Model-Based Testing and Formal Verification in IEC 61508-3 ed2.0

Mika Katara
Tampere University of Technology
Department of Software Systems
Outline

Motivation
IEC 61508: Verification & Validation
How to get started
Case
Lessons learned
Conclusions
Q&A
Motivation

Formal methods are making their way into the development of safety-critical systems.

Companies developing safety-critical systems must balance between safety requirements imposed by standards and productivity requirements.

On the one hand, the higher the safety integrity requirements (SIL/PL level), the more time and effort are needed for validation and verification activities.

On the other hand, companies producing less safety-critical systems often face fierce competition and are required to put more emphasis on the overall efficiency of the development process.
The machine manufacturers can use self-declaration in the case of most type of machines; for EU markets, the manufacturer can label the product with the “CE” marking without formal type examination.

However, certification by an independent assessment organization may still be required by customers and/or for marketing reasons. It is also seen as an important step if an accident should occur and investigation of the development practices takes place.

Profit margins are better on the higher SIL/PL levels where less competition exists (for the moment).
New version of IEC-61508, an important basic standard on functional safety
ed1.0 → ed2.0 (April 2010)
Not harmonized, but referenced by harmonized standards such as EN ISO 13849-1 and EN 62061 in the machinery sector
The new version contains a lot of information that was not present in the old version
Understanding the practical implications of the changes is difficult
It seems that there are now more choices available
It may not be so easy to use the standard as a “check-list”
Due to its size, the new version is far from trivial to apply and to check conformance with
For instance, there is still no requirement for “safety case”
The difficulty increases steeply with the SIL level (1 → 4)
For each SIL level, there is a set of Highly Recommended, Recommended and Not Recommended methods.

In addition, for the use of some methods the standard does not indicate any recommendation on certain SIL levels.

Model-based testing is a relatively new technology that is now recognized in the new version as an important software testing technique.

The background of the technology is in formal methods and formal verification.

Formal verification (proof) is another important verification technology that was already present in the old version.

While there can be great challenges in deploying these V&V techniques, they can also offer great benefits over competitors when used efficiently.
Model-Based Testing (two flavours)

Formal Verification

Proving (or disproving) the correctness of system with respect to a certain formal specification or property

Many flavors, main branches are model checking and theorem proving

Theorem proving requires manual effort while model checking can be automated to a large degree

In model checking, you need to give a system or its model and to specify a property that should hold in the system

If the property holds, the answer is YES

If the property does not hold, the tool provides you with a counter example

If the tool does not answer anything, the problem is probably too big for you computing resources
Annex A: Guide to the selection of techniques and measures

Table A.1 *Software safety requirements specification*
- Formal methods are an alternative to semi-formal methods
  - SIL 2&3: R (recommended)
  - SIL 4: HR (highly recommended)

Table A.2 *Software design and development – software architecture design*
- Formal design and refinement methods are an alternative to structured diagrammatic methods, semi-formal methods and automatic software generation
  - SIL 2&3: R
  - SIL 4: HR
Table A.4 Software design and development – detailed design
Formal design and refinement methods are an alternative to structured methods and semi-formal methods
SIL 2&3: R
SIL 4: HR

Table A.5 Software design and development – software module testing and integration
Model-based testing: SIL 1&2: R, SIL 3&4: HR
Formal verification: SIL 3&4: R (Note 3: the use of this technique may reduce the module and integration testing needed)

Table A.7 Software aspects of system safety validation
Modelling: SIL 1&2: R, SIL 3&4: HR

Table A.9 Software verification
Formal proof: SIL 2&3: R, SIL 4: HR
Annex B: Detailed tables (has changed from normative to informative in the new version)

Table B.2 Dynamic analysis and testing
Test case execution from model-based test case generation: SIL 1&2: R, SIL 3&4: HR

Table B.3 Functional and black-box testing
Test case execution from model-based test case generation: SIL 1&2: R, SIL 3&4: HR

Table B.5 Modelling
Finite state machines and formal methods are alternatives to Time Petri nets
Finite state machines: SIL 2: R, SIL 3&4: HR
Formal methods: SIL 2&3: R, SIL 4: HR
Table B.7 *Semi-formal methods*

Finite state machines/state transition diagrams are an alternative to Time Petri nets

SIL 1&2: R, SIL 3&4: HR
Since it is impossible to give exact rules how to combine different techniques, Annex C – Properties for software systematic capability has been added to the standard

Annex C supplements Annexes A and B

It should help in selecting specific techniques and outlining rationale for justifying the use of methods not listed in the tables

Annex C defines a concept of Rigour scale: R1, R2, and R3, from the least rigorous to the most rigorous

R3: enables achieving the required property with objective, systematic reasoning

R2: enables achieving the required property with high level of confidence given the objective acceptance criteria

R1: none or limited objective acceptance criteria
SIL 1&2: R1 is the minimum level to be aimed for
SIL 3: R2, where available, is the minimum level to be aimed for
SIL 4: highest rigour available

For example Table C.9 *Properties for systematic integrity – Software verification*

Formal proof is considered to be at level R3 with regards to the property of “correctness of verification with respect to the previous phase (successful completion)”

The techniques are selected subject to practical constraints in addition to their capabilities, example constraints include:

- How consistent and complementary the chosen methods, tools and languages are
- Whether the developers fully understand them
- How well they can be adapted to the specific problems during development
In part 7, the different methods and techniques listed in the tables are briefly discussed giving references to the literature.

Annex B (informative): Overview of the techniques and measures for E/E/PE safety related systems: avoidance of systematic failures

B.2.2 Formal methods

B.2.3.2 Finite state machines / state transition diagrams

Annex C (informative): Overview of the techniques and measures for achieving software safety integrity

C.2.4 Formal methods (describes CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM, VDM++, and Z)

C.5.12 Formal proof (verification)

C.5.12.1 Model checking

C.5.27 Model based testing (Test case generation)
How to get Started with Model-Based Testing and Formal Verification?

As seen in the previous slides, both model-based testing and formal verification are well presented in the requirements of the standard, at least on the high SIL levels (3-4).

Getting started with these techniques requires:
- An expert with necessary background with the associated tools
- Selecting the right tools
- Selecting the right sub-contractors

Different V&V methods can and should be used in complementary ways, formal verification of critical code and model-based techniques at the system level, for instance.

HR methods can be replaced by R methods if justification exists, like in the case of formal proof vs. test.

It is the whole V&V approach that counts, not the individual techniques.
In the past, there have been major impediments in using formal methods. Performance of the old tools and the computing power available was too limited in order to solve real life problems. Moreover, special expertise was required to use the tools. Nowadays, there is evidence in the literature that new tools can solve practical problems given the increased computing resources available. Unfortunately, however, there is still lack of user experience reports that would discuss the required expertise to use the modern tools.
Case: Verifying a 2oo3 Voting Scheme

The subject was very simple 2oo3 (2-out-of-3) voting scheme used for redundancy in a SC 3 level shutdown system.

The system development is being done according to the IEC 61508-2 and certification is being conducted by an independent organization.

For verification we used two bounded model checking tools, CBMC and EBMC (www.cprover.org).

Model checking as a technology does not require as high a level of expertise as, for instance, theorem proving.

Moreover, these tools were easily available and supported the input formats we were able to work with.

In addition, they support the existing development process and no major changes in the work flow are required.
always @(posedge clk or posedge rst_n) begin
  if(rst_n == 1'b 0) begin
    voter_state_i <= 1'b 0;
  end else begin
    if((ICH1_comparator_state_och_in == 1'b 1 &&
        ICH2_comparator_state_och_in == 1'b 1) ||
        (ICH1_comparator_state_och_in == 1'b 1 &&
        ICH3_comparator_state_och_in == 1'b 1) ||
        (ICH2_comparator_state_och_in == 1'b 1 &&
        ICH3_comparator_state_och_in == 1'b 1))
      begin
        voter_state_i <= 1'b 1;
      end
    else begin
      voter_state_i <= 1'b 0;
    end
  end
end
reg voter_state_check_in_pos;
reg voter_state_check_in_neg;
reg voter_state_check;

initial begin
    voter_state_check_in_pos = 0;
    voter_state_check_in_neg = 0;
    voter_state_check = 1;
end

always @(posedge clk or posedge rst_n) begin
    voter_state_check_in_pos <= rst_n & (1'b 0
            | (ICH1_comparator_state_och_in & ICH2_comparator_state_och_in)
            | (ICH1_comparator_state_och_in & ICH3_comparator_state_och_in)
            | (ICH2_comparator_state_och_in & ICH3_comparator_state_och_in)
    );
    voter_state_check_in_neg <= rst_n & (1'b 0
            | (!ICH1_comparator_state_och_in & !ICH2_comparator_state_och_in)
            | (!ICH1_comparator_state_och_in & !ICH3_comparator_state_och_in)
            | (!ICH2_comparator_state_och_in & !ICH3_comparator_state_och_in)
    );
    voter_state_check <= (!voter_state_check_in_pos | voter_state_i) &
        (!voter_state_check_in_neg | !voter_state_i);
    assert (voter_state_check);
end
Lessons Learned

First, formal verification is seen useful at least in simple cases like the one studied. It was possible to develop a generic assertion mechanism for the code translated from VHDL to Verilog, which should be reusable in the verification of similar designs and further supported by assisting tools. Training would still be needed, though, in order to get engineers to use the tools.

Second, the tools used in this study worked well, but their scalability is still unknown. It would also be better if the VHDL code could be checked directly without the translation process to Verilog, unless a (certified) translator that could be trusted is found.
Third, the design flow in this particular case could be improved by specifying the properties associated with the requirements more precisely. This would allow detecting errors and inconsistencies already in the requirements capturing phase, as this phase is widely recognized to be critical.

Fourth, experimenting first with tiny systems is highly recommended. Model checking suffers from the state explosion problem. Complex specifications are more error prone to write and harder to check.

One practical problem related to the tools might be to find a suitable formal verification tool. It might be more economical to buy formal verification as a service.
Conclusions

Model-based testing and formal verification are useful techniques in the development of safety-critical systems. While there still are many problems to be solved, the tools are getting more scalable and user-friendly. Moreover, the whole development process could be streamlined with the support of such tools. While the standards regulating the development practices in the safety-critical domain are recommending the use of formal verification tools, the biggest problem seems to be related to training. Methodological introduction into the development process could be eased with the help of simple assisting tools.
Q&A
THANK YOU

Contact:
Mika Katara
TUT
+358 40 849 0743
mika.katara@tut.fi

Ohjelmaturva project financiers: Tekes (Safety & Security programme), Metso Automation Oy, Sandvik Mining and Construction Oy, Konecranes Oyj, Bronto Skylift Oy Ab, ABB Oy, Epec Oy, John Deere Forestry Oy, Safety Advisor Oy, Sundcon Oy