Int J Softw Tools Technol Transfer (2004) 00: 1–12 / Digital Object Identifier (DOI) 10.1007/s10009-004-0152-y

Formal verification of fault tolerance in safety-critical reconfigurable modules

Jerker Hammarberg[*], **Simin Nadjm-Tehrani**

Department of Computer and Information Science, Linköping University, Linköping, Sweden
e-mail: simin@ida.liu.se

Published online: [redacted] 2004 – © Springer-Verlag 2004

Abstract. Demands for higher flexibility in aerospace applications has led to increasing deployment of reconfiguarble modules in the form of Field Programmable Gate Arrays (FPGAs). FPGAs act as digital hardware, but in the context of safety analysis they should be treated as software. Formal anlysis of safety-related properties of such components is thus essential for their use in safety-critical subsystems. The contributions of this paper are twofold. First, we illustrate a development process using a language with formal semantics (Esterel) for design, formal verification of high-level design, and automatic code generation down to synthesisable VHDL. We argue that this process reduces the likelihood of systematic (permanent) faults in the design and still produces VHDL code that may be of acceptable quality (size of FPGA, delay). Second, we provide a framework in which component fault modes and effects can be formally studied. We show how the design model can be modularly extended with fault models that represent specific or random faults (e.g. radiation leading to bit flips in the component under design) and transient or permanent faults in the rest of the environment. Some faults corrupt inputs to the component and others jeopardise the effect of output signals that control the environment. This process supports a formal version of Failure Modes and Effects Analysis (FMEA). The setup is then used to formally determine which (single or multiple) fault modes cause violation of the top-level safety-related property, much in the spirit of fault-tree analyses (FTA). All of this is done without building the fault tree and using a common model for design and for safety analyses. An aerospace hydraulic monitoring system is used to illustrate the analysis of fault tolerance.

[*] *Present address:* DST Control AB, Linköping, Sweden.

Keywords: Safety analysis – Formal verification – Fault tolerance – FPGA – Esterel

1 Introduction

The drive to produce faster, cheaper and better systems in the last decade has made the need for reuse of modules in development cycles of safety-critical systems a reality. Contrary to current life cycle processes in which several teams use various partially overlapping models and documents in almost independent activities, one now looks for ways to reuse the same component in different activities using different views. An important example of such component-based reuse is when a formal design model captures the functions of a component to be integrated into a system. In this paper we propose that the same (mathematical) *functional* model is used for *non-functional* (safety-related) analysis at the system level. This opens up for reuse of the component both in different parallel processes in the same development cycle and with respect to reuse in future generations.

This paper argues for such a reuse methodology in the context of a specific example: design of reconfigurable and safety-critical hardware. The results thus build on languages and tools that promote flexible and dependable hardware. However, the generic message of the paper is the merger between the functional and safety-related analyses that are by no means language or application dependent.

1.1 FPGAs

The combination of flexibility and efficiency requirements in the development of digital components has made the use of Field Programmable Gate Arrays (FPGAs) prevalent, not only in space electronics [17] but also in more traditional avionic systems. FPGAs have gained popularity during the last decade because they provide many of the advantages from both hardware and software components. Being a hardware component, an FPGA is efficient compared to software, and the rapid development of the technology results in growing capacities. At the same time, an FPGA can be configured, tested and reconfigured in the field, that is to say by the developers themselves, and this provides for flexibility that could earlier only be achieved by using a software component. Due to this flexibility and because FPGAs are standard components manufactured on a large scale, the use of FPGAs instead of more traditional hardware can drastically reduce both costs and time-to-market. In a recent study of an air intercept missile system, FPGAs were used at design stage to provide flexibility, but they were later carried over to the tactical system due to cost considerations as compared with the more prohibitive application-specific integrated circuits (ASICs) [14]. This trend necessitates development of techniques for evaluation of the safety risks associated with FPGAs.

When considering traditional safety evaluation techniques, FPGAs lie somewhere in between hardware and software. Although traditional quantitative safety and reliability analysis favours the use of hardware in safety-critical applications, recent results show that this may be a changing reality. Shivakumar et al. estimate that soft error rates per chip (bit flips in hardware as a result of cosmic radiation) of logic circuits will increase nine orders of magnitude in a 10-year perspective by 2011, at which point they will be comparable to the soft error rate per chip of unprotected memory units [25].

1.2 Design languages

The development of hardware with software-like properties calls for new, software-like development methods, and indeed this is now sought by industry. The traditional gate-level schematics and other low-level design methods are being abandoned in favor of high-level languages in order to cope with increasing complexity and to make use of the flexibility allowed by FPGAs. For safety-critical systems, where faults in a component may have disastrous consequences, there are additional safety and reliability demands that have to be taken into account. The use of VHDL and other hardware description languages is insufficient because they are still at a too low level, making them error prone for larger and more complex systems, and they are not well suited for formal analysis such as model checking and causality analysis. Here we argue that the use of synchronous languages such as Esterel for FPGA design can in many cases be a feasible solution to these problems. Esterel is modular, easy to use and based on formal semantics, allowing for efficient formal verification. Moreover, Esterel can now be automatically compiled to synthesisable VHDL with a commercially available compiler. In this article we consider the potential shortcomings of an FPGA design obtained in this manner by looking at the 'costs' associated with using Esterel. The main concern with automatic code generation from a language at a high abstraction level such as Esterel is the degraded efficiency and performance of the implementation. In particular, the additional logic blocks needed as a result of the overhead from the compilation could exceed the capacity of the FPGA.

1.3 Combined reliability and safety analysis

The above considerations motivate a serious study of system development processes that facilitate efficient integration of FPGAs and other software or hardware components in systems design, with improved levels of safety. In this paper we study the integration of reconfigurable modules in safety-critical aerospace applications by illustrating two aspects of fault management. First, we consider techniques that help the developer to remove or prevent systematic permanent design faults. Second, we put forward an analysis method that helps the system safety engineer to concentrate on combinations of external and random faults that should be the focus of the fault tolerance and containment techniques. These faults may be of a permanent or transient nature.

The suggested method combines analysis of functional correctness (thereby reliability) with safety analysis.[1] It builds upon elements of analysis normally performed to concentrate on faults that lead to system-level hazards, much in the spirit of fault-tree analysis (FTA) and failure modes and effects analysis (FMEA) techniques [13]. The novelty of our approach is the combination of this type of safety-related analysis with the model-driven development of a design, early formal verification, and automatic code generation. The contributions of the paper are therefore not in development of core techniques but in bringing together a number of ingredients that have so far existed as isolated activities, by providing a systematic approach to the introduction of engineering knowledge in a formal development process. This approach is illustrated on a real aerospace case study, the hydraulic control unit in an aircraft.

We round up the article with a discussion of the trade-off between high dependability and performance requirements, and compare the characteristics of Esterel with other candidates in a PID control application taken from an aircraft arrester system. More specifically, an Esterel-based design is compared with a hand-coded VHDL design.

The remainder of the paper is structured as follows. Section 2 presents the hydraulic system case study. Section 3 introduces some of the necessary concepts and

[1] The certification authorities use the term functional safety to denote the evaluation of safety of a piece of equipment during functional use, which is somewhat confusing for computer engineers that consider safety a non-functional property distinguished from reliability.

techniques involved in the development and safety analysis of the system. Section 4 describes our approach to a combined development of reusable design models and safety-related analyses. Section 5 provides some insights into the size/performance trade-offs for higher abstraction models and higher dependability. Section 6 compares with related works, and Sect. 7 concludes the paper and presents future projects.

2 Aerospace application

The ideas presented in this paper were motivated by the need for safety analysis of a subsystem inside the JAS 39 Gripen multi-role aircraft, obtained from the Aerospace division of Saab AB in Linköping, Sweden. The purpose of the system is to detect and stop leakages in the two hydraulic systems, which feed the moving parts, including the flight control surfaces and the landing gear, with mechanical power. Leakages in the hydraulic systems could in the worst case result in such low hydraulic pressure that the airplane becomes uncontrollable. To avoid this, some of the branching oil pipes are protected by shut-off valves. These valves can be used to isolate a branch in which a leakage has been detected. Then, although the leaking branch will no longer function, the other branches will still keep the pressure and be able to supply the moving parts with power.

The reading of oil level sensors and the controlling of the four shut-off valves are handled by three electronic components, as depicted in Fig. 1. The H-ECU is a software component that continually reads the oil reservoir levels of the two hydraulic systems and determines which shut-off valve to close accordingly. However, it would be very dangerous if some electrical fault caused more than one valve to close at the same time – this could result in the locking of the flight control surfaces and the landing gear. For this reason, two programmable logic devices, here called PLD1 and PLD2, continually read the signals to and the statuses of the valves, and if the readings indicate closing of more than one valve, they will disallow further closing. Thus, PLD1 and PLD2 add fault tolerance to the shut-off subsystem implemented in the H-ECU. PLD2 will only accept a request from the H-ECU for closing a particular valve if the check, which is partly done in PLD1, indicates that everything is OK. A valve will close only when both the low-side signal, which is the shut-off signal directly from the H-ECU, and the high-side signal, which is the checked signal from PLD2, are present.

Design for fault tolerance is an inherent part of satisfying safety requirements for a system. Although quantification of dependability using selected metrics is well studied in electromechanical systems, satisfaction of safety requirements in the presence of software or software-like hardware is still a major challenge. In this particular example, one wants to verify that no single fault, be it in the components, in the wires or in the valves, can cause more than one valve to close at the same time. Moreover, the safety engineer is interested in combinations of faults that might lead to violation of top safety requirements. These goals are traditionally pursued through the FTA/FMEA processes. In the following discussion, we will show how a design model of the system can be used to formally verify that it is tolerant to single faults. The proofs also pinpoint the significant combinations of potential double faults.

3 Background

This section introduces some concepts and earlier work that will be used for the analysis described in Sect. 4. An introduction to the properties of Esterel will be followed by a short description of fault-tree analysis and earlier related works. Since model-based development down to

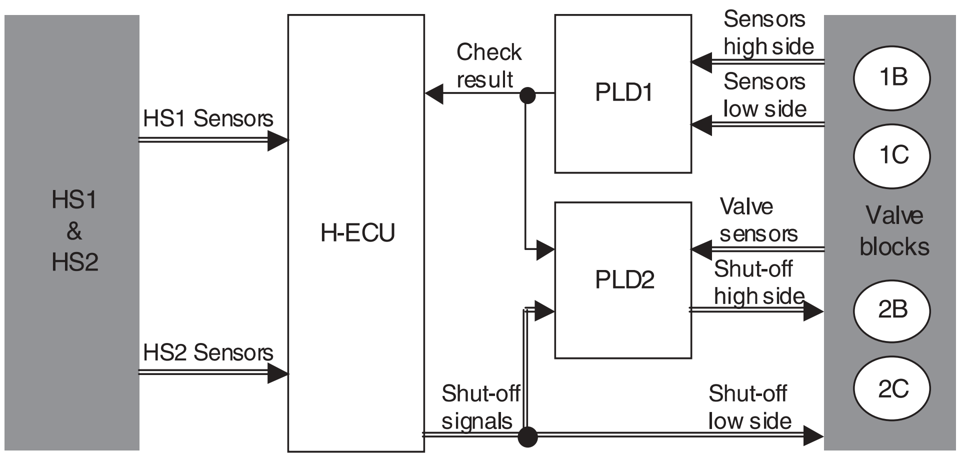


Fig. 1. Hydraulic leakage detection subsystem. *White boxes* indicate electronic components and *patterned boxes* indicate physical parts. *Arrows* indicate signal flows; *double arrows* are collections of several signals

code generation and formal verification is supported by the Esterel language and tools, these will be used to illustrate the ideas.

3.1 Esterel

Esterel [1] is a synchronous language with formal semantics, making it ideal for analyses with formal methods. This means that if systems are designed in Esterel, efficient formal verification is available directly on the actual design code.

Esterel is tailored for designing reactive systems [21], which continually read inputs and produce outputs. In Esterel, time is modelled as a discrete sequence of instants. Each instant, new outputs are computed from the inputs and from the internal state (control state, latched signals and variables) according to the imperative statements of the Esterel programme. Figure 2 shows some of the Esterel code for PLD2 in the aerospace example. Esterel systems and subsystems are always defined as modules, as seen in lines 1 and 15 that enclose the code for this example. Lines 2, 3 and 5 declare the input and output signals of this module, much like in a hardware description language. Lines 7 through 14 define an infinite loop, running one iteration at each instant that is represented by 'each tick'. This means that the code from line 8 to line 13, emitting the ShutOff_1B signal if a given condition is present, will be executed instantaneously at every instant. The output signal *ShutOff_1B* will be emitted only if there is an incoming request to close valve 1B, all input seems to be OK and the *CheckOK* signal was present at the previous instant.

Our experience [12] indicates that synchronous hardware systems can be written more easily and with fewer lines of code in Esterel than in hardware description languages such as VHDL [9]. This is due to the higher level of abstraction and the suitable constructs for modelling hierarchical reactive modules. Bug count per thousand lines of code (bugs/kloc) appears to be constant over different languages [23]. Hence, we believe that the use of Esterel could contribute to system correctness and maintainability at a lower cost (i.e. development time) compared to the design methods that are commonly used in industry today.

The main advantage with using Esterel for hardware development is, however, its tight coupling to formal methods for system analysis. Esterel compilers automatically check the design for causality loops, and the fact that Esterel is synchronous makes formal verification particularly efficient [11]. Non-deterministic control programmes are rejected at compilation stage, making the reliance on the output of a module at each instant possible. The commercial tool Esterel Studio [7] has two model checkers built in – one based on binary decision diagrams (BDDs) [22] and one based on propositional satisfiability (SAT) techniques [26]. Thus, many design faults should be quickly found and eliminated already at the design stage of the process.

The two model checkers bring the best of both worlds in the battle against complexity, namely state space explosion and long proofs with many open hypotheses, respectively. The tool used in our experiments was the SAT solver. It implements the Stålmarck proof technique that is based on the observation that many systems with large state spaces have short proofs.[2] The interested reader is referred to the tutorial by Sheeran and Stålmarck [27] for a full exposure to the method, which is beyond the scope of this paper.

In Esterel, the safety[3] properties to prove with the model checker are formalised as synchronous observers. The *observer* is a process, also written in Esterel, that runs in parallel with the actual system and monitors its input and output signals, as depicted in Fig. 3. As soon as the observer finds that the property is violated, it emits an *alarm signal* (sometimes also called *bug signal*). Prov-

[2] In the sense of Gentzen style deductions.

[3] In the formal verification sense: non-reachability of a bad state.

```
 1: module PLD2:
 2:   input CheckOK;
 3:   input ShutOffRequest_1B;
 4:   % more inputs
 5:   output ShutOff_1B;
 6:   % more outputs
 7:   loop
 8:      signal InputNotOK in
 9:        % some code emitting InputNotOK if something is wrong with the input
10:         present ShutOffRequest_1B and not InputNotOK and pre(CheckOK) then
11:            emit ShutOff_1B
12:          end present
13:      end signal
14:    each tick
15: end module
```

Fig. 2. Part of the Esterel code for PLD2

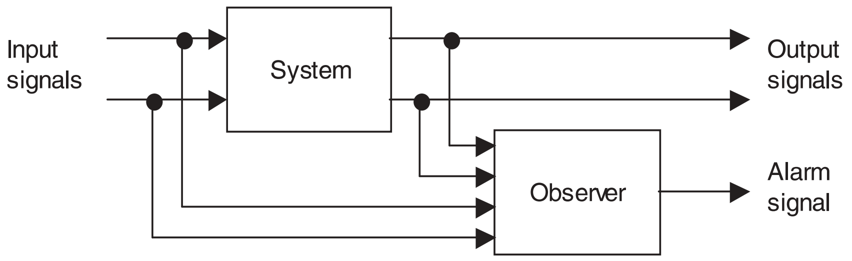


Fig. 3. A system being monitored by an observer. *Arrows* indicate signal flow or signal readings

ing the property is thus reduced to proving that the alarm signal will never be emitted. Note that the observer is used only during verification of the design. For code generation and during actual execution of the system, these modules are not present.

Finally, it should be noted that there are other synchronous languages, e.g. Signal [19] and Lustre [10], that support formal verification. However, none of them can presently be compiled to VHDL (and thus not synthesised to an FPGA) with a commercially available compiler. This is the main reason for Esterel being chosen in our studies.

3.2 Fault-tree analysis (FTA)

A traditional technique for safety analysis is to produce a *fault tree* that displays the hazardous events and their possible implicants. In a fault tree, each node represents an event, and the *top event* is the disaster that one wants to find the possible causes to. The children of an event in the tree are the events that, alone or together, may cause that event. By producing a complete fault tree, one can reveal what faults (or combinations of faults) are hazardous and must be attended to. In addition, if the probabilities for the leaf events are known, a probability for the top event can be calculated.

Consider now a fault tree for the aerospace application, where the top event is the airplane crashing. One event (internal node in the fault tree) that can single-handedly cause the top event is when two or more shut-off valves close simultaneously. To further derive the descendants of this event in the fault tree would require a complete analysis of the behaviour of the three electronic components H-ECU, PLD1 and PLD2, something that would be extremely tedious to do in separate models developed in FTA tools, and this fault tree would not be the only analysed fault tree for this system. Clearly, there is a need for tool support to create the fault tree automatically in a way seamlessly related to the design models. With current techniques every change in the design would potentially mean reconstructing the fault tree. Moreover, the current FTA/FMEA analyses assume that the event represented in the leaves are independent, and thus dependent faults or faults whose effects are non-monotonic in time (i.e. one fault masking the effect of another fault) cannot be analysed by traditional methods. Our approach to performing FTA/FMEA-like analyses based on the design models both reduces the need for constructing the fault trees and makes the formal analysis of complex software/hardware dependencies automatic. It also allows modelling dependencies among various fault modes and non-monotonic faults if needed by the application.

3.3 Related work

Earlier work on the combination of design models and FTA-like analysis is sparse. Where fomal analysis is combined with methods for safety/reliability analysis, the major works in the area try to improve the efficiency of the computations on fault trees or improve the tools that perform such computations by strengthening their mathematical underpinnings (including dynamic fault trees). An example of the first category of work is the study by Dutuit and Rauzy [4], and an example of the second category is the Galileo project at the University of Virginia (see e.g. [20]). Our work is mostly related to research that enables the use of common models for system development and for safety/reliability analyses. In fact, in our work we avoid building the fault tree.

Leveson points out the risks of confusing safety and reliability analyses or assuming that higher reliability for a software component automatically enhances safety [18]. We believe that similar arguments hold for reconfigurable hardware. Fenelon et al. [8] recognise the missing link between the design and safety analysis.

In 1999 Åkerlund et al. showed how FTA-like analysis of a hardware or a software component could be performed with the help of an SAT solver [29]. The component was first modelled as a set of logic formulas in the tool NP-Tools relating the input and output variables. By checking that the formulas that constitute the model imply a specific property, such as ‘at most one shut-off valve is signalled to close simultaneously’, compliance with a safety property (in the sense of non-reachability) in the absence of external faults could be proved. This is ordinary model checking. Then, the model was extended with additional input signals representing specific fault modes in the environment that could cause corruption of the real input to the component. A fault was modelled

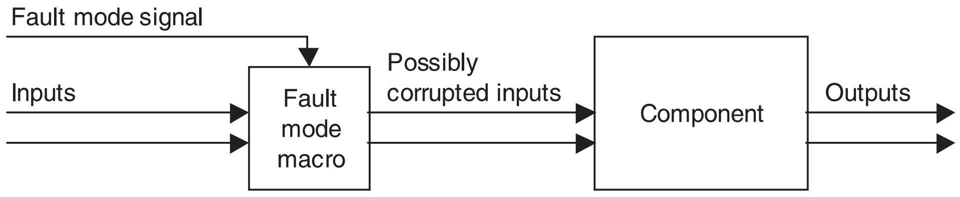


Fig. 4. Extending a component with a fault mode. *Arrows* indicate signal flow

by a macro that modifies the input signals if the fault mode signal is present, as illustrated in Fig. 4. By again verifying this extended model, one could either prove that the component was tolerant to the modelled faults or see which fault modes could cause violation of the property on the basis of the produced counter-examples. The counter-examples were automatically generated by the verification tool as a side effect of the analysis. Note that a fault tree was never built explicitly, but the set-up allowed the study of effects of single as well as multiple faults (including those with non-monotonic effects). These boolean expression descriptions of the model were considered as too low level for describing complete complex systems, but at that point there were no commercial tools with high-level models, code generation and formal verification. This paper extends the above work by working at a higher abstraction level and considering faults in the outputs generated by a module (as well as its inputs).

ESACS is an ambitious European project that attempts to bridge this gap by an approach similar to ours using an integration of the Statemate and SCADE environments respectively with the industrial FaultTree+ tool [6]. Recent work in the project by Buzzano and Villafiorita [2] uses the NuSMV2 model checker to generate minimal cut sets from a system model and definiton of failure modes. An elegant approach to modelling failures in mode automata and compiling into boolean formulas has recently been proposed by Rauzy [24]. However, some steps remain before exploitation of the techniques in commercial tools. The need for raising the abstraction levels in modelling embedded control systems is increasingly recognised in current projects that promote model-driven development.

VHDL is the dominant hardware description language in industry, which is why we have chosen it for comparative study in this paper. However, research environments that promote general-purpose programming languages for FPGA designs are emerging. For example, JHDL [15] provides a development environment that includes simulation and testing and makes future formal verification a possibility via verification tools that accept subsets of Java as input language.

4 Analysis of fault tolerance

In this section, we combine the formal modelling of the design and FTA/FMEA-like analyses to obtain several advantages. First, instead of low-level (circuit) modelling of the design as in the Åkerlund et al. study, we lift the design abstraction to a higher level. Due to the maturity of Esterel tools, high-level design can now be used for automatic code generation. Without this step, original ideas with automated FTA-like analysis are far from applicable in everyday industrial settings. Second, we extend the classes of considered faults. In addition to faulty inputs of the component under design, we model faulty outputs generated by the component itself as a result of failures, as well as faults in the environment that receives the output of the component. The higher abstraction level allows arbitrary classes of faults to be modelled based on detailed knowledge of the application. In the FPGA case study, for example, an interesting fault type was random errors due to radiation.

We present an extension of the current Esterel development environment in which templates are used to verify effects of the above fault types on any design module. Hence we remove the need for creating and managing two separate models (one for design verification and one for FTA/FMEA analysis of safety-related properties). The idea builds on a framework for combining individual components to form a verification bench that allows the study of the effects of the above fault classes.

Next we show the steps involved in formally analysing functional correctness and fault tolerance in an integrated way.

4.1 Development of verification bench

In order to verify the hardware and software components working together with physical parts of the system, it will be necessary to build an Esterel model of relevant parts of the environment. In addition to models of the physical parts, we include models of all necessary wires between various components as well as wires between the components and the physical parts. Next, observers running in parallel with the components and the environment are added. This construction will be referred to as a *verification bench*, which various components can be plugged into and analysed.

A verification bench for the aerospace application is illustrated in Fig. 5. It contains models of the four valves and of the wires between the components; it has empty slots for H-ECU, PLD1 and PLD2. The observer monitors the output of the valve models and emits an alarm signal as soon as more than one valve is closed at the same instant.

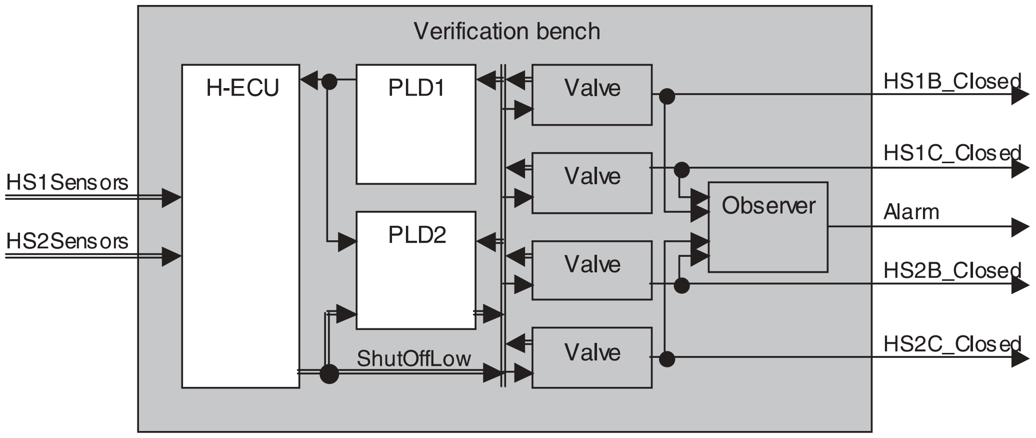


Fig. 5. Verification bench for aerospace application. *Grey boxes* indicate modules of the verification bench and *white boxes* indicate modules to be verified. *Arrows* indicate wires; *double arrows* are collections of several wires. The *vertical bar* in the middle is shorthand for a set of connections

```
module Main:
  sensor HS1Pressure : double;
  % more main inputs
  output HS1B_Closed;
  % more main outputs
  output Alarm;
  signal
    ShutOffLow_1B;
    % more wires
  in
    run HECU [
       signal HS1Pressure / HS1Pressure;
       % more connections
    ]
    ||
    run PLD1 [...]
    ||
    run PLD2 [...]
    ||
    run Valve [...]
    ||
    % more valves
    ||
    loop
       present HS1B_Closed and HS1C_Closed then
          emit Alarm;
       end present;
       % more checks
    each tick
  end signal
end module

module Valve:
   % valve model
end module
```

Fig. 6. Skeleton code for aerospace application verification bench in Esterel

Figure 6 shows a skeleton for the implementation of the verification bench in Esterel. All wires are modelled with local signals using the `signal` construct. The main body consists of calls to the components (`run` keyword), and the valve models run in parallel with the observer that checks that no pair of valve-closed signals is simultaneously present. The signal renamings in the `run` construct defines how the interface of the component is connected to the local signals or to the main inputs or outputs.

Note that the verification bench can be written independently of the components. This is useful for distributed development – upgrades and code from different departments, or even different companies, can easily be plugged in and immediately checked by formal verification. Also note that this allows verifying sequential circuits (modelled in Esterel) and arbitrary fault models that can be expressed as Esterel 'fault patterns'. For example, if the application demands that 'no transients that affect more than X cycles of computation should lead to the top event', then the templates for fault pattern can be made more complicated than those needed for this case study to reflect this.

From here on we assume that the design models for the components that are included in the safety analysis have been formally verified for compliance with their requirements specifications. This can be done in the Esterel environment (illustrated in Fig. 9 below).

4.2 Augmenting with fault modes

The next step in the process of checking for safety-related fault tolerance is to model faults in the verification bench. Here we will show how to model malfunctions in the chips on which the components run, in our case the microprocessor and the two PLDs. Many other classes of faults, such as electrical or mechanical faults in physical parts of the system, can also be modelled in the verification bench with some creativity.

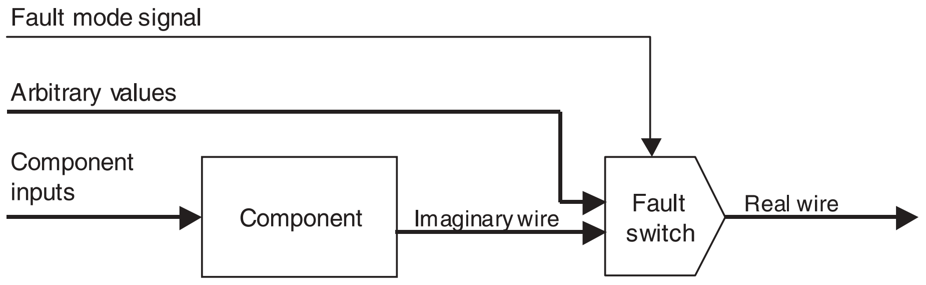


Fig. 7. Fault switch modelling component faults. *Thick arrows* indicate possibly several signals and the *thin arrow* indicates a pure signal

Faults in the hardware parts, such as FPGAs or microprocessors on which the system components run, can occur due to sudden power-down, overheating or radiation that flips over some bits inside the chip. Modelling such faults at fine granularity would be complicated, so we will opt for a coarse granularity strategy and assume that if such a fault occurs, the outputs of the component running on the chip can be anything. Besides being much simpler, this strategy also has the advantage that the component design module does not have to be altered; the malfunction can be completely modelled in the verification bench.

To induce completely arbitrary output from a component, one can add an additional block coming in between the outputs of the component and the following wires, as depicted in Fig. 7. This block will be referred to as the *fault switch* and can be seen as the formal verification counterpart of fault injectors used in test benches. Whereas in test environments the objective is to see what the output of a computation is in the presence of a fault, here the objective is to see whether a safety-related property that was earlier formally verified still holds in the presence of the fault. Figure 8 shows the Esterel implementation of a fault switch following the H-ECU, and it should replace the `run HECU` call in Fig. 6. The idea is to let through the correct output of the component into the wires as long as the fault mode signal is absent, but when it is present, arbitrary values will be fed into the wires instead. These arbitrary values can be taken from additional inputs of the verification bench since the verification tool will allow these to be anything. The component must furthermore be connected to the inputs of the fault switch instead of the wires.

In the hydraulic application case, the following 15 faults were modelled:

- Arbitrary malfunction in the H-ECU, in PLD1 or in PLD2 (e.g. due to radiation).
- Short-cut to ground on the incoming low-side shut-off signal to each of the four valves.
- Short-cut to ground on the low side of each valve.[4]
- Short-cut between the low and the high side of each valve.

4.3 Fault mode verification

In Esterel Studio, plugging in the designs of the components is simply a matter of loading them into the verifica-

[4] This is not the same as grounding of the incoming low-side shut-off signal since there are electronic components in the valve that, based on the incoming signals, produce a voltage that makes the valve close. This fault models grounding of the low side of the voltage.

```
...
run HECU [
   signal HS1Pressure / HS1Pressure;
   % more connections
   % feed output to local correct-values wire
]
||
loop
   present FaultHECU then
      % feed arbitrary values to local signals (ShutOffLow_1B etc)
   else
      % feed correct values to local signals
   end present
each tick
...
```

Fig. 8. Skeleton code for fault switch in Esterel

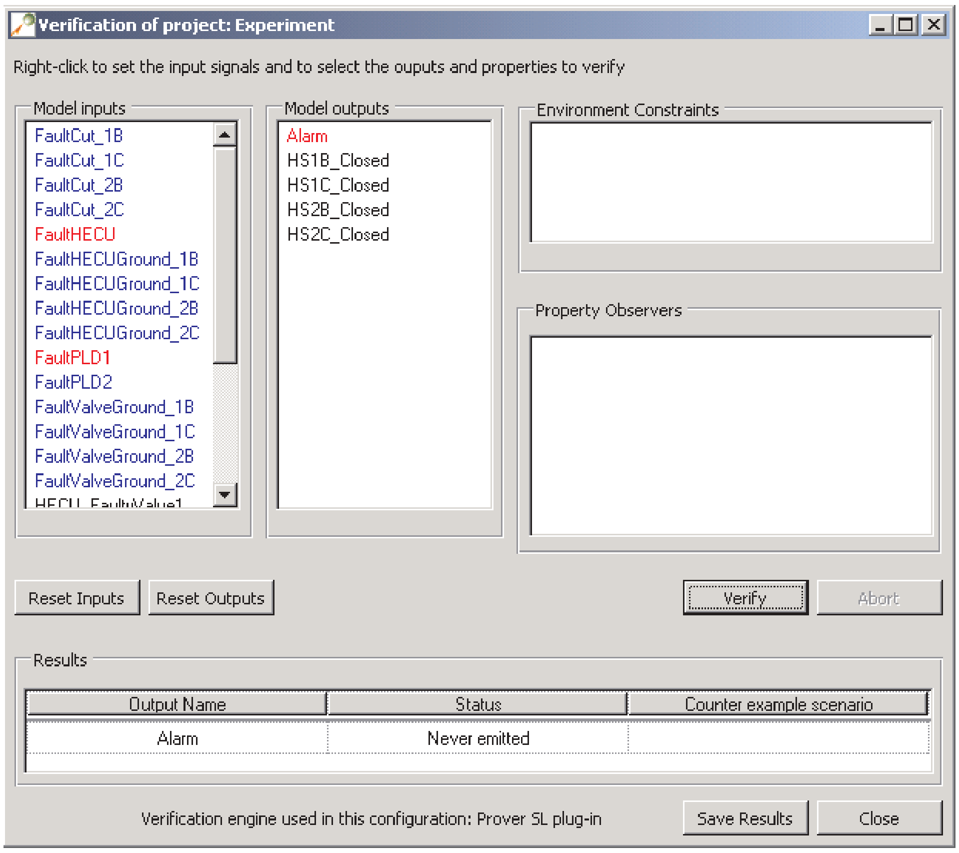
tion bench project. The verification bench and the components together then constitute a model of the complete subsystem that can be verified with the built-in model checker.

To analyse safety-related fault tolerance, we make use of the feature in Esterel Studio to constrain some input signals. In this case the fault mode signals can be restricted to analyse different scenarios. By further testing for the observer alarm signal emission, we can check for violation of the safety property in the presence of those faults. For example, if we are only interested in component malfunction to see what combinations of faulty components the system can withstand, we can constrain all other fault mode signals to be absent. This is simply achieved by marking the appropriate faults as absent in the input window within the Esterel verification environment (Fig. 9). Note that finding systematic design faults that are unrelated to the run-time environment is the same as finding violations of safety properties in the absence of external faults. This is done by designating *all* fault mode signals to be absent, which is typically done first, before analysis for fault tolerance begins. Next, single (and possibly double) faults can be found by allowing all single (and all pairs of) fault modes, constraining the other fault mode signals to be absent.

Figure 9 shows the model checker window in Esterel Studio when verifying the aerospace application verification bench. It should be clear that if this model were verified with all fault mode signals allowed to be present, then the safety properties would most probably be found false, unless the system is extremely robust and can withstand any fault. When a property is found false, a counter-example is produced that shows a sequence of combinations of faults and inputs that lead to safety violation. Note that if there are many fault modes, there will be too many of such combinations, so running the unrestricted fault scenario does not provide any insight to the engineer.

Using this technique on the aerospace application verification bench, consisting of 422 lines of code spread over 6 Esterel modules, we could verify the following:

– *The components do not contain design faults causing violation of the property.* This was shown by constraining all fault mode signals to be absent before running the verification.
– *No combination of the 12 valve faults can cause violation of the property.* This was verified by constraining the three component fault mode signals to be absent.
– *No single random fault can cause violation of the property.* The three component fault modes were first checked independently, constraining two of them to be absent and one present at a time. The other 12 faults were already cleared by the previous step.


Verification of project: Experiment

Right-click to set the input signals and to select the ouputs and properties to verify

Model inputs
FaultCut_1B
FaultCut_1C
FaultCut_2B
FaultCut_2C
FaultHECU
FaultHECUGround_1B
FaultHECUGround_1C
FaultHECUGround_2B
FaultHECUGround_2C
FaultPLD1
FaultPLD2
FaultValveGround_1B
FaultValveGround_1C
FaultValveGround_2B
FaultValveGround_2C
HECU_FaultyValve1

Model outputs
Alarm
HS1B_Closed
HS1C_Closed
HS2B_Closed
HS2C_Closed

Environment Constraints

Property Observers

Results

Output Name	Status	Counter example scenario
Alarm	Never emitted	

Verification engine used in this configuration: Prover SL plug-in

Fig. 9. The model checker window in Esterel Studio

– *The only double fault violating the property was shown to be when both the H-ECU and PLD2 were faulty.* This was checked by testing the relevant possible double faults, again constraining the other fault mode signals for each pair. Since the four valves are symmetrical, it was sufficient to check physical faults in one valve such as 1B, reducing the number of combinations to 12.

The model checking never took more than a few seconds. As long as the system is mainly combinatorial (as in this case), the complexity of the problem, in average case for an SAT solver, is linear to the size of the model [27]. This means that it should still be feasible to verify systems of this kind even if they are as large as thousands of lines of code. Alternatively, if testing for validity takes too long, one may opt for the bug-chasing strategy option in Esterel Studio. This option omits the proof-searching part of the model checking and only searches for counter-examples, which is considerably quicker and can thus not be used for proofs but only for finding fault mode combinations.[5]

5 Performance and resource considerations

The ability to verify a design at early stages and to extend the analysis to cover fault tolerance is obviously beneficial in safety-related systems. However, industry is interested in knowing what performance/size trade-offs are involved in a new technology. To evaluate the performance and resource usage of an FPGA produced using the above approach, we have studied Esterel in comparison with other languages typically used in control applications.

In the Saab case study we did not have access to the VHDL code on which the real implementation was based. We were simply presented with requirements specifications for the system in natural language. Thus, to make a detailed comparison against other potential languages, we used a simpler example for benchmarking purposes. The case study is presented in more detail elsewhere [12]. However, the details presented here suffice for comparison purposes.

The application is a brake control for an aircraft arrester system, provided by DST Control Systems AB, that works as follows. A large net is placed at the end of an airfield runway. If an aeroplane fails to take off, it will run into the net, which is connected to ropes that are rolled up on wheels in stations on both sides of the runway. The task of the brake control is to calculate an optimal brake force that is applied to the net in order to decelerate the aeroplane, and a PID controller applies the actual brake force accordingly. Since this is a safety-critical application, methods that minimise the development time while increasing safety and reliability of the product are of interest.

To evaluate the trade-offs, we will consider the size of the generated circuit and its delay time, which is a measurement of its performance. Delay refers to the time that the circuit takes to stabilise the values after a clock tick – this does indeed take some time since the voltage needs to transmit through all the components. In order to have full control over the comparative designs, we developed two versions of the PID controller module for the application; one in hand-coded and manually optimised VHDL and one in Esterel using the same latency. The Esterel code was then used to automatically generate VHDL (using version 4 of the commercial compiler released in August 2002). Both versions were then used to synthesise the PID on a Xilinx Spartan2 XC2S150 using the tool Synplify Pro 7.1 from Synplicity [28]. These results were then presented to the company that prior to this project had implemented an FPGA core on which the PID was running. The core had been generated using circuit schematics as a design language.

Table 1 shows the comparative results. The Size column indicates the percentage of logic blocks in the FPGA needed for the implementation as calculated by the synthesis tool. The Delay, that is the maximum stabilisation time after a clock tick, was also calculated by the tool. The Latency column shows the number of cycles for computing the control value. Note that this is entirely a design choice. Finally, the No. of lines column presents the number of lines of code for each implementation, excluding blank lines and comments.

The difference between Esterel and VHDL is clear by comparing the first and second rows. The hardware-tailored VHDL language leads to an implementation that is superior in terms of efficiency, computing the result three times faster and occupying less than half of the logic blocks. A large part of the extra space needed by the Esterel implementation turns out to be due to the wide integers. There is no way of specifying the range for an integer in Esterel; one can only choose between 16 and 32 bits for the whole system at once. The PID controller uses both 16- and 32-bit integers, so 32 bits for all integers was the only choice. As an experiment, the unnecessarily wide integers were manually narrowed in the Esterel-generated VHDL code, and then the size could be reduced to 33%. This fares much better against the handwritten VHDL code. However, such manual modifications effectively invalidate any formal verification made

Table 1. Comparison between two design approaches

Method	Size	Delay	No. of lines
Esterel	52%	82 ns	98
VHDL	22%	29 ns	139

[5] The model checking is an induction proof over the discrete time. The induction depth is increased until either the base step is found false or the inductive step is found true. The bug-chasing strategy omits the inductive step.

on the Esterel design and should therefore be avoided. In any case, the size of the design (in terms of lines of code) is always likely to be smaller in Esterel (a 30% decrease compared to the VHDL code size in this case).

The more interesting comparison would be between these two designs and the existing technique at the company for representation of the design as schematics. However, it is not possible to compare the area of the FPGA solution with both the processor core and the PID application that runs on it as a set of sequential instructions. While optimising latency and delay had not been the prime concern when designing the in-house FPGA, the application developers were clearly impressed by getting an FPGA that was formally verifiable and occupying an acceptable area of the chip.

To sum up the analysis, one may say that VHDL is still the appropriate design language for FPGAs if efficiency is the prime concern. In many cases, however, the acceptance criteria may simply be small enough to fit into the chip, as was the case in both the Saab and the DST case studies. Not fitting into an existing chip means using several chips or moving to a more expensive generation of chips (the more expensive solution in both cases). If the 'small enough' criterion is chosen, Esterel may produce implementations of fully acceptable size and speed in many existing applications. Moreover, one could argue that the additional cost of using a larger FPGA, should the implementation not fit, will in many cases be minor compared to the potential gains resulting from shorter development time and increased reliability. Ongoing research (see for instance [5] and [16]) aims at further improving the compilation of Esterel to hardware, and the language itself is being developed to allow for better control over the hardware resources.

6 Conclusions and future work

The quest to develop safe systems while incorporating modern technology and more complex functionality is a driving force behind the rising interest in FPGAs in safety-related systems. As with other reusable components, we need guidelines on how to incorporate such components into the development and safety-analysis processes. Furthermore, we need specific guidelines as to the treatment of FPGAs when building up safety arguments.

The analysis performed on some subsystem (e.g. the hydraulic subsystem in this paper) may be useful to carry over to another system that 'imports' this subsystem as a component. In this paper we did not consider the form and character of interfaces that make such a reuse in a new system efficient. When (reusable) software components become a reality in safety-critical applications, then well-defined interfaces that characterise the failure modes and effects of a component failure are necessary. This is a topic for our current research.

In this paper we provide some evidence that results of the last decade of research in language design, formal verification and tool development are reaching maturity levels that make a serious case for incorporating these techniques in real applications. We have illustrated how an FPGA design process can combine analyses for safety and functional correctness and guide the designer in finding the focus for system-level fault tolerance. The abstract (implementation-independent) design model was shown to be transformable to a VHDL implementation with acceptable loss of efficiency (still fitting in the circuit that was intended for the design) and at the same time supporting formal analysis of the design.

We proposed a formal verification bench for analysing systematic, specific and random faults in the external environment, using the standard technique of observers, and showed it to be an efficient means of pinpointing fault combinations that need more attention in safety evaluations. The use of verification benches for safety analyses should be applicable to any design language with formal verification support, not only to Esterel.

Ideally, the analysis should render a set of prime implicants of the system failure function, so that the engineer can see all the causes of a safety property violation with one push on a button rather than having to try all the fault mode combinations manually. However, this is not possible to accomplish in the current version of Esterel Studio. One algorithmic approach is presented in [3]. Current work on the Esterel verification bench includes extension of the tool so that prime implicants (or FTA-like cut-sets) are automatically generated. Another extension would be visualisations and combination with quantitative methods currently used by engineers' state-of-the-art FTA tools.

Acknowledgements. This work was partially financed by project SAVE supported by Swedish Strategic Research Foundation (SSF), the national aerospace programme NFFP3-428. The early grounding work was supported by the Swedish Board for Technical Research (TFR).

The authors gratefully acknowledge the valuable feedback of contact persons at Saab Aerospace, Lars Holmlund, Hans Sjöblom, Thomas Trei, Anna-Karin Rosén, and Marriane Almesåker, as well as contact persons at DST Control, Jan Nygård and Jan-Erik Strömberg.

References

1. Berry G, Gonthier G (1992) The Esterel synchronous programming language: design, semantics, implementation. Sci Comput Programm 19(2):87–152
2. Bozzano M, Villafiorita A (2003) Improving system reliability via model checking: the FSAP/NuSMV-SA safety analysis platform. In: Proceedings of the 22nd international conference on computer safety, reliability and security (SAFECOMP'03). Lecture notes in computer science, vol 2788. Springer, Berlin Heidelberg New York, pp 49–62
3. Deneux J (2001) Automated fault-tree analysis. Master's thesis, Uppsala University, Uppsala, Sweden
4. Dutuit Y, Rauzy A (2000) Efficient algorithms to assess components and gates importances in fault tree analysis. Reliabil Eng Sys Safety 72(2):213–222

5. Edwards SA (2002) High-level synthesis from the synchronous language Esterel. In: Proceedings of the international workshop on logic and synthesis (IWLS), New Orleans, June 2002
6. ESACS: Enhanced safety assessment for complex systems (2004) `http://www.cert.fr/esacs/principal.html`. Accessed 30 April
7. Esterel Technologies Web site (2004) `http://www.esterel-technologies.com`. Accessed 30 April
8. Fenelon P, McDermid JA, Nicholson M, Pumfrey DJ (1994) Towards integrated safety analysis and design. ACM SIGAPP Appl Comput Rev 1(2):21–32
9. Ghosh S (1999) Hardware description languages: concepts and principles. Wiley-IEEE Press, New York
10. Halbwachs N (1992) Synchronous programming of reactive systems. Kluwer international series in engineering and computer science, December 1992
11. Halbwachs N, Lagnier F, Raymond P (1993) Synchronous observers and the verification of reactive systems. In: Proceedings of the 3rd international conference on algebraic methodology and software technology (AMAST'93), workshops in computing. Springer, Berlin Heidelberg New York, June 1993
12. Hammarberg J (2002) High-level development and formal verification of reconfigurable hardware. Master's thesis LiTH-IDA-Ex-02/102, Linköping University, Linköping, Sweden
13. Henley EJ, Kumamoto H (1981) Reliability engineering and risk assessment. Prentice-Hall, Upper Saddle River, NJ
14. Holbrook D (2001) FPGA use for safety critical functions in an air intercept missile. In: Proceedings of the 19th international system safety conference, pp 618–628
15. Hutchings BL, Nelson BE (2000) Using general-purpose programming languages for FPGA design. In: Proceedings of the international conference on design automation. IEEE Press, New York, pp 561–566
16. INRIA TICK project Web page (2004) `http://www.inria.fr/recherche/equipes/tick.en.html`. Accessed 30 April
17. Katz RB (2000) Faster, better, cheaper space flight electronics – an analytical case study. In: Proceedings of the conference on Mil/Aero applications of programmable logic devices (MAPLD), September 2000
18. Leveson NG (2001) The role of software in recent aerospace accidents. In: Proceedings of the conference on international system safety, September 2001
19. Le Guernic P, Gautier T, Le Borgne M, Le Maire C (1991) Programming real-time applications with SIGNAL. Proc IEEE 79:1321–1336
20. Manian R, Coppit D, Sullivan KJ, Dugan JB (1999) Bridging the gap between systems and dynamic fault tree models. In: Proceedings of the annual symposium on reliability and maintainability. IEEE Press, New York, pp 105–111
21. Manna Z, Pnueli A (1992) The temporal logic of reactive and concurrent systems – specification. Springer, Berlin Heidelberg New York
22. McMillan KL (1992) Symbolic model checking – an approach to the state explosion problem. Technical Report CMU-CS-92-131, Carnegie Mellon University, Pittsburgh
23. Musa JD, Iannino A, Okumoto K (1987) Software reliability – measurement, prediction, application. McGraw-Hill, New York
24. Rauzy A (2002) Mode automata and their compilation into fault trees. Reliabil Eng Sys Safety 78(1):1–12
25. Shivakumar P, Kistler M, Keckler SW, Burger D, Alvisi L (2002) Modeling the effect of technology trends on the soft error rate of combinational logic. In: Proceedings of the international conference on dependable systems and networks, June 2002. IEEE Press, New York, pp 389–398
26. Sheeran M, Singh S, Stålmarck G (2000) Checking safety properties using induction and a SAT-solver. In: Proceedings of the international conference on formal methods in computer-aided design, November 2000
27. Sheeran M, Stålmarck G (2000) A tutorial on Stålmarck's proof procedure for propositional logic. In: Proceedings of the international conference on formal methods in computer-aided design, November 2000
28. Synplify Pro product Web page (2004) `http://www.synplicity.com/products/synplifypro`. Accessed 30 April
29. Åkerlund O, Nadjm-Tehrani S, Stålmarck G (1999) Integration of formal methods into system safety and reliability analysis. In: Proceedings of the 17th international conference on system safety, September 1999