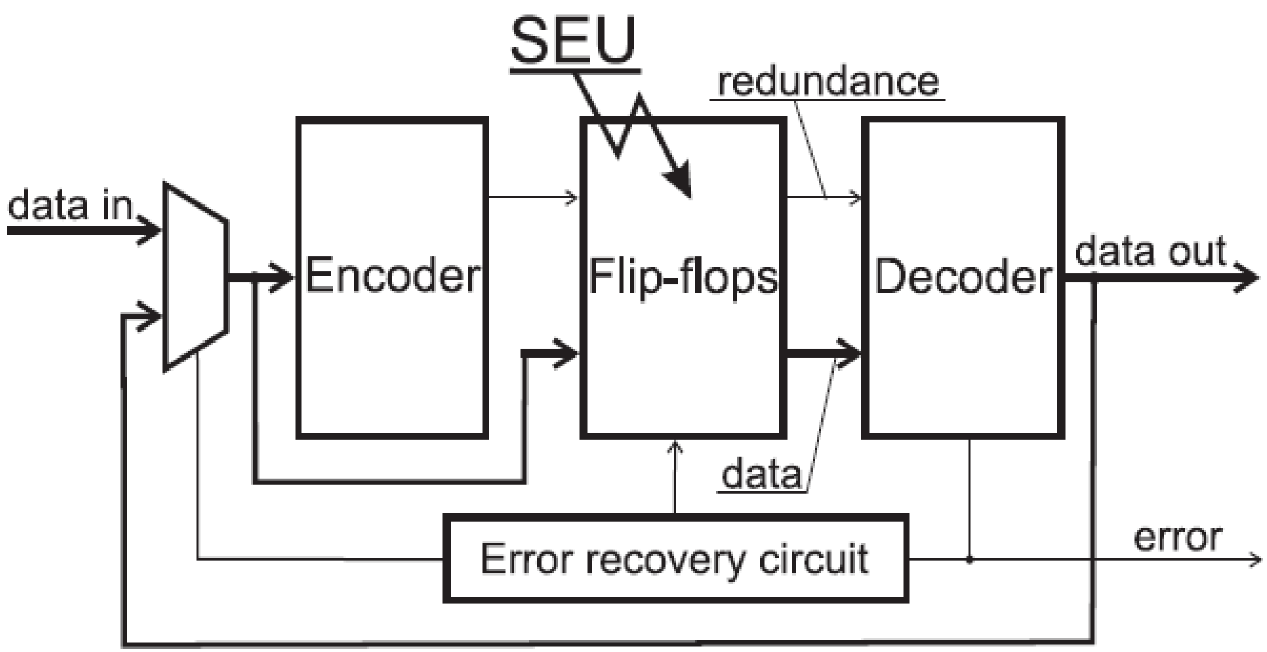


The comparison between a) dual and b) triple modular redundancy.



Exemplary voters: a) OR-NAND, b) NAND-OR, c) tristate voter of Xilinx Virtex FPGA.

Error Detecting and Correcting



The error detection and correction technique.

Software Implemented Hardware Fault Tolerance

SIHFT Features

- Pure software solution based on source code modification,
- Can be implemented at high-level or assembler-level programming language,
- Divided into two categories:
    - Data hardening algorithms,
    - Control flow hardening algorithms.

SIHFT – Pros and Cons

- Do not require any hardware modification,
- Can be used with unhardened Commercial-Of-The-Shelf equipment,
- Results in increase of program size and decrease program efficiency,
- Do not guarantee 100 % effectiveness.

Hardening Data

- Based on operation and data redundancy
- Can be implemented on several levels of granularity:
    - Instruction-level duplication,
    - Procedure-level duplication,
    - Program-level duplication.

Instruction-level Duplication

•

Original source code	*Modified source code*

res = search(a);
...
int search (int p)
{
    int q
    ...
    $q = p + 1$
    ...
    return (1)
}
	search(a_0, a_1, $\&res_0$, $\&res_1$);
	...
	void search (int p_0, int p_1, int $*r_0$, int $*r_0$)
	{
	    int q_0, q_1;
	    ...
	    $q_0 = p_0 + 1$
	    $q_1 = p_1 + 1$
	    if (p_0 != p_1)
	        _error();
	    ...
	    $*r_0 = 1$;
	    $*r_1 = 1$;
	    return;
	}

- Every variable in program must be duplicated,
- Every write operation has to be performed on both copies of variables,
- After each read operation on variable, checking for consistency has to be done. In case of inconsistency, appropriate error recovery procedure has to be called
- Redundancy in returned value

Procedure-level Duplication

Original source code	*Modified source code*

void A()
{
$a = B(b);$
$c = c + a;$
}
 	
int B(int b)
{
$d = 2 * b;$
return d;
}
	void A()
	{
	$a_1 = B(b_1);$
	$a_2 = B(b_2);$
	if $(a_1 \mathrel{!=} a_2)$ _error();
	$c_1 = c_1 + a_1;$
	$c_2 = c_2 + a_2;$
	if $(c_1 \mathrel{!=} c_2)$ _error();
	}
	
	int B()
	{
	$d = 2{*}b;$
	return d;
	}

- Every procedure should have duplicated instructions or should be executed twice,
- Procedure with non-duplicated statements should not call procedures with duplicated statements

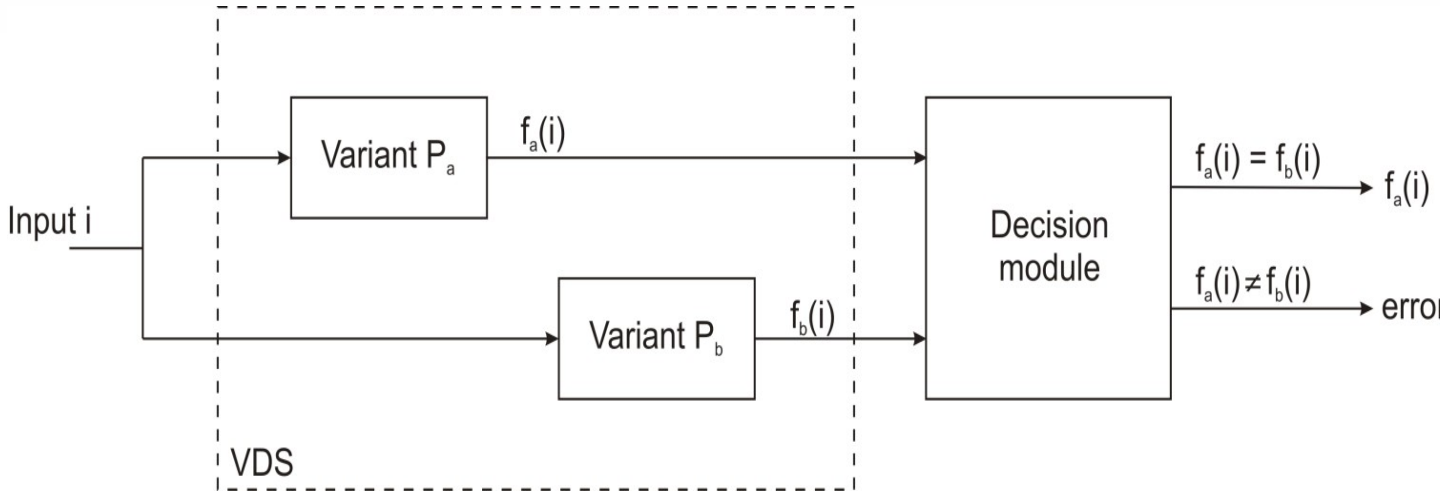
Achieving Fault Tolerance-Duplication and Checksum

Original source code	*Modified source code*
int a, b;	int a_0, a_1, b_0, b_1; int crc_0, crc_tmp_0;
...	...

$a = b + 7;$	crc_0 ^= a_0;
	 $a_0 = b_0 + 7;$
	 $a_1 = b_1 + 7;$
	 crc_0 ^= a_0;
	 if (b_0 != b_1)
	     $crc\_tmp_0 = a_0;$
	     crc_tmp_0 ^= b_0;
	     if ($crc\_tmp_0 == crc_0$)
	         $a_1 = a_0$; $b_1 = b_0$;
	     else
	         $a_0 = a_1$; $b_0 = b_1$;
	         $crc_0 = a_0$; crc_0 ^= b_0;

- Instruction-level duplication is able to detect faults but cannot correct errors,
- Calculation of checksums can be used to recover correct value of variables.

Program-level Duplication



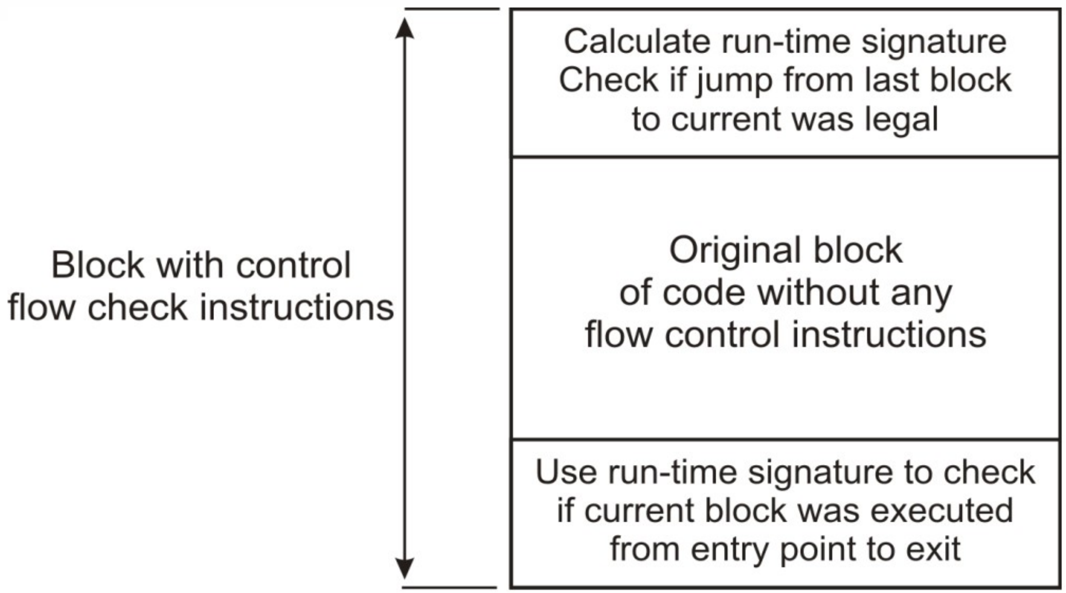
- Program-level duplication uses time redundancy to execute the same program multiple time,
- Transient errors are covered due to time redundancy, permanent errors can be covered by design diversity method,
- Single microprocessor systems can execute the same program multiple times (serial execution or simultaneously with task switching). Multiprocessor or multi-core systems can perform operation in parallel.

Algorithm-based Fault Tolerance

$$
\begin{array}{|ccc|}\hline 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ \hline 12 & 15 & 18 \\ \hline \end{array}
*
\begin{array}{|ccc|c|}\hline 1 & 2 & 3 & 6 \\ 4 & 5 & 6 & 15 \\ 7 & 8 & 9 & 24 \\ \hline \end{array}
=
\begin{array}{|ccc|c|}\hline 30 & 36 & 42 & 108 \\ 66 & 81 & 96 & 243 \\ 102 & 126 & 150 & 378 \\ \hline 198 & 243 & 288 & 729 \\ \hline \end{array}
$$

- More specialized approach than instruction-level duplication,
- Smaller overhead but introduce significant complication of algorithm development,
- Example:
  - Matrix operations like multiplication, LU decomposition, transposition,
  - FFT calculation.

Hardening of Control Flow



- A Control Flow Error is an error that causes a processor to fetch and execute an instruction different then expected,
- Control Flow Checking (CFC) methods compare reference signatures with run-time signatures to check flow correctness,
- Free program memory should be filled with escape instruction to safe procedure, i.e. jump to error handler or software interrupt.

Software Fault Tolerance on Compiler Level

- Based on GCC compiler modification,
  - Implementation independent on hardware architecture,
- Automatic implementation of composite data type protection algorithm,
  - Checksum based protection of variable,
  - Protection of local variables in functions based,
  - Source code optimization adapted to local variable protection algorithm.

Conclusions

- Application of hardware or software methods allows to improve systems reliability,
- This methods requires application of additional resources (hardware, processing power, etc...),
- Complexity introduces unexpected interactions,
- Methods inadequately used can decrease the whole system reliability or even result in unstable operation.

Thank you for your attention

ABTF Memory overhead