# Are You Safe Yet? Safety Mechanism Insertion and Validation


## Ping Yeung, Jin Hou, Vinayak Desai, Jacob Wiltgen

Mentor, A Siemens Business

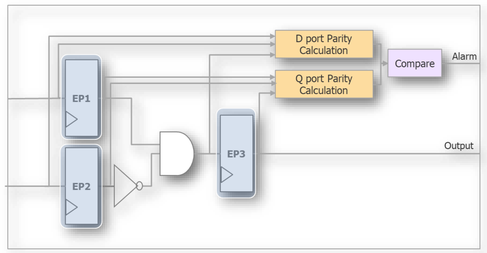

## Abstract


As functional safety becomes increasingly important in today's industrial and automotive designs, many legacy designs have to be "upgraded" to meet the safety goal of the system. An efficient approach is to use safety synthesis and formal verification to incorporate a safety architecture into the design. The flow can consist of these major steps: 1) explore areas of the design where better fault detections are required, 2) introduce the right safety mechanisms into the design with safety synthesis, 3) validate the design changes with formal verification, and 4) perform formal fault injection to measure the diagnostic coverage.


## Safety Mechanisms Insertion

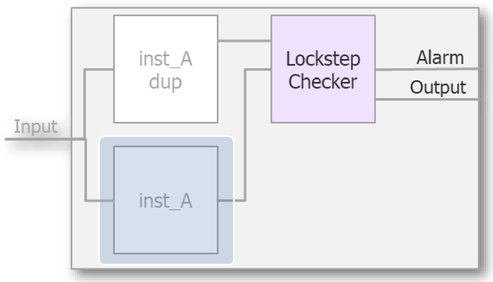
Register-level safety mechanisms include:

- Parity generation and checking for critical control elements.
- Double modular redundancy for a selected list of registers.
- Triple modular redundancy for a selected list of registers.
- Error correction, and single-error correction with double-error detection for banks of registers.
- Protocol checking ensures valid state transitions for finite state machines

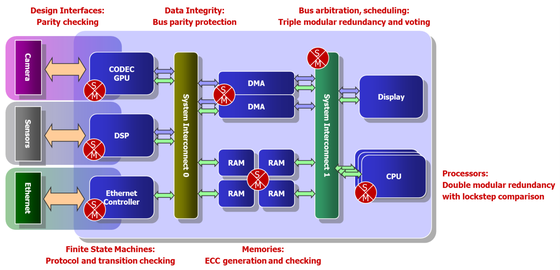



Module-level safety mechanisms include:

- Double modular redundancy along with lockstep checker.
- Triple modular redundancy along with lockstep checker and majority voting
- Input and output parity checking on groups of interface signals
- Memory parity generation and checking

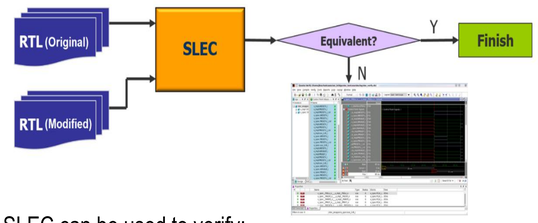



## Safety Mechanisms for a Safety-Critical Design




A high-level architecture is shown. For the design interfaces, parity checks can be performed to ensure accurate data transmission between the interface modules and the interface controllers inside the design. Once the data are inside the design, they can be protected with data parity on the buses and error-correction code (ECC) in storage elements. Critical control components such as FSMs and arbitration logic will best be protected with triple module redundancy and majority voting. Central and embedded processors can be protected with double modular redundancy along with lockstep checkers.

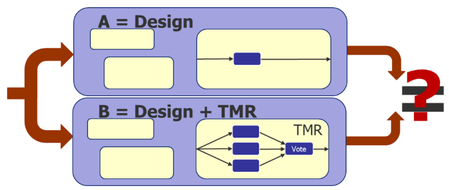
## Sequential Logic Equivalence Checking (SLEC)




SLEC can be used to verify:

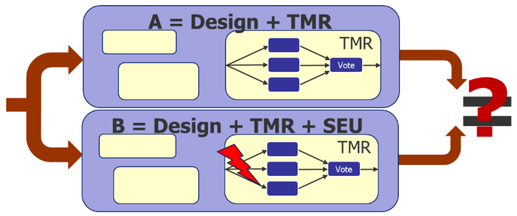
- Safety Mechanism Insertion, ensuring that the functionality of the original design is not changed by the addition of the safety mechanisms (SMs)
- Safety Mechanism Operation, ensuring that the inserted functional safety mechanisms are working as designed.

## Safety Mechanism Insertion Verification



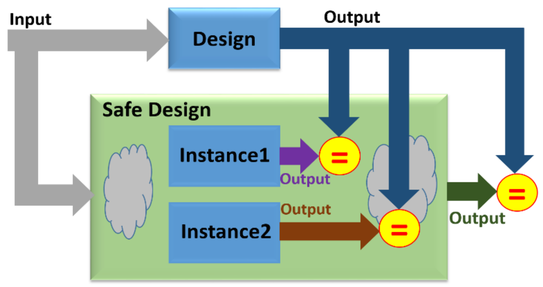

A triple modular redundancy (TMR) is used as the safety mechanism to protect the design. By comparing Design A (without safety mechanism) and Design B (with TMR), SLEC can mathematically prove that the TMR has been correctly inserted into the design.

## Safety Mechanism Operation Verification




A golden (no-fault) model and a fault injected model are used to perform on-the-fly fault injection and result analysis. By instantiating a design with a copy of itself, all legal input values are automatically specified for SLEC, just as a golden reference model in simulation predicts all expected outputs for any input stimulus. By comparing a fault injected design with a copy of itself without faults, the formal tool checks if there is any possible way for the fault to either escape to the outputs or go undetected by the safety mechanism.

## Implementation Environment




The tool, Austemper Annealer [7], was used to perform safety synthesis by duplicating part of the design for double modular redundancy. The figure above shows the "safe" design and the setup of verifying the double modular redundancy safety mechanism using SLEC.

The tool, Questa SLEC [8], was used for SLEC verification. We not only compare the outputs of the original design and the "safe" design but also compare the outputs of the original design and the outputs of the two instances in the "safe' design to make sure that the two instances behavior the same as the original design.

```
run: compile_designs run_slec

compile_designs:
        vlib work_spec
        vlog -f filelist_design -work work_spec
        vlib work_impl
        vlog -f filelist_safedesign -work work_impl

run_slec:
        qverify -c -od log -do " \
        slec configure -spec -d design_top -work work_spec;\
        slec configure -impl -d safedesign_top -work work_impl;\
        slec map -instance {spec impl.instance1} -target -output; \
        slec map -instance {spec impl.instance2} -target -output; \
        slec compile; \
        slec verify; \
        exit"
```

Compile the original design

Compile the safe design

SLEC configuration

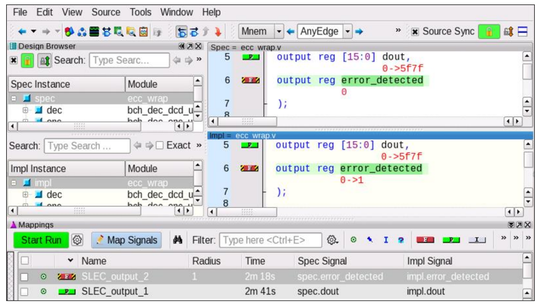
SLEC compile and run

Map the outputs of "design_top" and Instance1/Instance2 as target respectively

## Results

| | Safety Mechanism | SLEC result | | |
|---|---|---|---|---|
| | | Proven | Fired | CPU time |
| AMBA Block A | module duplication | 24 | 4 | 1s |
| AMBA Block B | register duplication | 19 | 0 | 1s |
| AMBA Block C | ECC logic | 16 | 0 | 36s |
| OpenRISC subsystem | module duplication | 42 | 0 | 1s |
| Ethmac design (design bug) | module duplication and register duplication | 31 | 23 | 2s |
| Ethmac design (bug fixed) | after bug fixed in safety mechanism | 54 | 0 | 2s |

This table summarized the many design blocks that have been "upgraded" with different safety mechanisms. For AMBA-based design block A and block B with duplication insertions, we have verified the equivalence between the outputs of the original block and the modified (original+safety mechanism) block. For block C with ECC insertion, we have verified the equivalency between the outputs of the original design and the modified design with ECC insertion.




"spec" of the original design while "impl" is the "upgraded" design with safety mechanisms. Even though a fault had been injected into the design, the output of the design was still correct (impl dout the same as the spec dout). The ECC safety mechanism had recovered the data from the fault correctly. The error detection signal, error_detected, was asserted to alert the user of this situation. One the other hand, if an injected fault had caused a failure at the comparison point, a waveform of the counter-example that captures the fault injection and propagation sequence is generated for debugging.

## References


[1] ISO 26262-5:2011 *Road vehicles Functional safety, Part 5: Product development at the hardware level*, https://www.iso.org/standard/51360.html

[2] Andrew Hopkins, *Silicon evolution for the automotive revolution.* ARM White Paper, 2019.

[3] Jacob Wiltgen, *Reducing Your Fault Campaign Workload Through Effective Safety Analysis*, Semi Engineering, Aug 2019.

[4] Avidan Efody, *Whose Fault Is It? Advanced Techniques for Optimizing ISO 26262 Fault Analysis.* DVCon 2016.

[5] Ping Yeung, et al., *Whose Fault Is It Formally? Formal Techniques for Optimizing ISO 26262 Fault Analysis.* DVCon 2018.

[6] Doug Smith, *It's Not My Fault! How to Run a Better Fault Campaign Using Formal.* Verification Academy, Jun 2018.

[7] Austemper Annealer User Guide, Mentor, A Siemens Business, 2019.

[8] Questa SLEC User Guide, Mentor, A Siemens Business, 2019.



TEMPLATE DESIGN © 2008
www.PosterPresentations.com