Gaisler Research

Lessons Learned from FPGA Developments

Technical Report

FPGA-001-01
Version 0.0
April 2002

Prepared by Sandi Habinc

EUROPEAN SPACE AGENCY CONTRACT REPORT

The work described in this report was done under ESA contract, No. 15102/01/NL/FM(SC) CCN-3.
Responsibility for the contents resides in the author or organisation that prepared it.

Stora Nygatan 13
411 08 Göteborg
Sweden

tel +46 31 802405
fax +46 31 802407
www.gaisler.com

1 INTRODUCTION

1.1 Scope

This document is a compilation of problems encountered and lessons learned from the usage of Field Programmable Gate Array (FPGA) devices in European Space Agency (ESA) and National Aeronautics and Space Administration (NASA) satellite missions. The objective has been to list the most common problems which can be avoided by careful design and it is therefore not an exhaustive compilation of experienced problems.

This document can also been seen as a set of guidelines to FPGA design for space flight applications. It provides a development method which outlines a development flow that is commonly considered as sufficient for FPGA design. The document also provides down to earth design methods and hints that should be considered by any FPGA designer. Emphasis has also been placed on development tool related problems, especially focusing on Single Event Upset (SEU) hardships in once-only-programmable FPGA devices. Discussions about re-programmable FPGA device will be covered only briefly since outside the scope of this document and will become the focus of a separate future technical report.

1.2 Reference documents

RD1	ASIC Design and Manufacturing Requirements, WDN/PS/700, Issue 2, October 1994, European Space Agency
RD2	VHDL Modelling Guidelines, ASIC/001, Issue 1, September 1994, European Space Agency
RD3	Space product assurance, ASIC development, ECSS-Q-60-02, Draft Specification, Issue 3, Revision 1, European Cooperation for Space Standardization
RD4	Logic Design Pathology and Space Flight Electronics, R. Katz et al., ESCCON 2000, ESTEC, Noordwijk, May 2000, European Space Agency
RD5	Small Explorer WIRE Failure Investigation Report, R. Katz, NASA Goddard Space Flight Center, 27 May 1999
RD6	Use of FPGAs in Critical Space Flight Applications — A Hard Lesson, Wally Gibbons & Harry Ames, MAPLD 99, 28-30 September 1999, The Johns Hopkins University-Applied Physics Laboratory, Laurel, Maryland, USA
RD7	An Experimental Survey of Heavy Ion Induced Dielectric Rupture in Actel Field Programmable Gate Arrays (FPGAs), G. Swift & R. Katz, RADECS 95
RD8	Entrusting EDA vendors to synthesize your FPGA is like having McDonald's cater your daughter's wedding, Ted Boydston, Harris Corporation, ESNUG Post 390 Item 3, 20 March 2002

1.3 Vendor and tool reference documents

RD9	Commercial to Radiation-Hardened Design Migration, Application Note, 5192644-0, September 1997, Actel Corporation
RD10	Design Migration from the RT54SX32 to the RT54SX32S Device, Technical Brief, 5192679-0/8.01, August 2001, Actel Corporation
RD11	Prototyping for the RT54SX-S Enhanced Aerospace FPGA, Application Note, 5192672-0/1.01, January 2001, Actel Corporation
RD12	Actel SX-A and RT54SX-S Devices in Hot-Swap and Cold-Sparing Applications, Application Note, 5192687-0/12.01, December 2001, Actel Corporation

RD13	Power-Up and Power-Down Behaviour of 54SX and RT54SX Devices, Application Note, 5192674-0/1.01, January 2001, Actel Corporation
RD14	Power-Up Device Behaviour of Actel FPGAs, Application Note, 5192663-1/03.02, March 2002, Actel Corporation
RD15	Two-Way Mixed-Voltage Interfacing of Actel's SX FPGAs, Application Brief, March 1999, Actel Corporation
RD16	eX, SX-A and RT54SX-S Power Estimator, Actel Corporation
RD17	JTAG Issues and the Use of RT54SX Devices, 20 September 1999, Actel Corporation
RD18	Using IEEE 1149.1 JTAG Circuitry in ACTEL SX Devices, 25 August 1998, NASA Goddard Space Flight Center
RD19	Testing and Burn-In of Actel FPGAs, Application Note, 5192662-0/4.00, April 2000, Actel Corporation
RD20	Design Techniques for Radiation-Hardened FPGAs, Application Note, 5192642-0, September 1997, Actel Corporation
RD21	Using Synplify to Design in Actel Radiation-Hardened FPGAs, Application Note, 5192665-0/5.00, May 2000, Actel Corporation
RD22	Minimizing Single Event Upset Effects Using Synopsys, Application Note, 5192651-0, July 1998, Actel Corporation
RD23	Package Characteristics and Mechanical Drawings, 5193068-1/2.01, v3.0, February 2001, Actel Corporation
RD24	EIA Standard Board Layout of Soldered Pad for QFP Devices (PQ/RQ208/CQ208), Actel Corporation
RD25	Analysis of SDI/DCLK Issue for RH1020 and RT1020, Technical Brief, 5192689-0/1.02, January 2002, Actel Corporation
RD26	Termination of the VPP and Mode Pin for RH1020 and RH1280 Devices in a Radiation Environment, Technical Brief, 5192683-0/10.01, October 2001, Actel Corporation
RD27	RadHard/RadTolerant Programming Guide, 5029106-1, August 2001, Actel Corporation
RD28	Metastability Characterization Report, Application Note, 5192670-0/8.01, August 2001, Actel Corporation
RD29	Using Synplify to Design With Actel Radiation-Hardened FPGAs, September 1999, Synplicity Inc.
RD30	Designing Safe VHDL State Machines with Synplify, 1999, Synplicity Inc.

1.4 Acronyms and abbreviations

ASIC	Application Specific Integrated Circuit
ECSS	European Cooperation for Space Standardization
EDAC	Error Detection And Correction
ESA	European Space Agency
FPGA	Field Programmable Gate Array
NRE	Non Recurring Expense
SEU	Single Event Upset

1.5 Acknowledgements

Many of the problems and their solutions discussed in this document have been contributed by Mr. Peter Sinander (formerly with ESA) and Mr. Richard Katz (NASA).

2 BACKGROUND

Field Programmable Gate Array (FPGA) devices have been used in space for more than a decade with a mixed level of success. Many problems have been encountered due to technological reasons as well as due to design methodology deficiencies. With the capacity of FPGAs continuously increasing the latter problem is growing, which has been demonstrated in some of the most recent satellite developments. The objective of this section is to characterize the increasing potential of FPGAs and to underline the commonly misleading perceptions of how they can be designed and used.

2.1 Potential

The capacity and performance of FPGAs suitable for space flight have been increasing steadily for more than a decade. For once-only programmable devices the increase has been from approximately 2 kgates to 72 kgates. For reprogrammable devices the increase has been from 40 kgates to 2 Mgates. The current technology trends for the space flight FPGA market are:

- from total dose hardness to total dose tolerance;
- from SEU sensitive registers to registers with built-in TMR;
- from once-only anti-fuse based programming to SRAM / EEPROM base re-configuration;
- from SEU sensitive registers to SEU sensitive FPGA configuration;
- from high to low voltage operation: from 5 V to 3.3 V, 2.5 V and 1.8 V;
- from simple module generators to Intellectual Property (IP) core usage.

The application of FPGAs has moved from simple glue logic to complete subsystems such as monolithic telecommand decoders and telemetry encoders. The potential for FPGA use in space is steadily increasing, continuously opening up new application areas. The FPGAs are more commonly being used in critical applications and are replacing ASICs on a regular basis.

The complexity of the designs that are made in FPGAs has increased at the same rate as for ASICs. In essence, the FPGA design task has become <u>much</u> more complex. It has never before been so easy to make so many mistakes so fast as now.

Until now only two major FPGA vendors supplied devices for the space market: *Actel* and *Xilinx*. New vendors are planning to offer flight FPGAs: *QuickLogic* and *Atmel*.

Since this report has been focused on existing once-only-programmable devices, the FPGA products from Actel Corporation will be discussed in some detail. What is therefore more suitable than to round off this section by presenting Actel Corporation's view on the advantages of flight worthy FPGA devices.

Flight worthy FPGAs offer the designer significant advantages over discrete logic, including:

- reduced weight and board space due to decrease in devices required;
- increased reliability with reduced solder connections;
- increased flexibility to make design changes after board layout is complete;
- lower cost of ownership with fewer vendors to qualify.

Flight worthy FPGAs offer the designer significant advantages over ASIC devices:

- increased flexibility to make design changes after board layout is complete with much shorter shipment lead times;
- lower cost of ownership with fewer vendors to qualify and no NREs required;
- lower risk since design does not have to be completed six months in advance of device delivery.

2.2 Perception

Many of the benefits of using FPGAs in commercial applications often become a problem when applying the devices to space applications. FPGAs are perceived as easy to modify and correct late in the development process. This often leads to design methods that would not be accepted in ASIC developments. The design methodology used is often more casual than when developing ASICs. The design of FPGA devices is still considered analogous to the design of electrical boards, rather than the design of ASICs.

There is an overall increase of FPGA usage due to a reduced number of space specific standard ASICs. Board developers employ FPGAs more frequently and the portion of electrical board functionality implemented in FPGAs increases. However, the improvement of the design methodology has not always increased at the same rate, nor is the risk awareness fully appreciated by the project management.

ESA often receive electronic equipment with FPGAs embedded for which there is no specific FPGA documentation, making it difficult to assess the correctness of the design. It becomes even worse when a problem is discovered late in the integration cycle and there is no visibility into the design or the verification process.

It has always been recommended that proper design methods are applied to FPGA designs, which initially increases the FPGA development cost but reduces downstream costs. However, a common excuse is frequently encountered: *This is a low cost project with minimal requirements for formal documentation and analysis of the FPGA designs.*

3 LESSONS LEARNED

Field Programmable Gate Array (FPGA) devices have been used in many satellite developments, ranging from breadboards to flight equipment, with a mixture of success stories and problem cases. The following real life cases will illustrate the risk of combining the increasing FPGA complexity with a naive perception of the efforts required to be designing with them. As for the success stories, in a majority of the cases the design methods applied have been at an expected level of standard which can explain the success in the first place.

3.1 The WIRE power-up mishap

This section is a shortened transcription of the failure investigation report, RD5, for the Wide-Field Infrared Explorer (WIRE) spacecraft. There are several other findings in the aforementioned report that are not covered in this document since being outside its scope.

The NASA WIRE spacecraft was launched in 1999 and a problem was detected during its second pass over the ground station, when the spacecraft was observed to be spinning. It was determined by the failure review board that the cover was ejected at approximately the time that the WIRE pyro box was first powered on. The instrument's solid hydrogen cryogen supply started to sublimate faster than planned, causing the spacecraft to spin up to a rate of sixty revolutions per minute once the secondary cryogen vent was open. Without any solid hydrogen remaining, the instrument could not perform its observations.

The probable cause of the WIRE mishap is logic design error. The transient performance of components was not adequately accounted for in the design. The failure was caused by two distinct mechanisms that, either singly or in concert, resulted in inadvertent pyrotechnic device firing during the initial pyro box power-up. The control logic design utilized a synchronous reset to force the logic into a safe state. However, the start-up time of the crystal clock oscillator was not taken into consideration, leaving the circuit in a non-deterministic state for a significant period of time. Likewise, the start-up characteristics of the FPGA were not considered. These devices are not guaranteed to follow their truth table until an internal charge pump starts the part and the uncontrolled outputs were not blocked from thepyrotechnic devices' driver circuitry. A contributing factor to the mishap was the lack of documentation for the FPGA's power-up transient characteristics in the device data sheet. This information is however available in the FPGA data book and design guide in two application notes.

At least three conclusions can be drawn from the WIRE mishap:
- the design should never rely on the default value of the device output during power-up
- the designers should continuously collect and read all available device documentation
- there should be a direct asynchronous path from the reset input for initialising the device state, being independent of any clock activity

3.2 The Single Event Upset review

At the end of the development phase of a recent satellite projected it was noted at the critical design review that there were FPGAs used in several sub-systems for which there was very little documentation. It was also noted that either no Single Event Upset (SEU) requirements had been established or they had not been followed for some of the designs. A task force was formed to investigate all FPGA designs in critical sub-systems such as power, computer and communications. A first inventory showed that there were more than fifty FPGA parts used in such systems for which there were at least ten different designs. A half dozen designs were selected for detailed review after an initial classification. The review covered the design methods, the SEU protection mechanisms, gate-level netlist inspection, fault injection by simulation etc. A design deemed critical was subjected to heavy ion testing to validate the applied review methods. Considerable effort was spent on the review work by the task force as well as by the companies responsible for the designs.

A variety of SEU related problems were uncovered, many linked to the use of synthesis tools. Some of the problems lead to modified spacecraft operation procedures in order to mitigate their effects on the mission. No FPGA design was in fact modified since flight models of the various sub-systems had already been delivered. The findings from this review will indirectly be discussed in detail in section 7.

At least five conclusions can be drawn from this review:

- designers are often unaware of how the synthesis tools work;
- little effort had been done to verify that SEU protection were actually implemented;
- when shortcomings caused by lack of SEU protection were in fact documented by the designers, they were no properly propagated in the project organisation to have an impact on spacecraft operations;
- it is extremely costly to perform a review a long time after the design has been completed;
- poor awareness in spacecraft projects regarding the sheer number of FPGA designs and parts embarked on spacecraft.

3.3 Risks in using FPGAs for ASIC validation

One of the first times FPGAs were used in an ESA funded activity was during a bus interface ASIC development. The design was first prototyped using four FPGAs. The design was then transferred to a standard cell ASIC technology. The transfer was made on the gate-level which resulted in many difficulties during layout. Although the FPGAs had been used in operational surroundings during the validation of the prototype, there were several bugs that were not discovered until after ASIC manufacture and validation.

The lessons learned from this development are that an application should initially be designed targeting the ASIC technology and only when the design is completely verified should it be partitioned to meet the requirements posed by the FPGAs. It might seem as a good idea to transfer the design on the gate-level, since the FPGAs were already tested, but the disadvantages outnumbered the potential advantages. For example, circuitry required for the partitioning between FPGAs were unnecessarily incorporated in the ASIC. The verification effort for the FPGAs was lower than it would have been for an ASIC since FPGAs are perceived as simple to correct after testing in the operational environment. The verification effort for the ASIC was also reduced since the design had already been validated using the FPGAs. Bugs that could have been easily detected during routine ASIC verification went therefore undetected.

3.4 Risks in using ASICs to replace FPGAs ongoing project

In two separate but similar developments it was decided to transfer multiple FPGA designs into a single ASIC which could be configured to operate as one of the FPGA designs at a time. This would reduce the cost of the development since the targeted FPGA parts were rather expensive compared to an ASIC solution. It was considered less expensive to qualify one ASIC than multiple FPGA designs. There were also improvements in power consumption and reliability performance to be gained by going to ASIC technology. In the first case the transfer was done in nine months, successfully transferring six FPGAs.

The second case was not that successful since many of the FPGAs needed to be upgraded or even completed and tested before the transfer could begin. In the end the transfer was not possible due to schedule limitation resulting in unforeseen FPGA usage for the flight equipment. The lesson learned is again that FPGA prototyping is not a guarantee that a transfer to ASIC technology will finally be successful.

3.5 The Preferred Parts List syndrome

The Preferred Parts List (PLL) for a spacecraft project included an FPGA device type. The FPGA device was subsequently used in some of the sub-system designs. The device had been placed on the PLL since it had been used in the mass memory controllers on other spacecraft for which extensive SEU protection and analysis has been performed. There were however no warnings or restrictions attached to this list regarding the SEU susceptibility of the FPGA device. A number of contractors used the device without considering the SEU risk. Before placing similar FPGA devices on future preferred parts lists it should be assured that the potential users are informed of any SEU limitations before embarking it in their designs.

3.6 Design and verification by the same individual

Although recommended that verification should be performed by someone else than the designer, this is rarely the case for FPGA developments. The obvious risk of masking errors done by the same individual during design and verification is often ignored. The reasons for this ignorance or calculated risk taking is often tight budget and schedule. It is costly to a have separate engineer doing the verification since the individual needs to a have an understanding of the application which is as good as that of the designer. To obtain this understanding it is necessary to specify and document the design extensively which again is costly. Instead, this whole bit is skipped by the company letting the designer verify his own work, saving on personnel and documentation. Needless to say, this is often false economy. In several satellite developments this has become a painful reality. An example is the design of a finite state machine, implementing the core reconfiguration handling of a spacecraft, for which the designer implemented something else than what had been specified and the only verification done was by visual inspection performed by the designer himself. The discrepancy was not discovered until an independent review was performed after the equipment had been delivered. In another example there were several cases of similar mistake masking since some parts of the design had been verified by the same engineer as had designed it, and not by the colleague as it had been planed. Unnecessary design iterations were required in this particular case.

3.7 Inadequate verification with inapt stimuli

With growing complexities of FPGA devices, the generation of a complete verification suite requires an ever increasing portion of the development time and resources. Today, even though extensive verification has been employed, FPGA designs with incorrect functional behaviour are often produced. Typically, the erroneous logical implementation has not been detected due to inappropriate verification criteria. This is often caused by inadequate or unsophisticated stimuli generation. In one specific FPGA project, the simulation process relied wholly on manual entry of test vectors and visual on-screen inspection of waveforms which is error prone and time consuming and induces a risk for an overall reduced verification effort. The simulation test cases were sometimes simplified, since cumbersome to tailor, e.g. the same data were written multiple times during a test which could mask errors. There were also no means for automatic regression testing. With such an awkward approach to stimuli generation it is not difficult to see why the resulting quality of the verification was poor.

3.8 The verification by testing pitfall

Verification of FPGA designs is commonly performed by means of logical simulation. A common way to short-cut the verification cycle is to skip simulation and quickly move to testing. In one particular flight computer development there were several FPGAs to be designed. The development philosophy was to proceed to hardware testing with real sensors as soon as possible to ensure a working design in the real world. The practice for the developing company was to implement and program FPGAs through step wise refinement, resulting in as many as eight iterations per FPGA design. Although this method provides fast access to the hardware, allowing early software integration and parallel development of hardware and software, testing of hardware cannot replace corner case simulation etc., which might be sacrificed to compensate the additional time required for hardware debugging. This method might lead to patching rather than designing, with the risk that problems are not discovered until late in the design process.

3.9 Design entry by gate-level pushing

The benefits of using a hardware description language such as VHDL for designing FPGAs should by now be known to most hardware designers and managers. This is however not always the case as was discovered in two recent flight FPGA projects.

In the first case it was partially justified to enter the design manually at the gate-level, often using predefined macros from the FPGA vendor, since the activity was assumed to be based on a previous design, only requiring a limited amount of design modifications. This design was already described on the gate-level which resulted in continuous usage of gate-level entry. In addition, the design team did not have any experience which VHDL. It is thus not obvious that the usage of VHDL would have reduced the design effort and required time in this particular case. For future project it is was however recommended that the team invest in VHDL tooling and training, which should simplify future design entry and verification.

In the second case VHDL was actually used for parts of the design. But its benefits were somehow lost since also gate-level entry, graphical block diagram entry, vendor specific macros and block generators were used for the rather small 4 kgates design. The mixing of design entry styles resulted in several problems. For example, the initial design was not possible to be read-in by the place and route tool since cells had been used from the wrong technology, macros with a low drive capability had been connected to too many gates, etc. The design did not work properly since simulation could only be performed at the gate-level, due to the mixed design entry style, which led to few test cases and no use of VHDL for stimuli creation. The design was subsequently redone using only VHDL which resulted in a functionally correct design in less than a day, including the addition of SEU protection mechanisms and diagnostics modes.

3.10 Running out of clock buffers

Although synchronous design methods are the safest approach to logical design independent of target technology or device type, it seems that asynchronous solutions are the ones that always seems to be the designers' preference. Combining asynchronous design with the limitations of the FPGA architecture often leads to inadequate results. In a particular FPGA design, the task was to measure the number of rising edge events on several input signals. The initial solution was to build separate counters for each input signal which would also act as the clocks for the counters. The first problem that was encountered was that there were not enough clock buffers in the FPGA architecture to support all input signals. The second problem was that it was not clear how to synchronise the counters with the control logic, since all resided in separate clock domains. The solution to the problems was simply to develop a synchronous design in which edge detectors for the input signals were implemented as local, one flip-flop only, clock domains, and the output of the edge detectors were oversampled with the system clock. All counters were implemented in the system clock domain, in which also the communication logic was implemented. The design job was largely reduced by moving to a synchronous solution which required only a moderate effort for the worst case timing and clock skew analysis.

3.11 Risks in not performing worst case timing analysis

Timing is everything when it comes to logical design since if the timing constraints are not met the logic operation of the design will be invalid. Still worst case timing analysis is often neglected in FPGA development since the designers believe that their design is within the performance capacity of the technology. In one particular case this was pointed out to a company which replied that there were no formal requirements for worst case timing analysis. In another case the timing problems were not discovered until the flight model underwent thermal cycling. The conclusion made by that company was that for only one of several similar signals the routing in the FPGA was unfavourable and therefore the timing constraints were not met. The timing analysis was only performed as part of the failure analysis. The electrical board hosting the FPGA had to be patched in order to mitigate the problem. Designers and managers often misinterpret the maximum clocking frequency figures provided in FPGA data sheets as promisees that are always true and therefore neglect the timing analysis. This has been seen in combination with manual gate-level entry for which timing analysis is often done manually.

3.12 Managers and reviewers

Finally, to illustrate how an FPGA development can be mismanaged, an example has been taken from a NASA *faster, better and cheaper* project. In one particular development the FPGA design had been reviewed according to normal procedures. In addition, special *red teams* performed reviews, finding no problems, even claiming that good design practices had been applied. This was in stark contrast to what some independent reviewers later found. A quote from one of the reviewers tells it all: *"Oh my God!"*. The design was full of mistakes and there were more problems introduced by the local design guidelines than were solved. The lesson learned from this example is that the reviewers should be qualified engineers and not managers.

Looking a the above examples I can only repeat myself: *FPGAs should be forbidden in space applications until the designers learn how to design with them.*

4 MOTIVATION FOR DEVELOPMENT METHODOLOGY

As can be derived from the preceding section many of the failures or problems involving FPGA devices in space applications are a result of applying an inadequate development methodology. A motivation for establishing adequate methods for FPGA developments will be provided in this section.

4.1 Comparison with electrical board development

When electrical boards carrying FPGAs are designed as part of a spacecraft or unit development, the FPGAs are often considered as part of the board and are treated as black boxes. The FPGAs are assumed to be covered by the board specification and there are seldom any specific documents associated with the FPGAs.

The FPGAs are not like any other discrete component that is added to a electrical design. Today most part of the logical design has been moved from the electrical board design to the FPGA design. The effort for designing the FPGA is easily larger than the effort required for the board itself. The FPGAs should therefore also be treated differently.

The way FPGAs are designed today often includes the use of a hardware description language such as VHDL or VERILOG. The board designers and the system engineers are normally not accustomed to these languages which results in a gap between them and the FPGA designer that is often left open. If the FPGA designs are not document in way that the behaviour of the design can be understood by the board designer or the system engineer, e.g. an extensive user manual or a data sheet, there is a risk that the design details will not be propagated upwards in the design hierarchy affecting operations and the documentation of the unit. From the reviewers point of view it is difficult to assess the correctness of these FPGAs. Things become even more difficult when one is summoned to solve problems discovered late in the development. Problems are always discovered late since nobody has bothered to verify the FPGAs properly before integrating them on the board. Since there are no specifications for the FPGAs it is difficult to verify their correctness independently.

The verification of the FPGA should not be done in isolation by the designer. To ensure that the FPGA will operate in its intended environment board level simulation should be employed. The design is often verified by the designer only, this being a great potential for error masking. The setup of the verification campaign and the interpretation of the results should involve not only the FPGA designer but also those responsible for the board and system design. This is to ensure that the design loop is being closed correctly.

4.2 Comparison with ASIC development

The following arguments are based on an excellent article written by Mr. Ted Boydston in ESNUG Post 390, RD8.

It is often claimed that there is an immense difference between the ASIC and FPGA design methodologies. It is however difficult to find a technical justification for this claim, since it is often solely driven by cost issues leading to reductions in documentation and verification efforts. The flows between the two technologies should instead be as similar as possible. Differences, like electromigration in ASICs or dealing with architectural FPGA peculiarities, should be taken as additions to the design process. Instead, FPGAs, because they are often reprogrammable, are treated differently. They have no NRE, hence no sense of failure. They are the software of the hardware world, encouraging engineers to quickly get to hardware test, ignoring good design practice. There have been cases at many companies where a device will spend months in the lab doing testing, because of flawed FPGA design flow. This is a waste of resources and schedule, but because there is no NRE looming its regarded as alright to continue with this flawed flow. Hereafter follow some examples of encountered FPGA flaws.

Since static timing analysis is not well supported in many FPGA synthesis tools it is often neglected by the designers. In the best case the static timing is analysed in the place and route tool. Why should there be a difference between ASIC and FPGA design methods in this case?

Clock skew is something many associate with ASIC design only. This is however also a life or death issue in FPGAs. Has anyone ever done a design with eight clocks on an FPGA that only has four clock buffers? At least four of the clocks will have a problems with skew.

Low power design is something not discussed when talking about FPGA designs, but with increasing clocking frequencies this is becoming a reality in the space segment. The same issues need therefore to be tackled as for ASIC design.

Once again, because its reprogrammable and no NRE is involved, there seems to be less interest to get it right the first time for FPGA designers. It seems that some FPGA vendors encourage designers to take their designs to the lab and begin testing as soon as possible, even unofficially claiming that simulation is a waste of time. The question is how do you locate a bug in a million gate FPGA chip featuring hundreds of pins?

4.3 Comparison with software development

The favourite argument from designers and managers is that FPGAs should not be treated as ASICs since they are more like software. The obvious answer is that if you want to treat FPGAs as software you should also follow the mandatory software engineering standards that apply to any space applications. This will most often discourage any further arguments along this line.

4.4 Guidelines and requirements

Although the European Space Agency (ESA) never explicitly wrote any design guidelines for FPGAs in the past, there have been at least two documents around in the last decade that are applicable to FPGA design. The *VHDL Modelling Guidelines*, RD2, were originally written for ASIC developments but are relevant to any FPGA design using VHDL as well. The design flow described in the *ASIC Design and Manufacturing Requirements*, RD1, are applicable to any FPGA development, although some parts can obviously be omitted.

A new document will replace RD1 and will be part of the *European Cooperation for Space Standardization* (ECSS) frame work. The document is named *Space Product Assurance, ASIC Development.* This document is currently only available in draft form and can be obtained from ESA on request. The document will explicitly cover FPGA development.

Finally, please read all available documents from the FPGA vendor. An engineer once asked the following during a design review: *Do you seriously expect us to read all these pages?*

5 DEVELOPMENT METHODOLOGY

The FPGA development methodology presented in this section is based on the *ASIC Design and Manufacturing Requirements,* RD1, and the *Space Product Assurance, ASIC Development,* RD3, documents. The methodology has been modified and enhanced to reflect what is specifically required for FPGA developments. There is no claim that the presented methodology is complete or exclusive. It should be used as a guideline and inspiration for preparing more detailed and powerful in-house development methodologies.

5.1 Design initiation

The objective of the design initiation is to establish a base for the FPGA development covering requirement collection, feasibility study and risk assessment. This stage should normally be performed before starting the actual FPGA development. It can be performed either by the customer or by the design house.

5.1.1 Requirements

An FPGA Requirements Specification should be established, defining all relevant system configurations and characteristics to a level allowing the FPGA device requirements to be derived. All system-level issues imposing requirements on the FPGA device should be covered. Thereafter the requirements on the FPGA device itself should be established, covering but not limited to the following:

- functionality and operating modes;
- performance and operating conditions etc.;
- identify all sources of requirements (interfaces of the FPGA to its system, environmental and radiation constraints according to the intended mission);
- consider protocols to external devices, including memory mapping;
- consider the proposed system partitioning;
- respect the overall power budget;
- define the operating frequency range;
- respect specific electrical constraints (e.g. supply voltage, power supply, drive capability);
- consider the system testing capabilities;
- derive functional specifications;
- define items for concurrent engineering.

Although it is recommended that each requirement is made traceable throughout the development phase, it has been experienced that individually numbered requirements can lead to hazardous simplifications. In one rare case the only requirement regarding the error correction capability of a microprocessor system was stated as *"It should have an EDAC"*. Obviously this is not sufficient as a requirement and needs to be extended with information on error detection and correction capabilities, data widths, effects on system performance etc. In the example case the numbered requirement was propagated throughout the design and verification stage and ended up as an item to be ticked off in a verification plan. The test bench verifying this item was developed and everybody was happy. The problem was that the test bench run perfectly well reporting no errors even if the EDAC was not included in the design. The point here is that requirements are sometimes difficult to catch in a single sentence and require more work, both when being specified and verified.

5.1.2 Feasibility study

The feasibility of the FPGA development should be studied, based on foreseen development conditions, and be documented in a feasibility report. The suitability of the selected FPGA technology for the foreseen application or mission should be assessed. A trade-off between implementing the required functionality in FPGA or ASIC technology should be performed. The purpose of the report should be to allow the feasibility of the development to be judged.

The feasibility report should include but not be limited to:

- availability and quality levels of the potential FPGA technologies;
- possibility that the requirements can be met by the potential FPGA technologies and whether the design will reach any technical limitations such as complexity, operating frequency and packaging limitations such as the number of pins.
- check the definition status of the given ASIC requirements. Assure that these requirements are settled and ambiguities are avoided, if possible;
- reflect the targeted power consumption and the speed performance;
- check the availability for the intended package; Conduct a preliminary worst case timing analysis of external interfaces and clock domains;
- check the availability of all required design tools and libraries;
- check the qualification status and the radiation performance of suitable processes and compare them with the system requirements;
- ensure that the checked processes have a remaining lifetime coinciding with the expected procurement stage;
- check that the system partitioning is reasonable and fixed.

5.1.3 Risk analysis

A risk analysis should identify critical issues and possible backup solutions, including but not limited to:

- maturity of foreseen FPGA manufacturers, including tools, design libraries supported;
- availability of engineering resources and their experience and familiarity with the design type, tools, technology and the potential foundries;
- stability of the requirements and possibility of requirement changes.
- underestimation of design and verification effort;
- underestimation of debug and repair efforts;
- overestimation of actual gate capacity and clocking frequency;
- understanding of the synthesis tools/libraries;
- understanding of the timing/simulation tools/libraries;
- technology suitability for mission and its duration
- total ionising dose requirements for given mission
- SEU susceptibility for given mission
- suitability of SEU protection mechanisms;
- irrecoverable lock-up states due to SEUs;
- undetermined I/O behaviour during power-up;
- high in-rush currents at power-up;
- stability of FPGA configuration for SRAM based devices;

- failure rate after programming (burn-in), life time aspects;
- component availability.

5.2 Initial analysis

The objective of the initial analysis is to establish a detailed functional specification and an FPGA development plan. The development plan is sometimes established already in the proceeding design initiation stage and need not be performed within the actual development.

5.2.1 Specification

As for all design disciplines, proper functional specification of each FPGA should be established before implementation begins. In case no functional specification is provided for the FPGA, it should be established in accordance with the outline listed hereafter. If a functional specification is provided, a critical review should be performed and incomplete or contradictory information should be identified.

The purpose of an FPGA functional specification should be to fully and clearly specify the device to be designed, fulfilling all requirements of the preceding FPGA requirement specification. In case a requirement cannot be fulfilled, the discrepancies should be fully documented. The functional specification should be self-contained and there should be no need to consult other documents in order to understand what should be implemented, except when external interfaces are specified to be compliant with certain identified standard protocols etc. It should be structured to easily be transformed into a data sheet or user manual for the FPGA. Trade-off discussions should be avoided in the functional specification, except when background descriptions are necessary for the overall understanding of the functionality. Issues unnecessary reducing the design space available for the architectural design should be avoided.

All functions and requirements should be specified in a clear and precise manner, including but not limited to:

- Description of all foreseen system configurations in which the device can be used, including external devices and interface protocols;
- Bit numbering and naming convention (to be maintained throughout the design flow);
- Format of data structures;
- Functionality, including operational modes and requirements for system test;
- Protocols and algorithms used;
- Error handling;
- Definitions of programmable registers and their initialization;
- State after reset for signals and internal registers;
- Identification of functions not included in the device, to avoid potential misunderstandings;
- High-level block diagram indicating data and control flows;
- Specification of all interfaces and signals, including power supply and test pins;
- Operating conditions;
- Specification of all electrical parameters;
- All target timing parameters with corresponding waveform diagrams;
- Baseline FPGA device and package.

The baseline FPGA technology and device to be used should be selected. In case a novel device is proposed, the consequences (qualification cost and schedule, radiation effects etc.) should be analysed and documented. The remaining life time of the selected technology should be confirmed by the foundry. Unless already performed, the feasibility w.r.t. device complexity, packaging, operating frequency and timing should be assessed.

5.2.2 Development plan

The purpose of the development plan should be to allow an assessment of the proposed development strategy. It should include issues applicable to the work to be done in the FPGA development activity. The development plan should include, but not be limited to:

- Design flow, e.g. for design capture, logic synthesis, back annotation etc.;
- Design analysis and verification approach;
- Identification of the baseline FPGA technology, device family and package;
- Identification of all design tools to be used, including their version and platforms. The availability of each tool should be explicitly stated. The status and availability of the FPGA libraries for each of the tools to be used should be assessed;
- Specification of in which formats design data will be transferred between design tools;
- Verification and stimuli generation approach, including code coverage verification approach;
- Identification of programming equipment;
- Validation approach.

5.3 Architectural design

The architecture of the FPGA design should be defined by partitioning the functions on the block level, clearly specifying the functions, interfaces and interactions of each block. The work should take into account the subsequent implementation.

As has been discussed in the preceding lessons learned section, it is difficult to justify not to use VHDL as design entry when developing FPGAs today. VHDL should be used even if performance might be better using gate-level pushing, since subsequent modifications, re-use and analysis is very much simplified.

The VHDL model should be developed according to the requirements for VHDL models for component simulation as specified in RD2. A neat thing is to repeat the requirement numbers from the FPGA functional or requirement specification in the VHDL source code for easy reference.

Verification plan should be established, defining how the functionality of the design should be verified. VHDL board-level simulation should be performed using the developed VHDL model in one or several VHDL test benches, demonstrating all operating modes and functions of the device in accordance with the verification plan. Surrounding components need only incorporate functions and interfaces (including timing modelling) required to properly simulate the device.

Using the above VHDL test benches, a test suite verifying the full functionality should be developed and its code coverage verified as specified in RD2. The verification should follow the established verification plan and should be performed by somebody other than the designer, to avoid that a mistake in the design is masked by the same mistake in the verification.

The timing of the external interfaces should be estimated based on the selected FPGA technology and the design complexity. Worst case timing analysis of the external interfaces should be performed, taking into account timing effects of the printed circuit board layout.

The feasibility should be re-assessed, considering issues such as complexity, packaging, operating frequency, metastability and other possibly critical issues. A full analysis need only to be performed for potentially critical cases.

Finally, the results of all work should be documented in an architectural design document. The purpose of the document should be to allow the proposed architecture to be analysed. All results should be documented to such level that the detailed design can commence using only the functional specification and the architectural design document as the specification of the design. There should be a clear correspondence between the functional specification and the architectural design document. Information already included in the functional specification need not be duplicated. An introduction to the architecture should be given, including: block partitioning and its rationale; internal protocols and data formats; interfaces and protocols to external devices, including memory mapping. Thereafter the architecture should be described in detail, including: block diagram corresponding to the actual decomposition of the design and all signals, together with a description of the block interaction; purpose and overview of functionality; detailed functionality, including description of finite state machines using graphs, type of state machine encoding and handling of unused states.

The verification of the architecture should be documented, including textual descriptions of the verifications performed and results obtained, descriptions of all stimuli and test benches and the VHDL code coverage results.

5.4 Detailed design

The FPGA gate-level design should be performed for the selected process and library, fulfilling the requirements of section 6. This is normally performed by synthesising the previously developed VHDL model of the design.

The FPGA gate-level design should be verified against the VHDL model, using the complete test suite from the architectural design; the function should be proved to be identical on a clock-cycle basis for best and worst case simulation. Any discrepancies need to be approved by customer. The VHDL model should then be updated w.r.t. functionality and timing parameters, and the complete test suite re-run. Board level worst case timing analysis should be performed and documented, taking timing parameter input from the FPGA gate-level verification described above. For designs with multiple FPGAs, critical interaction between FPGAs should be verified by means of simulation. If possible, equipment level simulation should be performed.

In order to establish the timing margins, static timing analysis should be performed, or alternatively the design should be simulated at higher operating frequency using appropriate stimuli. The worst case timing analysis of the external interfaces of the FPGA should be updated. Log files should be stored to allow independent analysis after the simulation campaign.

The design should be analysed with respect to the requirements expressed in section 6, which should be documented. Any non-compliancies w.r.t. the requirements and the functional specification of the FPGA should be clearly identified and agreed with the customer. A Failure Mode, Effects and Criticality Analysis (FMECA) should be performed on the board level, taking the FPGA components into account. An SEU analysis should be performed. Radiation effects need only be simulated in case the appropriate tools and libraries are available.

The purpose of the detailed design document should be to document that all requirements on functionality and design methodology have been met, including documentation of simulation and analysis results. All information necessary for the subsequent work, such as place and route and programming, should be clearly expressed. In addition, all information necessary to allow a possible future redesign or re-targeting to another FPGA technology should be included, to an extent allowing it to be performed by another designer. The detailed design document should be updated after the FPGA place and route stage. To avoid potential errors, information from already established documents can be referenced using a complete reference. However, in case information is only partly correct or applicable, it is better to include a fully correct version.

The gate-level simulation should be documented, including a textual description of the stimuli and expected responses. Analysis results of error and warning messages from the simulation should be documented. The comparisons between the VHDL and the gate-level simulation results should be documented. A summary of the gate-level simulation results should be included and a verification matrix should be established. If the VHDL model has been modified, the results of test suite re-run should be included.

The organization of the complete design database should be fully described, including the file structure, naming conventions and version control. All tools and FPGA libraries actually used during the detailed design should be documented and possibly archived for future use.

5.5 Place and route

Place and route should be performed using FPGA vendor approved tools. The use of constraint files should documented.

Post place-and-route simulation should be performed using the complete previously developed test suite to verify the proper operation. In case the design has been modified during the place-and-route, e.g. the netlist has been rebuilt, the simulation should be performed on the modified design. The function should be proved to be identical on a clock-cycle basis for best and worst case simulation for the complete test suite.

The post place-and-route timing margins should be established in the same way as in the detailed design stage. The clock skew should be analysed.

5.6 Programming

The programming file should be generated from FPGA vendor approved tools. Programming of the FPGA should only be made using equipment approved by the FPGA vendor. The equipment should be properly calibrated.

It should be noted that some equipment is approved for commercial or prototype devices, but is not approved for the programming of flight devices. As an example, in a specific satellite project the programming of the flight part was made using the Silicon Sculpture equipment from Actel, although this is strictly not recommended. This recommendation was made because the equipment did at that time not adhere to Actel's part failure support analysis. However, for correctly programmed parts this did not have any adverse effect on the neither the quality of the device nor the validity of the programming. It is however not recommended to repeat this procedure.

The successful programming of the device should be verified using FPGA vendor approved tools and methods. The fuse resistance for each part should be logged before programming, after programming and after burn-in. Blank checks should be performed. Lot failure analysis procedures should be followed in case of part programming error. Checksum verification should be used to ensure correct programming file usage. All programming files and log files should be stored for each programmed part.

5.7 Design validation

A validation plan should be established, where possible reusing parts of the verification plan. The programmed FPGA devices should be validated according to the validation plan, by designing and building a breadboard representative of a realistic application and performing validation tests covering the full functionality and all operating modes of the device. For generic FPGAs, the design should be validated beyond the obvious application cases to avoid problems when the design is used in a new environment. The results should be documented in a report. The functional specification should be updated based on the experience gained from designing the breadboard, the validation, and other possible sources.

The die differences between commercial and flight FPGAs should be taken into account when design validation is performed. It is recommended that design validation is performed on the actual flight die as far as possible to avoid any late surprises due timing differences etc. Any validation performed under thermal/vacuum conditions should be done on the flight die only.

5.8 Production and burn-in

The programming of flight parts should be done under clean room conditions obeying ESD regulations etc. Burn-in should be performed, preferably with functional activity going on in the FPGA during the burn-in. The functional testing capability of the programming equipment should be used as far as possible after the burn-in to verify the correctness of each FPGA device. Note that burn-in of FPGAs is normally not recommended by Actel for their devices, RD19.

5.9 Documentation

All documentation should be in English. The documentation should be well structured and easily readable, so that the design can be understood by an FPGA designer not being involved in the design work. Names of signals, blocks etc. should be chosen to indicate their function, and should be based on the English language. The documentation should be consistent, e.g. the same item should have the same name in all documentation. Block diagrams, control flow charts, timing diagrams and figures should be introduced where beneficial for the understanding of the text.

Every time an updated document is delivered it should include a detailed change list, and all significant changes marked using a change bar in the margin. If a document is delivered before being complete, each page with incomplete information should be clearly marked. Some of the information listed is normally better delivered separately or in appendices, such as layout plots, gate-level schematics, lists of undetected faults etc.

All documentation, covering both the documents and the design data base, should be under configuration control. Automatic version control should be used if possible for the design data base. Intranet could be used for distributing design data and documentation within the design and production teams to avoid the use of superseded copies.

All files produced by the design tools, such as log files and waveforms, should be stored, even if possible to be regenerated at a later stage. This is to ensure that independent reviews can be performed without the need to regenerate the design work, which can sometimes be impossible due to design tool changes and upgrades.

5.10 Reviews

Formal reviews should be held at the completion of each design stage. The individuals partaking in the reviews should be experts in the field of FPGA and ASIC design, as well as personnel responsible for the system, equipment and board in which the reviewed FPGA belongs. The development schedule should allow for potential re-design work after each review since few designs are without problems.

The objectives of the reviews should been to assure functionality and to avoid repeating known mistakes. One should realise that all reviewers will not be proficient in VHDL and FPGA design. It is important that all available design information is provided to the reviewers, in particular VHDL source code and synthesis scripts, code coverage, etc. Independent reviewers should be employed where possible. A design should be reviewed from Dr. Rod Barto’s perspective: *Every circuit is considered guilty until proven innocent.*

Although many companies have a clear distinction between FPGA designer, normally fetched from the ASIC development team, and board designer, it is important that no water tight responsibility barriers are built up between the two teams. The FPGA designers should be integrated with the equipment and board development teams to assure that the final design will meet all equipment level requirements. This is should however not be interpreted as a recommendation to have combine the FPGA and board designer in one individual.

5.11 Relief

Although the preceding sections describe a rather loaded design process, it is actually not that difficult to design and document an FPGA properly. The point is that one should think through the whole development process beforehand and act accordingly. Many of the required documents can be combined into one without losing any information, e.g. one requirement/ functional specification and data sheet, one architectural and design document including verification, one feasibility and risk report. The documents should be updated after each development stage and be kept alive throughout the development process.

6 DESIGN METHODOLOGY

The FPGA design methodology presented in this section is based on the *ESA ASIC Design and Manufacturing Requirements*, RD1, but has been extended and oriented towards design issues relevant to FPGA design.

6.1 Reset

There are several concerns about the reset of an FPGA device that need to be taken into account during the design of the device as well as of the board or equipment in which it resides. Some of the main points have been summarised hereafter.

6.1.1 Reset input

The reset input to an FPGA device is normally derived from some kind of power-up resistor-capacitor network which de-asserts the signal after a predefined time after power up. The time constant is normally set to cover the power up time of the FPGA and any oscillator or other circuitry generating the system clock of the FPGA. The nature of the reset signal is therefore not purely digital and has often rise and fall times that are outside the specification for the FPGA technology. The normal approach is to filter then reset signal externally with a Schmitt trigger inverter to avoid having an input signal with a slow rising edge that upsets the maximum rise time requirement. The Schmitt trigger inverter also protects the FPGA from a potentially damaging inrush current.

In one particular flight design the power on reset circuitry was designed using an output of the FPGA to determine when the device actually was power up before de-asserting the reset. By assuming tri-state outputs until the FPGA was powered up, it was believed that the circuit would maintain the reset active till the FPGA began to drive a high voltage on the fixed logical '*one*' output. This approach has some major flaws: there might be a potential current going into the FPGA through the output at power down; the actual reset input might act as an output during power up; and finally the application note upon which the design was based wrongly stated that the output would be in tri-state till the FPGA was properly powered up since subsequent lab tests showed that the output was driven to a high voltage level early than expected.

It is a matter of design style whether to use asynchronous or synchronous reset at the level of each individual flip-flop. This design style selection should not be confused with the synchronisation of the actual input reset signal and will be discussed somewhat later. To ensure the proper timing of the reset input signal on each flip-flop the signal needs to be synchronised with respect to the clock used to clock the flip-flop. This is however not always straight forward since sometimes there are actually no clock edges to synchronise with, as was described in the failure investigation report RD5. An elegant solution is to assert the internal reset signal asynchronously and to de-assert it synchronously with the clock. In this way the effect of the reset will be immediate, and the release will only occur when there is a functional clock signal. This solution is easily implemented using two flip-flops through which the input signal is synchronised and gated with the input signal itself before distributed to the rest of the flip-flops in the design. As long as both signals are not de-asserted, the design should stay in the reset state. The recovery timing critical de-assertion is timed by the two reset flip-flops. If additional delay is required, it should be implemented before the aforementioned common gate.

Coming back to the asynchronous or synchronous reset at the level of each individual flip-flop, each of the two approaches has its own benefits. Using synchronous reset does not pose any specific requirements on the routing of the reset signal since it is routed similarly as any of the functional signals. However, it requires a clock edge to reset the flip-flop. Using synchronous reset instead of asynchronous requires a justification since it could cause simulation problems when logic synthesis has been used. Using asynchronous reset allows an immediate reset of the flip-flop, but will most often pose tight timing requirements on the routing of the reset signal.

A common way of meeting the above timing requirement is to use one of the internal clock buffers in the FPGA for the reset signal. A recommendation is to instantiate the internal clock buffer directly in the VHDL code to ensure that the buffer will actually be in place after synthesis and to control the reset signal path in order to handle its skew and propagation delay. In many architecture the clock buffer is shared between internal or external usage, depending on programming of the FPGA. It is then required that the corresponding clock input pin of the clock buffer is electrically well defined even if only used internally.

For outputs that are critical for the system operation it is recommended that the corresponding flip-flops are reset asynchronously. The output could also be directly gated with the input reset signal to ensure the proper output value during reset, but this approach can be plagued with undesired output glitches.

In one specific flight design it was argued for not synchronising the reset input since the timing path between the synchronising flip-flop and the rest of the flip-flops in the design would be longer than the clock period, basically defying the purpose of the synchronisation. It was also claimed that all finite state machine and counters were designed in a way that would prevent a transition from the default reset state to another state immediately on the de-assertion of reset. Thus, the impact of any metastability caused by the completely asynchronous reset input would be small. It is however not recommended to repeat this procedure.

6.1.2 Internal state during and after reset

In general, the design should be fully deterministic and it should be in a fully known state after reset. This means that all finite state machines should be reset, all control registers should be reset etc. For some design it is neither necessary nor feasible to reset all flip-flops since they might be used as memory elements that are not read from until written to etc. The state during and just after a reset should be documented in detail.

6.1.3 Outputs during reset

All flip-flops driving output signals should be reset. Outputs that could activate external units should be de-asserted during reset, e.g. chip select and write strobes to memories. Bi-directional data and address buses should be placed in a high impedance state to avoid bus contention during reset. Specific attention should be paid to this issue, especially in the case the reset signal is internally synchronised and there is no asynchronous reset path since bus contention could occur between the time the reset is asserted and the next active clock edge occurs.

6.2 Clocks

An ineptly chosen clocking strategy for an FPGA can often lead to down stream problems that could have been avoided earlier on in the design stage. The most common concerns about clocking are discussed in the subsequent sections.

6.2.1 Synchronous design

The baseline approach should be to design an FPGA that is fully synchronous. Asynchronous solutions might be needed in many cases but should then be limited to areas where a synchronous scheme cannot be implemented. A synchronous design adheres to the following definitions. The number of clock regions should be minimised to reduce since FPGA normally have only a few dedicated clock buffers. Every latch and flip-flop within a clock region should be connected to the same clock source with only buffers inserted for the purpose of increasing the driving strength or using another clock edge. Clock gating should be avoided as far as possible, i.e. no logic functions or memory elements are allowed in the clock signal path.

Control and data signals between different clock regions should be synchronised. In some cases it is sufficient to synchronise control signals only, provided that the data signals are stable during read out in the receiving clock region, which should be possible to derive from the synchronised control signals.

6.2.2 Asynchronous design

A simple definition of asynchronous design is a situation when there is more than one clock from which signals are generated and sampled by. Asynchronous design issues should therefore be considered whenever there is a signal crossing the boundary between clock domains and on most external input interface. Asynchronous design also covers other design aspects which are not considered in this document since they are difficult to master in FPGA designs and should therefore not be used, e.g. asynchronous finite state machines.

The main issue with asynchronous signals is that they need to be synchronised with respect to the FPGA clock. The input signal should be synchronised before being used for any decision involving more than one flip-flop, since each flip-flop could interpret the state of the incoming signal differently. As a rule of thumb, each such input signal should be synchronised with at least one flip-flop before being propagated to the rest of the circuit.

In some cases it is required that the input signal is synchronised towards the opposite clock edge (as compared to the rest of the logic is clocked on) to reduce the delay induced by the synchronisation. Note that any type of sampling will always introduce quantisation uncertainties.

Whenever a clocked flip-flop synchronizes an asynchronous signal, there is a small probability that the flip-flop output will exhibit an unpredictable delay. This happens when the input transition not only violates the setup and hold-time specifications, but actually occurs within the timing window where the flip-flop accepts the new input. Under these circumstances, the flip-flop can enter a symmetrically balanced transitory state, called metastable.

Metastability is unavoidable in asynchronous systems. This extra output delay time is called the metastable resolution time, and can be much longer than the normal clock-to-output delay. To avoid the effects of metastability, the input signal is normally synchronised with two flip-flops (some times with only one). Metastability problems also occur on signals that transit between different clock domains. Formulas for metastability calculations can be obtained by the FPGA vendors, e.g. RD28. The possibility of metastability phenomena should be reduced. The impact of metastability failures should be assessed and documented for each design.

Any form of clock gating induces a level of asynchronous design and should be carefully considered before used. The device function should not be dependent on internal signal delays. An item affected by the above is the memory latches which are often clocked by a locally derived signal that in principle forms its own local clock domain.

6.2.3 Distribution of clock signals

Clock signals need to be distributed inside the FPGA device in a safe and stable way. The clock tree for each clock region should be optimally balanced, this is best achieved using a pre-routed or hard-wired clock signals in the FPGA. If clock tree branches with shorter or longer delays are used (e.g. for decreasing the clock-to-output delay) the impact should be analysed, which can be the case for clock trees not using per-routed clock signals.

The skew should be carefully analysed for any type of clock tree. It should be noted that even the pre-routed or hard-wired clock signals have a certain level of skew. Note also that some flight FPGA devices show a larger skew than their commercial counter parts. Contrary to what is sometimes believed, skew is independent of clock frequency and can ruin a design running at a frequency as low as a single Herz. Clock skew results in hold timing violations and is predominately visible in long shift registers. An excellent discussion on clock skew can be found in RD4.

6.2.4 Coding style

Although synchronous clock design is intended there are VHDL coding styles that might lead to unwanted results. In one specific design, the VHDL process sensitivity list of the synchronous part contained other signals than the usual clock and reset signals. This kind of coding style can lead to simulation and synthesis mismatches. A mixture between combinatorial and sequential logic in the same process should be avoided for the same reason as above. It is therefore recommended to follow simple VHDL source code templates that are available from the FPGA vendors and synthesis tool provides.

6.3 Power

Although power is a scarce resource in many space application it is often ignored during the design of flight FPGAs. For all FPGA designs, the power consumption should be kept in mind. There are several different approaches that can be taken to reduce power consumption at the VHDL source code level. The simplest approach is to reduce signal switching when not necessary. More advanced approaches require clock signal manipulations such as dividing, gating and stopping, that are in conflict with synchronous design methods and should be avoided for main stream applications. If such design methods are to be used, a trade-off versus using a fully synchronous design, including a risk assessment, should be performed. Power calculations should be performed for every FPGA design. The FPGA vendors sometimes supply spread-sheets, e.g. RD16, as a complement to the power calculation formulas provided in the data sheets.

One should always pay careful attention to the power up and down sequences of FPGA devices, since some technologies exhibit an uncontrollable behaviour on their input and output pins. The behaviour is dependent on the supply sequencing etc., one should ensure that no critical external interfaces are dependent on a deterministic power behaviour of the FPGA. One should carefully define and document the power-up/power-on-reset sequence, strictly following the FPGA vendor guidelines.

Attention should be paid to the differences in supply voltages for various parts of the FPGA. There are often differences between the core and input/output voltages that must be considered during the design. The ordering of powering up and down the different supplies must often be taken into account not to damage the devices. Some FPGA devices have a separate voltage to defined the input signal tolerance level that must be set in accordance with the expected electrical signal environment of the design. The number of power supply connections should be sufficient to guarantee the proper operation of the device in all foreseen applications. Maximum ratings should always be respected.

6.4 Interfaces

A functionally correct design can easily be wrecked by ignoring proper design of its electrical interfaces. FPGAs are often used as the glue logic between an application and its environment, implementing the interface circuitry, which requires that special care is taken of inputs and its outputs.

The input signals are often not arriving from pure digital devices, sometimes being even generated by analogue circuit networks. The voltage levels of the FPGA inputs must be respected, as must the rise and fall times of the incoming signal. It is common that the latter is ignored, resulting in input signals with rise and fall times longer than the maximum time recommended by the FPGA vendors. A simple way to reduce the risk for violating the rise and fall times is to place a Schmitt trigger inverter between the analogue source and the FPGA, which will also act as isolation. At least one case is known in which analogue signals were interfaced to the flight FPGA without any kind of protection in between. As it can be expected, the FPGA device was blown to pieces at least once due to not so careful handling.

Even when a design is synchronous, it sometimes happens that the outputs from the FPGA are generated by means of a combinatorial logical that could produce unintentional glitches on the outputs of the device. It is therefore recommended that all output are driven directly from a flip-flop, or from a multiplexer for which the selector signal is static during an operational phase. It should be noted that the combinatorial voter of a Triple Modular Redundancy flip-flop can often lead to glitches.It is also possible to filter the output signal at the receiving end to mitigate the effects of the glitches, but this should be avoided since it could hamper design reuse. Glitches on the outputs can also be produced during the start up of an FPGA due to several reason, e.g. power-up phenomena as discussed in section 6.3 or during the time period before the FPGA is fully configured as is the case for re-configurable devices. As a rule, it should be ensured that no output will produce glitches during normal operation.

Bus contention occurs when two devices attempt to drive the same bus with different logical values. The effects of bus connection can be catastrophic if its duration is lengthy. It should therefore be ensured that bus contention (or collision) cannot occur, internal as well as external to the FPGA. Special care should be taken during power-up and reset. Leaving nodes floating will however increase power consumption and should also be avoided. It should therefore be ensured that tri-state signals are always driven except for a short period (typically a half to two clock cycles) when switching between drivers, internally on the device as well as for external signals and buses (eliminating the need for external pull-up or pull-down resistors).

A major difference between using an ASIC and an FPGA is that the latter comes with several special pins that are not used by the application, but still need careful consideration during the design of the electrical board. Since this document is limited to once-only-programmable devices, all the following cases will be based on various FPGA devices from Actel Corporation. In general, it should be ensured that all special mode/test/unused pins are terminated properly, strictly following the FPGA vendor guidelines.

Unused user pins, or I/Os, should normally be left unconnected, since any unused pin would be programmed as an output by the place and route software. Refer to the FPGA vendor documentation for each specific technology. Pins marked as non connected, or NC, should be left unconnected. It is not recommended to connect I/O or NC pins directly to power or ground. Document properly the state of the unused pins.

The Actel MODE pin should be tied to ground in most applications. Please refer to RD4 for more details and examples of how this pin can be misused.

Input user signals used for configuration should normally be connected to power or ground by means of resistors instead of straps. The latter is potentially dangerous since the inputs can actually be driven from the FPGA part under unfortunate power up sequencing.

For the Actel DCLK and SDI pins, please refer to the corresponding data sheet and to RD4 for more details. A rule of thumb is that the two pins should be ground terminated (10 kOhm) if unused.

The Actel VPP pin should be tied to VCC for proper operation. Please refer to RD4 for more details and examples of how this pin can be misused.

The Actel PRA and PRB probe pins should be left unconnected, or only connected to a test pin, if the design has been configured for restricted probe pin usage in the place-and-route tool.

The IEEE JTAG 1149.1 pins should be handled with care as discussed in RD4. It is recommended, pin count permitting, to always configure the FPGA for restricted JTAG pin usage in the place-and-route tool. In this way there is no sharing between JTAG pins and user I/Os. The TCK, TDI, TMS and TRSTB inputs should then be terminated to ground via a resistor when not in use. The TDO output should be left unconnected.

It is advantageous to design an electrical board in such way that different devices from the same FPGA family can be used without a board redesign or layout. Although upward compatibility is often foreseen by the FPGA vendors, it is not always that it is fully supported and requires some work by the designers to succeeded. One should pay special attention to the power supplies since voltage levels might change between devices. One should look out for new special pins such as additional clock inputs etc.

The documentation of the pinout for a FPGA design should be made in way that leaves little room for interpretation mistakes. One successful and tried approach it for each pin to list the pin number, the FPGA vendor name for the pin (e.g. MODE), the design name of the pin (e.g. Address11), and finally to provide an indication how the signal should be connected on the board (e.g. connect to ground serially via 10kOhm).

6.5 Unsuitable and Unnecessary Circuitry

The design should not incorporate more circuitry than necessary. In particular, unwanted redundant and unused circuitry should be removed where possible since it decreases the testability and device reliability.

Redundant logic is often produced by the synthesis tool to meet timing requirements. This can include replication of sequential logical, which will be discusses in section section 7.3.4.5, and insertion of buffers in the combinatorial logic.

Library elements and circuitry unsuitable for the intended environment should not be used unless necessary. Dynamic logic should not be used. Care should be taken when using special circuitry such as for Phase Locked Loops and memory blocks. No cell inputs should be connected to the power supply rails, except where necessary for the functionality. Please refer to FPGA vendor for details.

6.6 Testing

Although it is not commonly performed, it is recommended that FPGAs are tested individually after being programmed, just as it is done for ASIC devices after manufacture. The programming equipment often supports some level of basic testing, acting in a similar way as a automatic test generator used by ASIC manufacturers. The FPGA designs should therefore be developed with testing in mind, covering controllability and observability.

6.7 Developing FPGAs using VHDL cores

The experience so far has been that commercial VHDL cores are very seldom, if ever, designed taking design guidelines applicable to space application into account. A proper review of any VHDL core is required before use in a flight FPGA. One should keep in mind that some modification of the VHDL core will most probably be required before usage.

The first serious in-house attempt made by ESA to design an FPGA using cores was a packet telemetry and channel encoder to be used in an in-flight transponder experiment. One conclusion from this development was that properly developed cores are easier to use than code that was not developed with reuse in mind. In this case it was more efficient to start from scratch, using previous design knowledge, experience and verification infrastructure, than try to comprehend the old source code written in a foreign HDL. A second conclusion drawn from this development was that modifications of a core are often required to overcome the low density properties of present flight worthy FPGAs.

7 SINGLE EVENT EFFECT MITIGATION

The main thing that differs between designing FPGAs for flight and commercial application is the target environment. The essential aspect that can be affected by the FPGA designer is the susceptibility of the design towards Single Event Effects (SEEs). Simplistically seen, all other problems are related either to process technology or to the electrical board design. This is the motivation for this extended section on single event effect mitigation.

7.1 Definitions

There are a number different types of hazardous effects that can be caused by energetic particle. A summary of the definitions is listed hereafter (compiled by Mr. Richard Katz, NASA):

- A Single Event Upset (SEU) is a change of state or transient induced by an energetic particle such as a cosmic ray or proton in a device. These are soft errors in that a reset or rewriting of the device causes normal device behaviour thereafter.
- A Single Event Disturb (SED) is a momentary disturb of information stored in memory bits.
- A Single Event Transient (SET) is a current transient induced by passage of a particle, can propagate to cause output error in combinatorial logic.
- A Single Hard Error (SHE) is an SEU which causes a permanent change to the operation of a device. An example is a stuck bit in a memory device.
- A Single Event Latchup (SEL) is a condition which causes loss of device functionality due to a single event induced high current state. An SEL may or may not cause permanent device damage, but requires power cycling of the device to resume normal device operations.
- A Single Event Burnout (SEB) is a condition which can cause device destruction due to a high current state in a power transistor.
- A Single Event Gate Rupture (SEGR) is a single ion induced condition in power MOSFETs which may result in the formation of a conducting path in the gate oxide.
- A Single Event Dielectric Rupture (SEDR) is basically an anti-fuse rupture.
- A Single Event Functional Interrupt (SEFI) is a condition where the device stops operating in its normal mode, and usually requires a power reset or other special sequence to resume normal operations. It is a special case of SEU changing an internal control signal.
- A Multiple Bit Upset (MBU) is an event induced by a single energetic particle such as a cosmic ray that causes multiple upsets or transients during its path through a device.
- A Single Event Effect (SEE) is any measurable effect to a circuit due to an ion strike. This includes (but is not limited to) SEU, SHE, SEL, SEB, SEGR, SEDR and SEFI.

This document will mainly concentrate on SEUs induced in flip-flops. Note that failures caused by SEUs due to changes in the FPGA configuration are not discussed in this document.

7.2 Specification

As for all engineering disciplines one should start with expressing the requirements. In the case of SEU mitigation, the overall specification of bit error rates should be derived from the mission and orbit parameters and spacecraft system study. When the environment and bit error rate is known for a certain mission, the effects on the selected FPGA technology should be assessed and a protection strategy should be devised. An SEU analysis should be performed and the results should be applied to each particular design. If the error rate is greater than expected, even with the built-in mechanisms in the FPGA technology, additional protection might be required.

7.3 Protection mechanisms

The most common protection mechanisms against SEUs is some kind of redundancy of which some have been listed hereafter (compiled by Mr. Kenneth A. LaBel, NASA):

- Parity, capable of detecting single bit errors;
- Cyclic Redundancy Code, capable of detecting errors;
- Hamming code, normally capable of detecting two and correcting one bit error;
- Reed-Solomon code, capable of correcting multiple symbol errors;
- Convolutional codes, capable of correcting burst of errors;
- Watchdog timer, capable of detecting timing and scheduling errors;
- Voting, capable of detecting and selecting most probable correct value;
- Repetition in time, capable of overcoming transient errors:

For a protection scheme which only detects errors, or can detect more errors than it is capable to correct, the occurrence of an error needs to be propagated to the system for decision making. The reporting and handling of such events needs to be built in to the equipment and be documented. One particular example of how things can be misused is the Built-In-Self-Test functionality in a telemetry encoder that was enable in most applications, but the error flag was always left unconnected. This resulted in a start up delay of about one thousand clock periods but there were no means to observe if a problem was ever encountered during the self test.

In a recent SEU analysis of the critical control equipment on a deep space probe there were two aspects that were considered critical. Firstly, could the system hang up due to an SEU in an FPGA? Secondly, could a critical event be triggered due to an SEU? This illustrates two aspects that should be considered during any FPGA design; functions that could stop working permanently and functions that could be triggered spontaneously. The former can often be globally protected by means of reset or power cycling, whereas the latter requires active protection mechanisms in the design.

7.3.1 No protection

The simplest approach to SEUs is to ignore them completely. This is often based on an optimistically calculated probability of an SEU happening in any critical part of the design. This is not a joke. It has been used in some parts of very critical designs, where it was not possible to protect all parts of the FPGA against SEUs.

It is often misleading to believe that there are parts that are non-critical as will be shown in the following example. In a particular design it was considered safe not to protect certain data registers since they would only affect the value being read out by the external microprocessor. However, a careful analysis showed that if the read out values were too much off their expected levels the software would take an incorrect decision that could be potentially dangerous. In this specific case, the problem was solved thanks to system level redundancy, allowing the software to read out the value from three different sources.

In another case the length of a specific time period could be changed by fifty percent, either way, due to an SEU hitting an unprotected counter. The time change would in the worst case lead to the unsuccessful switching of a relay, which would require sensing and confirmation by the software to ensure that the intended effect had taken place.

Another mistake easily done is to terminate a counter count by means of the explicit relation operator '=' instead of '>='. This difference in VHDL coding style could, in combination with an SEU, lead to the counter entering a state from which there is no recovery. If instead the counter was designed using an implicit wrap-around structure, one SEU could lead to an approximate 100% counting error in the worst case.

Even if no explicit redundancy is employed in a design, it is always wise to ensure that the design will never enter a state from which it cannot recover. Depending on the application, it is sometimes possible to confine the effects of an SEU to a single communication packet or to a single picture, which can be acceptable for some purposes.

7.3.2 Error detection

In some applications it is sufficient to detect an error caused by an SEU and to flag the affected data as invalid or corrupted. There must however be means for communicating the occurred problem to the application or user. There are several types of error detection schemes of which some are listed in section 7.3.

In other cases it is sufficient to mask the effects of an SEU by employing dual modular redundancy, e.g. two flip-flops must have the same value to make any effect on a strobe, using a simple logical '*and*' function. One should however be careful with unintentional glitches on such voted outputs.

7.3.3 Triple Modular Redundancy

A straight forward approach to SEU protection is to use Triple Modular Redundancy (TMR) for all registers. TMR refers to triple voting, a register implementation technique in which each register is implemented by three flip-flops or latches such that their vote determine the state of the register. Note that TMR can also be applied to complete designs or part of circuits, not only flip-flops as discussed in this document.

The inclusion of TMR can be done directly in the VHDL source code, which does not necessarily require a too great an effort. An example of this is the LEON SPARC microprocessor developed by ESA in which flip-flops are made TMR directly in source code.

There are also synthesis tools, such as Synplify from Synplicity Inc., that can replace any flip-flop with a TMR based flip-flop, without the need for rewriting the VHDL source code as discussed in section 7.3.3.2.

Some FPGA technologies, such as the SX-S family from Actel Corporation, include TMR flip-flops directly on silicon as discussed in section 7.3.3.3.

Due to the simplicity of implementing TMR, it is recommended that it is applied consistently in all crucial designs (including data paths, in particular status and configuration registers), especially in those applications for which no power cycling or reset is possible.

However, the use of TMR does not come entirely without problems which will be demonstrated in the subsequent sections.

7.3.3.1 Refresh of TMR

Although TMR based flip-flops can tolerate one SEU, they cannot tolerate a second one before being refreshed. The refresh cycle of the flip-flops can be compared with the scrubbing of memory protected by EDAC. The idea is that any bit error should be corrected before the occurrence of a second error. In order for TMR protection to be efficient it requires that the refresh rate of the flip-flops is high enough to avoid an accumulation of multiple bit errors (assuming that the flip-flops are feed back with the voted result). The refresh period needs to be shorter than the expected bit error period. Ignoring this fact can lead to a TMR based design that is not as efficient as presumed.

In one particular case, a critical register was implemented using a TMR flip-flop to store mission critical information. The value was to be written in as one of the last actions before the complete satellite was placed in long term hibernation which could last as long as a year. But since the TMR flip-flop was clocked by the rising edge of write strobe produced by a microprocessor, the register would not be refreshed during hibernation. This could lead to an accumulation of SEU induced errors in the TMR flip-flop, which would lead to an incorrect result being output. In this example this could have lead to an involuntary modification of the on-board time counter. In addition, using the write strobe as a clock signal for the flip-flops is considered as asynchronous design. In view of the above, it is recommended that TMR flip-flops are refreshed as often as possible to counteract the possibility of SEU induced error accumulation.

7.3.3.2 Automatic protection insertion

The Synplify design tool will be used as an example of automatic TMR insertion, since being known to the authors. Synplify provides three different techniques for protecting flip-flops, of which two are TMR based. For older technologies, the C-C technique uses combinatorial cells with feedback to implement storage rather than flip-flop or latch primitives. The C-C techniques provides an improvement over the usa of unprotected sequential primitives in those technologies. The TMR technique is implemented with sequential primitives and a combinatorial voter. The TMR_CC technique uses a triple-module-redundancy technique where each voting register consists of combinatorial cells with feedback rather than flip-flop or latch primitives. The latter is only used with some older technologies. The Synplify *Syn_RadHardLevel* attribute is used for controlling the insertion of protection mechanisms. The attribute can be applied, globally, on block level or per register. The attribute can either be included in the VHDL source code or in a separate constraints file. If necessary, the attribute can be applied globally to the top-level module or architecture of the design and then be selectively overridden for different portions of the design.

7.3.3.3 Built-in Triple Modular Redundancy

As mentioned earlier, there are FPGA families that have built-in TMR flip-flops which do not require any particular design effort in order to be used. One should however carefully compare the specified SEU tolerance with what has been specified for one’s application. The SEU tolerance levels of the current technologies employing TMR is not high enough to cover all types of missions. Even if the built-in TMR seems sufficient, it is still possible to minimise the effects of SEUs in areas where they could be fatal to the application.

7.3.4 Finite State Machines

The design of synchronous Finite State Machines (FSM) poses some challenges when trying to make them insensitive to SEUs using modern synthesis tools. The main issues is the existence of unused states that could be entered due to an SEU and from which it is not possible to recover. Unused states are formed when there are less used and defined states than any integer of the power of two. For example, an FSM with three defined states; Idle, Read and Write, would be implemented with two flip-flops for sequential or gray encoding, or with three flip-flops for one-hot encoding. In both case there is at least one possible state that is not defined. An SEU could move the state of the FSM into such an unused state.

Good design practice has always been to ensure that there is an exit path from all unused states. In case the unused state would be entered due to an SEU, the FSM would exit from that state and operation could resume, without a risk for dead locking the FSM. There are several ways of protecting an FSM to enter an illegal state due to SEUs. Unfortunately, most of these constructs are optimised away by the synthesis tool, since the constructs are inherently redundant and the synthesis tool’s job is to remove unnecessary logic.

7.3.4.1 FSM and TMR

The simplest and most extensive SEU protection for FSMs is to implement all state registers using TMR based flip-flops. Since at least two SEUs are required to upset the register, the likelihood for the FSM entering an unused state becomes small.

7.3.4.2 When others

If TMR based flip-flops are not used then one has to ensure that the FSM would exit from unused and illegal states by other means. There are several ways in which this could be described in VHDL, but the problem is that modern synthesis tools have a tendency to remove all such redundant precautions. The basic approach is to declare a special case for all unused states, normally using a ‘when others’ statement in a case statement in VHDL. But since the unused states have no entry point from any of the used states, the logic required for exiting from the unused states will be considered as redundant and will be removed by the synthesis tool. There are means of forcing the tools not to do this optimisation as will be discussed further in section 7.3.4.6. The Synplify design tool uses the *Syn_Keep* attribute to prevent flip-flops from being optimised away.

7.3.4.3 Encoding

There are different encoding styles that can be used for FSMs. In some cases the encoding is predefined by the designer in the VHDL source code. In other cases the encoding is left to the synthesis tool in order to achieve the best possible result. Gray coding, in which only one flip-flop changes value between state transitions, is normally considered as safe for flight applications, but it is not always the most optimal solution area and timing wise. One-hot encoding uses the largest number of flip-flops, since each state is represented by one flip-flop, and would thus also run the largest risk for experiencing SEUs.

7.3.4.4 Automatic recovery

There are synthesis tools that provide as an option the possibility to implement automatically logic that will detect if an unused state has been entered and which will reset the FSM to its initial legal state. The implementation of such automatic recovery is not always efficient or attractive and should be carefully analysed before taken in use. In one such specific implementation the problem is solved using flip-flops clocked on the opposite edge of the FSM flip-flops, which is not fully coherent with synchronous design practices and could also limit the operating frequency of the design.

The Synplify design tool uses the *Syn_Encoding* attribute to enable automatic recovery from illegal states. In addition to the encoding value, i.e. *gray*, *sequential* and *onehot*, the *safe* value can be added for the above purposes.

7.3.4.5 Replication

As mentioned earlier, flip-flop replication is used by synthesis tools to optimise the design for timing. The normal flip-flops in a design are replicated to share the load that needs to be driven by the original flip-flops. From SEU sensitivity point of view this poses a great problem.

In one particular case a five state FSM was implemented with three flip-flops, for which all used states were hard coded in the VHDL source code and the “when others” statement had been used. During synthesis, the tool replicated the state registers to eight flip-flops in total. The number of unused states was increased from three to 251. A gate level analysis showed that an SEU occurring while the FSM was in its idle state, in which most of the time was spent, could lead to a number of unwanted results; the functional event sequence could be inadvertently triggered, the FSM could lock up without the possibility for recovery, the FSM could start oscillating between illegal states without the possibility for stopping, and the FSM could enter an illegal state but due to input signal values it could be recovered. In view of the above, the synthesis tools should be directed not to allow flip-flop replication for FSMs if possible.

The Synplify design tool uses the *Syn_Preserve* attribute to prevent flip-flops from being optimised away or being replicated.

7.3.4.6 A safe approach

Since there are several different synthesis tools on the market and new versions are releases ever so often, it is difficult provide a silver bullet for the FSM problem. One attempt has been tried out for the Synplify design tool and is presented hereafter. It is however no guarantee that this approach will work for future releases and the results should therefore always be analysed on the netlist level.

Note that the objective is to design an FSM that will always exit an unused state, independent of input signal values. This statement is important since it is sometimes virtually impossible to analyse a design in such detail that one can determine that an FSM will exit from unused states due to input signal combinations.

The proposed approach is as follows. The FSM itself written with a VHDL case statement and constants are used in the case comparisons. The first version of the VHDL code included an enumerated type for the FSM state signal and the elements of the enumerated type had the same names as used in the case statement comparisons. The Synplify FSM compiler and explorer were enabled to find the optimal coding and the result was gray coding. Without the FSM explorer the result was one hot encoding which was fast, but the gray encoding gave a smaller design.

The state signal was then redefined as a vector of bits. Eight constants were defined with the same names as the elements in the previous enumeration type to avoid rewriting the case statement comparisons. The constants were gray encoded, based on the previously obtained results. The FSM compiler and explorer were then disabled and the synthesis was run without any attributes and the result was smaller than the FSM compiler/explorer result but slower than the one hot encoding. The Synplify *Syn_Preserve* attribute was then applied to the state signal to avoid replication and the *Syn_Keep* attribute was applied to avoid optimization of the intentionally redundant logic that should bring the state machine out from the unused states. The result was smaller and faster than the two other implementations. As for all other protection mechanisms, the results of this approach should be verified at the netlist level.

7.3.5 Optimization during synthesis and place-and-route

As has been discussed earlier, the logical synthesis tools perform optimisation with the aim of removing unnecessary and redundant logic. This is in stark contrast with the objective of adding redundancy to protect a design from SEUs. In addition to removing the logic for implementing the "when others" statement mentioned above, replicating flip-flops and adding buffers, the tools also performs optimisation of which the effects sometimes are not obvious and need analysis to be understood properly.

In one particular case, a designer coded an FSM with two flip-flops using all four states. Since the purpose of the FSM was to filter and detected a critical incoming rising edge pulse it was important that the output from the machine was not triggered incorrectly by a possible SEU. When inspecting the encoding of the FSM it appeared that in one particular state it was sufficient to receive a single SEU in order to change state and incorrectly trigger the output. The actual implementation of the output signal was a logical '*and*' of the two state flip-flops, both had to be set in order to assert the output. To mitigate the potential SEU problem, an additional flip-flop was added to the FSM. The idea was to increase the hamming distance between the two aforementioned states from one to two, requiring two simultaneous SEUs to trigger incorrectly an output. The intended implementation of the output was to perform logical '*and*' between three flip-flops, instead of two as previously done. Due to unfortunate encoding, the value of the additional flip-flop had been set to zero for all the three other legal states. This lead to a surprisingly efficient optimisation of the output logic. Instead of looking at all flip-flops, the tool derived that it was sufficient to look at the value of the additional flip-flop only. This is logically correct, since it has the value zero for all other legal and used states and no further comparison is needed. The result was that the design could now trigger an unwanted output due to a single SEU in any of the legal states. The design was thus made more sensitive to SEUs than it had been before.

The place-and-route tools provided by the FPGA vendors are also capable of optimising the number of modules used in the design by recombining the modules and compacting the design.

The gate-level netlist produced by the place-and-route tool is not necessary equivalent to the one that had been provided to the tool.

The conclusion is that it is important to analyse the results of the synthesis and the place-and-route at the netlist level to ensure that the intended SEU protection logic has been implemented. This is the subject of the next section.

7.3.6 Verification of protection mechanisms

The verification of the SEU protection mechanisms can be performed in several ways, of which gate-level netlist inspection and simulation with fault injection are two approaches that have been used in a real satellite development.

The netlist inspection approach consists of analysing the protection mechanism for each flip-flop in a design and can be tedious labour. For each flip-flop the intended protection should be documented and verified by means of logic deduction. The logic should be scrutinised to detect any deviations from the intended implementation. Any suspect findings should be documented and brought to the attention of the designer and the equipment managers. Any unprotected flip-flops should be documented. One of the best analysis seen so far was made by a company that studied how a single SEU in any of the flip-flops could affect any other flip-flop in the design. More simple analysis cases involved only checking that the number of flip-flops had been triplicated due to the TMR insertion.

As a complement to inspection, gate-level simulation can be used to exercise complicated protection mechanisms or part of the design for which it is not obvious how the resulting implementation actually will act under influence of SEUs. Simple fault injection can be performed by re-running an existing test bench to a certain point in time at which a flip-flop value is intentionally changed and the simulation is resumed. The effects of the fault can then be studied. One should be careful when injecting a fault in a simulation in order not to be too pessimistic. For example, if the output of a flip-flop is forced to a value to simulate an SEU, it is important to remove the forced value after a while since it is not ensured that the simulator will actually be able to override this value. This should be done to avoid a permanent force due to lack of events or transactions generated by flip-flop if its output value does not change. The best way is to force the flip-flop output, wait for one clock period, and then to remove the force on the output and to observe state of the flip-flop after another clock period has passed.

It is highly recommended that the SEU effects discovered during the verification are propagated to users on system level and to assess the impact of those effects on the system.

7.3.7 Validation

On rare occasions, complete board including FPGAs are submitted to heavy ion radiation in order to determine if it they are susceptible to SEUs. This kind of validation is costly and requires a lot of preparation, including adaptation of test equipment to allow for an analysis of the obtained results.

8 RECOMMENDATION

There is no endorsement of FPGAs in general or any specific FPGA technology. The contractor should therefore perform an overall risk and feasibility assessment on the system level before deciding for FPGA usage in flight equipment.

Employing FPGAs for general use is only recommended when appropriate design methods are followed and when obvious improvements are made on cost, schedule and performance.

Employing FPGAs for critical use is only recommended when appropriate risk analysis has been performed and when the contractor can prove that the selected FPGA will fulfil its task in a given application and environment. The choice of FPGAs instead of ASICs must be justified early in the development. Cost or schedule benefits are insufficient on their own to justify FPGA usage in critical applications. The importance is that the unit actually works as required.