# Xilinx Tools facilitate development of FPGA Applications for IEC61508

Giulio Corradi
Xilinx
ISM (Industrial Scientific Medical)
Munich, Germany
giulio.corradi@xilinx.com

Romuald Girardey, Jürgen Becker
Institut für Technik der Informationsverarbeitung – ITIV
Karlsruhe Institute for Technology (KIT)
Karlsruhe, Germany
{girardey, becker}@itiv.uni-karlsruhe.de

*Abstract*— **In industrial automation and electrical motors drives, functional safety applications are growing steadily driven by market demand and by the adoption of the IEC61508 Edition 2 [3]. This second edition specifically includes directives on functional safety designs for FPGAs. Compliance with IEC61508 directives in general may prove very challenging even for the most seasoned practitioners. The functional safety management process directives in the norm ranks the development tools in categories T1, T2 and T3 in function of the specific impact on the final FPGA design, the higher the number more critical the impact. T1(data entry), T2(verification) and T3(transformation) type of tools. Design preservation, isolation design flow, the hierarchical design and associated floor-planning supplement the design flow to achieve added confidence on FPGA designs for functional safety. This paper address the isolation design flow as T2 tool aiming to decreasing the Beta factor βIC as required by the IEC61508 part 2 Annex E.**

***IEC61508, FPGA, design isolation, redundancy***

## I. INTRODUCTION

FPGAs are being increasingly used in the high-integrity and safety-critical domains. However, at present there is a lack of consensus of how FPGAs can be safely deployed and certified. One issue is whether the device should be treated as hardware or software during the certification process. Another issue is the lack of literature and shared information on the subject and on the determination of the risk associated to the FPGA technology.

In addition FPGAs possess features such as parallelism, reconfiguration, separation of functions, and self-healing capabilities that are compelling for creating redundancy and independent blocks and to increase the overall availability, however those features are not generally well known especially to safety assessors.

This document goes through the application of IEC61508 to Xilinx® FPGAs in regard to methods help to achieve a possible certification in accordance to the functional safety recommendation of the IEC61508.

## II. FPGAS AND SAFETY SYSTEM OF INTEREST

FPGAs have a good fit in safety applications, however, to perform the necessary validations the system of interest has to be defined.

Fig.1 defines the subsystem controlling the equipment under control (EUC). Each subsystem is equipped with sensors (S) and actuator elements (A) connected to a logic system (LS). This connection is done through an input interface (I) and an output interface (O).

According to the IEC61508 standard the different system levels are referred to as follows:

- The Electrical/Electronic/Programmable Electronic System (E/E/PES) design level describes how different subsystems are correlated.
- The subsystem design level describes how one of these subsystems is internally designed.
- The FPGA design level describes how the FPGA is internally designed.

The scope is to discover and mitigate failures having the potential to cause hazardous outcomes. There are a number of sources of failures, which can be classified according to the origin of the failures:

- Design: incorrect design;
- Implementation: faults introduced by the generation of the bit-stream from the design, loaded onto the FPGA and running the programmed FPGA;

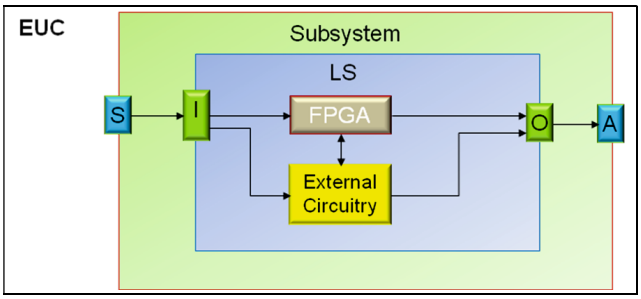


Figure 1 FPGA in IEC61508 system of interest

- Build: physical faults in the FPGA when it is produced or when it is built into the system;
- Wear: physical failures caused by continued use or storage;
- External: failures caused by the external environment in which the FPGA is operating.

In this article the basic assumption is that the overall design requirements for the FPGA are correct; issues with requirements are a more general problem and the IEC61508 requires that the safety management process takes care of them.

## III. Failure Mitigation

The way in which these failures can be mitigated is normally achieved with redundancy by duplicating an already existing function. It increases:

- The availability of a system.
- The robustness of the system.

Redundancy may be implemented at different system levels, depending on the requirements to be met. Because safety-related functions might be very complex and expensive, normally only a specific part of such a function is duplicated. Several types of redundancy can be defined.

### *A. Hardware Redundancy*

Hardware redundancy consists of duplicating all or parts of the electronic hardware in a way that if one part fails to operate, at least one of the duplicated parts will still deliver the correct service. There are several possible realizations:

- Single Chip redundancy
- Separate Chip redundancy
- Separate and diverse Chip redundancy

Additionally the redundancy can be N-module redundancy; simple redundancy, triple-mode, etc.

### *B. Software Redundancy*

Software redundancy consists of duplicating all or parts of the software, so that if one part of the software fails to operate at least one of the duplicated parts will still deliver the correct service. Examples of software redundancy may be:

- Additional redundant conditions before initiation of a critical event in conjunction with special bit patterns (instead of single bits) for critical flags.
- Implementation of different redundant data paths between critical inputs and their corresponding outputs.

### *C. Information Redundancy*

Information redundancy consists of duplicating all or part of critical information so that if one part of the information becomes corrupted at least one part containing the correct information remains preserved. Examples of information redundancy may be:

- Multiple storage of any type of critical information.
- Adding a checksum or signatures to preserve the integrity.
- Special coding, such as anti-valent bits; 0 is represented with two bits 01 and 1 is represented with 10.

### *D. Time redundancy*

Time redundancy consists of duplicating all or part of critical control at several times, to decrease the probability that faulty information is transferred resulting in a fault. Example of time redundancy may be:

- Duplicating the transmission of a message at different time instants.
- Duplicating the reading of a signal status within a time window.

### *E. Diversity*

Diversity is a type of redundancy realized with different implementations ensuring the independence of common development errors of the redundant components. Diversity can be performed in software or hardware. The two major forms of diversity realizable with an FPGA at the design level are N Self-Checking Programming and N-version programming. A self-check adds redundancy so that it can check its own dynamic behavior during execution. It may consists of either a variant and an acceptance test or two variants and comparison algorithm. In an N-version software system, each module is made with up to N different implementations. Each variant accomplishes the same task in a different way submitting its answer to a voter which determines the correct answer. VHDL development and Verilog development with the same specifications are an example of N-version programming to minimize the common cause effect of the HDL synthesizer. The full diversity can be realized using components, or subsystems designed and delivered by different manufacturers with the same initial specification.

### *F. Diagnostics*

It is common to adopt mechanisms for diagnosing the system in order to detect if the system operates out of its specification via testing. The result of the diagnostic test becomes input parameters to subsystems that affect the system operation and they should be considered in the safety system requirements specification. Redundancy and diversity in conjunction with diagnostic functions ensure that critical failures are detected and handled in time to prevent the loss of safety-related functions.

## IV. IEC61508 Requirements for the On-Chip Redundancy

The standard in its second edition fills a major gap left by the first edition providing requirements techniques and

measures for the avoidance and control of systematic failures as well as random hardware failures in ASICs and FPGAs. Annex E of the IEC61508 specifies the requirement for on-chip redundancy with the aim of reaching hardware fault tolerance (HFT) greater than 0 and at most SIL3. HFT of N indicates that N+1 faults cause loss of safety function in the system. According to IEC61508 the system is fail-safe if the faults are detected and the system degrades reaching a safe state without violating the safety conditions in adequate "short time". If the system is also able to maintain the mission functionalities in presence of such faults the unit degrades as fail-operational; another term for fail-operational is fail-functional. Implementing HFT with on-chip redundancy offers advantages, such as reduced space and power consumption compared to non-single chip solutions. However, the on-chip redundancy also brings more risks because of common cause failure which impacts several channels of a redundant system. Therefore, Annex E defines a set of stringent requirements to avoid or reduce the probability of such common cause failures and amongst them the requirement of separation. Each channel, as well as each diagnostics must be electrically and thermally separated. A proper fence created with a minimum distance between each block sufficient enough to achieve a clear separation between blocks. Interconnect-lines from one channel must not be routed through another channel. Each channel must have its own Input/Output (I/O) pins. Furthermore the effects on a channel of an increasing of temperature resulting from a fault in another channel must be mitigated, by implementing temperature monitors for example. The effects of a failure in the power supply must also be taken into account, by implementing voltage monitors. Other requirements are specified in the standard they contribute to increase or decrease the beta factor, but they will not be further detailed in this document. They can be found in [3] Part 2 Annex E.

## V. Interpretation of the IEC61508 Redundancy Clauses

The modalities for implementing the failure mitigation shall account the directives of the IEC61508 as practicable with special attention in getting the right interpretation of its clauses. Being the edition 2 extended with the ASIC-FPGA coverage there are a mix of rules and recommendation that are very conservative and sometimes, if taken literally, may lead to difficult interpretation amongst each other. For example while in the IEC61508 Part 2 Annex E the boundary scan method to increase test coverage seems a penalty, in the Part 7 Annex A clause A.2.3 the boundary-scan is encouraged as a mean to increase testability. It is recognized practically that might be beneficial having those requirements applied with a grain of salt, the mitigation mechanisms scrutinized and also applying the norm with the necessary safety judgment to avoid over specification and doubtful safety usefulness.

Thus assuming that from a functional safety standpoint a proper safety module has been identified, its redundancy concept analyzed, including the common cause failures, one of the common questions is: how to achieve physical separation between two or more modules into a single FPGA, for the purpose of augmented diagnostic or redundancy? (See Fig.2)

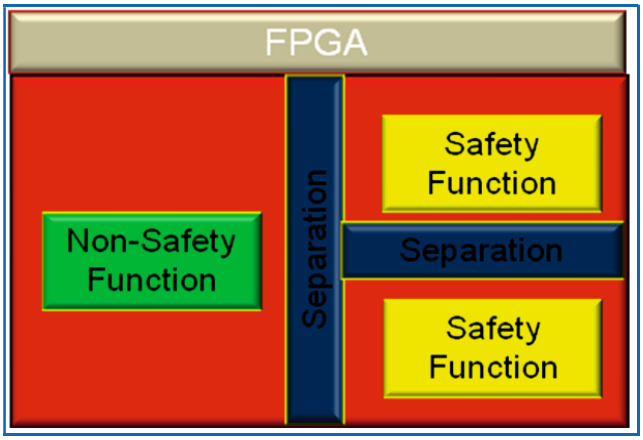


Figure 2 Example of separation

Among the possible techniques two are considered in this article:

- Xilinx Isolation Design Flow via IDF
- Detailed N-Version Isolation Design

The Xilinx Isolation Design Flow is a technique using floorplanning, pin placement, and constraints to create fences amongst two or more modules supplemented with a software tool that checks for the integrity of the fence in such a way that there are no contaminations between isolated modules.

The other technique, detailed N-Version isolation design, is the result of an experience realized by one of the authors to implement the isolation design with own special designed tools. This technique covers specifically Spartan 3E that is not available in the IDF.

## VI. Design Isolation

Xilinx Isolation Design Flow (IDF) valid for Virtex®-4, Virtex-5, and Spartan®-6 provides for multiple physically isolated/independent functions to be implemented within a single FPGA device, utilizing a fence of unused device components between each function. This paper addresses the Spartan 6 flow. Each isolated function is separated by this fence, generating isolated regions within the device. The flow uses early floorplanning, modular design, modular synthesis, and adherence to a set of guidelines and considerations to guarantee isolation between desired functions. Once a design is implemented, the Xilinx Isolation Verification Tool (IVT) can be used as a design rule check that isolation has been successfully implemented. The fence can also be considered as a structure that separate and decouples physical blocks as is defined in the Part 2 Annex E Table E.2 of the IEC61508 standard.

With the development of the IDF, functional safety and non functional safety processing logic can reside on the same FPGA, allowing designers of the safety system to realize the full capability of programmable logic.

The IDF was developed to allow independent functions to operate on a single device. The process is achieved using

Xilinx Synthesis Technology (XST), IVT and the PlanAhead™ software as tools [1].

### *A. IDF flow description*

The module synthesis is the first step, followed by system floorplanning, timing analysis, IVT on User Constrain File (UCF), design implementation and IVT on Native Circuit Description (NCD). (See Fig. 3)

A code reported in Fig. 4 shows the top instance of an extremely simplified dual channel redundant realization of a simple speed checker. Two modules called SafeRight and SafeLeft accept clock, reset, start, speed and run inputs and produce fail outputs. All the relevant signals are represented as anti-valent coding except for clock. Another module, the non-functional-safety is also implemented. Constraints are applied to all the modules to include I/O within isolated areas and floorplanning to ensure that proper isolation is achieved in logic, routing, and I/O blocks (IOB).

To achieve the proper separation each isolated function must be implemented in its own partition. Each partition must consist of a single module instantiation. A fence must be used to separate isolated partitions within a single chip. IOBs must be inferred or instantiated inside isolated partitions for proper isolation of the IOB.

The geometric constraints specified by the designer may be relaxed by the implementation tools provided required isolation is implemented. The fence is created indirectly by applying appropriate logic and routing constraints to each region to be isolated.

Specifically an AREA_GROUP constraint is applied at HDL level to the instances to be isolated

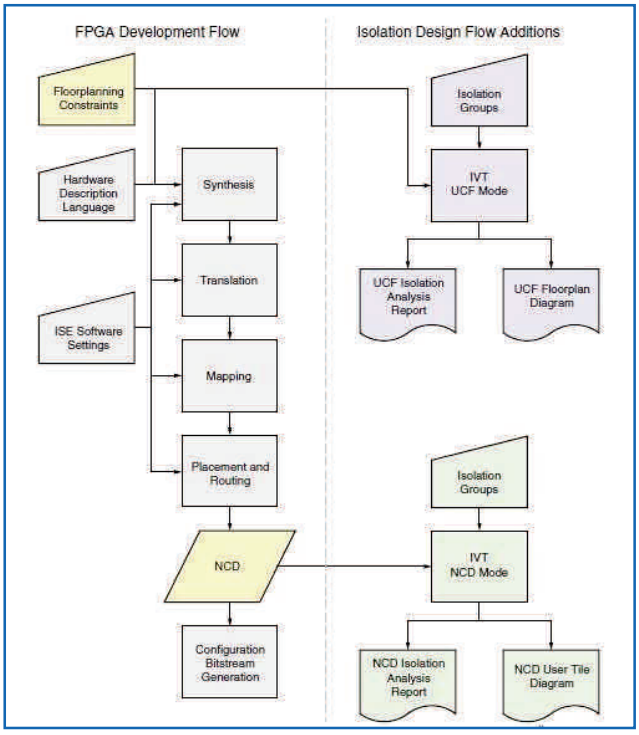


Figure 3 IDF design flow

```
package top_package is

component SafeRight
port(
    clk       : in  std_logic;
    reset     : in  std_logic_vector(1 downto 0);
    start     : in  std_logic_vector(1 downto 0);
    speed     : in  std_logic_vector(1 downto 0);
    run       : in  std_logic_vector(1 downto 0);
    fail      : out std_logic_vector(1 downto 0);
   );
end component;

component SafeLeft
port(
    clk       : in  std_logic;
    reset     : in  std_logic_vector(1 downto 0);
    start     : in  std_logic_vector(1 downto 0);
    speed     : in  std_logic_vector(1 downto 0);
    run       : in  std_logic_vector(1 downto 0);
    fail      : out std_logic_vector(1 downto 0);
   );
end component;

component NonFuntionalSafe
port(
    clk       : in  std_logic;
    reset     : in  std_logic;
    switch1   : in  std_logic;
    switch2   : in  std_logic;
    load_f1   :out std_logic;
    load_f2   :out std_logic;
    led       : out std_logic –
   );
end component;

end package;
```

Figure 4 Safety module example

The PlanAhead tool should be used to create the AREA_GROUPs with physical blocks (PBLOCKS) such a way that there is specific mapping between the isolated regions and the proper, design dependent, physical area in the FPGA.

When placing I/O buffers or pins, it is imperative to consider the physical location of I/Os in relation to the logic regions they interface with and I/Os need to be physically placed within that region. All available resources should be included, even if the logic is not used, because excluding them also excludes using their respective routing resources. This includes clock components such as DCMs, PLLs, and BUFGs, even though such components are instantiated (logically owned) at the top level. Naturally, all used components must be included in the isolated area and specifically the IOB deserve special care. If the IOBs are not included in the isolated region any routing from the isolated function to the IOBs cannot be contained to stay in the isolated region, and thus isolation cannot be guaranteed. Fig. 5 shows a properly included IOB in the isolated region.

Usually communications, cross checking or interlocks commonly find in the safety functions go off-chip, these signals must have their IOBs inferred inside an isolated partition. Thus the IOB to function routing must be specifically controlled at the lowest level partition. FPGA IOBs are commonly defined all in the top-level port map (for VHDL

users) and are inferred by the synthesis tool at the highest level of the target hierarchy. The only signals that should have IOB buffers inferred at the top level are those that do not require isolation. The IOBs are defined at the top level to allow simulation. The IDF is modular and each isolated function must be implemented independently of the other, however the development tools treat each module port declaration as being the design top level. Thus a method must be employed to assign the proper hierarchical level using some synthesis attributes that allow port definition but defer its instantiation in the lower level blocks. Those are (in VHDL):

attribute buffer_type: string;

attribute buffer_type of signal_name: signal is "none";

The IOBs are still identified in the top level of the hierarchy to enable design verification through standard functional and back-annotated timing simulations. However the IOB placement is determined at the lowest level applying the above constraints to only the signals that stay on chip while the remaining signals that have the attribute buffer_type unspecified will be inferred.

The definition of whether a function is isolated is determined in the PlanAhead tool by setting the module as a partition and then attaching the SCC_ISOLATED attribute to it.

IVT software must be used now. It verifies that an FPGA design that has been partitioned into isolated modules meets the isolation requirements. IVT is a batch application with a command line and file-based user interface. At this stage IVT must be used to perform a series of design rule checks on floorplans and pin assignments. It checks the UCF file to identify potential isolation problems before commitment to board layout. After the design is complete, IVT must be used again to validate NCD file such that the required isolation is built into the design and its results can be supplied to a certifying agency for verification. IVT runs under Windows 2000, Windows XP, Windows 7, and Linux operating systems. IVT, in UCF mode, checks the following:

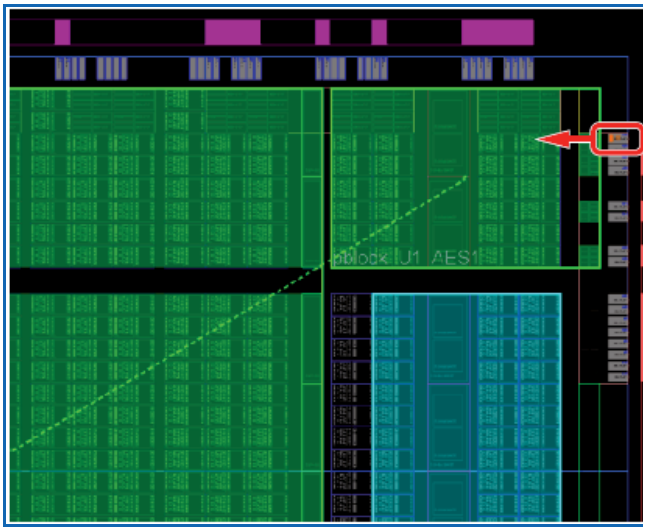

Figure 5 Floorplanning of isolated region

• Pins from different isolation groups are not physically adjacent, vertically or horizontally, at the die.

• Pins from different isolation groups are not physically adjacent at the package. Adjacency is defined in eight compass directions: north, south, east, west, northeast, southeast, northwest, and southwest.

• Pins from different isolation groups are not co-located in an IOB bank.

• The AREA_RANGE constraints in the UCF are defined such that a minimum of a one tile-wide fence exists between isolated regions.

The IVT shall then be run in NCD mode and it checks as the above list against the NCD file.

IVT outputs a text file that is a detailed report with faults and other items to be used for official evaluation. Additionally IVT outputs a graphical file representing a high-level overview of the user design that is helpful in quick identification of isolation faults.

IVT design is described in much more detail in [7] and applications are reported in [9] and [8] describes the proper hierarchical design.

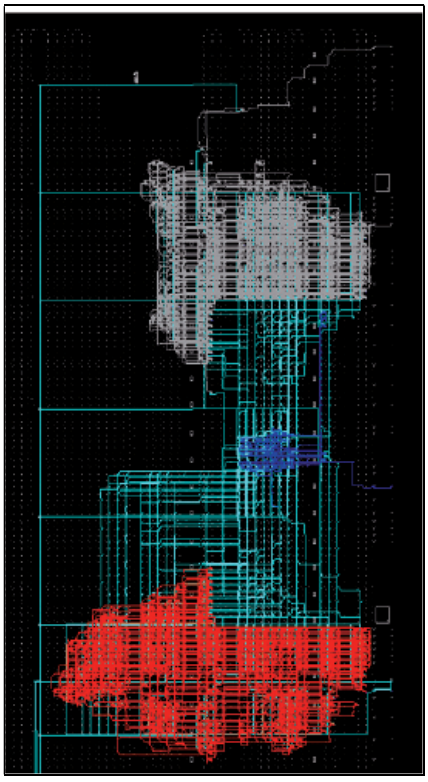

Figure 6 Floorplanning with the isolation

From the IEC61508 perspective the goal is achieving the confidence that separation is achieved without contamination and it leads to beta factor reduction to achieve the required 25% starting from the initial 33% as per Annex E. IEC61508 allocates a decrease in the beta factor of around 4% when separation is implemented. The example design of Fig. 4 for a SIL2 system uses separation and diversity and the Xilinx single event upset macro to diagnose the configuration memory at run time; conservatively it is assigned +2% as point 3a in table E1 of Annex E. The example requires the detection of speed that is evaluated by an algorithm using the FPGA hardened DSP (DSP48) for one channel, and look up tables (LUT) for the other channel. The finite state machines (FSM) are implemented as binary-safe state machine, for one channel, and one-hot safe for the other channel; thus achieving functional diversity. Starting with βIC= 33% as defined by the Annex E, the calculation lead to βIC = 33% + 2% (configuration memory)– 4% (fences and I/Os connected to ground) – 6% (diversity) = 25% ≤ 25%.

## VII. EXAMPLE OF AD-HOC IDF FLOW

A different approach is now presented using ad-hoc methodology that did not use IDF. This method has been applied to Spartan®3E. Fig. 7 presents the application where the Xilinx FPGA is used in a functional safety application for a smart pressure transmitter in the process automation market. It is composed by a transducer, also named sensor, which converts the process pressure into an electrical signal that is then filtered, amplified and digitalized by an analog module. The signal is then supplied to the signal processing part of the device, whose task is to compensate the effects of temperature, linearize the signal and calibrate it. The resulting digitally processed signal is thereafter supplied to an analog output module, which will draw a proportional current from the 4 to 20 mA interface. The aim of the system is to achieve SIL3.

Since the computational part has the highest probability to produce a failure, either systematic or random hardware, compared to the rest of the device, this part has been built around a triple redundant architecture, and especially the Two out of Three (2oo3) architecture defined by the standard IEC61508. This architecture is realized with three channels, which do the same tasks in parallel, and a voter. In the case of the 2oo3 architecture, the voter is a majority voter. This increases the availability compared to other types of voters. The voter compares all three channel's outputs together. If an output shows a difference compared to the two others, i.e. a fault occurred in this channel, the voter will mask this output and continue to work with the two other outputs. Only if a second fault occurs in the remaining channels, or if all three outputs show different values at the same time, the voter will achieve an alarm state, which is the safe state of the system. This architecture combines a high safety with a high availability. Furthermore, the system is implemented as a diverse redundancy, also known as diversity or N-version redundancy. The Three channels are not identical such as in a homogenous redundancy, but they are realised in a diverse manner with a DSP, a microcontroller and an analogue channel. The advantage is that not only the random hardware failures are detected, but also the systematic failures.

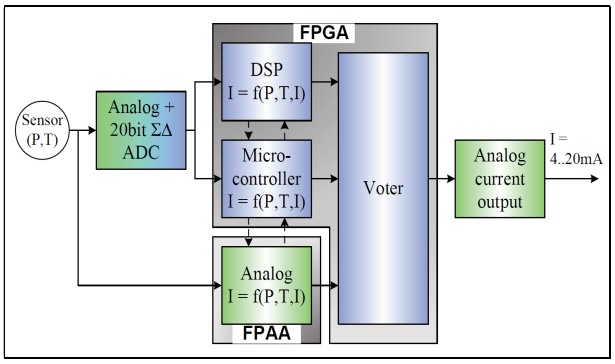


Figure 7 Coarse grained mixed-signal TMR architecture

Two channels and the voter are implemented into a Spartan 3 E 1600 FPGA from Xilinx. The remaining channel is implemented into a Field Programmable Analog Array (FPAA) from Anadigm. The three channels are realized in a diverse manner: a DSP, a microcontroller and an analog module. This arrangement decreases systematic faults that will impact all channels at the same time, and will therefore not be detected by the voter. The analog module further increases the diversity. The second advantage of the arrangement is the possibility to repair a fault in one channel. Indeed, thanks to the ability of FPGAs to be partially and dynamicly reconfigured, the voter can repair a fault detected in one channel, by reconfiguring it. This is known as a self-healing process. This process can further increase the availability and the safety compared to a classical 2oo3 system. Indeed, after the detection of a fault and its reparation, the system is again triple redundant. For the user and the safety, the process is transparent; the fault is automatically corrected. There is no reduction of channel, as in classical 2oo3 systems. A more detailed description of the system can be found in [5].

The modules implemented in the FPGA build a so-called on-chip redundancy, which was presented together with the requirements from the standard in the section IV.

### *A. Ad-hoc Safety Aware Place and Route for on-chip redundancy in FPGA*

While the IDF is mainstream, for other FPGA families like the Spartan 3, alternative techniques should be used. In such a case in the fence all resources, such as for the Xilinx FPGA, the Configurable Logic Blocks (CLB), Block RAMs (BRAM), multipliers, Digital Clock Manager (DCM)and IOs, as well as the interconnect lines, are configured or connected to ground. The size of the fence is dependant of the size of interconnect lines. The interconnect lines of the Spartan 3 E are organized in four groups: direct connection, double lines, hex lines and long lines, from the shorter to the longer respectively. The size of the fence must be defined in such a manner that no Hex lines can connect one channel or function block with another one. Therefore, the fence is at least 8 CLBs wide. Since the long line spans the whole FPGA, they cannot be used; otherwise they will connect two different channels together. Therefore, they will be connected to ground. The fence also acts as a potential ring, which electrically separates the different channels. Indeed, if a failure in one channel would faulty

connect a signal line of the channel with an interconnect line of the fence, this will produce a short circuit in the concerned channel, but not influence the other channels. This avoids common cause failures. A complete description of the fence and all other failure avoidance measures can be found in [6].

### *B. Safety Aware Place And Route tools*

Implementing the fence without using PlanAhead is not so simple, since all resources must be configured or connected to ground. The solution is to implement the fence during the Place And Route (PAR) phase defining a hard macro by hand by using the tool FPGA Editor form Xilinx. With a hard macro, it is possible to save a placed and routed function block for reuse in another design. In this case, the fence can be defined and routed by hand, then saved into a hard macro, and finally used in the standard Xilinx toolflow. The Xilinx tools will implement the macro without modification, and place and route the rest of the design around it.

However, realizing the fence by hand will take so much time, that it is not practicable. For example, the fence used to separate the three blocks in the system presented above and represented in Fig. 9, comprises 750 CLBs, 4 BRAMs and multipliers, 4 DCMs, 116 IOs and about 550 000 interconnect lines. Furthermore the probability to forget one resource or interconnect line is huge. In this example a tool has been implemented by one of the authors based on Microsoft Excel for the graphical part, on Microsoft Visual Basic for the program part and uses a SQLite3 database. The user must define the form and size of the fence and all other function blocks in an excel sheet, as represented in Fig. 8. All resources: CLBs, BRAMs, Multipliers, DCMs, IOs and Switch Matrix (SM) for a defined FPGA, here a Spartan 3E1600FG320, are already defined and placed in the Excel sheet. Each cell represents a resource. The user only needs to set the color of a resource according to the rules defined above (8 CLB/SM wide fence). In Fig. 8, the fence is gray, the microcontroller module is red, the DSP module is yellow and the voter is green.

The tool first creates a UCF template with the pin assignment for each block and the fence. The user can use it to make their individual assignment. The tool also generates a view of the pin assignment for the PCB Layout. The user can layout the board according to this information. As stated by the standard, the layout is also important for the implementation of an on-chip redundancy.

The tool then creates a SQLite3 database, with all resources present in the fence. For each CLBs, DCM, BRAMs, multipliers and IOs, the tools uses configuration templates. These templates are defined in such a manner that the maximum of internal resources, such as Look-Up Tables (LUT), Flip-Flops (FF) and routings lines, are configured or connected to ground.

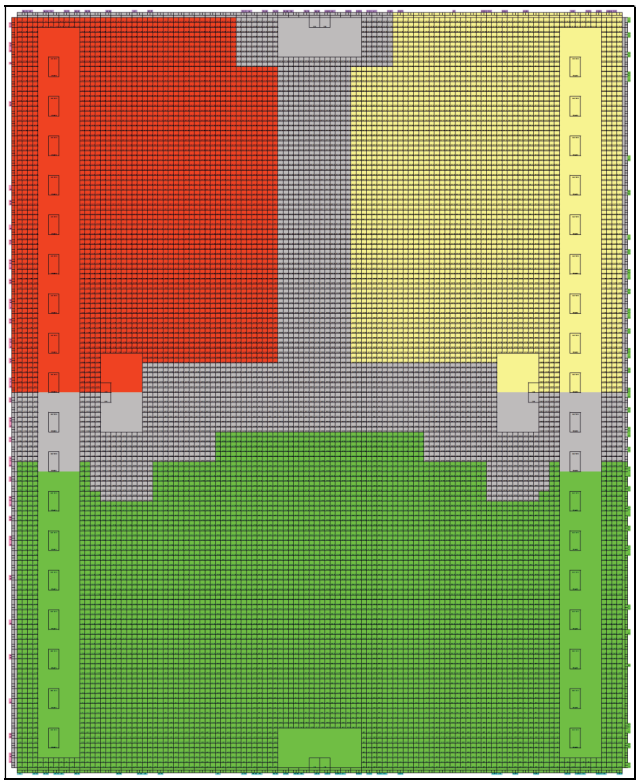

Figure 8 Safety Aware PAR tool

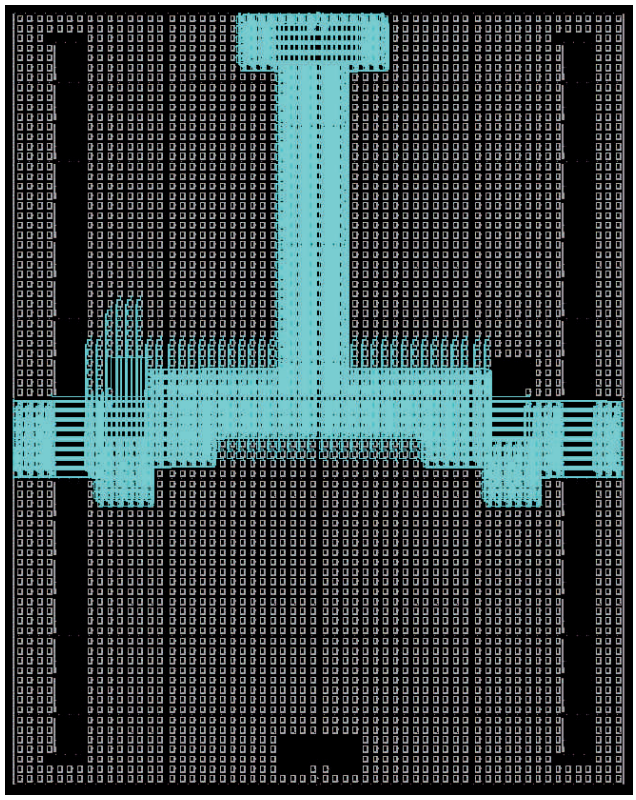

Figure 9 Fence in a Spartan 3E1600

The tool creates a XDL file, with the configuration of all resources present in the fence. The XDL file is a configuration description file for Xilinx FPGAs. This file type comes with a homonymous tool, XDL, which converts XDL files into hard macro file (.nmc) and vice versa. With XDL, it is also possible to retrieve all information about a defined resource, such as all interconnect lines of a CLB and their destinations. This possibility is used by the Safety Aware PAR tool: all interconnect lines of a resource in the fence are saved into the database. Afterwards, the tool configures each interconnect lines to ground, which starts and ends in the fence, according to the information present in the database. This is a compromise to reduce the interconnect lines usage without compromising the purpose of the fence. Nevertheless it is ensured that no interconnect lines cross the fence. As already mentioned, the long lines are connected to ground. The configuration of the interconnect lines is also written into the XDL file. Finally, the complete XDL file can be converted into a hard macro file using XDL. The resulting fence is represented as hard macro in blue in Fig. 9. For clarity, long lines are not represented here. The generated macro can be then instantiated in the top level HDL file together with all other function blocks. It will be implemented as it is in the FPGA. The other blocks will be placed and routed around.

## VIII. CONCLUSION

The advancements in design methodologies and tools like PlanAhead help in achieving the requirements of the IEC61508 rev. 2 with a smooth design flow. The use of PlanAhead achieves the placement, the definition of isolation regions and timing verification. IVT makes it possible to automate the verification step in such a way that possible human error is eliminated. The isolation design flow is part of the IEC61508 package for functional safety under the IEC61508 rev. 2. While Xilinx`s main focus is on the Spartan 6 and the next 7 Series product, another ad-hoc mechanism has been presented for Spartan 3 in which is possible to reach a similar result via specific application of ad-hoc tools. It is the hope of the authors that this article will highlight the potential of the FPGA in safety applications in compliance with the IEC61508.

[10]

[1] PlanAhead http://www.xilinx.com/tools/planahead.htm

[2] ISE DesignSuite http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm

[3] International Electrotechnical Commission, “IEC 61508 Second Edition: Functional Safety of Electrical/Electronic/Programmable Electronic Systems”, 2010.

[4] Xilinx IEC61508-Guideline

[5] R. Girardey, M. Hübner, J. Becker, “Dynamic Reconfigurable Mixed-Signal Architecture for Safety Critical Applications”, FPL2009, Prague, Czech Republic, 2009

[6] R. Girardey, M. Hübner, J. Becker, “Safety Aware Place and Route for On-Chip Redundancy in Safety Critical Applications”, ISVLSI2010, Lixouri, Greece.

[7] Steve McNeil, “Developing Secure Designs with the Spartan-6 Family Using the Isolation Design Flow”, XAPP1145, (v1.0) August 23, 2011

[8] Xilinx UG748, “Hierarchical Design Methodology Guide”

[9] John D. Corbett, “The Xilinx Isolation Design Flow for Fault-Tolerant Systems”, WP412, (v1.0) January 30, 2012