Component-Based Safety Analysis of FPGAs

Philippa Conmy and Iain Bate, *Member, IEEE*

***Abstract*—Component-based and modular software development techniques have become established in recent years. Without complementary verification and certification methods the benefits of these development techniques are reduced. As part of certification, it is necessary to show a system is acceptably safe which subsumes both the normal and abnormal (failure) cases. However, nonfunctional properties, such as safety and failures, are abstraction breakers, cutting across multiple components. Also, much of the work on component-based engineering has been applied to software-based systems rather than field programmable gate array (FPGA)-based systems whose use is becoming more popular in industry. In this paper, we show how a modular design embedded on a FPGA can be exhaustively analyzed (from a safety perspective) to derive the failure and safety properties to give the evidence needed for a safety case. The specific challenges faced are analyzing the fault characteristics of individual electronic components, combining the results across software modules, and then feeding this into a system safety case. A secondary benefit of taking this approach is that there is less uncertainty in the performance of the device, hence, it can be used for higher integrity systems. Finally, design improvements can be specifically targeted at areas of safety concern, leading to more optimal utilization of the FPGA device.**

***Index Terms*—Component-based, field programmable gate arrays (FPGAs), safety analysis.**

I. Introduction

SAFETY critical and safety related systems typically undergo some sort of certification process prior to their deployment. The certification details are dictated by domain specific standards and guidance, and cover many dependability issues such as reliability, availability, security, and safety. There are a number of common practices and procedures within these standards. One common item is a safety case which demonstrates how potentially hazardous failures within the system are prevented or mitigated. This involves two distinct phases. First, how to construct a system to increase the likelihood that it is safe [1], and second, how to demonstrate whether the system is safe. This paper concentrates on the latter of these. Safety is a crosscutting concern and may link multiple components which are otherwise physically or logically separated (e.g., a system hazard could be caused by two physically separated subsystems misbehaving at the same time or interacting in an unanticipated way). Thus, safety analysis of an individual component must be combined with analysis of other components in order to ensure the system is acceptably safe.

Field programmable gate arrays (FPGAs) are becoming increasingly popular for use within safety critical systems. They contain thousands of logic gates and are highly complex once configured. Hence it is extremely difficult to determine the effect that a single low-level fault can have at the system level, or upon multiple functions which may be embedded on the same device. If this information cannot be provided, a conservative design approach (e.g., Triple Modular Redundancy (TMR) [2], [3]) has to be taken. This protects against single points of failure and improves reliability, but is inefficient in terms of power and weight costs. FPGAs naturally support high levels of parallelism so that multiple software components can be executed at the same time. Again, this ability is not being fully exploited in the safety critical domain due to difficulties in assessing failure and safety relationships.

In this paper, the authors propose an approach which traces very low-level FPGA faults to high-level system hazards. This is achieved by performing exhaustive bottom-up failure analysis of the FPGA's circuit to determine how potentially hazardous outputs can occur at the Input Output (IO) pins of the FPGA device. A hierarchical component-based approach is then used to manage scale. The approach is shown to be compatible with current widely used international safety standards such as DO254 [4] and IEC61508 [5]. An example mine-pump control system embedded on an FPGA is used to demonstrate the techniques.

This paper is laid out as follows. Section II describes FPGAs in more detail. Section III describes the necessary steps in certifying a device for safety, and hence the constraints within which a safety analysis technique must fall. The section also compares common techniques used for dealing with FPGA faults outside of the safety critical domain, and argues why these are not satisfactory. Section IV describes the overall approach presented in this paper. Sections V and VI describe, in turn, two processes associated with the approach, and discuss how they are used in combination. Section VII demonstrates the technique using an example. Finally, conclusions are presented.

II. Field Programmable Gate Arrays (FPGAs)

An FPGA is programmable logic device which has thousands of connected logic cells. The basic structure of these cells is identical but they can be configured to perform different operations. For example, each cell typically contains a Look-Up-Table (LUT) which can be configured to perform different binary comparisons on input signals such as AND or XOR. These cells are connected together via a series of interconnects to perform higher level processing tasks. Due to its design, an FPGA can perform many tasks in parallel, e.g., to perform digital signal processing.

Manuscript received May 21, 2009; revised September 08, 2009 and December 08, 2009; accepted December 12, 2009. First published March 01, 2010; current version published May 05, 2010. This work was supported in part by the U.K. Ministry of Defence funded Software Systems Engineering Initiative. Paper no. TII-09-05-0074.

The authors are with the Department of Computer Science, University of York, York, YO10 5DD, U.K. (e-mail: philippa.conmy@cs.york.ac.uk; iain.bate@cs.york.ac.uk).

Digital Object Identifier 10.1109/TII.2009.2039938

FPGAs can have different hardware implementations, especially for the interconnects. FPGAs with SRAM interconnects can be reconfigured as many times as the user requires. Antifuse interconnects can only be configured once, however they are smaller and offer more routing flexibility. In addition, some FPGAs include extra hardware devices such as static memory, multipliers and even traditional sequential microprocessors. This assists with tasks that the FPGA is not well suited for, e.g., dealing with floating point numbers, but can add further complexity to the certification process as each different device will need to be assessed. The approach taken in this paper is to perform a technology neutral analysis, although parts of the tools need to be altered for the particular data formats from the tools used, of a single configuration of the device.

A. Component-Based Design on FPGA

FPGAs are configured by synthesizing code written in a Hardware Description Language (HDL) such as VHDL [6] or Verilog. The first stage of synthesis converts the code into a netlist which describes the hardware parts and connections which will be used to implement the code. The netlist then undergoes a place and route operation that describes which logic cells will actually be used on the FPGA and how they are connected. The routing will be calculated based on criteria such as acceptable timing performance.

VHDL can be used to construct a modular design. It is possible to synthesize this into a modular layout on the FPGA with each module separated within the logic cells. However, the synthesis tool may optimize and integrate some modules in order to meet timing and power requirements. In addition, while the design might appear modular, if the same data needs to be used by multiple modules they may be closely physically located on the FPGA once synthesized and share a single link. In addition, if data is processed by one module prior to it being received by another, then delays are introduced across the board per clock cycle and further relationships are established. This paper does not identify different strategies for synthesis or design in order to avoid this (after all, they are a side effect of good design and necessary optimizations for performance purposes), but rather seeks to illuminate where these relationships occur, and further illuminate how these relationships can affect the overall safety of FPGA functions and its host system. Instead an assumption is made that the design has been mapped down onto a technology-specific format that is readable by both human and computer. However available tools, such as PlanAhead from Xilinx [7], provide a powerful means to influence placement and maintain componentization.

B. FPGA Faults and Dependability

FPGAs are susceptible to a number of different random hardware faults, but those which are most often cited are grouped as Single Event Effects (SEEs) [8]. An SEE causes a bit flip in the device, i.e., from 0 to 1 or *vice versa*. The most commonly discussed is a Single Event Upset (SEU) in which a flipped bit may be stuck on a certain value until the device or the cell is reset. Other SEEs are transient (reset without intervention) or permanent. Several SEEs may be found in a group together, e.g., caused by a radiation burst.

One often quoted mantra in the safety community is that *"safety is a system property"* [9], [10]. This encapsulates the idea that the safe behavior of a system may cross physically independent subsystem boundaries and forms a different view of the system. Therefore, care must be taken to ensure that any independent safety analysis of a component can be traced up to the system level [11]. One of the difficulties in analyzing an FPGA is determining the effect (if any) that an SEE has at the system level. In some instances an SEE may never cause a failure, for example if an SEE causes an incorrect input into a multi-input LUT the output may still be correct depending on the internal bit comparison tables and other input values. For example, if two values undergo an AND then the same output will be achieved for 00, 10, and 01. More detailed discussion of SEE effects can be found in [12] and [13].

On the other hand, the fault may be very significant if a flipped value is used by many other cells, effectively becoming a common mode failure across cells, and hence for multiple functions. However, it may be that even though outputs from the FPGA are affected there is actually no safety concern. Also, an SEE which affects multiple cells may be of little concern as it is easier to detect. However, an SEE which causes an output value to be subtly incorrect but credible may be of more interest. This reflects the inherent complexity of an FPGA which is not just due to the number of internal cells but also to the number of permutations of interactions between them.

In the authors' experience, and that of others [2], [14], the inability to determine the effect of an SEE has led to safety critical systems developers making pessimistic assumptions about the reliability of FPGAs. As a result, conservative (and expensive) system designs such as multiple lanes or external monitoring devices are used [2], [14]. Additionally, FPGAs are often not even considered as an option by developers of systems of a very high criticality, even when they may offer considerable benefits (e.g., due to their inherent parallelism). Therefore, one driver for this work is to help determine the safety effect of an SEU, leading to design improvements in areas of significance. This has the potential to improve both the ability to certify an FPGA-based system and also to support less conservative designs. In order to combat the problem of complexity, the approach is composable, with independent analysis of modules at both VHDL and synthesized code level. These analysis results are then combined.

One feature of this work which differs from others is that hardware faults of concern other than SEEs are considered, in particular, clock errors which can lead to synchronization, propagation, and timing issues.

C. Certification Standards and Guidance

Certification is used within this paper as a general term for the process of demonstrating that a high-integrity system performs as intended and/or required to an authority of some kind. For the safety critical systems being discussed here, safety analysis will form a large part (or in some cases all) of the certification process. Safety analysis is the term used to describe any method which shows how a failure within a system could lead to a hazardous event (i.e., one which could lead to an accident, e.g., engine fire). Here, we use the term failure to indicate some problem with the performance or output of a system,

in other words, a failure is a deviation from the desired or expected intent. This may be caused by an internal fault, or by a logic problem (bug), or due to an incorrect specification for the system. This paper mainly concentrates on the faults within an FPGA which lead to failures in board output and performance. One major issue for FPGA developers is that there is some confusion over whether an FPGA should be classified as software or hardware. On one hand, it is highly programmable or reprogrammable, but on the other hand the code is run directly in hardware. The pragmatic approach is to assume that development of the HDL code will be performed using processes similar to that for software (e.g., a V lifecycle), as well as examining the hardware using established techniques for electronics analysis. This paper adopts a dual approach. After analysis, it should be possible to show how potentially hazardous failures have been managed. There are various techniques for this, some or all of which are prescribed by standards or guidance within a domain. For example, IEC 61508 [5] is used in many different domains in multiple countries. It is a generic standard used for the development of electronics and programmable electronics. To meet the standard the developer must first assign a Safety Integrity Level (SIL) to the system or its functions based on the risks associated with it malfunctioning. Note that a function may not necessarily map to a specific component but may instead map to multiple components or multiple parts of component. Hence, the safety view of a system may be very different to an architectural view of the system and component groupings and relationships need to be considered.

SIL assignment is assisted by various safety analyses. Then, the rigor of processes applied during development is dictated by the assigned SIL and the need to demonstrate that risks are managed (e.g., in a high SIL a formal proof of code might be required but not for lower SIL). Software and hardware are treated separately in the standard. It is likely that some of the recommended software practice would need to be adapted for application to an FPGA, e.g., how to ensure code is fully tested, and how to assess the synthesis tools and their output, however, the general principles still hold.

Hardware guidance requires (amongst other things) an estimation of the failure of safety functions due to random hardware failures. This is of significance when certifying an FPGA since it is very hard to identify the effect of an SEU so, even if an estimate of the rate of occurrence is known, it cannot be linked easily to a specific safety function at a system level. This can lead to a very pessimistic failure rate assumptions when an FPGA is used. For example, in [12], Isaac suggests assuming a much higher failure rate for FPGAs than tests would indicate, purely due to uncertainty in determining failure effects.

Another applicable document is the internally mandated civil aviation guideline DO254 [4] which provides guidance for design assurance of programmable electronics in avionics safety systems. It is based on similar concepts to IEC 61508, in other words processes and design methods are dictated based on a derived level of integrity for a function. Again, the safety function view of the system may be very different to the architectural view. In addition, some of the suggested methods may be hard to apply to FPGAs. It advocates the use of very low-level elemental analysis of high integrity system parts, for example, at the individual electronic component level of an FPGA. The FPTC analysis approach presented in this paper would be suitable for this purpose.

III. Related Work

The previous section established that determining the safety effect of faults within an FPGA would be extremely helpful to support the certification process. From this discussion, two basic requirements for FPGA safety analysis have been formed.

1) Identification of safety relationships within FPGA functionality and between other system components.
2) Determination of the effect of low-level FPGA faults on those relationships.

As a backdrop to this, it is assumed that it is desirable to minimize the need for physical replication to mitigate against failures, either via board or code replication within the FPGA cells.

This section first examines typical techniques used to determine the effect or significance of SEEs outside of the safety critical domain. Then, different generic failure and safety analysis techniques are detailed and their applicability to FPGAs considered. A discussion on the choice of the demonstrated approach in this paper is also presented.

A. SEE Detection

In [3] the authors describe a method for determining the effect of SEU in a TMR system, i.e., one in which three identical lanes are used for redundancy in calculations and a monitor determines whether there is (intolerable) disagreement in their output. In their case all three lanes are on the same FPGA. They use a tool to determine if an SEU at a given location can affect more than one lane, and, if it can, they consider it significant. The advantage of their approach is that it can highlight the extent of the effect of an SEU. However, it is limited to a certain type of design solution, and, at present, does not determine whether there is an actual safety effect, rather it is assumed that any effect is undesirable. As discussed in the previous section, the main driver for the research presented in this paper is to support the safety analysis of FPGAs, but also there is a wish to avoid TMR type replication, if possible. Therefore, this technique does not meet the requirements.

A common method for detecting the effect of SEUs is to use fault injection to simulate an SEU. One example of this can be found in [13] where the authors use two FPGAs, one loaded with a correct configuration file and one with an configuration file with a simulated SEU inserted. A third FPGA is used to compare the output of the two FPGAs and note when they differ. If they do, it is assumed that the SEU is significant. Again, this technique suffers from an inability to determine whether the SEU is significant in terms of safety effect, although it could be adapted to do so. This paper does not use this type of method, as the approach presented here looks at a broader class of fault than simply SEEs. However, an advantage of the fault injection approach is that test case generation can be fully automated.

A third method to detect permanent SEEs is described in [15] by Emmert *et al.* who use "Roving STARs" (Self-Test AReas) within reconfigurable FPGAs. They reconfigure the FPGA cells around any permanent SEEs detected and, by

moving the STARs around during operation, they can cover the entire FPGA. The disadvantages of such an approach (apart from the fact that only permanent SEEs are detected) is that it requires the FPGA to be reconfigured around the test areas during operation, which can have an effect on system timings and cause output to be interrupted. As safety critical systems typically have hard real-time deadlines, a reconfigurable approach may introduce intolerable or unpredictable execution times. In addition, the effect of an SEU occurring in a non test area would be different for each configuration. Hence, safety analysis would need to consider all potential configurations. The issue of certification of a system using reconfiguration is not addressed in this paper, instead this work is seen as a natural precursor to reconfiguration as the first necessary step is to give evidence that a single configuration is safe. A discussion of one possible approach can be found in [14]. It is of note that a system which can reconfigure around faults is potentially more reliable and, hence, can offer safety benefits compared to a static one. However, this raises other issues in terms of justifying the integrity of the reconfiguration method. In [16], the authors discuss different SEEs and describe their generic effects upon the operation of an FPGA. However, the authors do not attempt to discern individual effects of an SEE. Instead the use of TMR is advocated and partial scrubbing (reloading of part the configuration file) to mitigate against any effects. As discussed, reconfiguration (even only scrubbing), would interrupt the operation of the device which may have timing effects. Also avoiding the overuse of replication, e.g., TMR, is one of our aims.

The authors of [17] demonstrate how VHDL for a safety-critical avionics systems can be auto-generated from an Esterel model. As part of their work they look at fault modelling. They do this at a coarse grained level, viewing all faults at the edges of an FPGA only. This means they cannot effectively determine the risk of any of the faults on the output actually being caused by internal faults. As a result, pessimistic assumptions would have to be made as to the cause and effect of internal faults.

Different low-level FPGA faults are described in [18]. They perform fault emulation, by injecting the different fault types into an FPGA. Traces of execution were then examined and outputs classed as failure (output differed to that expected), latent (failure remaining in the system but not manifesting straight away), and silent (no change in expected output). Again, there is no attempt to differentiate between failures with a safety effect, and those with incorrect output with no safety effect.

B. Failure and Safety Analysis

There are numerous different safety analysis techniques which can be applied at different times during a systems development lifecycle. A good overview of these can be found in [9]. At early stages of development these help provide insight into whether a design needs to be adapted to deal with identified failures and at later stages of development the results of the analysis can be used as evidence to demonstrate failures are adequately managed. Techniques can be top-down, i.e., starting with a hazardous system level event then working down to individual components, to see how they could contribute to its occurrence. A typical top-down technique is Fault Tree Analysis which shows how component failures combine (and their probabilities) to cause a single event. Other techniques are bottom-up, i.e., they show how individual component failures can contribute to one or more hazardous events at the system level. A typical technique would be Failure Modes and Effects Analysis (FMEA) which looks at individual failures and considers their effects. FMEA can be supported using different guide words which suggest different failure types, e.g., late arrival, data omission, or value error for computer data analysis. FMEAs can be applied at a very low-level or at a higher architectural level to help assess preliminary design and requirements generation. Top-down analysis, by its very nature, includes significant manual effort as only a human can judge the severity of failures for a given operational context of a system.

A bottom-up technique would best support the second requirement to determine the overall effect that a single SEE, or low-level failure, can have, i.e., how it could fan out to multiple system level events. However, FMEA-based techniques are generally manually applied, and an FPGA can have thousands of different connections and routes to consider. Hence, it is impractical to use manual analysis. Various works [19]–[21] have looked at how FMEAs can be automated, however none of these have been demonstrated with a low-level starting point of a FPGA's circuit design. Also, as noted by the authors of [22], the majority of automated fault analysis techniques do not support feedback loops between components, hence are unsuitable for use modelling FPGAs. Two potentially useful techniques are described in [23] and [24]. The first of these two approaches considers hardware at an abstract level and does not describe specifically how to link an FPGA design to the FMEA analysis presented. The second of these is tied to a specific language, Esterel, which has formally defined syntax and semantics. However, Esterel is not universally adopted in critical systems.

Another two techniques, which support cyclic loops, are the Architecture Analysis and Description Language (AADL) [22] and the Fault Propagation and Transformation Calculus (FPTC). AADL allows rich modelling of fault characteristics including state transitions, property sets, port maps and events. However, the low-level components of an FPGA do not require this depth of description, and the models would be over complex to generate with no additional benefit from the features. Hence, FPTC was used for this case study as it supports the expressive requirements for an FPGA, while still being relatively simple to generate models.

FPTC [25] is a bottom-up safety analysis approach in which individual components can be analyzed independently to identify their fault characteristics, and then a tool can be used to calculate the actual faults which could propagate through a network of these components. An analyst can then look at the concatenated results in order to determine overall safety effects. This technique has the potential to assist in the assessment of the failure effect of faults in FPGAs. FPGAs are constructed of the same types of hardware components, replicated many times. Hence, the faults of each component type can be analyzed individually and the results reused. Then, a network can be constructed to represent how they are connected together on the

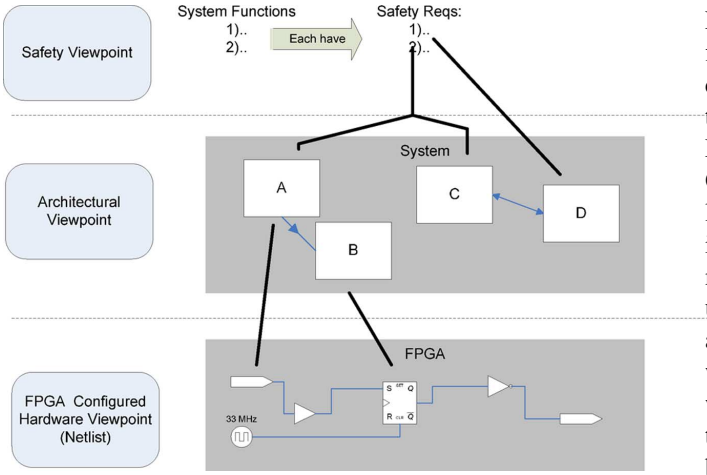

Fig. 1. System and components viewpoints.

FPGA (based on the synthesis outputs) and assess how an individual SEE or clock drift would propagate through the system outputs. However, some automation to help generate FPTC networks and to guide the analyst may be required for a device as complex as an FPGA, due to the sheer number of internal components. FPTC has also mainly been applied at a higher level for analysis in previous case studies (e.g., software process level [25]), so it is efficacy for such a low-level analysis needs to be assessed. Industrial partners, including both engineers and system safety analysts, have showed considerable interest in the FPTC approach, indicating that there is a need for this level of analysis.

A top-down technique would best support the first requirement to establish safety relationships between different components. There are many existing techniques which support this down, breaking down the system into relatively large components, e.g., subsystems or software modules. However, tracing these relationships down to the depth of individual FPGA gates would be a lengthy complex process, closely coupled with the final FPGA configuration. Since the top-down analyses are usually performed during the initial design stages to drive the development process, this would be inappropriate. Therefore, this paper proposes an approach which takes full advantage of both top-down and low-level bottom-up analyses, meeting halfway. Furthermore it is usable in a component-based way in order to support scalability and incremental certification.

The evidence generated by applying the technique can be used to strengthen the safety case. It would be used alongside other evidence, for example, that the synthesis tools are trusted and that functional requirements are valid.

IV. Proposed Approach

In order to support the mixed approach, this paper considers the system from three different viewpoints, with well defined relationships between each level, as shown in Fig. 1. The top-level is the safety viewpoint, at which a set of system functions and their related safety requirements are defined. The second-layer is the architectural design level, at which architectural components and complete functional blocks (such as VHDL or software modules or subsystems) are defined. For the purposes of this paper, the bottom-level is the much lower level electronic FPGA component level which includes items such as I/O pins, flip-flop gates and clocks. Obviously, other types of components could also be examined at the lowest level, but this is outside the scope of this paper. To a certain extent, this is similar to the Model Driven Architecture (MDA) approach described by the Object Management Group (OMG) [26] which uses different levels of hardware abstraction. However, the purpose of MDA is to automatically generate code from a series of models and mappings–always moving downwards, whereas this approach uses only partially automated technique at the lowest level. In addition, alterations and updates can occur upwards and downwards between layers, whereas MDA techniques move downwards only. Also, generation of the FPGA configuration uses traditional VHDL design and synthesis tools rather than UML-based models and transformation languages.

The thicker lines in Fig. 1 represent the relationships between the entities at one level to another. For example, one safety requirement is mapped onto components A and C at the architectural level, and another maps only onto D. These relationships are identified using top-down analysis.

Between the FPGA level and the architectural level, the lines indicate which individual electronic components perform the functionality described in VHDL modules. Bottom-up FPTC analysis identifies how architectural components are affected by faults in hardware components.

Also shown are the functional relationships within layers, for example, A sends data to B, and C shares data with D at the architectural level. It would appear that these are potentially areas where further safety coupling between components exists, for example if safe operation of C can be affected by the data D sends. This will be determined by failure analyses within each level, compared with the top-down safety requirements. Note that a failure condition does not necessarily have a safety effect.

V. Top-Down Analysis—Requirement 1

As discussed, in previous sections, there are a number of existing (informal and manual) safety analyses which can be used to identify safety requirements and relationships. As these are well established this paper does not offer any new techniques. Using those found in [9] and [27], the following steps are followed for top-level analysis. These steps are largely manual but no more labor intensive than for systems where FPTC analysis is not used.

- Identification of hazards associated with the system via:
  - Prior knowledge and engineering judgment.
  - FMEA or FTA type analyses that only look at architectural components.
- Determination of methods for dealing with these hazards leading to:
  - Allocation of methods/requirements to system components.
  - Alteration of system design to avoid hazards.

If system design alteration is required then the analysis should be started again as there is a possibility that new hazards have been introduced. In fact, as with aspects of software engineering, this can be an iterative process with a number of repeats.

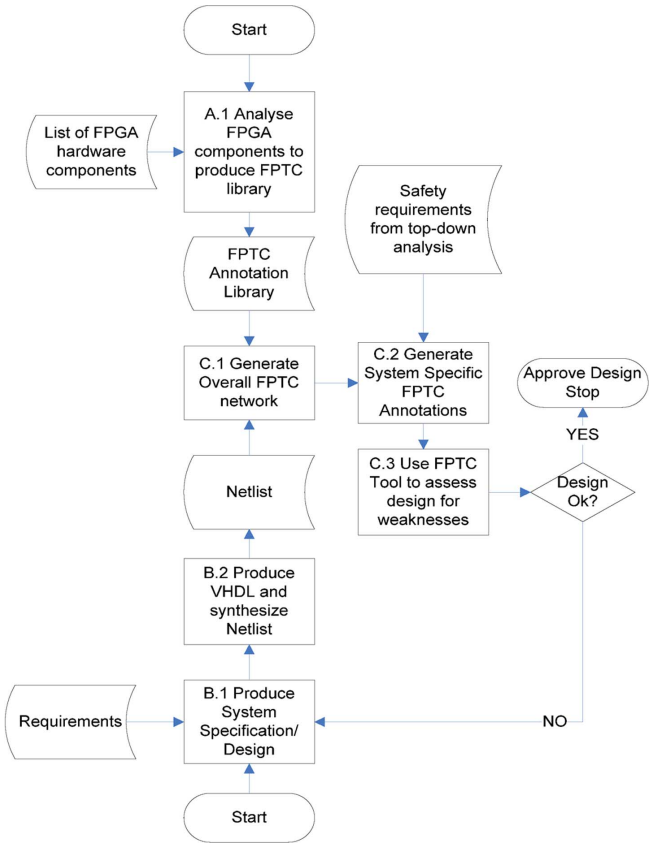

Fig. 2. Summary of bottom-up FPGA analysis process.

VI. Bottom-Up Analysis—Requirement 2

This section discusses the low-level bottom-up analyses designed to satisfy the second analysis requirement. Based on the previous discussion, our aims in this analysis are to provide evidence about the effects of low-level failures on system level events and improve the design, and also to enhance confidence in an FPGAs ability to perform in more critical systems, thus supporting more efficient designs. A summary of the overall FPTC process is shown in Fig. 2. It comprises of three parts: Generic reusable FPGA component analysis (A.1), generation of FPGA hardware design (B.1–B.2), and system specific safety analysis (C.1–C.3). These are now described in turn. All the stages in the approach can be automated (given the results from the system hazard analysis) except for the "Design ok?" and "Generate System Specific FPTC Annotations" steps. It is necessary that these steps are partly manual as currently the decision as to whether the system is sufficiently safe is a subjective activity using concepts such as ALARP [5].

However, the other parts can still be automated without compromising safety as long as the manual checks on outputs are in place. It is noted that for the approach to be used in real systems, any tools providing automation would need to be qualified. As the FPTC analysis itself cannot directly introduce errors into the system, the tools would be classed as verification tools, and could be qualified as such using the criteria found in guidance such as DO178B [28].

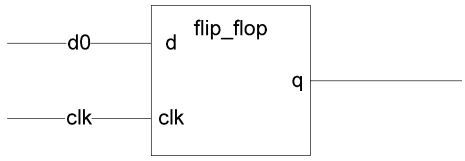

Fig. 3. Flip-flop component.

TABLE I
Flip-Flop Input Fault

Input/Output	Fault	Comments
clk	Early/late	The clock may be running correctly or may be early or late due to drift
d0	Value	The value of *d0* may be correct or may be false high/low or undetermined. We express this as a single fault of "value"

A. Part A: FPTC Component Analysis

This section describes the FPGA component analysis process (A.1) used to generate a reusable library of FPTC annotations. An FPGA is made of numerous different logic cells. These cells typically contain components such as LUTs, flip-flop circuits, multiplexors, inputs, outputs, and clock signals [13]. As we are using the Xilinx ISE, we are basing our analysis on their lists of generic design elements for a typical recent Xilinx FPGA as described on their website [29]. It is worth noting the process and analysis are not tied to these tools and could easily be ported to others. Each component can have some kind of internal fault, for example, a physical degradation of materials meaning a bit is stuck at a particular value. The fault causes a component to produce an incorrect response which is summarized as to its type, e.g., early, late, data omission, incorrect value.

Fig. 3 shows a flip-flop component of the type typically found in an FPGA. This component copies the value of d0 to its output q on the low to high clock (clk) transition. In order to produce an FPTC annotation, an analyst needs to consider the possible faults which could be provided on input and how they would affect the component. The possible input faults to the flip-flop are shown in Table I. They have been derived by considering the internal faults of the input components and categorizing them. The analyst must also consider any internal faults which could be introduced by the component itself. Finally, a set of output fault categories is produced based how both sets might be transformed or manifest themselves. The output faults for the flip-flop are summarized in Table II.

The results are expressed in the FPTC notation as shown in Fig. 4 (see [25] for full syntax and semantics). The left-hand side of an FPTC annotation indicates the values which are received on input, and the right hand side (after the arrow) indicates the corresponding outputs. Each separate input and output set is contained in curled braces, faults are labelled as "fault x". The asterisk "*" symbol is used to indicate that no fault is received on input, and the underscore "_" is used to indicate that any value can be received on a given input. If multiple input faults lead to a single output type of fault (as is true in this case

TABLE II
FLIP-FLOP OUTPUT FAULTS

Input Fault	Output Fault	Comment
clk Early	stale_value	If the clock triggers the flip-flop to copy the value of *d0* early then *d0* may not contain the most recent value, hence stale_value is produced
clk Late	stale_value	If the clock triggers a late copy of d0 to q then it is possible the value of q has been read already hence the output fault is again stale_value
d0 Value	Value	If the value of *d0* is incorrect then the flip-flop will simply pass this on
None	Value	It is possible that the flip-flop may have been affected by an SEE hence it could be the source of a value fault

```
({*}, {fault early, fault late})->({fault stale_value})
()->({fault value})
({fault value},{*})->({fault value})
```

Fig. 4. FPTC annotations for flip-flop component.

for both early and late faults), then these can be concatenated together.

FPTC components are linked together to form a network using the Eclipse-based analysis tool described in [30]. The network is an acyclic graph that represents hierarchies of components starting from the bottom with a single FPGA component (e.g., flip-flop) to a VHDL module and then to the whole program. The stages between the VHDL module and the whole program are defined by the designer of the VHDL. In practice, when using the technique the number of low-level components produced by the synthesizer for each module varies depending on the nature of that code, but typically 50 lines of code could be represented by about 15–20 actual components. The synthesizer will typically map a VHDL module onto the FPGA components. The tool can be used to either generate the entire set of potential faults throughout the network, or to analyze the propagation of an individual injected fault anywhere in the network. Essentially, the tool attempts to match inputs to a component with the left-hand side patterns, as provided in the annotations. Every time there is a match, the right-hand side is provided as input to next component it is connected to. An example of this is shown in Fig. 5. In this example, a clock is the source of a *late* fault (shown on the output arrow) and another input (*Input1*) is correct. These two inputs match with the first annotation on *FLIP_FLOP* but not the second, and hence its output is *stale_value* only.

Note that the fault tokens are entirely generated by the analyst and not dictated by the notation or the tool. While this gives great flexibility, it is up to the analyst to ensure that token names are consistently spelled and that there are appropriate matches between components.

Using the method laid out in this section, we have analyzed a number of common FPGA hardware components. These results are used in the next analysis stage described in Section VI-C.

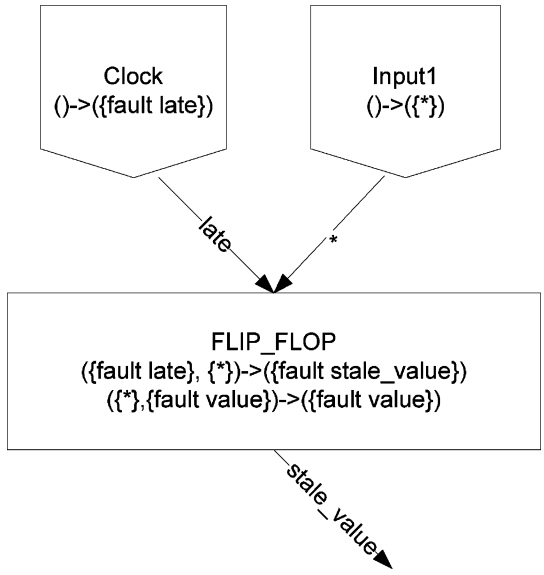

Fig. 5. FPTC matching example.

B. Part B: FPGA Design

This section describes the processes corresponding to B.1 and B.2 in Fig. 2. The aim of this work is to investigate the potential safety effects of hardware faults within an FPGA. Therefore, a hardware representation of the intended design is required. To this end, VHDL descriptions of potential designs are produced based on system requirements, and then synthesized currently using Xilinx ISE Project Designer. However, the method is independent of specific tools. Currently, netlist files are used as this makes producing tools easier and the results are more straightforward to validate. Taking this approach is not uncommon, e.g., structural testing of software is often done at the source code level. In practice, the approach can easily be ported to a lower level or limited low-level validation performed to ensure problems have not arisen in the later parts of synthesis, e.g., due to optimizations. The different areas of concern in terms of safety effect are the identification of systematic design flaws, i.e., flaws in the program logic and also analysis of the effect of random hardware faults upon the design.

In both cases, the ideal situation is to improve the design so that a safety weakness is no longer included. However, if that is not possible, then an attempt will be made to alter the program design to mitigate against unacceptable faults. Also, it is the intention that results of the analysis (once changes are made) can be used as evidence to support a system safety case. The key point is that modifications can be targeted better, e.g., adding limited replication, rather than coarse grained decisions such as replicating the whole design.

C. Part C: FPTC Analysis of FPGA Design

This section describes the processes corresponding to C.1–C.3 in Fig. 2.

Once a netlist has been generated using the ISE tool, it is converted to a network of FPTC components instances and their

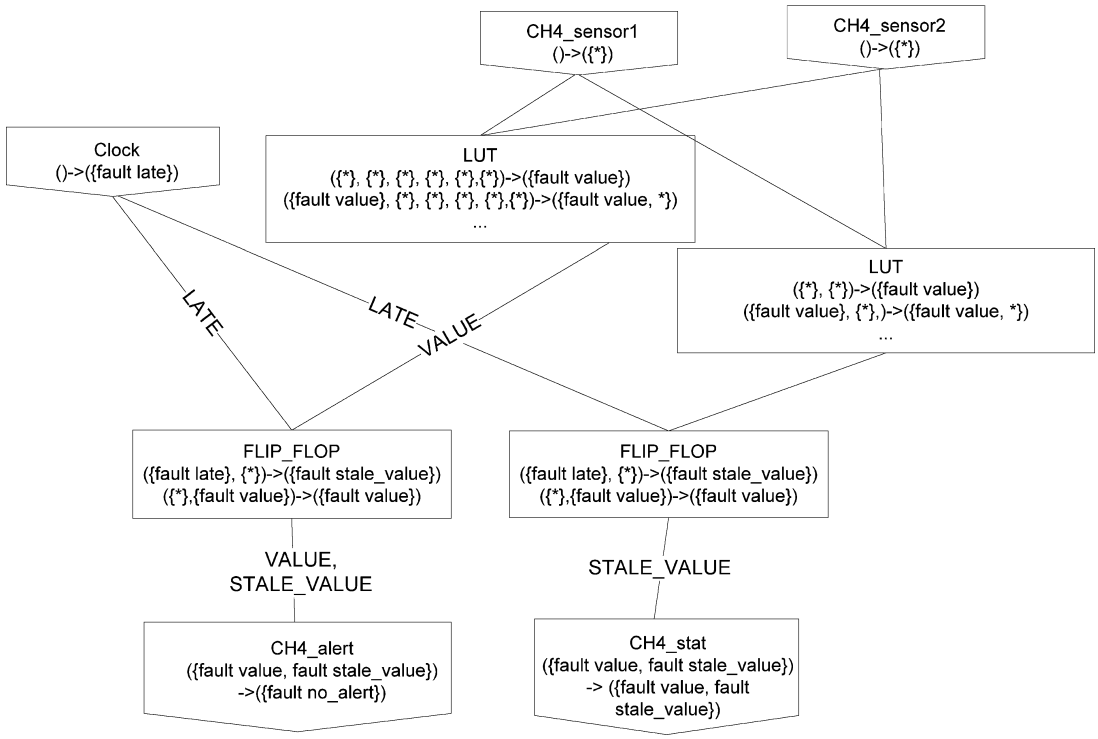

Fig. 6. CH4 sensor processing module FPTC analysis.

outputs within the Eclipse tool. This process is currently performed manually but could in the future be automated using a MDA merging program [26] to link the FPTC annotations to the netlist. This corresponds to C.1 in Fig. 2. Fig. 6 shows an example.

The ISE tool has a compilation option which can be used to ensure that the VHDL component hierarchy is maintained. This means that each module of VHDL corresponds to a specific area of the netlist. Hence, each module can be examined as a separate FPTC network. This has two advantages. First, it improves the scalability of the approach as only a section of the system needs to be examined at once. Second, when an alteration is made in one VHDL module then it should have no impact on any of the other synthesized modules. To date, no case has been found where this didn't hold true during experiments. This means that the only FPTC network which needs to be regenerated and examined is that relating to the module being changed.

C.2 involves the production of system specific fault annotations on the output pins and some internal pins on the edge of components. The purpose here is to link faults to specific system level events which may or may not be hazardous, thus linking with the top-down analysis. This allows the generic internal hardware fault types to be traced to one or more system level events, consequently supporting the aim of identifying how internal failures cause system level hazards. This is described in more detail in Section VII using a case study for illustration.

VII. Case Study: Mine Pump Control System

In order to test the proposed approach, the mine pump control system described in [31] was used as a case study. The main functions of the control system are to ensure that water percolating into the mine is kept to a safe level, and to ensure that the air quality is sufficiently safe. A water pump is automatically deployed when water rises above a certain level, but it can also be controlled by a human operator if they are worried about a rapid rise. In the event of either a pump failure or inadequate air quality the mine needs to be evacuated. In order to achieve this, the system has sensors to detect the following: Carbon Monoxide levels, Methane Levels, Sufficient Air Flow, High Water Mark, Low Water Mark. An evacuation alarm is raised if any of the sensors detect gas at intolerable levels. The pump must not be used if methane levels are above a certain amount as there is a risk of explosion.

The proposed design uses seven VHDL modules, synthesized for an FPGA, in order to mange the sensors and pump. These are the systems architectural components in our viewpoints. Three individual modules are used to monitor each different air sensor, and a single module monitors both water sensors for validity of data. An operator panel module collates information and controls a series of indicators. These are viewed by an operator so that they can assess whether an evacuation is necessary. An automatic pump control module activates the pump if the high water mark is breached and switches off when the low water mark is reached (if methane levels are acceptable). In addition, there is a data logging module and an overall control module.

The designers have been asked to see if the amount of equipment can be minimized (to lower power requirements and costs). Therefore, the safety analysis will explore whether the safety of the system can be maintained with a single board running all modules.

TABLE III
TOP-DOWN SAFETY ANALYSIS

Hazard	Safety Requirement	Safety Relationships
1	Air flow is measured to ensure fresh air supply	None
	Carbon monoxide levels are measured to ensure they are within safe limits	None
	Methane levels are measured to ensure they are within safe limits	None
2	Pump is automatically activated when water levels breach acceptable high water mark	None
	Pump can be activated by operator if water flow judged to be unacceptable	Between pump controller and operator panel
3	Pump is not activated when methane levels beyond limit X	Between methane sensor monitor and pump controller

A. Top-Down Analysis

This section shows how the safety relationships between the modules can be identified. First, three hazards are associated with the system.

1) Poor air quality for mining staff.
2) Flooding with associated danger of drowning.
3) Methane ignition caused by pump activation.

Hazards 1 and 2 are derived via engineering knowledge, whereas hazard 3 is derived with an understanding of the actual system components via an architectural FMEA. Table III shows the safety requirements (not other functional requirements) and relationships derived to manage these hazards.

Note that there are other relationships and dependencies between components, such as data sharing between the operator panel and sensor monitors, but these relationships are not directly safety critical. There is a need for accurate and reliable data or an accurate assessment of data validity of course, but this is mitigated by operator intervention as discussed in the results section.

B. Bottom-Up Analysis of Mine Pump

In order to generate an FPTC network for analysis, the seven VHDL modules described at the beginning of this section have been coded and converted into a netlist. The resultant network included six modules, each with multiple LUTs, flip-flops, and inverters, a clock and the required input/output pins to carry data. These are contained within an overarching network, corresponding to the overarching VHDL Module. The components could be categorized into just five different types with the same basic failure patterns, with differing numbers of inputs and outputs. The VHDL modules were synthesized using the "keep hierarchy" option so that each module could be turned into a separate FPTC network and examined individually as described in the next section.

C. Results

The outputs of each module's network were examined manually one by one, in order to identify whether the fault categories listed could contribute to any of the high-level hazards and break individual safety requirements. If they could then they were annotated with an additional fault describing this scenario. For example, the water flow indicator value from the *water_sensor_management* module was annotated with outputs of "fault value, fault stale value" to indicate the generic fault categories which could be introduced. An additional output of "fault unreliable_flow_indicator" was added in order to indicate the specific effect which could contribute to a failure to activate the pump. This allows the safety analyst to trace back possible paths within the module network which could lead to this potential safety effect and consider the risk associated with it. Note that as there is a safety relationship between this module and the operator panel they must consider this within their risk assessment. From the assessment, they can then identify potential areas where the design needs to be strengthened, or isolated within that module. For example, any single component that could introduce an unsafe output can be replicated at the netlist level, via alterations to the HDL. This may be achieved using additional comparisons, translating to the use of multiple LUTs in the netlist to compare data. This helps protect against single points of failure, thus ensuring that our aim of using a single board can be supported. This type of design response could be viewed as similar to crude TMR, but it is targeted towards an actual known safety issue in a specific part of the design, rather than being applied across the whole design.

The key is safety insight is provided to the developer to aid their decision process. When the design has been finalized the annotated networks can be used as additional evidence to support the system safety case and show that contributions to hazards have been considered and managed.

The water flow indicator value is an input into the operator panel module. The specific bespoke annotation about unreliable data is not added to the FPTC network for the operator panel, however. There are two reasons for this. First, injecting the value/stale_value fault types is adequate for considering the types of problems which could propagate within an individual network and cause issues on the outputs. Second, the ability to create the FPTC networks automatically will be lost if bespoke fault types need to be considered on inputs as well as outputs. The analyst of this module will also need to consider a safety relationship, for example if one of the inputs is from an untrusted module they may consider that the risk of that input being incorrect is higher than one from a trusted module. Again, if some internal protection against that can be made then they need to consider it as part of their own design strengthening strategy. In other words some knowledge is required of the overall system design, but the modules internal faults can still be examined in isolation.

Fig. 6 shows an FPTC network for the methane sensor module. This small module example is representative of some of the larger networks examined in the case study. There are two sensor value inputs (from two different sensors replicated to mitigate against a single point of failure) and also the clock

input. In the figure, the clock has a late failure, and a fault in one LUT has introduced a value fault. The two LUTs have a different number of inputs, but their failure behavior is essentially the same. They can introduce a fault, pass on a fault, or absorb a fault. The full set of LUT FPTC annotations is not shown in order to simplify the diagram, but involves permutations of those shown.

There are also two clocked flip-flops which are affected by the late fault on the clock, thus introducing a stale_value fault. In addition, the value fault introduced via the LUT is also passed on. These can cause faults on the two outputs. The *CH4_alert* output has the failure of no_alert for both stale_value and value. This failure is hazardous since a false negative on the alert could mean that personnel are exposed to dangerous levels of methane. A false positive is not listed as a possible failure since it is not hazardous, and so is out of scope. It may, of course, be costly if an unneeded evacuation is ordered. Note that an additional safety relationship was identified between the CH4 monitor and the water pump manager module. In fact, the *CH4_alert* value is passed onto the pump manager and so if a false negative is passed on then the hazard of methane ignition remains as expected. In order to counter this, an independent check of the CH4 sensor data validity has been introduced into the operator panel module, rather than via the *CH4_alert* value. In other words, some targeted redundancy has been introduced. This has a couple of effects. First, if the pump is still activated when the gas alert is displayed on the operator panel, then the operator can override and shut down the pump. Thus, the hazard is mitigated. However, this breaks the modularity of the design and introduces a new implicit safety relationship, whereby the operator panel should not be altered to remove this functionality and just use the *CH4_alert* as the system then becomes less safe. Thus, our list of safety relationships needs to be updated.

The *CH4_stat* output is used as an input to the *operator_panel* module where it is used to determine whether certain warning lights should be displayed to the operator. There are a number of failures uncovered by the FPTC analysis which could mean the air monitor status potentially contradicts with the alert status (e.g., an alert is shown but the sensor data is shown to be faulty). In this situation, the operator can perform further investigations and take action (e.g., order precautionary evacuation) if necessary. In other words, although there was a safety effect it was not of high risk (safety related rather than safety critical). Hence, no design alterations were necessary. One other important point, is that the signals were passed around the network via each clock tick, and, due to the parallel execution performed on the FPGA, some data shared between modules could be slightly out of date. This is not an issue as long as the timing constraints can be met. For example, the *CH4_sensor* module output data is not processed by the pump controller or operator panel until the next clock tick. However, if the total length of processing time is within the timing constraints then there is no timing concern. An examination of the synthesis reports generated indicated that the initial timing requirements described in [31] could be met.

The VHDL source for the case study consisted of 320 lines of useful code (ignoring comments and dead space). The output netlists contained 97 components in total, reduced to 65 by combining IO buffers and ports as single items. Converting these into FPTC format took an afternoons manual effort, obviously this would be considerably reduced with automation. Running the FPTC tool took at most a couple of seconds for each netlist. Manual analysis of the outputs was again approximately one afternoons work. It should be noted that undertaking a manual failure analysis, such as an FMEA, is an extremely time consuming process. In fact it can take so long it exceeds design time [20]. Therefore, even with only partial automation there are significant potential benefits by being able to feed the results into the design process.

VIII. Conclusion

This paper has demonstrated how analysis of individual faults within electronic components, which are supporting a modularized design embedded on an FPGA, can be linked with cross-cutting safety analysis in order to enhance and verify the safety properties required. The paper used the semi-automated FPTC analysis technique tailored to individual fault types found on an FPGA. The analysis exposed a number of potential issues, and the mine pump system design was given minor alterations to mitigate against those with a safety effect. Thus, we were able to strengthen the design in areas of concern, without having to resort to crude methods of TMR due to pessimism about the FPGAs failure characteristics. Therefore, the aim of supporting less conservative designs is supported.

In order to develop the technique, the following areas will be examined. First, the possibility of formalizing the safety relationships as contracts will be examined as this will allow more rigorous mathematical analysis to determine whether properties are broken. Second, conversion of netlists to FPTC networks will be partially automated in order to improve the scalability of the technique. Although high integrity functionality is often kept relatively simple in order to aid analysis, a reasonably complex netlist may still be generated. This could lead to mistakes being made during a manual conversion. In addition, the FPTC network may be difficult for the analyst to assess. Therefore, the FPTC tool is being extended to highlight certain fault propagation paths for the analyst, and to check automatically for common failure patterns. Also, further work will examine whether the FPTC only needs to be targeted at certain areas of code that are considered the most critical.

Finally, there is potential to extend the FPTC technique to include information such as probabilities of faults [32], in order to further support a system safety case.

References

[1] J. Hilton and G. Hall, "Developing critical systems with PLD components," *Proc. Foundations Softw. Eng.*, pp. 72–79, 2005.

[2] F. L. Kanstensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," *IEEE Design, Autom., Test in Europe DATE*, pp. 1290–1295, 2005.

[3] L. Sterpone and M. Violante, "A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs," *IEEE Trans. Nuclear Sci.*, vol. 52, no. 6, pp. 2217–2223, 2005.

[4] RTCA/EUROCAE, "*Design assurance guidance for airborne electronic hardware*, DO-254/ED-80," 2000.

[5] IEC, "Functional safety of electrical/electronic/programmable electronic safety-related systems (IEC 61508)," 2001.

[6] Design automation standards subcommittee of the design automation technical committee of the computer society of the IEEE and automatic test program generation subcommittee of the IEEE standards coordinating committee 20. *IEEE Standard VHDL Language Reference Manual,* 1988.
[7] Xilinx: Design Tools May 2009. [Online]. Available: http://www.xilinx.com/tools/designtools.htm, Xilinx
[8] E. Normand, "Single event effects in avionics," *IEEE Trans. Nuclear Sci.*, vol. 43, no. 2, pp. 461–474, 1996.
[9] N. G. Leveson, *Safeware*. Reading, MA: Addison-Wesley, 1995.
[10] N. Audsley, P. Conmy, S. Crook-Dawkins, and R. Hawkins, "Safety challenges for model driven development," in *Proc. Int. Workshop on Metamodelling for MDA*, York, U.K., 2003.
[11] P. Conmy and I. Bate, "Safe composition of real time software," in *Proc. 9th IEEE High Assurance Syst. Eng. Conf.*, 2005, pp. 79–88.
[12] T. A. Isaac, "Firmware in safety critical subsystems," in *Proc. Int. Syst. Safety Conf.*, 2004, pp. 469–478.
[13] P. Graham, M. Caffrey, J. Zimmerman, and D. E. Johnson, "Consequences and categories of SRAM FPGA configuration SEUs," in *Proc. Military and Aerosp. Programmable Logic Devices Int. Conf.*, 2003, pp. 1–9.
[14] B. Rousseau, P. Manet, D. Galerin, D. Merkenbreack, J. D. Legat, F. Dedeken, and Y. Gabriel, "Enabling certification for dynamic partial reconfiguration using a minimal flow," *Design, Autom. Test in Europe*, pp. 983–988, 2007.
[15] J. M. Emmert, C. E. Stroud, B. Skaggs, and M. Abramovici, "Dynamic fault tolerance in FPGAs via partial reconfiguration," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2000, pp. 165–174.
[16] D. Weigand and M. Harlacher, "A radiation-tolerant low-power transceiver design for reconfigurable communications and navigation applications," in *Proc. Earth Sci. Technol. Conf.*, Pasadena, CA, 2002, pp. 6–0.
[17] J. Hammarberg and S. Nadjm-Tehrani, "Development of safety-critical reconfigurable hardware with esterel," *Electronic Notes in Theoretical Computer Science*, vol. 80, pp. 219–234, 2003.
[18] D. de Andres, J. C. Ruiz, D. Gil, and P. Gil, "Fault emulation for dependability evaluation of VLSI systems," *IEEE Trans. Very Large Scale Integration*, vol. 16, no. 4, pp. 422–431, Apr. 2008, 2008.
[19] L. Grunske, P. A. Lindsay, N. Yatapanage, and K. Winter, "An automated failure mode and effect analysis based on high-level design specification with behavior trees," *Integrated Formal Methods, Lecture Notes in Computer Science*, vol. 3771, pp. 129–149, 2005.
[20] Y. Papadopoulos, D. Parker, and C. Grante, "Automating the failure modes and effects analysis of safety critical systems," in *Proc. 8th IEEE Symp. High Assurance Syst. Eng.*, 2004, pp. 310–311.
[21] J. Elmqvist and S. Nadjm-Tehrani, "Tool support for incremental failure mode and effects analysis of component-based systems," in *Proc. Design, Autom., Test in Europe*, 2008, pp. 921–927.
[22] L. Grunske and J. Han, "A comparitive study into architecture-based safety evaluation methodologies using AADL's error annex and failure propagation models," in *Proc. 11th IEEE High Assurance Syst. Eng. Symp.*, Nanjing, China, 2008, pp. 283–292.
[23] Y. Papadopoulos, J. A. McDermid, R. Sasse, and G. Heiner, "Analysis and synthesis of the behavior of complex programmable electronic systems in conditions of failure," *Reliability Eng. Syst. Safety*, vol. 71, no. 3, pp. 229–247, 2001.
[24] J. Hammarberg and S. Nadjm-Tehrani, "Formal verification of fault tolerance in safety-critical reconfigurable modules," *Software Tools for Technology Transfer*, vol. 7, no. 3, pp. 268–279, 2005.
[25] M. Wallace, "Modular architectural representation and analysis of fault propagation and transformation," in *Proc. 2nd Int. Workshop on Formal Foundations of Embedded Software and Component-Based Software Architectures*, 2005, pp. 53–71.
[26] Object Management Group (OMG), *MDA Guide*, ver. 1.1, 2003.
[27] N. Storey, *Safety-Critical Computer Systems*, 1st ed. Reading, MA: Addison-Wesley, 1996.
[28] RTCA/EUROCAE, *"Software considerations in airborne systems and equipment certification,"*, DO-178B/ED-12B, 1992.
[29] Xilinx, Xilinx: Design Elements, 2008. [Online]. Available: http://toolbox.xilinx.com/docsan/xilinx5/data/docs/lib/lib0050_34.html
[30] R. F. Paige, L. M. Rose, X. Ge, D. S. Kolovos, and P. J. Brooke, "Automated safety analysis for domain-specific languages," in *Proc. Workshop on Non-Functional System Properties in Domain Specific Modeling Languages*, 2008.
[31] M. Joseph, "Real-time systems specification, verification and analysis, 2001. [Online]. Available: http://www.ebook.ieeelab.com/Embedded/RTSbook.pdf
[32] X. Ge, R. Paige, and J. A. McDermid, "Automated and probabilistic failure analysis," in *Proc. SAFECOMP*, 2009, pp. 215–228.

Philippa Conmy received the Ph.D. degree in computer science from the University of York, York, U.K., in 2006.

She is a Research Associate within the High Integrity Systems Engineering Group at the University of York. Her research interests include all aspects of safety critical software design, with a particular focus on field programmable gate arrays, distributed systems and model driven development.

Dr. Conmy is a member of the British Computer Society.

Iain Bate (M'00) is a Lecturer in real-time systems. His research includes scheduling and timing analysis, design and analysis of safety-critical systems, and engineering of complex systems of systems including sensornets.

Dr. Bate is Editor-in-Chief of the *Journal of Systems Architecture* and a frequent member of program committees for distinguished international conferences.