Design Validation & Reliability Verification of HW-SW Systems for Safety-Critical Applications*

Fabian Vargas

Catholic University - PUCRS
Electrical Engineering Dept.
Av. Ipiranga, 6681. 90619-900 Porto Alegre, Brazil
vargas@computer.org

* *This work has been supported in part by CNPq and FAPERGS.*

Summary

- 1. Motivation: *an overview on previous works*
- 2. The Proposed Methodology
  - 2.1. System Description Validation
  - 2.2. System Reliability Insertion & Verification
- 3. Case Studies
- 4. Conclusions

Motivation of the Work

Previous works suggest how to design systems towards:

monetary cost, performance, communication rates, power consumption, silicon area, testability, memory size .

None of the above strategies suggest how to *validate the design* & *verify the reliability* of a system in a *HW/SW common basis*.

For embedded systems which are *safety-critical* and particularly *complex to design*, validate the design and integrate *reliability constraints* during *HW-SW partitioning* may have very good returns.

Motivation of the Work

In an attempt to improve this point, we have been motivated to develop a novel approach well suited to design (at a high-level description) *embedded safety-critical computing systems*

- ✓ First, we *validate the C++/VHDL description* of the system.
- ✓ Next, we *insert reliability functions* into the HW and SW parts of the system.
- ✓ Then, we *verify* the system *reliability* against *transient or permanent faults*.

2. The Proposed Methodology

System Description Validation

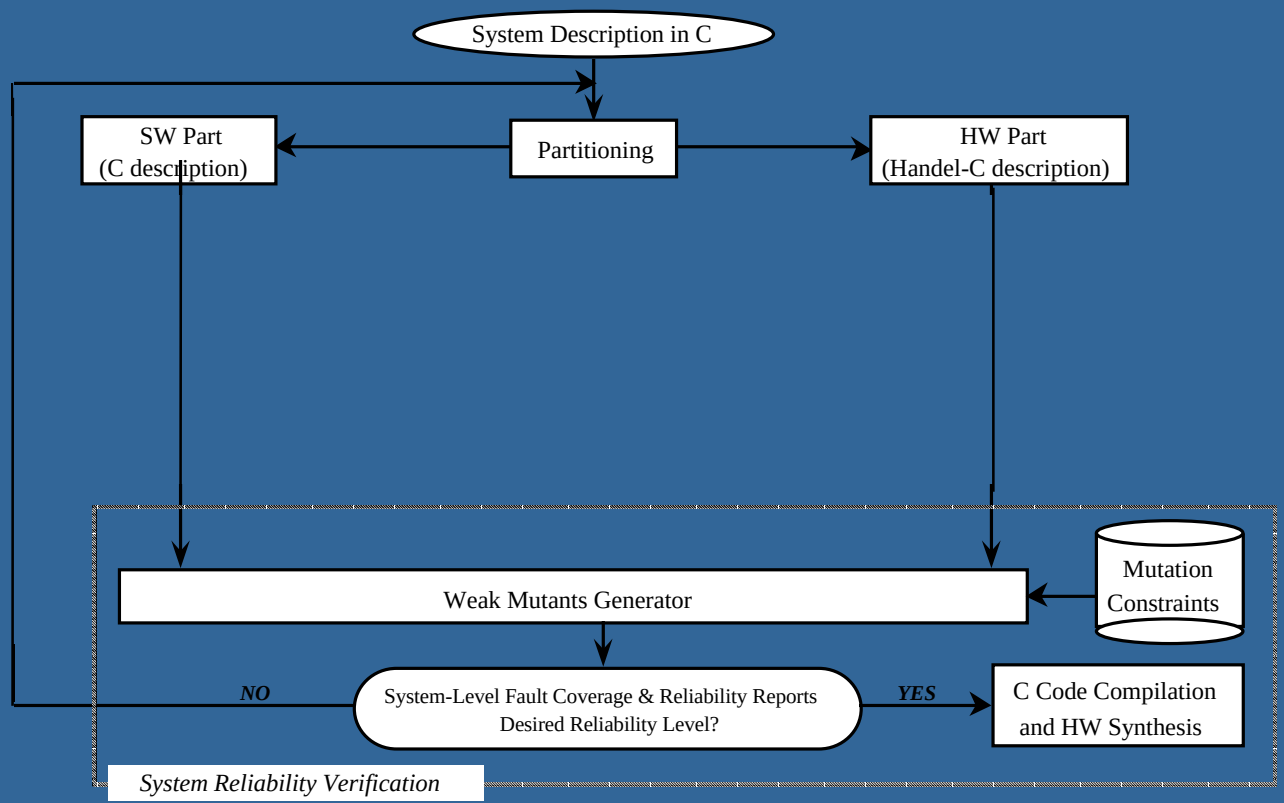

Fig. 1a. Design Flow: *system description validation*.

2. The Proposed Methodology

System Reliability Insertion & Validation

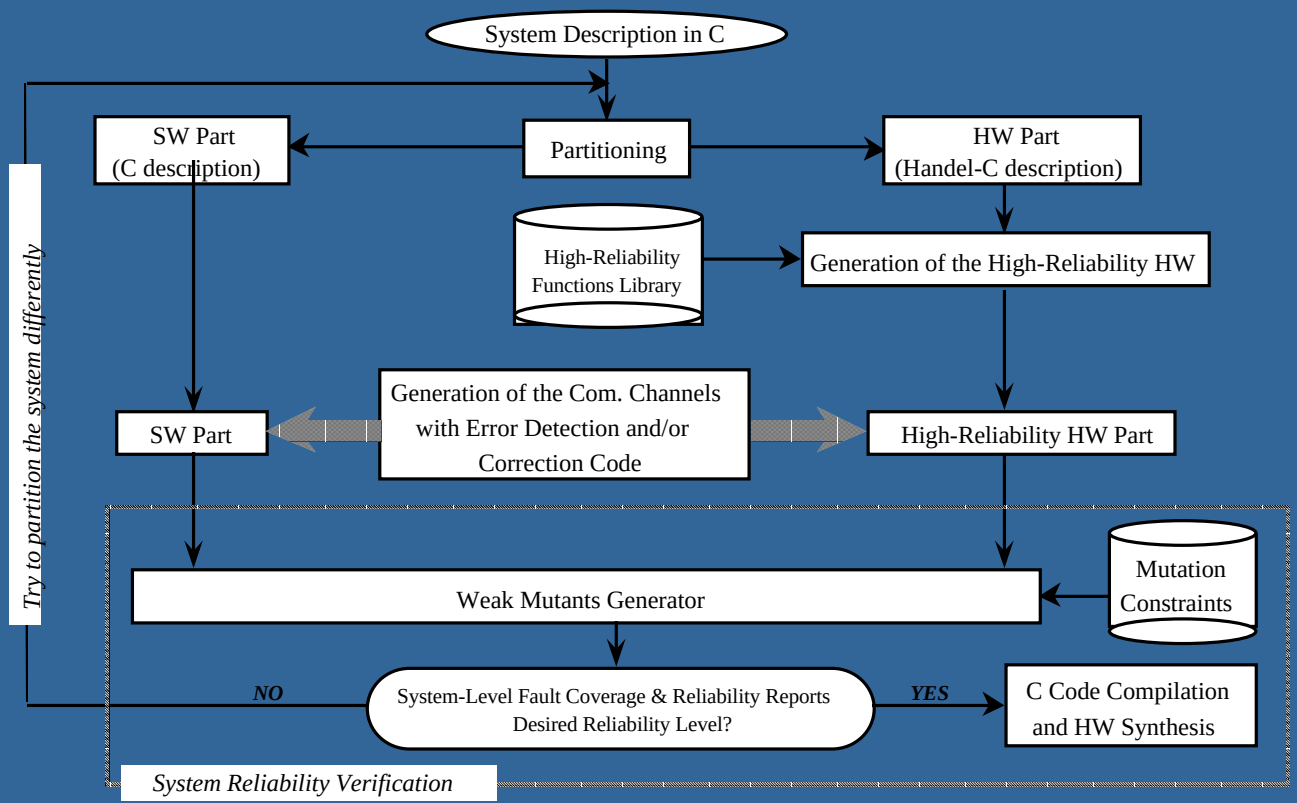

Fig. 1b. Design Flow: *system reliability insertion & verification*.

2. The Proposed Methodology

✓Embedded Reliability Functions:

✓HW:

- Built-In Self Test (*BIST*)
- Triplication with Voter (*Fault Masking*)
- Duplication with Comparator (*Performance Degradation*)
- Error Detecting & Correcting Codes

2. The Proposed Methodology

✓ Embedded Reliability Functions:

✓ SW:

- Robustness
- Acceptance Tests (*Specification & Placement*)
- Capability Check (*Checks for System Capabilities at a Given Time*)
- Recovery Blocks (*Primary & Alternate Programs*)
- Stress Testing (*Abnormal Situations*)
- Performance Testing (*Real-Time Applications*)

2. The Proposed Methodology

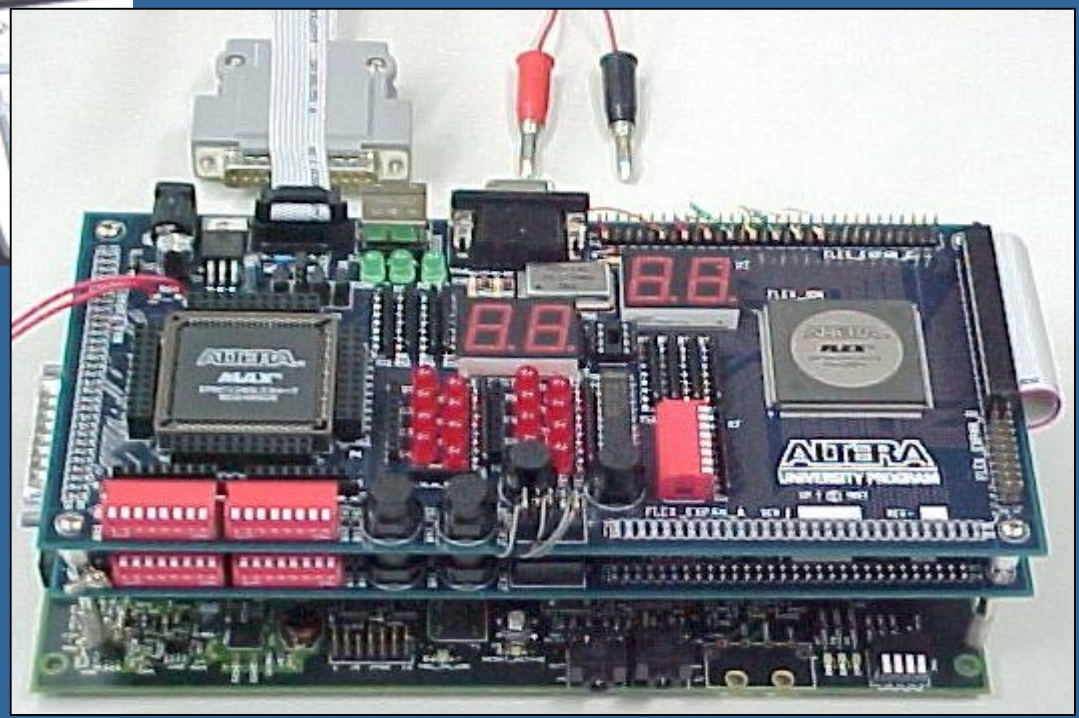

Photo 1. Previous Prototyping Environments.

2. The Proposed Methodology

- ***The idea***: **validate the description and estimate the reliability of the system (at a high-level description) against transient or permanent faults.**

- ***The solution***: **adaptation of the Mutation Analysis Approach, originally proposed for software testing in 1978 by DeMillo. (Goal development of a criterion for the selection of test vectors).**

- ***The proof***: **we need first to show that the stuck-at fault coverage at the gate level is equal to or greater than the one obtained by means of mutation analysis in a VHDL HW description, at the system level.**

2. The Proposed Methodology

The implementation: fault injection by means of generating small syntactic changes in the original code and determination of which test vectors were able to detect the mutated versions of the code.

The measurement of the fault-coverage:

If a program has *M* mutants, *E* of which are equivalent, and a test set *T* kills *K* mutants, the mutation score is defined to be:

$$MS(P,T) = \frac{K}{(M - E)}.$$

2. The Proposed Methodology

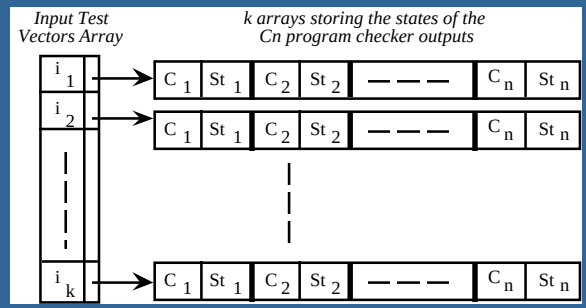

Fig. 2. Mutant Data Structure (MDS) for the weak mutation generation procedure.

The MDS for a given program consists of two parts: an ***array I*** representing the program input test vectors, in which each element points to an ***array C*** containing the name and state of all comparators outputs in the program during program execution.

2. The Proposed Methodology

If any of the comparators outputs is set to "1", then the injected fault (i.e., the mutated statement in the code) is detected, otherwise the fault can be classified as redundant or undetectable by the additional *HW blocks/SW routines* (thus, lowering system testability).

2. The Proposed Methodology

Type	Description
AOR	Arithmetic Operator Replacement
ABS	Absolute Value Insertion
CR	Constant Replacement
CVR	Constant for Variable Replacement
LOR	Logical Operator Replacement
ROR	Relational Operator Replacement
ODR	Operation for Delay Replacement
OSR	Operation for Skip Replacement
VCR	Variable for Constant Replacement
VR	Variable Replacement
UOI	Unary Operator Insertion
BOR	Bit Operator Replacement

Table 1. Mutation operators set for VHDL/C++ functional descriptions.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all
entity CRYPT is
port ( entrada_info : in integer range 0 to 3;  entrada_chave : in integer range 0
to 100;  saida : out integer range 0 to 100 );
end CRYPT;

Architecture ARCH_NAME of CRYPT is
begin
process(entrada_info, entrada_chave, saida)
variable temp1 : integer range 0 to 6;
variable temp2 : integer range 0 to 18;
variable temp3 : integer range 0 to 18;
variable temp4 : integer range 0 to 118;
constant sum_const : integer := 3;
constant mul_const : integer := 2;
constant sub_const : integer := 1;
        begin
        temp1 := entrada_info + sum_const;
        Δ temp1 := entrada_info - sum_const;   -- Mutant 1: AOR
        Δ temp1 := entrada_info + temp2;       -- Mutant 2: CVR
        temp2 := temp1 * mul_const;
        Δ temp2 := temp1 * sum_const;          -- Mutant 3: CR
        Δ delay;                               -- Mutant 4: ODR
        temp3 := temp2 - sub_const;
        Δ temp3 := sum_const - sub_const;      -- Mutant 5: VCR
        Δ temp3 := temp3 - sub_const;          -- Mutant 6: VR
        temp4 := entrada_chave + temp3;
        Δ skip;                                -- Mutant 7: OSR
        saida <= temp4;
        end process;
end ARCH_NAME;
```

Fig. 3. Example of fault injection in a VHDL description. (The symbol Δ identifies mutated statements.)

3. Case Studies

This section illustrates of the System Reliability Verification Procedure. Hereafter we compare the stuck-at fault coverage with the one obtained by means of mutation analysis in a VHDL HW description level.

Circuit	Number of Gates	Number of test vectors generated	Number of detectable stuck-at faults	Number of stuck-at faults detected
Multiplier 2x2	19	9	116	100 %
Multiplier 4x3	110	15	622	100 %
Multiplier 6x6	431	30	2420	99.50 %
Multiplier 8x4	353	23	1996	99.30 %
Multiplier 8x6	565	25	3211	99.78 %
Multiplier 8x8	809	36	4548	99.74 %

Table 2. Stuck-at fault testing summary for the 6 Multiplier Circuit operand widths.

Circuit	Number of Gates	Number of test vectors generated	Number of generated mutants	Number of mutants killed
Multiplier 2x2	19	9	22	95.45 %
Multiplier 4x3	110	15	106	94.34 %
Multiplier 6x6	431	30	432	88.19 %
Multiplier 8x4	353	23	364	88.88 %
Multiplier 8x6	565	25	600	88.67 %
Multiplier 8x8	809	36	832	88.34 %

Table 3. Mutation analysis summary for the 6 Multiplier Circuit operand widths.

3. Case Studies

Circuit	Number of Gates	Number of test vectors generated	Number of detectable stuck-at faults	Number of stuck-at faults detected
ALU - 4 bit	71	18	452	99.55 %
ALU - 8 bits	155	22	980	98.98 %
ALU - 12 bits	239	21	1508	98.80 %
ALU - 16 bits	323	21	1908	98.53 %

Table 4. Stuck-at fault testing summary for the 4 ALU operand widths.

Circuit	Number of Gates	Number of test vectors generated	Number of generated mutants	Number of mutants killed
ALU - 4 bit	71	18	92	94.56 %
ALU - 8 bits	155	22	204	92.15 %
ALU - 12 bits	239	21	316	92.40 %
ALU - 16 bits	323	21	428	87.38 %

Table 5. Mutation analysis summary for the 4 ALU operand widths.

3. Case Studies

Adder Arcuitecture (4 bits)	Number of Gates	Number of test vectors generated	Number of detectable stuck-at faults	Number of stuck-at faults detected
Simple Adder	49	10	296	100 %
Manchester	76	12	343	99.56 %
Carry Lookahead	64	10	372	99.73 %

Table 6. Stuck-at fault testing summary for the 3 Adder Circuit architectures.

Adder Arcuitecture (4 bits)	Number of Gates	Number of test vectors generated	Number of generated mutants	Number of mutants killed
Simple Adder	49	10	46	86.95 %
Manchester	76	12	57	83.37 %
Carry Lookahead	64	10	62	82.26 %

Table 7. Weak mutation analysis summary for the 3 Adder Circuit architectures.

4. Conclusions

- We have presented: new approach to design fault-tolerant systems & verify their reliability.
- The approach is based on the Weak Mutation Analysis Technique.
- *Case studies:* the fault coverage obtained based on the mutation analysis of the VHDL description is equal to or lower than the one obtained based on the stuck-at fault model, at the gate level structure.
- These results show that the weak mutation analysis provides a conservative verification of fault coverage (fault-tolerance) when compared to the one at the gate level structure.

4. Conclusions

This condition allows us to use the proposed technique to design & verify the reliability level of critical-application systems specified in VHDL/C++ descriptions.

SiSC

Systems, Signals & Computing Lab

www.epo.pucrs.br/~sisc