



NUREG/CR-7006
ORNL/TM-2009/20

Review Guidelines for Field-Programmable Gate Arrays in Nuclear Power Plant Safety Systems

AVAILABILITY OF REFERENCE MATERIALS IN NRC PUBLICATIONS

NRC Reference Material

As of November 1999, you may electronically access NUREG-series publications and other NRC records at NRC's Public Electronic Reading Room at <http://www.nrc.gov/reading-rm.html>. Publicly released records include, to name a few, NUREG-series publications; *Federal Register* notices; applicant, licensee, and vendor documents and correspondence; NRC correspondence and internal memoranda; bulletins and information notices; inspection and investigative reports; licensee event reports; and Commission papers and their attachments.

NRC publications in the NUREG series, NRC regulations, and *Title 10, Energy*, in the Code of *Federal Regulations* may also be purchased from one of these two sources.

1. The Superintendent of Documents
U.S. Government Printing Office
Mail Stop SSOP
Washington, DC 20402-0001
Internet: bookstore.gpo.gov
Telephone: 202-512-1800
Fax: 202-512-2250
2. The National Technical Information Service
Springfield, VA 22161-0002
www.ntis.gov
1-800-553-6847 or, locally, 703-605-6000

A single copy of each NRC draft report for comment is available free, to the extent of supply, upon written request as follows:

Address: Office of the Chief Information Officer
Reproduction and Distribution
Services Section
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
E-mail: DISTRIBUTION@nrc.gov
Facsimile: 301-415-2289

Some publications in the NUREG series that are posted at NRC's Web site address <http://www.nrc.gov/reading-rm/doc-collections/nuregs> are updated periodically and may differ from the last printed version. Although references to material found on a Web site bear the date the material was accessed, the material available on the date cited may subsequently be removed from the site.

Non-NRC Reference Material

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, and transactions, *Federal Register* notices, Federal and State legislation, and congressional reports. Such documents as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings may be purchased from their sponsoring organization.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at—

The NRC Technical Library
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852-2738

These standards are available in the library for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from—

American National Standards Institute
11 West 42nd Street
New York, NY 10036-8002
www.ansi.org

Legally binding regulatory requirements are stated only in laws; NRC regulations; licenses, including technical specifications; or orders, not in NUREG-series publications. The views expressed in contractor-prepared publications in this series are not necessarily those of the NRC.

The NUREG series comprises (1) technical and administrative reports and books prepared by the staff (NUREG-XXXX) or agency contractors (NUREG/CR-XXXX), (2) proceedings of conferences (NUREG/CP-XXXX), (3) reports resulting from international agreements (NUREG/IA-XXXX), (4) brochures (NUREG/BR-XXXX), and (5) compilations of legal decisions and orders of the Commission and Atomic and Safety Licensing Boards and of Directors' decisions under Section 2.206 of NRC's regulations (NUREG-0750).

212-642-4900

DISCLAIMER: This report was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any employee, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for any third party's use, or the results of such use, of any information, apparatus, product, or process disclosed in this publication, or represents that its use by such third party would not infringe privately owned rights.

Review Guidelines for Field-Programmable Gate Arrays in Nuclear Power Plant Safety Systems

Manuscript Completed: October 2009
Date Published: February 2010

Prepared by
M. Bobrek¹, D. Bouldin², D.E. Holcomb¹, S.M. Killough¹
S.F. Smith¹, C. Ward¹, and R.T. Wood¹

¹Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831-6010

²University of Tennessee
419 Ferris Hall, 1508 Middle Drive
Knoxville, TN 37996-5483

M.E. Waterman, NRC Project Manager

NRC Job Code N6351

Office of Nuclear Regulatory Research

ABSTRACT

This report is a compilation of safe field-programmable gate array (FPGA) design practices that can be used by NRC staff as guidance for reviewing FPGA-based safety systems in nuclear power plants. It can also serve as a basis for development of specific activities that will support the licensing process such as FPGA-specific review procedures and acceptance criteria. The report follows on the investigation of existing regulatory documents and standards related to design and review of safety-related FPGA systems. Since the existing regulatory documents are not specific about FPGA design practices, this report also serves as the complement to the standards that cover general issues related to digital and software safety systems in nuclear power plants.

FPGA design practices are classified into three major groups—FPGA hardware design practices, FPGA design entry methods, and FPGA design methodologies. Within these major groups, design practices are further classified according to four top-level attributes—reliability, robustness, traceability, and maintainability according to the framework used in NUREG/CR-6463.

The report focuses on listing and describing FPGA design practices that are potentially unsafe as well as on suggesting which ones are acceptable for safety-critical designs. Additionally, the report outlines a design life cycle that could be used by the designers and the reviewers for FPGA-based safety systems.

FOREWORD

Although field programmable gate arrays (FPGAs) (and complex programmable logic devices (CPLDs)) have not been used extensively in safety-related nuclear applications to date, it is highly likely that this technology will be used in safety systems submitted for NRC review in the foreseeable future.

A common misunderstanding about FPGAs (and CPLDs) is that these devices are less complex than microprocessor-based systems and therefore are inherently safer than microprocessor-based safety systems. FPGA devices are fundamentally complex software designs implemented by hardware engineers. As more functions are moved onto single integrated circuit (IC) chips, greater attention should be given to the system development process. Experience has shown that FPGA disciplined design methodologies have not progressed at the same rate as the capability to add functionality to FPGAs, implying that project managers may not fully appreciate the risk to safety. There also could be a perception that the process is improved by using automated design tools. In fact, there may be an over-reliance on these design tools, as indicated by several projects in which problems were linked to improper use of the tools or unintended loss of redundancy arising from the tools optimizing the intended design into unintended functionality.

From a safety perspective, it is difficult to assess the correctness of FPGA devices without extensive documentation, tools, and review procedures. Therefore, several aspects of these technologies should be addressed during safety reviews. First, NRC reviewers should review vendor information about FPGA design processes, including software design tools and development methodologies (similar to that used for current software reviews). Second, NRC reviewers should consider specific device design information (over and above the system-level documentation) for the system under review, such as requirements and design specifications, data sheets, user manuals, programmer manuals, and so forth. Third, NRC reviewers should review device failure mode information, including mitigating fault tolerant designs (e.g., triple modular redundancy and concurrent cell error detection) and workaround design changes resulting from faults found too late in the design process to correct without extensive cost (a potential issue with third-party FPGA devices). Fourth, to ensure consistent reviews of FPGA safety systems, the NRC should acquire tools and review procedures (as well as corresponding training) to support staff evaluations of safety functions implemented in FPGAs.

The primary objective of the research described in this report was to develop comprehensive guidance the NRC staff could use to confirm that FPGA-based safety systems are in conformance with NRC regulations, including Title 10 of the Code of Federal Regulations, Part 50 (10CFR50), Appendix A, *General Design Criteria for Nuclear Power Plants*, Criterion 21, *Protection system reliability and testability*. Criterion 21 states, "The protection system shall be designed for high functional reliability and inservice testability commensurate with the safety functions to be performed. Redundancy and independence designed into the protection system shall be sufficient to assure that (1) no single failure results in loss of the protection function and (2) removal from service of any component or channel does not result in loss of the required minimum redundancy unless the acceptable reliability of operation of the protection system can be otherwise demonstrated. . . ." The use of unsafe design practices could decrease the reliability of a safety system. Consequently, unsafe design practices should be identified during the development of the system by the vendor or licensee, and verified to be not in the system by the NRC reviewer during the licensing process.

Because the regulatory Offices are ultimately responsible for revising their standard review plans and interim staff guidance documents, the research is described in a NUREG-series report suitable for translating into regulatory guides and standard review plans. This report emphasizes the importance of a proactive approach to safety assessments of FPGA-based safety systems. The results of this work also could be used to develop NRC policy and regulations regarding acceptable FPGA design and application practices and procedures.

CONTENTS

	Page
ABSTRACT	iii
FOREWORD	v
LIST OF FIGURES	ix
LIST OF TABLES	ix
ACRONYMS	xi
1. INTRODUCTION	1
1.1 Background	1
1.2 Document Organization	1
2. FPGA HARDWARE DESIGN PRACTICES	3
2.1 Reliability	3
2.1.1 Board-Level Design Attributes	3
2.1.2 FPGA Internal Logic Design Attributes	6
2.2 Robustness	9
2.3 Traceability	10
2.4 Maintainability	11
3. FPGA DESIGN ENTRY METHODS	13
3.1 Reliability	13
3.2 Robustness	35
3.3 Traceability	36
3.4 Maintainability	41
4. FPGA DESIGN METHODOLOGIES	45
4.1 Design for Safety	45
4.2 FPGA Selection	45
4.3 Design Tool Selection	46
4.4 Design Flow	46
4.4.1 Design Requirements	46
4.4.2 Architectural Design	46
4.4.3 Detailed Design	48
4.4.4 Design Review	48
4.4.5 Behavioral Description	48
4.4.6 Behavioral Simulation	49
4.4.7 Logic Synthesis	49
4.4.8 Logic-Level Simulation	50
4.4.9 Physical Implementation	50
4.4.10 Post-Layout Simulation	50
4.4.11 Prototype	50
4.4.12 FPGA Verification	51
4.4.13 Hardware Implementation	51
4.4.14 Functional Hardware Verification	51
5. CONCLUSIONS	53
APPENDIX A: SURVEY OF FIELD PROGRAMMABLE GATE ARRAY DESIGN GUIDES AND EXPERIENCE RELEVANT TO NUCLEAR POWER PLANT APPLICATION	A-1
APPENDIX B: RESULTS OF SEARCH FOR TECHNICAL STANDARDS RELATED TO FPGA DESIGN	B-1

LIST OF FIGURES

Figure		Page
1	Metastability occurs at Q_A and causes error at Q_B	6
2	Reset synchronizing circuit 1.....	7
3	Reset synchronizing circuit 2.....	7
4(a)	Three different time constraints.....	8
4(b)	Single time constraint.....	8
5	FPGA design flow.....	47
6	Testing procedure for each design step.....	51
A-1	Documents required throughout the software life cycle.....	A-9

LIST OF TABLES

Table		Page
1	Recovery time for XC2VP4.....	6

ACRONYMS

ASIC	application-specific integrated circuit
BIST	built-in self-test
BRAM	block random access memory
CPLD	complex programmable logic device
DCM	digital clock manager
EDA	Electronic Design Automation
EDAC	error detection and correction
EDIF	Electronic Design Interchange Format
ESA	European Space Agency
ESD	electrostatic discharge
ESF	engineering safety feature
FIFO	first-in-first-out
FMEA	failure mode effects analysis
FPGA	field-programmable gate array
FTA	fault tree analysis
HDL	hardware description language
I&C	instrumentation and controls
IC	integrated circuits
I/O	input/output
IP	intellectual property
IV&V	independent verification and validation
JTAG	joint test action group
LUT	lookup table
mil-spec	military specifications
MSB	most significant bit
MTBMO	mean time between metastability occurrences
NASA	National Aeronautics and Space Administration
NRC	Nuclear Regulatory Commission
ORNL	Oak Ridge National Laboratory
PC	personal computer
PCB	printed circuit board
PGA	pin grid array
PLD	programmable logic device
PLL	phase locked loop
PROM	programmable read-only memory
rad-hard	radiation hardening
RAM	random access memory
ROM	read-only memory
RTL	register transfer level
QA	quality assurance
QFP	quad flat pack
SEE	single-event effect
SEU	single-event upset
SRAM	static random access memory
TDDDB	time-dependent dielectric breakdown
TID	total ionizing dose
TMR	triple-modular redundancy
UVA	University of Virginia

VHDL
V&V

very high speed integrated circuits HDL
verification and validation

1. INTRODUCTION

Oak Ridge National Laboratory (ORNL) has been engaged by the U.S. Nuclear Regulatory Commission (NRC) Office of Nuclear Regulatory Research to develop the technical basis for assessing field-programmable gate array (FPGA) technology in safety-related systems within nuclear power plants. In particular, ORNL has investigated programmable digital logic technology and implementation practices to support development of review guidance. In the first part of this study, ORNL surveyed information on the use of FPGA technology for high-assurance applications. A finding of these surveys, along with a summary of particularly relevant programmable logic device standards, is given in Appendix A of this document. In the second part of the study, ORNL developed FPGA review guidance as a set of suitable design practices for safety-critical systems. Additionally, a FPGA-specific design life cycle is outlined including specific design steps and the corresponding verification steps. The term FPGA in this document refers to complex programmable logic devices (CPLDs) as well. When there is a safety-related issue specific only to CPLDs, then the term CPLD is specifically used.

Information for this report was obtained through publicly available sources such as published papers and presentations. No proprietary information is represented.

1.1 Background

Existing regulatory documents for instrumentation and control systems in nuclear power plants are tailored for pre-FPGA technology including analog circuitry, discrete digital electronics, and microcontrollers. Additionally, a number of existing regulatory documents address safety issues related to software applications running on standard platforms such as personal computers (PCs) or embedded hardware platforms. Given the FPGA's dual nature as being both hardware and software, and its inherent complexity, there is a need for a FPGA-specific regulatory document that would address issues such as system safety assessment, design life cycle, verification and validation (V&V), configuration management, documentation requirements, etc. Many of the existing documents such as DO-254, IEEE 1012, IEEE 603, IEEE 7-4.3.2, and IEC 61508 address these issues in more general terms and, as such, may be used for the review of FPGA-based systems. However, FPGA-specific areas that are not sufficiently addressed in any of the existing documents should be addressed in a separate FPGA review guidance document. Some of these areas include FPGA hardware design practices, FPGA design entry methods, and FPGA design tools.

The main issue in achieving sufficiently comprehensive guidance arises from the need to tailor the design life cycle and V&V process to account for the specific characteristics of FPGAs. Even a relatively simple FPGA design in most cases prohibits 100% testing during code simulation and hardware verification. Therefore, FPGA-based safety systems need to be treated as complex systems as defined in DO-254, Section 1.6, and a suitable design cycle including the V&V process is needed for FPGA-based safety-related designs. The V&V process may also include design tool verification, such as that established in IEEE 7-4.3.2. Also, comprehensive guidance should include specific FPGA design practices that will lead to readable, traceable, and verifiable designs. Some of these practices should include using combinatorial design only, partitioning the design to verifiable pieces, redundancy, error reporting, etc.

1.2 Document Organization

The review guidance document contains acceptable design practices and V&V procedures that will improve the safety of FPGA designs. These practices are divided into three chapters: (1) FPGA Hardware Design Practices, (2) FPGA Design Entry Methods, and (3) FPGA Design Methodologies. This document

adopted the framework of NUREG/CR-6463 where the acceptable practices are organized in a three-level hierarchy. Only the top-level attributes are exactly the same as those in NUREG/CR-6463, while the lower-level attributes are specific to FPGA design practices. The four top-level attributes used throughout this document are reliability, robustness, traceability, and maintainability.

In Chapter 2, the FPGA hardware design practices are divided into two groups: the board-level design practices and FPGA internal logic design practices. The board-level design practices deal with issues related to FPGA placement on the printed circuit board (PCB) and issues related to FPGA interfaces to other circuitry residing on the same board. The internal logic design practices include general digital design safe practices as well as FPGA-specific design practices.

The Chapter 3 covers safety design issues related to design entry methods based on VHDL and Verilog, the most commonly used Hardware Description Languages (HDLs). The safe coding practices are listed as base-level attributes grouped into the four top-level attributes. The base-level attributes are accompanied by examples in both HDLs to show preferable coding practices that should produce reviewable, traceable, and verifiable codes. Some of the base-level attributes indicate potentially unsafe coding practices and suggest appropriate coding examples. The others present a preferred way to code common digital blocks such as state machines, encoders, decoders, shift registers, counters, read only memories, random access memories, and FIFO memories to help the safety evaluation review.

In Chapter 4, a FPGA-specific design life cycle is described, and an appropriate design flow through the life cycle is established. In most cases, FPGA designs are complex to the level that makes the 100% testability too costly and time consuming. The proposed design life cycle including the V&V process should be used as a substitute for 100% testability. Also in Chapter 4, the design life cycle includes a number of sequential design steps accompanied with their corresponding V&V steps. Every step is described in detail and necessary output documentation is specified.

Appendix A summarizes the survey of FPGA design guides and experience relevant to nuclear power plant applications.

2. FPGA HARDWARE DESIGN PRACTICES

This chapter discusses the safety-critical issues related to the hardware of FPGA design safety-related systems. It includes both the board-level design issues and the FPGA internal logic design issues. This chapter is organized into four sections, each addressing one of the top-level attributes.

2.1 Reliability

FPGA hardware design reliability is the ability of a system or component to perform its required safety and non-safety functions under stated conditions for a specified period of time. The base-level attributes for reliability are divided into two groups—board-level design attributes and FPGA internal logic design attributes.

2.1.1 Board-Level Design Attributes

The following base-level attributes facilitates the board-level reliability of FPGA designs:

- *FPGA Power-Up*
- *FPGA Configuration Time*
- *Input/Output Standards*
- *Power Pin Decoupling*
- *Simultaneous Switching Outputs*
- *Output Slew Rate*
- *Output Current Drive*
- *Unused I/O Pins*
- *External Reset*
- *Ground and Power Planes*
- *Power Dissipation/Consumption*
- *Clock Traces*
- *FPGA Programming Pins*
- *Printed-Circuit-Board Layer Stacking*

FPGA Power-Up

Most FPGAs use multiple power supplies, at least one for the FPGA core and one for the input/output (I/O) buffers. In addition, some FPGAs use separate reference voltage supplies for differential I/O buffers. Proper power sequencing is required in some of FPGAs to avoid an unpredictable behavior of the input and output buffers during power-up and power-down. Additionally, most FPGAs have power ramp-up timing requirements which set the upper and the lower limit of the power ramp time.

An additional issue is the startup current which in some cases can exceed the operational current. Care should be taken to implement voltage regulators that can support excess currents during the power-up.

FPGA Configuration Time

After power-up, reconfigurable FPGAs are configured from an internal or an external memory which can take up to several hundreds of microseconds depending on the FPGA size and the configuration clock. Care should be taken to prevent transient behavior of the external circuitry driven by the FPGA

during the configuration time. Appropriate measures include keeping the interfacing circuitry disabled during the configuration time or pulling up or down the critical inputs of the interfacing circuitry.

Input/Output Standards

FPGA outputs and inputs can be configured for different input/output (I/O) standards. It is essential that these standards are compatible with the I/O requirements of the external circuitry connected to the FPGA.

Power Pin Decoupling

Due to a simultaneous switching of a large number of gates in an FPGA, the supply current has a shape of periodic spikes which can cause spike-like voltage bounces at the ground pins. To reduce the ground bounce and the digital noise, every power pin should be decoupled using decoupling capacitors. Some of the FPGAs have their decoupling capacitors built in.

Simultaneous Switching Outputs

FPGA outputs that transition at the same time should be distributed across the FPGA output pins so that the simultaneous switching output requirement satisfies the FPGA technical specifications.

Output Slew Rate

To reduce the digital noise due to the output switching, the slew rate of the FPGA output pins should be set to the minimum value defined in the FPGA technical specifications. The use of high-slew-rate outputs should be limited to the case where the fast output switching is essential for proper FPGA interfacing.

Output Current Drive

Most of the FPGAs have a selectable current drive at the output pins. To reduce the digital noise due to the output switching, the current drive of the FPGA output pins should be set to the minimum value unless a higher current drive is needed for proper FPGA interfacing.

Unused I/O Pins

To reduce the power dissipation, the unused I/O pins should be properly programmed or tied to the ground or the power, depending on the particular FPGA I/O specifications. All unused differential I/Os should be configured as single-ended to save the power used by the differential I/O bias circuitry.

External Reset

If an external reset signal is provided to an FPGA, it should be bounce-free and have a sufficiently fast transition between the logic levels. Since different FPGAs may require different rise times at their inputs; the use of a dedicated reset integrated circuit (IC) is suggested.

Ground and Power Planes

The analog and the digital parts of the design residing on the same board should be physically separated to reduce injection of the digital noise into the analog signals. Also, separate power supplies and separate ground and power planes should be used. The analog and the digital ground planes should be connected in a single point to reduce the current loops in the planes.

Power Dissipation/Consumption

The timing behavior of the FPGA is guaranteed by the design tools only for a given junction temperature range. The power dissipation of the FPGA should be estimated and later measured to see if additional cooling is needed to keep the junction temperature within the required range.

Clock Traces

Long signal lines, if not properly terminated, can cause reflections and crosstalk. This is especially important for the clock and other high-speed lines that should be terminated at the source and/or the end of the line using serial and/or parallel termination networks.

FPGA Programming Pins

FPGA programming pins, such as joint test action group (JTAG) pins, may be subject to noise if not pulled up or down by resistors. This noise can either alter or erase the configuration memory. Also, FPGAs with external configuration memory require specific rules for the layout of the connections between the external configuration memory and the FPGA programming pins. To prevent accidental or malicious reconfiguration of CPLDs or external configuration programmable read-only memories (PROMs), the configuration memory should be write-protected by using write security bits. Malicious reconfigurations should be prevented by design and/or by administrative controls. For example, the FPGA design can include error detection circuitry or built-in self test (BIST) to constantly monitor the health of the FPGA system. Making JTAG connector physically accessible only to the trusted personnel may be one of the administrative measures.

Printed-Circuit-Board Layer Stacking

To ensure a high integrity of the FPGA board design, the stacking of the board layers is critical. Preferred layout-stacking alternates the signal layers and the power/ground layers. Below are shown two examples of a preferred 8-layer PCB stacking.

---Signal----	---Signal/Power---
---Ground----	-----Ground-----
---Signal----	---Signal/Power---
---Power----	-----Ground-----
---Ground----	-----Power-----
---Signal----	---Signal/Power---
---Ground----	-----Power-----
---Signal----	---Signal/Power---

2.1.2 FPGA Internal Logic Design Attributes

This section discusses the following base attributes that facilitate the predictability of the FPGA internal logic design:

- *Asynchronous Design*
- *Metastability*
- *Internal FPGA Reset*
- *Phase Locked Loop Locking Time*
- *Timing Constraints*
- *State Machines*
- *Multiple Clock Domains*
- *Latches*
- *High Fan-out Lines*

Asynchronous Design

The FPGA design should be synchronous as much as possible. Asynchronous designs are prone to glitches, bus skews, and other timing issues. Furthermore, the FPGA design tools do not generally support asynchronous timing constraint and analysis. If asynchronous designs are used for 100% testability or any other reason, appropriate measures need to be taken to make sure that the output glitches and the bus skews are not affecting safe operation of the FPGA design. These measures may include use of registered I/Os or analog filtering of the FPGA outputs.

Metastability

Metastability can occur when an asynchronous input gets clocked within the FPGA, and it is expressed as an undetermined state at the output of a flip-flop (Figure 1). The undetermined state resolves itself after the *recovery time*, which is on the order of several ns to several tens of ns for most of FPGAs. Table 1 shows the recovery times and their corresponding mean times between metastability occurrences (MTBMO) for XC2VP4 FPGA from Xilinx. For example, metastability with a recovery time of 1ns occurs on average every microsecond while the metastability with a 3 ns recovery time is extremely rare. The FPGA designer should consider the recovery time when determining the maximum clock speed of the design.

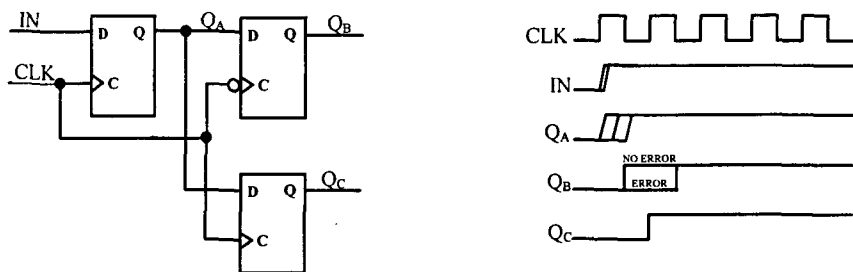


Figure 1. Metastability occurs at Q_A and causes error at Q_B .

Table 1. Recovery time for XC2VP4

Recovery time	1 ns	1.8 ns	2.1 ns	2.4 ns	2.7 ns
MTBMO	1 us	1 day	1 year	1,000 years	1,000,000 years

Internal FPGA Reset

Usually, all flip-flops in FPGAs are cleared after the power-up unless specified differently by the designer, in which case a dedicated synchronous/asynchronous reset network controlled by the reset logic inside the FPGA is used. Care should be taken to ensure synchronous reset of all flip-flops across the FPGA. This is especially critical when an asynchronous reset signal is used to clear counters or state machines that need to run synchronously. Figures 2 and 3 show two different methods for synchronizing an asynchronous reset line to ensure reset of all flip-flops within the same clock period. The approach in Figure 3 assumes that the clock period is greater than the resolution time for the flip-flops as well as skews on the reset network are less than half of the clock period. Both of these requirements are easily satisfied in modern FPGAs, since most FPGAs have dedicated low-skew lines for the reset signal and use low-resolution time flip-flops.

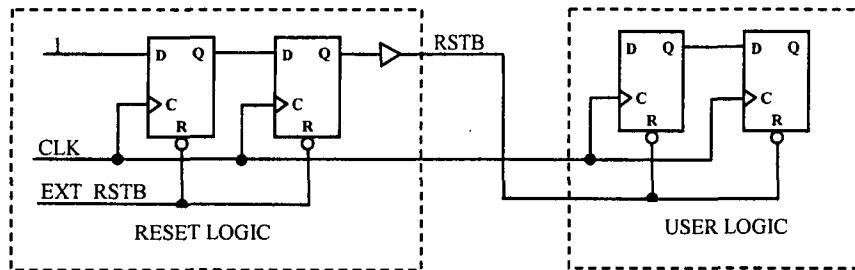


Figure 2. Reset synchronizing circuit 1.

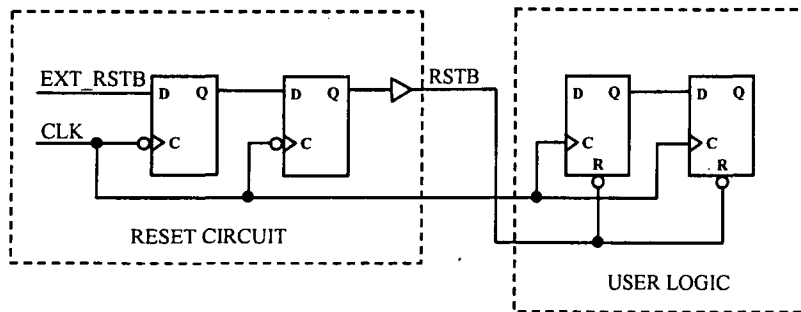


Figure 3. Reset synchronizing circuit 2.

Phase Locked Loop Locking Time

The Phase Locked Loops (PLL) inside FPGAs may be used to generate synchronous clocks or to fine-control signal delays. They require an additional time to lock after the FPGA power-up. The user logic should monitor the PLL lock signal, which indicates when the signals coming from the PLL are valid.

Timing Constraints

Most of FPGA design tools base their timing constraints and analysis on synchronous design. The timing constraints include the period, the pad-to-clock delay, the clock-to-pad delay, the skew, and the from-to delay.

Appropriate timing constraints are critical in high-speed and safety-related FPGA designs. Due to different FPGA architectures and design tools, it is important to standardize and simplify the way the timing constraints are assigned and verified. As Figure 4(a) shows, one should specify three different timing parameters to ensure proper FPGA performance at the required clock speed (T_{in} , T_p , and T_{out}). By using registered I/O buffers, one can reduce the number of timing constraints to one as shown in Figure 4(b).

Another critical timing issue is the maximum delay and/or maximum skew on high-fan signals such as RESET and EN. Designers should ensure that the maximum delay on these lines is less than the clock period to ensure the resetting and/or enabling of all flip-flops in the design occur within the same clock period.

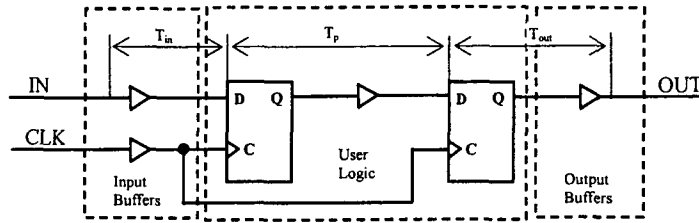


Figure 4(a). Three different time constraints.

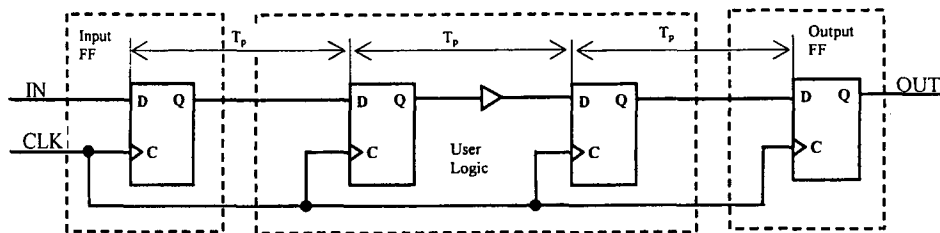


Figure 4(b). Single time constraint.

State Machines

Behavior of a state machine should be defined not only for the used states but also for the unused states. Most synthesis tools will ignore unused states and synthesize a state machine that can become stuck in an undefined state after entering it unexpectedly. A similar situation occurs if a state in the state machine is not defined for all possible combinations of its inputs. Therefore, all states and state transitions in a state machine should be explicitly defined.

Multiple Clock Domains

When designing the interface between clock islands that use different synchronous or asynchronous clocks, one should use double registers for edge-sensitive transfers to mitigate the occurrence of metastability.

Latches

Even though latches use fewer gates than conventional flip-flops, special care should be taken when the use of latches is necessary as the noise occurring at the latch inputs may propagate to the latch outputs. Use of flip-flops eliminates this problem.

High Fan-Out Lines

FPGAs have dedicated high-speed routing lines for clocks and other high fan-out signals such as RESET or EN. However, it is the designer's responsibility to properly assign these lines using the FPGA design tools. Also, the designer should impose necessary timing constraints with enough margin to ensure proper functioning of the FPGA in extreme environmental conditions.

2.2 Robustness

Robustness refers to the capability of an FPGA-based safety system to operate in an acceptable manner under abnormal conditions or events over the entire lifetime of the system. The following are the base-level attributes relevant to robustness of the hardware design of FPGA-based safety-critical systems:

- *Timing Performance*
- *Single-Event Upsets (SEU)*
- *Configuration Data Retention*
- *Program/Erase Cycles*
- *Lead-Based Soldering*

Timing Performance

Appropriate timing constraints with sufficient margin should be set by the designer during the synthesis and/or place and route process to ensure correct functioning of the time-critical paths of the FPGA design. The timing reports should be checked after the implementation of the design to ensure the constraints are satisfied and that there is enough margin in the timing performance.

Single Event Upsets

Single-event upsets (SEU) can affect both the FPGA logic as well as the FPGA configuration memory. Even in a radiologically benign environment, cosmic rays and alpha particles from the FPGA packaging material can produce enough charge inside an FPGA to change a state in one or more flip-flops or memory cells. Occurrence of SEUs over the entire lifetime of the FPGA system could be significant enough to require protective measures during the design as well as use of the system. These measures can be divided into two groups—the measures to mitigate SEUs in the user logic and the measures to mitigate SEUs in the configuration memory. The following are the possible measures to mitigate SEUs in the user logic:

- Redundancy in the design to ensure detection and/or correction of a SEU
- Error detection and correction (EDAC) for setpoints and memory data
- Error detection and/or correction in state machines and control logic
- Using SEU-immune or SEU-tolerant FPGAs
- Periodic resetting of the FPGA logic
- Watchdog timers

Measures to protect the FPGA configuration from an SEU are as follows:

- Using one-time programmable FPGAs
- Selecting the flash-based over the static random access memory (SRAM)-based reconfigurable FPGAs
- Run-time checking of the configuration memory
- Scheduled and error-initiated reloading of the FPGA configuration memory.

Configuration Data Retention

Configuration data retention for CPLDs, flash-based FPGAs, and programming devices used to configure SRAM-based FPGAs can be from less than 10 years to more than 20 years. During the operation period of an FPGA system, the programming data should be stored on a separate high-reliability memory media and then periodically used to refresh the configuration memory in the FPGAs/CPLDs. For one-time programmable FPGAs the concern is the self healing of the configured interconnects. The data sheets and reliability reports for one-time programmable FPGAs needs to be checked to determine the life time of the programmed interconnects.

Program/Erase Cycles

The program/erase cycles for the configuration memories are limited, ranging from several thousands to several tens of thousands. A log file regarding the number of program/erase cycles should be established to ensure there are enough program/erase cycles remaining in the FPGA throughout the entire life-cycle of the system.

Lead-Based Soldering

Growth of tin whiskers over time is especially critical for FPGAs due to the high pin count and the small pitch of the Pin Grid Array (PGA) and Quad Flat Pack (QFP) packages. Avoiding the use of lead-free tin parts and PCBs is an administrative control measure to eliminate failures due to whisker growth. Design techniques that include error detection and built-in self test can be an alternative solution.

2.3 Traceability

Traceability refers to attributes of a safety-critical FPGA design that support V&V of the design throughout the development design life cycle as well as the safety evaluation review during the licensing process. The traceability attributes cover issues such as use of predefined vendor- or tool-specific macros, use of nonstandard HDL libraries, implementing a hierarchical design, using version control tools, using a particular design style, and the design documentation. These attributes are more applicable to the Design Entry Methods and the Design Methodology that are described in separate chapters. The following are the traceability attributes that apply to the Hardware Design Practices:

- *Board-Level Design Documentation*
- *FPGA Internal Logic Design Documentation*

Board-Level Design Documentation

The documentation that is critical for the traceability of the FPGA board-level design includes PCB schematics, mechanical PCB drawings, board layout design files, bills of material, and the data sheets for the parts used on the board. This documentation should be included in the licensee/vendor design report to facilitate the review of the reliability and the robustness of the board-level design including FPGA power-up, external FPGA interfacing, external FPGA reset, power pin decoupling, unused I/O pins, PCB layer stacking, and other features described in this guidance.

FPGA Internal Logic Design Documentation

The documentation necessary to verify and review the FPGA internal logic design includes FPGA design files, FPGA data sheets, pin out and package information, application notes, and white papers. This documentation should be included in the licensee/vendor design report to facilitate the review of the FPGA internal logic design including FPGA configuration start-up, simultaneous switching outputs,

output slew rate, output current drive, FPGA power dissipation, metastability, timing constraints, state machines, high fan-out lines, and other features described in this guidance.

2.4 Maintainability

Maintainability of FPGA-based safety systems relates to attributes that facilitate the maintenance of a system during the operation period. The maintainability attributes address issues such as PCB design for maintainability, FPGA configuration maintainability, and FPGA design tool maintainability. The following are the base attributes relevant to maintainability of FPGA-based safety-related designs:

- *Modular Design*
- *Standardized PCB Format*
- *Test Points and Indicators*
- *Reprogrammability*
- *FPGA Selection*

Modular Design

To facilitate maintenance of FPGA-based designs, designs should be modular as much as possible (i.e., well-defined functions such as high-pressure trip, low-pressure trip, and high-flux trip) should be implemented on separate PCBs. Placing multiple FPGAs on a single board can complicate maintenance processes as well as unnecessarily challenge the safety system due to a faulty power supply for the board or due to mechanical failure of the PCB, for example.

Standardized PCB Format

FPGA-based PCBs should use one of the board standard formats where the size, the back plane connectivity, and the power supply are predefined to fit one of the hardware platform standards such as VME ANSI/IEEE 1014-1987, PCI, PCI-X, PC/104 IEEE P966.1, etc.

Test Points and Indicators

FPGA-based designs should include a number of test points and indicators for quick board diagnostic and for health monitoring of the board. These test points may include power supplies, clock sources, critical signals, temperature sensors, circuit aging sensors, etc. BIST logic for health monitoring can be placed inside the FPGA to monitor entire functions such as bus activity, occurrence of erroneous data, and time-out circuits.

Reprogrammability

FPGA-based PC boards should have provisions for reprogramming of the FPGA during the entire equipment design life. These provisions include physical access to the FPGA programming pins, necessary software and hardware tools for the programming of a particular FPGA, and the configuration files stored on a secure long-term storage media.

FPGA Selection

Selection of the vendor and the type of the FPGAs used in a design should be driven by maintainability of the system as well. The leading FPGA vendors with long-term presence on the FPGA market that support their legacy parts and the design tools should be considered. Furthermore, the backward compatibility of the design tools may not always be guaranteed by some vendors, which would

require the end user to store and maintain the version of the tool used during the initial design of the system.

3. FPGA DESIGN ENTRY METHODS

This chapter discusses the safety-critical issues related to very high speed integrated circuits HDL (VHDL) and Verilog, the most commonly used hardware description languages for FPGA designs. The schematic entry is still used for mainstream FPGA designs, and it is recommended for safety-critical designs as well. Using the schematic entry, the designer creates the gate-level representation of the design requirements that is otherwise done by the synthesis tool in the case of HDL design entry. While it gives the designer more control over the design process, the schematic entry is more time consuming and is expected to be replaced by HDL design over time.

The following sections discuss design practices that can lead to unreliable FPGA design, and therefore should be avoided. Also, proper design practices are suggested, and corresponding code examples are given for both design languages (i.e., VHDL and Verilog). The examples cover the most commonly used HDL structures such as state machines, multiplexers, decoders, read-only memory (ROM), random access memory (RAM), first-in-first-out (FIFO) memory, etc. This chapter is organized into four sections, each addressing one of the top-level attributes.

3.1 Reliability

The reliability of FPGA design entry methods is the ability of the method to capture the design requirements and to translate the requirements into the FPGA logic. A reliable HDL code gives an exact textual representation of the design requirements and provides unambiguous instructions to the logic synthesis tool. These goals cannot be completely achieved during the design entry phase of the development cycle since VHDL and Verilog are not formal languages. Therefore, the development life cycle for an FPGA-based, safety-critical system needs to include verification steps following the synthesis, place and route, configuration, and hardware implementation. The following are the base-level attributes relevant to reliability of FPGA design entry methods:

- *Mixed Language Design*
- *Nonsynthesizable Code*
- *Port Data Types*
- *Sensitivity List*
- *Implementing Resets*
- *Implementing Latches*
- *Multiple-Clock Domains*
- *If and Case Statements*
- *Multiplexers/Encoders*
- *De-multiplexers/Decoders*
- *Comparators*
- *Adder Trees*
- *Arithmetic Overflow*
- *Binary Scaling*
- *Counters*
- *Shift Registers*
- *ROM Design*

- *RAM Design*
- *FIFO Design*
- *State Machines*

Mixed Language Design

Even though the mixed language designs are supported by most of design tools, using different languages to code different blocks in the same design should be avoided due to potential language incompatibility and different synthesis rules. However, different languages can be used to design redundant identical blocks, redundant FPGAs, or redundant boards to provide additional diversity.

Nonsynthesizable Code

Nonsynthesizable code does not get materialized into the logic by the synthesizers. It is used only for simulation and it should reside in simulation-only files or test benches. The design files should contain synthesizable code only. Mixing synthesizable and nonsynthesizable code reduces readability of the code and increases probability of errors. Typical examples of nonsynthesizable codes include **wait**, **after**, and **assign** statements as shown below:

<i>--VHDL</i>	<i>//Verilog</i>
wait for 10 ns;	# 10;
Q <= '0' after 20 ns;	assign #20 Q=0;

Port Data Types

Use **std_logic** and **std_logic_vector** in VHDL code for port declaration to avoid restrictions that may apply to other data types. For example, using **buffer** type is convenient because it allows internal use of an output port, but it can cause errors during synthesis. Arrays in port declarations in VHDL should be avoided as well because of the inability of VHDL to represent and store array ports in the netlists such as Electronic Design Interchange Format (EDIF). Furthermore, the array ports can create incorrect correlation of the placed pin names from the names used in the port declaration. The following example shows the buffer port declaration.

```
--VHDL buffer port

entity acc_RAM is
  port (CLK   : in std_logic;
        DIN   : in std_logic_vector(7 downto 0);
        DOUT  : buffer std_logic_vector(7 downto 0));
end acc_RAM;

architecture behavioral of acc_RAM is
begin

  process (CLK)
  begin
    if CLK'event and CLK = '1' then
      DOUT <= DOUT + DIN;
    end if;
  end process;
end behavioral;
```

Sensitivity List

While using the **process** statement in VHDL or the **always** statement in Verilog, the sensitivity list should be completely specified. Some of the synthesis tools may assume the presence of signals in the sensitivity list even when these signals are omitted from the list. Synthesis tools need to be unambiguously directed by the HDL code to avoid unpredictable outputs of the synthesis process.

Implementing Resets

Resets should be synchronous unless the reset signal comes from a different clock region or is an external signal. In that case, the asynchronous reset signal should be resynchronized to avoid metastability when the reset is released. Since the synchronous reset can be executed only in the presence of the clock, asynchronous reset should be used in the case where the clock is not running continuously. The examples below show the synchronous reset code, the asynchronous reset code, and two different reset synchronizer codes that are HDL versions of the reset synchronizers from Figures 2 and 3.

```
--VHDL synchronous reset
process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' then
      Q <= '0';
    else
      Q <= D;
    end if;
  end if;
end process;

--VHDL asynchronous reset
process (CLK,RST)
begin
  if RST = '1' then
    Q <= '0';
  elsif CLK'event and CLK = '1' then
    Q <= D;
  end if;
end process;

//Verilog synchronous reset
always @(posedge CLK)
begin
  if (RST)
    Q = 1'b0;
  else
    Q = D;
end

//Verilog asynchronous reset
always @(posedge CLK or posedge RST)
begin
  if (RST)
    Q = 1'b0;
  else
    Q = D;
end
```

```

--VHDL reset synchronizer1
process (CLK)
begin
  if CLK'event and CLK = '1' then
    RST1 <= RST_EXT;
    RST <= RST1;
  end if;
end process;

```

```

//Verilog reset synchronizer1
always @ (posedge CLK)
begin
  RST1 = RST_EXT;
  RST = RST1;
end

```

```

--VHDL reset synchronizer2
process (CLK,RST_EXT)
begin
  if RST_EXT = '1' then
    RST1 <= '1';
    RST <= '1';
  elsif CLK'event and CLK = '1' then
    RST1 <= '0';
    RST <= RST1;
  end if;
end process;

```

```

//Verilog reset synchronizer2
always @ (posedge CLK or posedge RST_EXT)
begin
  if (RST_EXT) begin
    RST1 = 1'b1;
    RST = 1'b1;
  end
  else begin
    RST1 = 1'b0;
    RST = RST1;
  end
end
end

```

Implementing Latches

Latches should be avoided in safety-critical FPGA designs because of difficult timing analysis and their transparenence to glitches. However, latches can be unintentionally generated by ambiguous HDL statements intended to generate flip-flops. Most often, the latches are generated by incomplete *if* statements or by missing clock edges in the sensitivity list. Examples below show a correct latch implementation.

```

--VHDL
process (E,D)
begin
  if E = '1' then
    Q <= D;
  end if;
end process;

```

```

//Verilog
always @ (E or D)
begin
  if (E)
    Q = D;
  end
end

```

Multiple-Clock Domains

Multiple-clock domains in designs of safety-critical systems should be generally avoided due to difficult verification of the interface between the clock domains. If the multiple-clock domains are absolutely necessary for the proper functionality of the design, then double-registered interfacing between the clock domains should be implemented. Single-bit interfaces are relatively straightforward to design and verify, but transfer of the parallel data may be difficult to design and verify properly. Therefore, a thorough analysis of the interface should be performed to estimate the probability of occurrence of long metastability resolution times, their impact on design behavior, and necessary mitigation techniques that need to be implemented.

If and Case Statements

If and case statements may be used to design purely combinatorial logic such as multiplexers, encoders, de-multiplexers, and decoders as well as sequential logic such as state machines. When using **if-elsif** or **case** statements, all the branches should be defined explicitly for all possible input combinations. Also, all the outputs should be defined for every branch. If priority is not needed, use **case** statements instead of **if** statements to minimize the logic created by synthesis tools. Failing to completely define **if** or **case** statements can cause some synthesis tools to insert latches that are difficult to simulate. Examples related to **if** and **else** statements are given in the following two sections.

Multiplexers/Encoders

All possible multiplexer or encoder branches should be explicitly defined to avoid the possibility of creating latches and to improve the readability of the code. Case statements should be used if priority is not required. Below are the examples of **if-** and **case-**based multiplexers and encoders.

--VHDL 4-to-1 priority mux

```
process (RST,SEL,IN1,IN2,IN3,IN4)
begin
  if RST = '1' then O <= '0'; -- reset
  elsif SEL = "00" then O <= IN1;
  elsif SEL = "01" then O <= IN2;
  elsif SEL = "10" then O <= IN3;
  else O <= IN4;
  end if;
end process;
```

//Verilog 4-to-1 priority mux

```
always @ (RST or SEL or IN1 or IN2 or IN3 or IN4)
begin
  if (RST == 1'b1) O = 1'b0; //reset
  else if (SEL == 2'b00) O = IN1;
  else if (SEL == 2'b01) O = IN2;
  else if (SEL == 2'b10) O = IN3;
  else O = IN4;
end
```

--VHDL 4-to-1 mux using case statement

```
process (SEL,IN1,IN2,IN3,IN4)
begin
  case SEL is
    when "00" => O <= IN1;
    when "01" => O <= IN2;
    when "10" => O <= IN3;
    when others => O <= IN4;
  end case;
end process;
```

//Verilog 4-to-1 mux using case statement

```
always @ (SEL or IN1 or IN2 or IN3 or IN4)
begin
  case (SEL)
    2'b00 : O = IN1;
    2'b01 : O = IN2;
    2'b10 : O = IN3;
    2'b11 : O = IN4;
  endcase
end
```

--VHDL 4-to-2 priority encoder

```
process (RST,IN1)
begin
  if RST = '1' then O <= "00";
  elsif IN1 = "0000" then O <= "00";
  elsif IN1 = "0010" then O <= "01";
  elsif IN1 = "0100" then O <= "10";
  elsif IN1 = "1000" then O <= "11";
  else O <= "00";
  end if;
end process;
```

//Verilog 4-to-2 priority encoder

```
always @ (RST or IN1)
begin
  if (RST == 1'b1) O = 2'b00;
  else if (IN1 == 4'b0001) O = 2'b00;
  else if (IN1 == 4'b0010) O = 2'b01;
  else if (IN1 == 4'b0100) O = 2'b10;
  else if (IN1 == 4'b1000) O = 2'b11;
  else O = 2'b00;
end
```

--VHDL 4-to-2 encoder using case statement

```
process (IN1)
begin
  case IN1 is
    when "0001" => O <= "00";
    when "0010" => O <= "01";
    when "0100" => O <= "10";
    when "1000" => O <= "11";
    when others => O <= "00";
  end case;
end process;
```

//Verilog 4-to-2 encoder using case statement

```
always @ (IN1)
begin
  case (IN1)
    4'b0001 : O = 2'b00;
    4'b0010 : O = 2'b01;
    4'b0100 : O = 2'b10;
    4'b1000 : O = 2'b11;
    default  : O = 2'b00;
  endcase
end
```

De-multiplexers/Decoders

All possible de-multiplexer or decoder branches should be explicitly defined to avoid the possibility of creating latches and to improve the readability of the code. The **case** statement should be used if priority is not required. Below are the examples of **if-** and **case-**based de-multiplexers and decoders.

--VHDL 1-to-4 priority demux

```
process (RST,SEL,IN1)
begin
  if RST = '1' then O <= "0000"; -- reset
  elsif SEL = "00" then O <= "000"&IN1;
  elsif SEL = "01" then O <= "00"&IN1&'0';
  elsif SEL = "10" then O <= '0'&IN1&"00";
  else O <= IN1&"000";
  end if;
end process;
```

//Verilog 1-to-4 priority demux

```
always @ (RST or SEL or IN1)
begin
  if (RST == 1'b1) O = 4'b0000; --reset
  else if (SEL == 2'b00) O = {3'b000,IN1};
  else if (SEL == 2'b01) O = {2'b00,IN1,1'b0};
  else if (SEL == 2'b10) O = {1'b0,IN1,2'b00};
  else O = {IN1,3'b000};
end
```

--VHDL 1-to-4 demux using case statement

```
process (SEL,IN1)
begin
  case SEL is
    when "00" => O <= "000"&IN1;
    when "01" => O <= "00"&IN1&'0';
    when "10" => O <= '0'&IN1&"00";
    when others => O <= IN1&"000";
  end case;
end process;
```

//Verilog 1-to-4 demux using case statement

```
always @ (SEL or IN1)
begin
  case (SEL)
    2'b00 : O = {3'b000,IN1};
    2'b01 : O = {2'b00,IN1,1'b0};
    2'b10 : O = {1'b0,IN1,2'b00};
    2'b11 : O = {IN1,3'b000};
  endcase
end
```


--VHDL 2-to-4 priority decoder

```
process (RST,IN1)
begin
  if RST = '1' then O <= "0000"; -- reset
  elsif IN1 = "00" then O <= "0111";
  elsif IN1 = "01" then O <= "0110";
  elsif IN1 = "10" then O <= "1100";
  else O <= "1110";
  end if;
end process;
```

//Verilog 2-to-4 priority decoder

```
always @ (RST or IN1)
begin
  if (RST == 1'b1) O = 4'b0000; // reset
  else if (IN1 == 2'b00) O = 4'b0111;
  else if (IN1 == 2'b01) O = 4'b0110;
  else if (IN1 == 2'b10) O = 4'b1100;
  else O = 4'b1110;
end
```

--VHDL 2-to-4 decoder using case statement

```
process (IN1)
begin
  case IN1 is
    when "00" => O <= "0111";
    when "01" => O <= "0110";
    when "10" => O <= "1100";
    when others => O <= "1110";
  end case;
end process;
```

//Verilog 2-to-4 decoder using case statement

```
always @ (IN1)
begin
  case (IN1)
    2'b00 : O = 4'b0111;
    2'b01 : O = 4'b0110;
    2'b10 : O = 4'b1100;
    2'b11 : O = 4'b1110;
  endcase
end
```

Comparators

Before using the greater than (>) or less than (<) statement, the designer should be aware of the format of the numbers being compared. As the example below shows, the result of the comparison depends on whether the numbers are signed or unsigned.

--VHDL compare statement

```
O <= '1' when A > X"00" else '0';

-- if A = X"F0" is signed, O equals '0'
-- if A = X"F0" is unsigned, O equals '1'
```

//Verilog compare statement

```
assign O = (A > 8'h00) ? 1'b1 : 1'b0;

// if A = 8'hF0 is signed, O equals 1'b0
// if A = 8'hF0 is unsigned, O equals 1'b1
```

Adder Trees

Addition of multiple numbers should be implemented as a synchronous cascade of two-number adders to avoid different synthesis implementations and to improve timing performance. Also, care should be taken to properly size the data path to avoid overflows in the addition tree. Below are examples of five-number adder trees.

--VHDL pipelined adder tree with 5 inputs and 3-cycle latency

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    LEVEL_ONE_SUM1 <= IN1 + IN2;
    LEVEL_ONE_SUM2 <= IN3 + IN4;
    LEVEL_ONE_SUM3 <= IN5;
    LEVEL_TWO_SUM1 <= LEVEL_ONE_SUM1 + LEVEL_ONE_SUM2;
    LEVEL_TWO_SUM2 <= LEVEL_ONE_SUM3;
    FINAL_SUM <= LEVEL_TWO_SUM1 + LEVEL_TWO_SUM2;
  end if;
end process;
```

//Verilog pipelined adder tree with 5 inputs and 3-cycle latency

```
always @ (posedge CLK)
begin
  FINAL_SUM = LEVEL_TWO_SUM1 + LEVEL_TWO_SUM2;
  LEVEL_TWO_SUM1 = LEVEL_ONE_SUM1 + LEVEL_ONE_SUM2;
  LEVEL_TWO_SUM2 = LEVEL_ONE_SUM3;
  LEVEL_ONE_SUM1 = IN1 + IN2;
  LEVEL_ONE_SUM3 = IN5;
  LEVEL_ONE_SUM2 = IN3 + IN4;
end
```

Arithmetic Overflow

To avoid overflow/underflow during addition or subtraction, the most significant bit (MSB) padding should be applied to the operands before the operation is executed. Alternatively, the overflow/underflow can be detected and the result saturated if the application allows it. To prevent unnecessary data-path growth, appropriate LSB trimming may be applied with the choices of rounding off to the ceiling, to the floor, or around the mid-point. The examples below show MSB padding together with the bit-trimming.

*--VHDL MSB padding when adding two 8-bit numbers
-- outputs are trimmed to 8-bit numbers*

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    FULL_SIZE_SUM <= (IN1(7)&IN1) + (IN2(7)&IN2);
    FULL_SIZE_DIFF <= (IN3(7)&IN3) - (IN4(7)&IN4);
  end if;
end process;
FINAL_SUM <= FULL_SIZE_SUM(8 downto 1); -- LSB dropped
FINAL_DIFF <= FULL_SIZE_DIFF(8 downto 1); -- LSB dropped
```

```
//Verilog MSB padding when adding two 8-bit numbers
// outputs are trimmed to 8-bit numbers
```

```
always @ (posedge CLK)
begin
    FULL_SIZE_SUM = {IN1[7],IN1} + {IN2[7],IN2};
    FULL_SIZE_DIFF = {IN3[7],IN3} - {IN4[7],IN4};
end
assign FINAL_SUM = FULL_SIZE_SUM[8:1]; // LSB dropped
assign FINAL_DIFF = FULL_SIZE_DIFF[8:1]; // LSB dropped
```

Binary Scaling

Binary scaling is prone to errors due to potential mishandling of 2's complement numbers. A proper scaling requires that the MSB is preserved during the binary up- and down-scaling, as shown below.

```
--VHDL scaling of a 16-bit number
```

```
A_DIV_BY4 <= A(15)&A(15)&A(15 downto 2);
A_MULT_BY2 <= A(15)&A(13 downto 0)&'0';
```

```
//Verilog scaling of a 16-bit number
```

```
assign A_DIV_BY4 = {A[15],A[15],A[15:2]};
assign A_MULT_BY2 = {A[15],A[13:0],1'b0};
```

Counters

Among all the choices for counters, the binary counters are sufficient to cover most of the design needs. Gray counters may be used in cases when the single-bit-per-count change is absolutely necessary. Ripple counters should be avoided in safety-critical designs due to difficult timing simulation and verification. The examples below show the binary counter implementation as well as the ripple counter implementations.

```
--VHDL 8-bit binary counter
```

```
process (CLK)
begin
    if CLK'event and CLK = '1' then
        if RST = '1' then
            CNT <= X"00";
        else
            CNT <= CNT + '1';
        end if;
    end if;
end process;
```

```
//Verilog 8-bit binary counter
```

```
always @ (posedge CLK)
begin
    if (RST)
        CNT = 8'h00;
    else
        CNT = CNT + 1;
    end
```

--VHDL 3-bit ripple counter

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    Q0 <= not Q0;
  end if;
end process;

process (Q0)
begin
  if Q0'event and Q0 = '1' then
    Q1 <= not Q1;
  end if;
end process;

process (Q1)
begin
  if Q1'event and Q1 = '1' then
    Q2 <= not Q2;
  end if;
end process;

process (Q2)
begin
  if Q2'event and Q2 = '1' then
    Q3 <= not Q3;
  end if;
end process;
```

//Verilog 3-bit ripple counter

```
always @ (posedge CLK)
begin
  Q0 = !Q0;
end

always @ (posedge Q0)
begin
  Q1 = !Q1;
end

always @ (posedge Q1)
begin
  Q2 = !Q2;
end

always @ (posedge Q2)
begin
  Q3 = !Q3;
end
```

Shift Registers

There are many different HDL implementations of shift registers depending on their functionality. The examples below show implementations of the parallel-to-serial shift registers and the serial-to-parallel shift registers.

--VHDL P-to-S 8-bit shift register, MSB first

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' then -- reset
      SHIFT <= X"00";
      SHIFT_CNT <= "000";
    elsif LOAD = '1' then -- load
      SHIFT <= PDATA;
      SHIFT_CNT <= "000";
    elsif SHIFT_CNT < "111" then -- shift
      SHIFT(7 downto 1) <= SHIFT(6 downto 0);
      SHIFT_CNT <= SHIFT_CNT + '1';
    end if;
  end if;
end process;
SDATA <= SHIFT(7);
```

//Verilog P-to-S shift register, MSB first

```
always @ (posedge CLK)
begin
  if (RST) begin // reset
    SHIFT = 8'h00;
    SHIFT_CNT = 3'b000;
  end
  else if (LOAD == 1'b1) begin // load
    SHIFT = PDATA;
    SHIFT_CNT = 3'b000;
  end
  else if (SHIFT_CNT < 3'b111) begin // shift
    SHIFT[7:1] = SHIFT[6:0];
    SHIFT_CNT = SHIFT_CNT + 1;
  end
end
assign SDATA = SHIFT[7];
```

--VHDL P-to-S 8-bit shift register, LSB first

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' then -- reset
      SHIFT <= X"00";
      SHIFT_CNT <= "000";
    elsif LOAD = '1' then -- load
      SHIFT <= PDATA;
      SHIFT_CNT <= "000";
    elsif SHIFT_CNT < "111" then -- shift
      SHIFT(6 downto 0) <= SHIFT(7 downto 1);
      SHIFT_CNT <= SHIFT_CNT + '1';
    end if;
  end if;
end process;
SDATA <= SHIFT(0);
```

//Verilog P-to-S shift register, LSB first

```
always @ (posedge CLK)
begin
  if (RST) begin // reset
    SHIFT = 8'h00;
    SHIFT_CNT = 3'b000;
  end
  else if (LOAD == 1'b1) begin // load
    SHIFT = PDATA;
    SHIFT_CNT = 3'b000;
  end
  else if (SHIFT_CNT < 3'b111) begin //shift
    SHIFT[6:0] = SHIFT[7:1];
    SHIFT_CNT = SHIFT_CNT + 1;
  end
end
assign SDATA = SHIFT[0];
```

--VHDL S-to-P 8-bit shift register, LSB first

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' then -- reset
      SHIFT <= X"00";
      SHIFT_CNT <= "000";
    else -- shift
      SHIFT(6 downto 0) <= SHIFT(7 downto 1);
      SHIFT(7) <= SDATA;
      SHIFT_CNT <= SHIFT_CNT + '1';
    end if;
    if SHIFT_CNT = "000" then
      PDATA <= SHIFT; -- parallel data
      PDATA_EN <= '1'; -- data ready
    else
      PDATA_EN <= '0';
    end if;
  end if;
end process;
```

//Verilog S-to-P shift register, LSB first

```
always @ (posedge CLK)
begin
  if (RST) begin // reset
    SHIFT = 8'h00;
    SHIFT_CNT = 3'b000;
  end
  else begin // shift
    SHIFT[6:0] = SHIFT[7:1];
    SHIFT[7] = SDATA;
    SHIFT_CNT = SHIFT_CNT + 1;
  end
  if (SHIFT_CNT == 3'b000) begin
    PDATA = SHIFT; // parallel data
    PDATA_EN = 1'b1; // data ready
  end
  else
    PDATA_EN = 1'b0;
  end
end
```

--VHDL S-to-P 8-bit shift register, MSB first

```

process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' then -- reset
      SHIFT <= X"00";
      SHIFT_CNT <= "000";
    else -- shift
      SHIFT(7 downto 1) <= SHIFT(6 downto 0);
      SHIFT(0) <= SDATA;
      SHIFT_CNT <= SHIFT_CNT + '1';
    end if;
    if SHIFT_CNT = "000" then
      PDATA <= SHIFT; -- parallel data
      PDATA_EN <= '1'; -- data ready
    else
      PDATA_EN <= '0';
    end if;
  end if;
end process;

```

//Verilog S-to-P shift register, MSB first

```

always @(posedge CLK)
begin
  if (RST) begin // reset
    SHIFT = 8'h00;
    SHIFT_CNT = 3'b000;
  end
  else begin // shift
    SHIFT[7:1] = SHIFT[6:0];
    SHIFT[0] = SDATA;
    SHIFT_CNT = SHIFT_CNT + 1;
  end
  if (SHIFT_CNT == 3'b000) begin
    PDATA = SHIFT; // parallel data
    PDATA_EN = 1'b1; // data ready
  end
  else
    PDATA_EN = 1'b0;
  end
end

```

ROM Design

Read-Only Memory (ROM) can be implemented as a look-up table using the **case** statement or as a synchronous ROM. Examples are shown below. Also, ROM can be implemented using vendor-specific core generation tools such as CORE Generator from Xilinx, MegaWizard from Altera, or SmartGen from Actel. The generated ROM components are then instantiated in the HDL code. The ROM cores generated by these tools should be simulated before instantiation to avoid different interpretations of the ROM design by different vendors.

--VHDL ROM design using case statement

```

process (ADDRESS)
begin
  case ADDRESS is
    when "00" => DATA <= X"7";
    when "01" => DATA <= X"6";
    when "10" => DATA <= X"C";
    when others => DATA <= X"E";
  end case;
end process;

```

//Verilog ROM design using case statement

```

always @(ADDRESS)
begin
  case (ADDRESS)
    2'b00 : DATA = 4'h7;
    2'b01 : DATA = 4'h6;
    2'b10 : DATA = 4'hC;
    2'b11 : DATA = 4'hE;
  endcase
end

```

--VHDL synchronous ROM

```
entity syncc_ROM is
  port (CLK : in std_logic;
        ADDR : in std_logic_vector(3 downto 0);
        DOUT : out std_logic_vector(7 downto 0));
end syncc_ROM;

architecture behavioral of syncc_ROM is
  type rom is array(15 downto 0) of std_logic_vector(7 downto 0);
  constant SROM : rom :=(X"01", X"20", X"AA", X"A0", -- ROM table
                        X"0F", X"1C", X"34", X"C4",
                        X"99", X"32", X"70", X"DD",
                        X"E3", X"B6", X"DB", X"12");

begin

  process (CLK)
  begin
    if CLK'event and CLK = '1' then
      DOUT <= SROM(conv_integer(ADDR)); -- output data
    end if;
  end process;
end behavioral;
```

//Verilog synchronous ROM

```
module SROM (CLK,ADDR,DOUT);
  input CLK;
  input [7:0] ADDR;
  output [7:0] DOUT;
  reg [7:0] SROM [255:0];
  reg [7:0] DOUT;
  initial begin
    $readmemb("rom_table.list",SROM); // reads ROM data from a file
  end
  always @ (posedge CLK)
  begin
    DOUT = SROM[ADDR]; // output data
  end
endmodule
```

RAM Design

Random Access Memory (RAM) can be in different forms, including single-port synchronous, dual-port synchronous, and dual-port asynchronous. Single-port synchronous RAM can be used whenever simultaneous writes and reads from two different memory locations are not required. This is the simplest RAM implementation that can be easily tested and verified. If simultaneous reads and writes are necessary, dual-port synchronous RAM should be used. Care should be taken to explicitly define RAM

behavior during simultaneous reading and writing to the same memory cell. Depending on the application, there may be either write-first or read-first RAM implementations. Dual-port asynchronous RAM should be generally avoided in safety-critical designs because of their cross-clock boundaries that are difficult to simulate and verify. Below are examples of single-port synchronous RAM and the dual-port, read-first synchronous RAM implementations. RAM can also be instantiated after the RAM component has been generated using one of the vendor-specific core generators. The cores generated by these tools should be simulated before instantiation to avoid different interpretations of the RAM design by different vendors.

--VHDL single-port synchronous RAM

```
entity sp_sync_RAM is
  port (CLK   : in std_logic;
        WE    : in std_logic;
        ADDR  : in std_logic_vector(7 downto 0);
        DIN   : in std_logic_vector(7 downto 0);
        DOUT  : out std_logic_vector(7 downto 0));
end sp_sync_RAM;

architecture behavioral of sp_sync_RAM is
  type ram is array(255 downto 0) of std_logic_vector(7 downto 0);
  signal SP_RAM : ram;
begin

  process (CLK)
  begin
    if CLK'event and CLK = '1' then
      if WE = '1' then
        SP_RAM(conv_integer(ADDR)) <= DIN; -- RAM write
      end if;
      DOUT <= SP_RAM(conv_integer(ADDR)); -- RAM read
    end if;
  end process;
end behavioral;
```

//Verilog single-port synchronous RAM

```
module sp_sync_RAM (CLK, WE, ADDR, DIN, DOUT);
  input CLK;
  input WE;
  input [3:0] ADDR;
  input [7:0] DIN;
  output [7:0] DOUT;
  reg [7:0] SP_RAM [15:0];
  reg [7:0] DOUT;
  always @ (posedge CLK)
  begin
    DOUT = SP_RAM[ADDR]; // RAM read
    if (WE) SP_RAM[ADDR] = DIN; // RAM write
  end
endmodule
```


--VHDL dual-port read-first synchronous RAM

```
entity dp_sync_RAM is
  port (CLK   : in std_logic;
        WE    : in std_logic;
        RE    : in std_logic;
        WADDR : in std_logic_vector(7 downto 0);
        RADDR : in std_logic_vector(7 downto 0);
        DIN   : in std_logic_vector(7 downto 0);
        DOUT  : out std_logic_vector(7 downto 0));
end dp_sync_RAM;

architecture behavioral of dp_sync_RAM is
  type ram is array(255 downto 0) of std_logic_vector(7 downto 0);
  signal DP_RAM : ram;
begin

  process (CLK)
  begin
    if CLK'event and CLK = '1' then
      if WE = '1' then
        DP_RAM(conv_integer(WADDR)) <= DIN; -- RAM write
      end if;
      if RE = '1' then
        DOUT <= DP_RAM(conv_integer(RADDR)); -- RAM read
      end if;
    end if;
  end process;
end behavioral;
```

//Verilog dual-port read-first synchronous RAM

```
module dp_sync_RAM (CLK, WE, RE, WADDR, RADDR, DIN, DOUT);
  input CLK;
  input WE;
  input RE;
  input [7:0] WADDR;
  input [7:0] RADDR;
  input [7:0] DIN;
  output [7:0] DOUT;
  reg [7:0] DP_RAM [255:0];
  reg [7:0] DOUT;

  always @(posedge CLK)
  begin
    if (WE) DP_RAM[WADDR] = DIN; // RAM write
  end
  always @(posedge CLK)
  begin
    if (RE) DOUT = DP_RAM[RADDR]; // RAM read
  end
endmodule
```

FIFO Design

First-in-first-out (FIFO) structures are used for memory storage, data delay, data rate change, and data format change. They are usually designed as dual-port synchronous RAM with write and read counters controlling the write and the read address. The asynchronous FIFO should be generally avoided in safety-critical designs because their cross-clock boundaries are difficult to simulate and verify. Below are examples of synchronous FIFO implementations. Also, FIFO can be instantiated after the FIFO component has been generated using one of the vendor-specific core generators. However, the cores generated by these tools should be simulated before instantiation to avoid different interpretations of the FIFO design by different vendors.

--VHDL synchronous FIFO

```
RAM_WRITE:process (CLK)
begin
  if CLK'event and CLK = '1' then
    if WE = '1' and FF = '0' then -- write to FIFO if not full
      FIFO_RAM(conv_integer(WADDR)) <= DIN;
    end if;
  end if;
end process;

RAM_READ:process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RE = '1' and EF = '0' then -- read from FIFO if not empty
      DOUT <= FIFO_RAM(conv_integer(RADDR));
    end if;
  end if;
end process;

WRITE_POINTER:process (CLK,CLR)
begin
  if CLR = '1' then -- clear write pointer
    WADDR <= X"00";
  elsif CLK'event and CLK = '1' then
    if WE = '1' and FF = '0' then -- increment write pointer if not full
      WADDR <= WADDR + '1';
    end if;
  end if;
end process;
```

```

READ_POINTER:process (CLK,CLR)
begin
  if CLR = '1' then -- clear read pointer
    RADDR <= X"00";
  elsif CLK'event and CLK = '1' then
    if RE = '1' and EF = '0' then -- increment read pointer if not empty
      RADDR <= RADDR + '1';
    end if;
  end if;
end process;

```

```

FFLAG:process (CLK,CLR) -- active high
begin
  if CLR = '1' then -- clear full flag
    FF <= '0';
  elsif CLK'event and CLK = '1' then
    if RE = '1' then -- clear full flag when read
      FF <= '0';
    elsif WE = '1' and WADDR = RADDR - '1' then
      FF <= '1'; -- set full flag
    end if;
  end if;
end process;

```

```

EFLAG:process (CLK,CLR) -- active high
begin
  if CLR = '1' then -- clear empty flag
    EF <= '1';
  elsif CLK'event and CLK = '1' then
    if WE = '1' then
      EF <= '0'; -- clear empty flag when write
    elsif RE = '1' and WADDR = RADDR + '1' then
      EF <= '1'; -- set empty flag
    end if;
  end if;
end process;

```

//Verilog synchronous FIFO

```
always @ (posedge CLK) // write to FIFO if not full
begin
    if (WE & !FF) FIFO_RAM[WADDR] = DIN;
end

always @ (posedge CLK) // read from FIFO if not empty
begin
    if (RE & !EF) DOUT = FIFO_RAM[RADDR];
end

always @ (posedge CLK or posedge CLR) // write pointer
begin
    if (CLR) WADDR = 8'h00; // clear write pointer
    else if (WE & !FF) WADDR = WADDR + 1; // increment write pointer if not full
end

always @ (posedge CLK or posedge CLR) // read pointer
begin
    if (CLR) RADDR = 8'h00; // clear read pointer
    else if (RE & !EF) RADDR = RADDR + 1; // increment read pointer if not empty
end

always @ (posedge CLK or posedge CLR) // full flag active high
begin
    if (CLR) FF = 1'b0; // clear full flag
    else if (RE) FF = 1'b0; // clear full flag when read
    else if ((WE) & (WADDR == RADDR - 1)) FF = 1'b1; // set full flag
end

always @ (posedge CLK or posedge CLR) // empty flag active high
begin
    if (CLR) EF = 1'b1; // clear empty flag
    else if (WE) EF = 1'b0; // clear empty flag when write
    else if ((RE) & (WADDR == RADDR + 1)) EF = 1'b1; // set empty flag
end
```

State Machines

Safe state-machine design assumes explicit assignments for all states for all possible input combinations. Also, the state-machine outputs should be completely defined for every state. If a state machine contains states that transition to themselves for all input combinations (deadlock states), an external reset signal should be implemented to move the state-machine from the deadlock state. Furthermore, the reset signal is necessary to ensure that the state machine initially starts in a known state. If the reset signal is not synchronous, it should be resynchronized properly. Below are shown safe state-machine design examples.

--VHDL state machine with 4 states, two inputs and three outputs using if-elsif statements

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' then -- synchronous reset
      O <= "000";
      STATE <= "00";
    elsif STATE = "00" then -- state zero
      if IN1 = "00" then
        O <= "000";
        STATE <= "00";
      elsif IN1 = "01" then
        O <= "111";
        STATE <= "01";
      elsif IN1 = "10" then
        O <= "101";
        STATE <= "00";
      else
        O <= "001";
        STATE <= "11";
      end if;
    elsif STATE = "01" then -- state one
      if IN1 = "00" then
        O <= "000";
        STATE <= "00";
      elsif IN1 = "01" then
        O <= "110";
        STATE <= "01";
      elsif IN1 = "10" then
        O <= "001";
        STATE <= "10";
      else
        O <= "110";
        STATE <= "11";
      end if;
    elsif STATE = "10" then -- state two
      if IN1 = "00" then
        O <= "000";
        STATE <= "10";
      elsif IN1 = "01" then
        O <= "101";
        STATE <= "11";
      elsif IN1 = "10" then
        O <= "010";
        STATE <= "00";
      else
        O <= "110";
        STATE <= "01";
      end if;
    elsif STATE = "11" then -- state three
      if IN1 = "00" then
        O <= "001";
        STATE <= "11";
      elsif IN1 = "01" then
        O <= "100";
        STATE <= "11";
      elsif IN1 = "10" then
        O <= "001";
        STATE <= "00";
      else
        O <= "101";
        STATE <= "01";
      end if;
    end if;
  end if;
end process;
```

--VHDL state machine with 4 states, two inputs and three outputs using case statements

```

process(CLK,RST)
begin
  if RST = '1' then -- reset
    O <= "000";
    STATE <= "00";
  elsif CLK'event and CLK = '1' then
    case STATE is
      when "00" => -- state zero
        case IN1 is
          when "00" =>
            O <= "000";
            STATE <= "00";
          when "01" =>
            O <= "111";
            STATE <= "01";
          when "10" =>
            O <= "101";
            STATE <= "00";
          when others =>
            O <= "001";
            STATE <= "11";
          end case;
        when "01" => -- state one
          case IN1 is
            when "00" =>
              O <= "010";
              STATE <= "01";
            when "01" =>
              O <= "111";
              STATE <= "01";
            when "10" =>
              O <= "101";
              STATE <= "00";
            when others =>
              O <= "110";
              STATE <= "11";
            end case;
          when "10" => -- state two
            case IN1 is
              when "00" =>
                O <= "000";
                STATE <= "10";
              when "01" =>
                O <= "101";
                STATE <= "01";
              when "10" =>
                O <= "100";
                STATE <= "00";
              when others =>
                O <= "001";
                STATE <= "11";
              end case;
            when others => -- state three
              case IN1 is
                when "00" =>
                  O <= "101";
                  STATE <= "11";
                when "01" =>
                  O <= "110";
                  STATE <= "01";
                when "10" =>
                  O <= "000";
                  STATE <= "00";
                when others =>
                  O <= "001";
                  STATE <= "10";
                end case;
              end case;
            end if;
          end case;
        end process;
    end case;
  end if;
end process;

```

//Verilog state machine with 4 states, two inputs and three outputs using if-else if statements

```
always @(posedge CLK)
begin
    if (RST == 1'b1) begin // reset
        O = 3'b000;
        STATE = 2'b00;
    end
    else if (STATE == 2'b00) begin // state zero
        if (IN1 == 2'b00) begin
            O = 3'b000;
            STATE = 2'b00;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b111;
            STATE = 2'b01;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b101;
            STATE = 2'b00;
        end
        else begin
            O = 3'b001;
            STATE = 2'b11;
        end
    end
    else if (STATE == 2'b01) begin // state one
        if (IN1 == 2'b00) begin
            O = 3'b000;
            STATE = 2'b00;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b100;
            STATE = 2'b01;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b001;
            STATE = 2'b10;
        end
        else begin
            O = 3'b110;
            STATE = 2'b11;
        end
    end
    else if (STATE == 2'b10) begin // state two
        if (IN1 == 2'b00) begin
            O = 3'b000;
            STATE = 2'b10;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b101;
            STATE = 2'b11;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b010;
            STATE = 2'b00;
        end
        else begin
            O = 3'b110;
            STATE = 2'b01;
        end
    end
    else if (STATE == 2'b11) begin // state three
        if (IN1 == 2'b00) begin
            O = 3'b001;
            STATE = 2'b11;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b100;
            STATE = 2'b11;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b001;
            STATE = 2'b00;
        end
        else begin
            O = 3'b101;
            STATE = 2'b01;
        end
    end
end
```

//Verilog state machine with 4 states, two inputs and three outputs using case statements

```
always @(posedge CLK)
begin
    if (RST == 1'b1) begin // reset
        O = 3'b000;
        STATE = 2'b00;
    end
    else if (STATE == 2'b00) begin // state zero
        if (IN1 == 2'b00) begin
            O = 3'b000;
            STATE = 2'b00;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b111;
            STATE = 2'b01;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b101;
            STATE = 2'b00;
        end
        else begin
            O = 3'b001;
            STATE = 2'b11;
        end
    end
    else if (STATE == 2'b01) begin // state one
        if (IN1 == 2'b00) begin
            O = 3'b000;
            STATE = 2'b00;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b100;
            STATE = 2'b01;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b001;
            STATE = 2'b10;
        end
        else begin
            O = 3'b110;
            STATE = 2'b11;
        end
    end
    else if (STATE == 2'b10) begin // state two
        if (IN1 == 2'b00) begin
            O = 3'b000;
            STATE = 2'b10;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b101;
            STATE = 2'b11;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b010;
            STATE = 2'b00;
        end
        else begin
            O = 3'b110;
            STATE = 2'b01;
        end
    end
    else if (STATE == 2'b11) begin // state three
        if (IN1 == 2'b00) begin
            O = 3'b001;
            STATE = 2'b11;
        end
        else if (IN1 == 2'b01) begin
            O = 3'b100;
            STATE = 2'b11;
        end
        else if (IN1 == 2'b10) begin
            O = 3'b001;
            STATE = 2'b00;
        end
        else begin
            O = 3'b101;
            STATE = 2'b01;
        end
    end
end
```


3.2 Robustness

Robustness of design entry methods refers to the capability of the generated design to operate in an acceptable manner under abnormal conditions or events over the entire lifetime of the system. The following are the base-level attributes relevant to robustness of FPGA design entry methods:

- *Input Overflow*
- *Input Activity*
- *SEU Handling*

Input Overflow

A robust HDL design should be able to properly handle a potential input overflow. In the first step, the overflow should be recognized either by sampling an external overflow flag or by checking whether the input is in an acceptable range. In the second step, proper actions are taken such as ignoring the current input sample, replacing it by a default value, accepting the overflow, or reporting the overflow.

Input Activity

To prevent the hang-ups in the design due to an inactive or disconnected input, the HDL code should implement a logic that will respond to an inactive input. The response may include a specific internal action, such as ignoring the input or setting an error flag.

SEU Handling

Any potential changes of internal states of the design caused by an SEU should be properly handled by the HDL code. To detect an error, double redundancy logic can be implemented at the module level or at the FPGA level, and the comparison of the outputs can be done. Proper action may include a scheduled internal reset of the entire FPGA or reporting the error to an operator. Some parts of the design, such as setting points or state machines, may require error correction to be implemented. Possible methods include triple module redundancy and Hamming coding. Examples below show double module redundancy with self-reset on error and self-correctable triplicated flip-flop code with majority voting logic that can correct a single SEU within the clock period.

```
--VHDL double module redundancy
-- with reset-on-error

COMPONENT1:process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' or ERROR = '1' then
      CNT1 <= X"00"; -- reset when error
    else
      CNT1 <= CNT1 + '1';
    end if;
  end if;
end process;
```

```
//Verilog double module redundancy
//with reset-on-error

always @(posedge CLK)
begin
  if (RST | ERROR)
    CNT1 = 8'h00; // reset when error
  else
    CNT1 = CNT1 + 1;
  end
```

```

COMPONENT2:process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' or ERROR = '1' then
      CNT2 <= X"00"; -- reset when error
    else
      CNT2 <= CNT2 + '1';
    end if;
  end if;
end process;

```

```

always @ (posedge CLK)
begin
  if (RST | ERROR)
    CNT2 = 8'h00; // reset when error
  else
    CNT2 = CNT2 + 1;
  end
end

```

```

COMPARE:process (CLK)
begin
  if CLK'event and CLK = '1' then
    if CNT1 = CNT2 then
      ERROR <= '0';
    else
      ERROR <= '1'; -- error set
    end if;
  end if;
end process;

```

```

always @ (posedge CLK)
begin
  if (CNT1 == CNT2)
    ERROR = 1'b0;
  else
    ERROR = 1'b1; // error set
  end
end

```

--VHDL Self-correctable, triplicated flip-flop

//Verilog Self-correctable, triplicated flip-flop

```

process (CLK)
begin
  if CLK'event and CLK = '1' then
    if RST = '1' then
      Q1 <= '0';
      Q2 <= '0';
      Q3 <= '0';
    else
      Q1 <= D; -- flip-flop one
      Q2 <= D; -- flip-flop two
      Q3 <= D; -- flip-flop three
    end if;
  end if;
end process;
--voter
Q <= (Q1 and Q2) or (Q2 and Q3) or (Q1 and Q3);

```

```

always @ (posedge CLK)
begin
  if (RST) begin
    Q1 = 1'b0;
    Q2 = 1'b0;
    Q3 = 1'b0;
  end
  else begin
    Q1 = D; // flip-flop one
    Q2 = D; // flip-flop two
    Q3 = D; // flip-flop three
  end
end
//voter
assign Q = (Q1&Q2)|(Q2&Q3)|(Q1&Q3);

```

3.3 Traceability

Traceability of the HDL code refers to attributes that support V&V of the design throughout the design life cycle as well as safety evaluation review during the licensing process. Traceability attributes cover issues related to the HDL coding style, such as using hierarchical design, indenting and spacing, and code comments. The following traceability attributes apply to FPGA design entry methods:

- *Code Line Length*
- *Naming Conventions*
- *Port Mapping*
- *Constants and Parameters*
- *Indenting and Spacing*
- *Dynamic Parameters*
- *Hierarchical Code*
- *Code Comments*

Code Line Length

Code lines should be restricted to 80 character spaces, as a rule of thumb, to improve code readability. Longer lines can be broken with the continuation character and aligned with the first line.

Naming Conventions

The design entry file names should match the entity/module name of the VHDL/Verilog code contained in the file. This helps code readability and simulation. Names for signals, variables, wires, instances, etc., should be concise but meaningful using underscore where appropriate. Example: EVENT_COUNTER.

Port Mapping

To improve code readability, use named rather than positional association for the port mapping when instantiating a sub-module. Also, a single port mapping per line is preferred over single-line port mapping.

<pre> --VHDL named port mapping INST0 : tbuf port map (I => DATA_IN, O => DATA_OUT, T => DATA_ENB); </pre>	<pre> //Verilog named port mapping tbuf INST0 (.I (DATA_IN), .O (DATA_OUT), .T (DATA_ENB)); </pre>
<pre> --VHDL positional port mapping INST0 : tbuf port map (DATA_IN, DATA_OUT, DATA_ENB); </pre>	

Constants and Parameters

Using constants and parameters to substitute numbers helps readability and portability of the code.

--VHDL constants

```
constant INIT : std_logic_vector(1 downto 0) := "00";
constant SLOW : std_logic_vector(1 downto 0) := "01";
constant FAST : std_logic_vector(1 downto 0) := "10";
signal STATE : std_logic_vector(1 downto 0);
```

```
begin
  if STATE = INIT then
    O <= '0';
  elsif STATE = SLOW then
    O <= IN1;
  elsif STATE = FAST then
    O <= not IN1;
  else
    O <= '1';
  end if;
```

//Verilog parameters

```
parameter INIT = 2'b00;
parameter SLOW = 2'b01;
parameter FAST = 2'b10;
wire STATE;
```

```
always @(posedge CLK)
begin
  if (STATE == INIT) begin
    O = 1'b0;
  end
  else if (STATE == SLOW) begin
    O =;
  end
  else if (STATE == FAST) begin
    O = !IN;
  end
  else begin
    O = 1'b1;
  end
end
```

Indenting and Spacing

To have better readability of the code and to reduce coding errors, use proper indentation and spacing. The following example shows proper code indentation.

--VHDL

```
process (CLK,RST)
begin
  if RST = '1' then
    O <= '0';
  elsif CLK'event and CLK = '1' then
    if SEL = '1' then
      O <= I1;
    else
      O <= I2;
    end if;
  end if;
end process;
```

//Verilog

```
always @(posedge CLK or posedged RST)
begin
  if (RST)
    O = 0;
  else if (SEL == 1'b1)
    O = I1;
  else
    O = I2;
end
```

Dynamic Parameters

Implementing variable bus and array widths using **generic-s** and **parameter-s** helps code reuse and readability.

```
--VHDL                                     //Verilog

entity adder is                               module (IN1, IN2, OUT);
  generic (                                   parameter IN_WIDTH = 16;
    IN_WIDTH : integer := 16;               parameter OUT_WIDTH = 16;
    OUT_WIDTH : integer := 16;             input IN1 [IN_WIDTH-1 : 0];
  );                                         input IN2 [IN_WIDTH-1 : 0];
  port (                                     output OUT [OUT_WIDTH-1 : 0];
    IN1 : in std_logic_vector(IN_WIDTH-1 downto 0);
    IN2 : in std_logic_vector(IN_WIDTH-1 downto 0);
    O : out std_logic_vector(OUT_WIDTH-1 downto 0);
  );
end adder;
architecture behavioral of adder is

begin

  O <= IN1 + IN2;

end behavioral;
```

Hierarchical Code

Avoid using flat-module designs where all the code resides in a single file. Using hierarchical design makes the code easier to read, trace, and verify. Also, it facilitates team work on large designs. In hierarchical code, use the top-level code for component/module declaration and instantiation. The behavioral code should generally be placed at the lowest hierarchical level. Examples below show hierarchical design code for both VHDL and Verilog.

--VHDL file top.vhd

```
entity top is
port (
    IN1 : in std_logic_vector(15 downto 0);
    IN2 : in std_logic_vector(15 downto 0);
    IN3 : in std_logic_vector(15 downto 0);
    O : out std_logic_vector(15 downto 0)
);
end top;
architecture Behavioral of top is

component adder is
port (
    IN1 : in std_logic_vector(15 downto 0);
    IN2 : in std_logic_vector(15 downto 0);
    O : out std_logic_vector(15 downto 0)
);
end component;

signal SUM1 : std_logic_vector(15 downto 0);

begin

INST0 : adder port map (
    IN1 => IN1,
    IN2 => IN2,
    O => SUM1
);

INST1 : adder port map (
    IN1 => IN3,
    IN2 => SUM1,
    O => O
);

end Behavioral;
```

--VHDL file adder.vhd

```
entity adder is
port (
    IN1 : in std_logic_vector(15 downto 0);
    IN2 : in std_logic_vector(15 downto 0);
    O : out std_logic_vector(15 downto 0)
);
end adder;
architecture Behavioral of adder is

begin

O <= IN1 + IN2;

end Behavioral;
```

```
//Verilog file top.v
```

```
module (IN1, IN2, IN3, OUT);
input IN1 [15 : 0];
input IN2 [15 : 0];
input IN3 [15 : 0];
output OUT [15 : 0];
wire SUM1 [15 : 0];
always @ (IN1, IN2, IN3)
begin

    adder INST0 (
        .IN1 (IN1),
        .IN2 (IN2),
        .OUT (SUM1)
    );

    adder INST1 (
        .IN1 (IN1),
        .IN2 (SUM1),
        .OUT (OUT)
    );
end
end module;
```

```
//Verilog file adder.v
```

```
module (IN1, IN2, OUT);
input IN1 [15 : 0];
input IN2 [15 : 0];
output OUT [15 : 0];
always @ (IN1, IN2)
begin
    OUT = IN1 + IN2;
end
end module
```

Code Comments

Add comments to the code to describe the purpose and functionality of each of the components in the design. Also, comment on critical lines and segments of the code to help verification, traceability, and maintainability of the code.

3.4 Maintainability

Maintainability of the design entry methods relates to attributes that facilitate the maintenance of HDL code during the operation period. The following are the base attributes relevant to maintainability of HDL code:

- *Generic Code*
- *Synthesis Directives*
- *Place and Route Directives*
- *Vendor-Specific Hard Macros*
- *Vendor-Specific Intellectual Property Cores*

Generic Code

To allow for possible regeneration of the FPGA configuration file using different design tools or different FPGAs, the HDL code should be as generic as possible without tool-specific or FPGA-specific directives and structures. Examples in the previous sections of this chapter represent the generic code examples covering the most common HDL structures needed for safety-critical FPGA design. The example below shows the generic VHDL code to implement 4-to-1 multiplexer as well as the VHDL code that uses Xilinx M4_1 macro for 4-to-1 multiplexer.

--VHDL 4-to-1 mux using generic code

```
entity top is
port (
    SEL : in std_logic_vector(1downto 0);
    IN1  : in std_logic;
    IN2  : in std_logic;
    IN3  : in std_logic;
    IN4  : in std_logic;
    O    : out std_logic
);
end top;
architecture Behavioral of top is

begin

process (SEL,IN1,IN2,IN3,IN4)
begin
    case SEL is
        when "00" => O <= IN1;
        when "01" => O <= IN2;
        when "10" => O <= IN3;
        when others => O <= IN4;
    end case;
end process;

end Behavioral;
```

--VHDL 4-to-1 mux using Xilinx M4_1 macro

```
entity top is
port (
    SEL : in std_logic_vector(1 downto 0);
    IN1  : in std_logic;
    IN2  : in std_logic;
    IN3  : in std_logic;
    IN4  : in std_logic;
    O    : out std_logic
);
end top;
architecture behavioral of top is

begin

inst: M4_1 port map (
    D0 => IN1,
    D1 => IN2,
    D2 => IN3,
    D4 => IN4,
    S  => SEL,
    O  => O
);

end Behavioral;
```

Synthesis Attributes and Constraints

Avoid synthesis attributes and constraints in HDL code that are specific to a particular synthesis tool. Future versions of the synthesis tool, even from the same vendor, may not support old synthesis attributes and constraints. Below are some examples of the tool-specific synthesis constraints used by XST Xilinx synthesis tool.

USE_DSP48 — uses FPGA-specific DSP block for arithmetic operations
RAM_STYLE — defines whether to use distributed or block RAM

Place and Route Directives

Place and route directives are very specific to a particular FPGA vendor and even to a particular FPGA family from the same vendor. Avoid directives in the HDL code that are not critical for the design, such as particular placement of design components or use of specific logic and routing resources. Essential place and route constraints, such as pin placements, global clock lines, and registered I/Os, are generally transparent to most of FPGA place and route tools. The examples below show some essential place and route constraints that should be used as well as some non-essential constraints that should be avoided. The examples show Xilinx place and route constraints specified in the user constraint files (UCF).

4. FPGA DESIGN METHODOLOGIES

This chapter discusses the FPGA design methodologies that should be used to develop safety-critical FPGAs that are components in nuclear power control systems. The term *design methodology* refers to the development life cycle, which includes requirements capture, conceptual design, detailed design, implementation, and V&V. More specifically, the FPGA design methodology defines guidance on the use of mainstream FPGA design tools for safety-critical designs. The specific steps in the methodology need to be incorporated into the design life-cycle model adopted by the vendor according to its own management development plan. The chapter is organized into four sections, each covering a base-level attribute related to safe design practices for using FPGA design tools. The main objective in the selection of safe design practices is the development of a reliable, robust, traceable, and maintainable FPGA design. The following are the base-level attributes describing the suggested FPGA design methodologies in more detail:

- *Design for Safety*
- *FPGA Selection*
- *Design Tool Selection*
- *Design Flow*

4.1 Design for Safety

Design for safety is a different goal for most designers and Electronic Design Automation (EDA) tools which traditionally target speed and/or power consumption. A reliable FPGA should perform its intended functions in a predictable and consistent manner with the desired timing. It is essential that the design be verifiable and reviewable at each level (behavior simulation, logic simulation, physical simulation, and prototype measurement). Many EDA vendors offer verification tool suites that can greatly help in testability analysis and design simulation. Advanced simulation techniques such as constraint-random stimuli generation, assertion-based verification, and formal verification together with the code coverage analysis should be encouraged for use in the verification of safety-critical FPGA designs as an integral part of the design life cycle. Additional requirement for the design for safety is that the errors that occur while the FPGA is in use due to abnormal conditions and events should be detected, reported, and fixed.

4.2 FPGA Selection

Advances in the manufacturing of hardware components over the past few decades have produced FPGAs that contain millions of logic elements and interconnections. However, FPGAs in nuclear power plant control systems are generally required to implement relatively small designs requiring only a few thousand elements, and therefore, these designs do not warrant the use of the newest models of FPGA components. Further, it is best to employ one or more small FPGAs with low capacity rather than a single, large-capacity device. As a rule of thumb, if the FPGA occupancy is below 30%, the FPGA is too big for the application, and if the occupancy is over 80%, the FPGA is too small for the application. In many cases where designs do not require a large number of memory elements, CPLDs can be a safer choice than FPGAs due to their on-chip configuration memory, their rich combinatorial circuitry, and better timing predictability. The flash-based FPGAs have build-in configuration memory that is more resistant to single event upsets (SEUs) than the SRAM-based FPGAs. They are a valid alternative to CPLDs, especially for large designs that can not fit in any of available CPLD. Finally, one-time programmable FPGAs have advantages over the reprogrammable FPGAs in cases where reprogrammability is not necessary or not desired. The configuration in one-time programmable FPGAs is

immune to single event upsets (SEUs), is protected from accidental or malicious reprogramming, and does not require special maintenance.

4.3 Design Tool Selection

Electronic design automation (EDA) tools are continually being enhanced to handle larger and more complex designs. However, newer versions of tools generally have not been tested as much as older versions, so designers of FPGAs intended for nuclear power plant control systems may intentionally prefer to use stable versions of EDA tools to increase the likelihood that errors are not present in the tools. FPGA EDA tools that are used by only a few thousand designers are less desirable for safety system development compared with those for main-stream processors, which are used by potentially millions of users. Since no EDA vendors offer certified versions of their tools for safety-critical applications, the reviewer needs to rely on the review of the entire design life-cycle. In this process, the review of the EDA tools should establish whether the vendor exercised the following activities to assess the tools:

- Identifying the tool
- Identifying which design and verification processes the tool supports
- Independently assessing the tool output
- Identifying tool relevant history
- Identifying known tool issues
- Identifying tool settings used for the design
- Assessing designers' experience with the tool

4.4 Design Flow

Recommended design flow, as shown in Figure 5, includes the design path and the verification path where every step in the design path has a corresponding step in the verification path. Depending on the results of each of the verification steps, the corresponding design step may need to be redesigned until the verification requirements are satisfied. Also, the verification steps may require revision of the initial requirements and consequently require an update of all previous design steps. In the following paragraphs, the design and verification steps are described in more detail.

4.4.1 Design Requirements

The design begins with a narrative description of the requirements for the FPGA-based component. This description should state the desired functionality of the device and describe the context in which the device will be employed. Additional requirements, such as critical timing, power, and board size, may be added to the requirements.

4.4.2 Architectural Design

The most critical part of the architectural design process is the top-level design partitioning. The main criteria in the design partitioning should be design reliability, design traceability, and design verifiability. Other criteria may include functionality, clock domains, SEU mitigation, and safety vs nonsafety separation. These criteria are not mutually exclusive as they can lead to the same partitioning of the design. Below is a more detailed description of the design partitioning criteria.

- Partitioning for reliability refers to dividing the design into sub-blocks that best represent the initial requirements as some of the requirements may be loosely correlated to each other or not correlated at all. This will also help traceability of the initial requirements throughout the design life cycle.
- Partitioning for verifiability divides the design into blocks that can be separately simulated and verified throughout the development life cycle.
- Partitioning by digital functions divides the design into blocks that perform a specific function such as state machines, memories, memory controllers, I/O interfaces, etc.
- Partitioning by clock domains divides the design into separate clock islands that may or may not interconnect.
- Partitioning by SEU mitigation includes redundant blocks to implement error detection and/or error correction.
- Partitioning by safety separates safety functions from the nonsafety functions residing on the same FPGA device. Note that nonsafety function blocks cannot be sufficiently isolated from safety functions residing on the same FPGA device. As a consequence, the nonsafety blocks should receive the same rigor of verification as the safety blocks. Implementing safety and nonsafety functions on separate FPGAs or separate boards and then implementing a proper isolation on the interfacing lines is a better partitioning approach.

Additional architectural consideration may include critical timing of the internal FPGA design, FPGA interfacing with outside circuitry, external reset, external clock sources, power dissipation, etc.

4.4.3 Detailed Design

Detailed design objective is to define specific implementation blocks such as state machines, memories, internal and external interfaces, data-path sizes, and clock speeds. Additionally, circuitry for design health monitoring, built-in self test, redundancy, and observability needs to be specified in detail. The output of this step is most likely a block diagram showing functional blocks, their interfaces, and other information necessary for HDL coding or schematic design that follow in the design flow.

4.4.4 Design Review

Output of the architectural design and detailed design steps in the form of textual or graphical description of the design partitioning and other design requirements should be reviewed in the design review process. The result of the review may require a different design partitioning or correction of the initial requirements. The review can be conducted initially by the designer but, subsequently, by other members of the designer's organization or an independent third party.

4.4.5 Behavioral Description

In the next design step, the designer refines the architectural specifications into behavioral descriptions that delineate the functionality of each module and their interactions. The behavioral description is done by schematic diagram, block diagram, or HDL such as VHDL and Verilog which use register transfer level (RTL) representation of digital, two-valued logic. It is essential that the hierarchy in the behavioral description exactly represent the design partitioning done during the architectural design step. Also, the hardware design practices and the HDL coding practices suggested in previous chapters should be followed in the behavioral description. Outputs of the behavioral design step such as schematic diagrams, block diagrams, VHDL, or Verilog code can serve as the documentation base for the review of this design step, to trace back to the original design requirements, and to generate input stimuli for the behavioral simulation.

Other design entry methods exist that use a higher level of abstraction such as Matlab, C, or LabVIEW. Additional tools (code generators) are then used to translate the high-level behavioral description into HDL code. Using higher levels of abstraction saves development time but leaves numerous details up to the synthesis tool to interpret. On the other hand, specifying the design at a low level of detail requires more effort on the part of the designer, but the final implementation is more likely to reflect the original intent. This situation is analogous to an upper-level manager providing directions which are then interpreted by subordinate managers for final disposition by staff.

Whenever these higher level abstraction methods are employed, it is essential that thorough testing and evaluation be performed to ensure that the final implementation corresponds to the intent of the designer. Even though the higher level of abstraction increases productivity in the mainstream digital design, it may not be suitable for safety-critical designs as the generated HDL code may be difficult to read and trace back to the original design requirements.

4.4.6 Behavioral Simulation

Behavioral simulation, also referred to as presynthesis simulation, uses standard simulation tools and is based solely on the graphical or textual description of the design requirements. Input stimuli and expected output behavior vectors should be specified according to the initial design requirements. Simple designs that do not contain hierarchical blocks can have a single set of top-level input and output simulation vectors. In the case of hierarchical designs with multiple blocks, the simulations should be done for each block separately to achieve a higher level of simulation coverage.

After the block-level simulation is performed, the top-level simulation may be performed to confirm proper connectivity between the blocks. Special care should be taken when generating the input stimuli for safety-critical designs as a high-level of testability is generally required. Small combinatorial designs or design blocks can be 100% tested; however, even small sequential designs containing just a few tens of flip-flops may be practically impossible to test 100%. To achieve a reasonable confidence in the design, the input stimuli should not only cover the expected input combinations but also those input combinations that are not expected in normal use of the design. To increase confidence in the design, fault injection techniques can be used to change the internal state of the design to simulate SEUs.

Outputs of this step, the timing diagrams, and textual reports are necessary documentation needed to verify correct behavior of the design. There may be a need to correct the behavioral description of the code until the initial requirements are met. Sometimes, the initial requirements may need to be corrected or the behavioral simulation may show that the initial requirements are incomplete, inconsistent, or otherwise deficient.

4.4.7 Logic Synthesis

Separate design tools such as VHDL or Verilog synthesizers are used to generate the gate-level representation of the RTL description contained in HDL code. Synthesis tools optimize digital logic described in HDL code and produce either textual outputs such as EDIF files or graphical (schematic) outputs. Most synthesizers generate FPGA-independent schematic representation of the HDL code as well as the FPGA-specific schematic representation. The FPGA-independent schematic uses AND, OR, and NOT gates and flip-flops to represent the design, while the FPGA-specific schematic utilizes look-up tables to replace the combinatorial logic and different macros to represent higher-level functional blocks such as memories, multipliers, adders, etc. To be able to understand what possible optimizations and design changes are applied during the synthesis, designers and reviewers should carefully examine the synthesis report documents generated by the synthesis tool. Usually, the warnings in the report may give important information about how the synthesis tool understands the HDL code and, thereby, whether the original designer's intent remained unchanged.

4.4.8 Logic-Level Simulation

Since different synthesis tools perform different optimization of the HDL code and produce different FPGA-specific schematic diagrams, a post-synthesis simulation is required to confirm that the initial design requirements are preserved during the synthesis. While presynthesis simulation is technology independent, post-synthesis simulation uses technology-dependent elements that are FPGA specific. It is important to understand that the post-synthesis simulation does not accurately model the interconnection delays or critical timing parameters of the involved logic. The exact timing analysis is done by post-layout simulation performed after physical implementation of the design.

It is essential for the post-synthesis simulation that the same input stimuli that were used during the behavior simulation are used for post-synthesis simulation as well. If new stimuli are added at any level or changes are made to any descriptions, regression tests should be performed at each level to ensure that all levels have been subjected to the same tests and their responses are consistent. The verification process at this step should confirm that the outputs of the post-layout simulations are equivalent to those achieved during the behavioral simulation. Any differences between the simulations should be analyzed to determine how they affect the original design requirements.

4.4.9 Physical Implementation

Following synthesis, the design is physically implemented, which in the case of FPGAs involves placing and routing steps or in the case of CPLDs, the fitting step. During this design phase, an EDA tool maps the logic abstraction into the physical elements and interconnections on the FPGA device. At this design step, additional design requirements may need to be implemented. These requirements include the timing requirements, I/O placement, I/O parameters, use of FPGA-specific logic elements, etc.

4.4.10 Post-Layout Simulation

Post-layout simulation should be performed by the designer using the same input stimuli from the previous two simulation steps. Also, the outputs of the simulation should be compared to the simulation results from previous steps. As a result of exact timing parameters of the involved logic and interconnects, additional information is generated by the post-layout simulation. Designers and reviewers should use the additional timing information to verify whether the design can operate at the required speed or whether there are other timing issues such as glitches, excessive signal skews, etc. As a part of the verification process at this design step, the post-layout report files should be analyzed to confirm that all the timing requirements are met and within margin; the utilization of the FPGA resources is acceptable; the I/O pins are placed at the requested positions; and the I/O parameters such as slew rate or drive current correspond to the design requirements. Most of the EDA tools include chip viewing utilities that can be used to see how the design elements are positioned inside the FPGA device. The utilities also provide information about the fan-outs and the delays for each connection path.

4.4.11 Prototype

The main purpose of the prototyping is to perform hardware verification of the FPGA design that has been previously verified in software. After the post-layout simulation confirms that the design conforms to the initial requirements, a configuration file is generated by the design tool and downloaded into the target FPGA device. FPGA prototyping is needed to verify that the downloaded configuration file produces a configured FPGA with the intended behavior. Prototyping includes the FPGA board design where the FPGA device is placed in the interfacing environment that is expected in the final board design.

Because the software-based simulation from previous steps was done on both modular and top levels, additional access I/O pins such as JTAG boundary scan pins may be needed to perform hardware verification of the blocks inside the FPGA device as the functional I/O pins may not allow access to the

internal signals of interest. These additional pins provide the outputs of the internally connected modules as well as the inputs to the modules that need to be multiplexed with the existing internal inputs. Using this approach, all the internal signals relevant in the hardware verification will be accessible from the FPGA device I/O pins, thereby allowing separate verification of internal blocks as well as the entire design. In the final hardware implementation, these additional pins may be left unconnected with the input pins pulled high or low. The internal logic used to bring in input signals from the outside may be left inside the FPGA device as they do not use significant FPGA resources.

4.4.12 FPGA Verification

FPGA verification should be performed using hardware generated input stimuli that are identical to the stimuli used in the software-based simulation. By comparing the hardware simulation outputs with the software simulation outputs, one can verify whether the hardware implementation conforms to the initial design requirements both at the module level and at the top level. The verification process applied in the design flow is shown schematically in Figure 5, where different intermediate design steps undergo the same simulation process.

The unit under test in Figure 6 represents either a block within the design or the entire design. The stimuli used in the simulation should be designed to perform 100% or as-high-as-practically-achievable testability of the unit under the test.

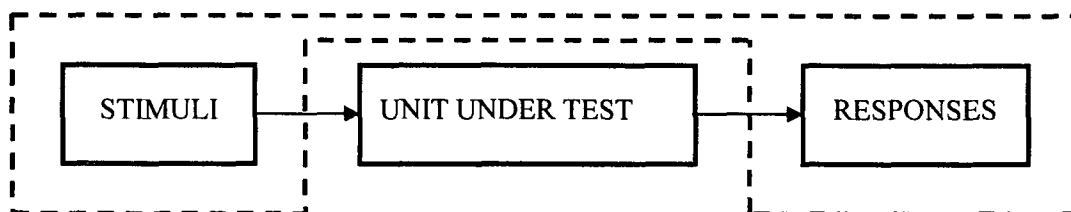


Figure 6. Testing procedure for each design step.

4.4.13 Hardware Implementation

Hardware implementation refers to final board/module design and fabrication. The board may include one or more FPGA devices and the necessary interfacing circuitry. The FPGA hardware design guidance from Chapter 2 should be used by the designers and the reviewers during the hardware implementation. Documentation needed to verify this step of the development life cycle includes schematic diagrams of the board, bill of materials, and data sheets for components on the board interfacing the FPGA device(s).

4.4.14 Functional Hardware Verification

At the end of the development life cycle, functional hardware verification should be performed to confirm correct board/module functionality in the operational environment. In contrast to the verification process described in Figure 6, functional hardware verification is performed on the entire FPGA board with a limited set of input stimuli that covers the input combinations expected during normal operation of the FPGA. Additionally, the input stimuli should include input combinations that are not expected during a normal board/module operation to confirm that there are no unexpected responses.

Functional hardware verification should also include design issues related to the circuitry residing on the same board with the FPGA device. These issues include verifying FPGA device interfaces with

external circuitry including proper I/O logic levels, output driving currents, external resets, external clock sources, FPGA power regulators, etc. FPGA hardware design practices described in Chapter 2 should be used as guidance for the hardware verification.

5. CONCLUSIONS

The FPGA review guidance has been developed to address the gaps in existing regulatory documents used to review instrumentation and controls (I&C) systems in nuclear power plants. The existing documents address the principal relevant issues, such as system safety assessment, development life cycle, V&V, configuration management, and documentation requirements, but they do not offer specific review guidance for FPGA-based systems. For example, the system safety assessment and the design life cycle are addressed in several regulatory documents such as DO-254, IEEE 1012-2001, and IEC 61508, and they can be used for FPGA-based system as a general guide. Also, the V&V process, configuration management, and the development life cycle documentation are addressed in the above documents and in IEEE Std 7-4.3.2. The FPGA-specific review guidance in this document includes FPGA hardware design practices, FPGA design entry methods, and FPGA design methodologies. Several key statements summarize the specific safety design issues in FPGA-based systems.

- FPGAs should be considered a mix of hardware and software and, therefore, specific design and review guidance should be applied. Their static configuration after programming can lead to a conclusion that they are purely hardware systems. However, since the configuration is based on a software-based design process using software design tools, FPGAs are software systems as well.
- The FPGA designer adds to the functionality of an empty FPGA from the FPGA manufacturer as a quality-assurance (QA)- tested and qualified part. To have sufficient assurance that the added functionality satisfies safety requirements, a rigorous and well-defined development and verification process should be applied.
- FPGAs are relatively complex digital components, and, in most of the cases, the FPGA designs are sufficiently complex that 100% testing is practically impossible. In a general case, the FPGA design will contain a significant number of memory elements (i.e., flip-flops). To perform a 100% testing of such a design, not only should the FPGA inputs be exercised for all possible combinations but also all possible internal states should be exercised as well.
- Even if a hardware-accelerated simulation is used, 100% testing may take a prohibitively long time to complete. Therefore, the review of the entire development life cycle should be performed to verify FPGA-based systems unless the design is purely combinatorial with no more than a couple of tens of inputs allowing for 100% testability.
- Design tools used for safety-critical FPGA designs are the mainstream tools that are usually used for large, high-speed FPGA designs. Tool manufacturers typically offer additional support for the designers to fix possible tool errors during the life cycle of the tool version. New tool versions or software packages are offered to handle larger and faster designs as well as to fix all known errors in the previous version. It is, therefore, likely that the mainstream tools will not pass the tool certification process for use in safety-critical design without significant changes and additional development cost. Consequently, the vendor needs to assess the design tools before using it for FPGA design and verification.

APPENDIX A

SURVEY OF FIELD PROGRAMMABLE GATE ARRAY DESIGN GUIDES AND EXPERIENCE RELEVANT TO NUCLEAR POWER PLANT APPLICATION

APPENDIX A

SURVEY OF FIELD PROGRAMMABLE GATE ARRAY DESIGN GUIDES AND EXPERIENCE RELEVANT TO NUCLEAR POWER PLANT APPLICATION

A.1 INTRODUCTION

Oak Ridge National Laboratory (ORNL) has been engaged by the U. S. Nuclear Regulatory Commission (NRC) Office of Nuclear Regulatory Research to develop the technical basis for assessing field programmable gate array (FPGA) technology in safety-related systems within nuclear power plants. In particular, ORNL has investigated programmable digital logic technology and implementation practices to support development of review guidance. As part of this study, ORNL has surveyed information on the use of FPGA technology for high-assurance applications. This report presents the findings of these surveys, along with a summary of particularly relevant programmable logic device standards.

Information for this report was obtained through publicly available sources such as published papers and presentations. No proprietary information is represented.

A.1.1 Background

An FPGA is a digital device containing programmable logic components and programmable interconnects between the logic components. The logic components can be programmed to duplicate the functionality of basic logic gates such as AND¹, OR², XOR³, NOT⁴, or more complex combinational⁵ functions such as decoders or simple math functions. Also, the programmable logic can include memory elements such as simple flip-flops⁶ or more complete blocks of memories.

FPGAs emerged more than two decades ago as a normal process of constantly increasing integration level in digital electronics. They offered a significant improvement in the digital design by moving the logic block interconnects from the designer's responsibility to the specialized synthesis, place, route, and simulation tools. This simple paradigm enabled several major advantages of the FPGA-based design over the existing glue-logic design that was based on extensive board-level interconnects.

First, FPGA design tools offer automatic detection and/or correction of many typical errors that were much more difficult to detect and correct in the old design environment. Second, moving the interconnections from the board level to the silicon level enables a huge reduction in size, power, and price of digital systems. Third, many of the tedious design steps that were prone

¹ AND is a digital logic gate that only outputs a high (1) result when both inputs are high (1). Otherwise, the output is low (0).

² OR is a digital logic gate that outputs a high (1) result when an input is high (1). Otherwise, the output is low (0).

³ XOR is a digital logic gate that outputs a high (1) result when only one input is high (1). Otherwise, the output is low (0).

⁴ NOT is a digital logic gate that is essentially an inverter. If the input is low (0) then the output will be high (1) and vice versa.

⁵ Combinational logic is logic whose output is a function of the present input.

⁶ Flip-flops are electronic circuits with two stable states that are capable of serving as one bit of memory.

to errors are now performed by the FPGA manufacturer so that the whole process can be standardized, tested, and constantly improved. Fourth, FPGA feature sizes are continually decreasing, allowing manufacturers to pack more logic into a single chip. FPGAs today use 65nm technology that can provide nearly 10 million basic logic gates in a single chip. Many of the mentioned advantages of the FPGA technology make it inherently more reliable and safe for use in critical applications. Obviously, the increased complexity requires special care when FPGAs are used in safety-critical systems.

In further detail, a hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. An FPGA designer can program these logic blocks and interconnects after the chip manufacturing process (hence the term “field programmable”), so that the FPGA can perform whatever logical function is needed. Not all FPGAs are truly field reprogrammable. Vendors often sell less flexible versions of their FPGAs, which cannot be modified after the design is committed. For safety applications, this has the advantage of permanently committing the logical functions into an invariant form.

The FPGA design process begins with the designer creating a hardware description language (HDL)⁷ or a schematic design of the desired logical functions. Common HDLs are VHDL⁸ and Verilog⁹. Then, using an electronic design automation tool, a technology-mapped netlist¹⁰ is generated. The netlist can then be fitted to the actual FPGA architecture using a process called place-and-route, usually performed by the FPGA company’s proprietary place-and-route software. Different companies’ place-and-route software packages are likely to implement the same logical functions in different physical layouts. The next step in a typical FPGA design process is for the designer to validate the place-and-route results via timing analysis and performance simulation. Once the design and validation process is complete, the generated binary file (also using the FPGA company’s proprietary software) is used to (re)configure the FPGA.

In an attempt to reduce its complexity, the abstraction level of the HDL design can be raised. A number of FPGA design tools that use high-level languages such as System-C™, LabVIEW™, Matlab™, SystemVerilog, SystemVHDL, and Handel-C™ have been recently developed. To further simplify the design of complex systems in FPGAs, libraries of predefined complex functions and circuits, with widely varying performances in speed, accuracy, reliability, etc., are commonly employed as a block to avoid having to recreate previously developed logic.

Generally, FPGAs can perform any arbitrary logic function for which they have been programmed, so they can be deployed in nuclear power plants in place of any logic function component such as trip logic units, engineered safety feature (ESF) actuation decision logic, or digital communication interface priority logic. Due to their technical capabilities, FPGAs are currently widely deployed for industrial applications requiring fixed or infrequently changing logical functions.

FPGAs constitute a broad technology class with differing implications for their application to safety systems based on the particular details of the implementation. In its simplest form, an FPGA could be restricted to implementing small logic blocks such as interdivisional voting. This type of implementation would likely lack any system memory (signal history), and therefore may be sufficient to be completely and deterministically analyzed and tested.

⁷ HDL describes the components operation, design, and organization.

⁸ VHDL stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language and is the most common software for FPGA and ASIC designs.

⁹ Verilog is another HDL for FPGAs and ASICs. Its syntax is similar to C programming language.

¹⁰ A netlist describes the connections that need to be made for the design.

In a more advanced form, the FPGA could possess memory functions and a set of basic math functions. This type of FPGA would very likely be designed, validated, and tested using computer-based tool sets based on some form of formal verification. The logic implementing this type of FPGA would almost certainly be too complicated to be completely validated analytically due to the extremely large number of possible logic states. Even more advanced logical functions can be implemented within FPGAs, including embedded microprocessors with their related peripheral components, enabling the creation of system-on-a-chip devices. These types of systems include both the digital logic and software designs, greatly increasing the overall complexity of the validation process.

Even though designing FPGA-based systems involves hardware implementation of logical functions, the design process itself is highly software intensive. Hence, errors within the software design process can result in undesired behavior of logical functions implemented in hardware. In this case, the software validation process extends to assessment of the FPGA design tools.

A.1.2 Research Approach

From a safety perspective, it is difficult to assess the correctness of FPGA devices without extensive documentation, tools, and review procedures. NUREG/CR-6463, "Review Guidelines on Software Languages for Use in Nuclear Power Plant Safety Systems [52]," provides guidance to NRC staff for auditing of safety system programs written in ten high-level languages. A uniform framework for the formulation and discussion of language-specific programming guidelines was employed. Comparable guidelines based on a similar framework are needed for FPGA-based systems. It is the objective of this project to develop the technical basis for these guidelines.

The first task in this research project involved evaluation of regulatory experience gained by other countries and other agencies, and experience captured in existing standards to identify regulatory approaches that can be adopted by NRC. If existing regulations do not provide a sufficient framework for adopting relevant regulatory approaches that are uncovered, ORNL was tasked with identification of the gaps. This report presents the findings of this initial research activity.

A.1.3 Report Organization

This report contains summaries of documents discussing the use of FPGAs in safety-critical systems. The summaries are divided into two main sections: 1) Regulatory Approaches by Other Countries and Agencies, and 2) Existing Standards. The first section contains documents found with regulatory experience from Japan and France. The remaining section is composed of documents dealing with the aerospace industry, which includes those from the European Space Agency (ESA), the National Aeronautics and Space Administration (NASA), and general FPGA findings for space, the nuclear industry, and the automotive industry. The final section includes the existing standards such as "Design Assurance Guidance for Airborne Electronic Hardware," DO-254¹¹ [11].

¹¹ DO-254 is a standard for complex electronic hardware created by the Radio Technical Commission for Aeronautics (RTCA), which develops standards for the Federal Aviation Administration (FAA).

A.2 FPGA-RELATED TECHNICAL STANDARDS/ PUBLISHED MATERIAL

Internet searches of technical standards related to FPGA design were performed. The main goal was to find FPGA design standards from other countries, industries, etc. to assess and possibly adopt as a basis for reviewing FPGA-based nuclear power plant safety system applications. Although the searches did not result in any standards specifically dedicated to the use of FPGAs, a total of more than 80 documents were identified as relevant. The material was then examined more closely for topics related to the following:

- Integrity of the FPGA programming process and methods of FPGA code and hardware verification and validation (V&V),
- Single event effects (SEEs) and the techniques to reduce/eliminate their impact on the FPGA functionality, and
- Safe hardware design practices specific to FPGAs

Of the 83 documents, 22 were selected based on relevance to the above criteria for further review. The selected documents are listed below:

Regulatory Approaches by Other Countries and Agencies

- Transition and Current Status of NPP C&I System of BWRs in Japan [15]
- PLD-Based Safety Critical Systems: An Introduction and Survey [31]
- A Comparison of Radiation-Hard and Radiation-Tolerant FPGAs for Space Applications [23]
- Formal Verification of Fault Tolerance in Safety-Critical Reconfigurable Modules [28]
- Lessons Learned From FPGA Developments [29]
- Application-Specific Integrated Circuit (ASIC) Design and Manufacturing Requirements [39]
- Independent Verification and Validation: First Year Summary Report for the Programmable Logic Devices Research [63]
- A Preliminary Practitioner's Guide to Defect Detection in VHDL Based Designs [66]
- Architectural Principles for Safety-Critical Real-Time Applications [55]
- Harmonization of the Licensing Process for Digital Instrumentation and Control Systems in Nuclear Power Plants [36]
- AP1000 Instrumentation and Controls [41]
- Design, Test, and Certification Issues for Complex Integrated Circuits [37]
- FPGA Space Qualification Presentation [56]
- Suitability of Reprogrammable FPGAs in Space Applications [67]
- VHDL Modeling Guidelines [38]
- Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems [42]
- Reliability Considerations for Automotive FPGAs [62]
- Embedded Digital System Reliability & Safety Analyses (NUREG/GR-0020) [34]

Existing Standards

- Design Assurance Guidance for Airborne Electronic Hardware (DO-254) [11]
- IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations (IEEE 7-4.3.2) [40]

- International Standard for Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems – Part 2: Requirements for Electrical/Electronic/Programmable Electronic Safety-Related Systems (IEC 61508-2) [54]
- IEEE Standard for Software Verification and Validation (IEEE 1012-2004) [80]

A.3 SUMMARIES OF SELECTED STANDARDS/ PUBLISHED MATERIAL

Summaries have been provided for each of the 22 selected documents. The summaries are meant to highlight issues/solutions to designing safety-critical systems with FPGAs.

A.3.1 Regulatory Approaches by Other Countries and Agencies

A.3.1.1 Japan

Transition and Current Status of NPP C&I System of BWRs in Japan [15]

This document overviews the number and status of nuclear plants in Japan. The document also presents the plan for controls and instrumentation (C&I) modernization using FPGA-based modules. The Power Range Neutron Monitor safety system design is based on a one-time programmable FPGA that has been recently developed and is currently undergoing quality assurance. Many other C&I systems under development are to be based on the FPGA technology.

A.3.1.2 France

PLD-Based Safety Critical Systems: An Introduction and Survey [31]

This report was prepared by the University of Virginia (UVA) for Électricité de France. The first half of the report is a survey of FPGA devices. The report then discusses the use of mil-spec parts, test coverage (off-line and on-line), and triple-modular redundancy (TMR). Fault tolerance was also included and highly advocated due to degradation and/or disruption of the configuration memory. The report did raise concerns about the lack of independent fault containment regions, clocking, and power issues if replicated modular fault tolerance were to be achieved on an FPGA. Formal verification methods such as equivalence checking and model checking were recommended as good design practices. The report also identified the DS-00-54 [83] as well as citing DO-254 [11] as the best guidance to date for defining the desired performance rather than prescribing how to achieve it.

A.3.1.3 Aerospace Industry

A.3.1.3.1 General findings

A Comparison of Radiation-Hard and Radiation-Tolerant FPGAs for Space Applications [23]

The document compares rad-hard FPGA families, static random access memory- (SRAM-) based Xilinx® FPGAs, and one-time programmable Actel® FPGAs based on their performances for space applications. The performance characteristics compared are: total ionizing dose (TID) performance, single event upset (SEU) performance, SEU mitigation techniques, fan-out¹², package quality, operating temperature, operation clock speed, set-up and configuration time, power consumption, and known quality issues. The document also mentions Aeroflex™ and Atmel™ rad-hard FPGAs.

¹² The term fan-out is used to indicate the maximum number of digital inputs that can be fed by the output of a single logic gate.

Formal Verification of Fault Tolerance in Safety-Critical Reconfigurable Modules [28]

This document considers Esterel, a formal verification language for FPGA-based safety systems. It describes a design process that includes top-level design and verification as well as automatic code generation for synthesizable VHDL. The document states that this process reduces the likelihood of systematic faults. Also, it reveals how Esterel can be used for failure mode and effects analysis (FMEA) and for fault tree analysis (FTA).

Design, Test, and Certification Issues for Complex Integrated Circuits [37]

This document mainly focuses on ASICs but does contain some information on SRAM and antifuse FPGAs. It stated that as more sequential logic becomes available in a device, the more difficult it becomes to test that logic. Signal delays in FPGAs were discussed with causes listed as 1) signal wire characteristics, 2) programmable elements, 3) amount of cascaded logic cells, and 4) propagation delay of each logic cell.

FPGA Space Qualification Presentation [56]

This document describes qualification of FPGA parts for space applications. The aerospace community and the United States government updated two main space qualification standards (MIL-STD-1546 and MIL-STD-1547) and published those updates as *Aerospace Technical Operating Report*. Then in 2006 the standards were updated again to include more stringent requirements and will be published again as MIL-STD-1546 and MIL-STD-1547. These documents will be used to qualify FPGA manufacturing processes as well.

Suitability of Reprogrammable FPGAs in Space Applications [67]

This document investigates SEU issues related to FPGAs emphasizing Xilinx® XC4000 and Virtex® FPGAs, both rad-hard and general use versions. It briefly describes reprogrammable FPGA technology and its susceptibility to SEUs. The SRAM-based FPGAs have been used recently in applications such as avionics, space exploration, and high performance reconfigurable processors (different SEU mitigation techniques are reported for these applications). The most common is the TMR technique, but others include adding idle cycles for concurrent error detection, Hamming codes, and other parity codes, built-in self-test (BIST), etc. To increase rad-hardness and reduce SEU sensitivity, most FPGA manufacturers use some kind of rad-hardened adjustment of their standard commercial foundry.

The document also describes three kinds of SEU in SRAM-based FPGAs. These are configuration memory upsets, used logic upsets, and architectural upsets [joint test action group (JTAG) upsets]. Several sensitive FPGA structures have been identified such as sequential and combinatorial logic, half-latches, lookup tables (LUTs), block random access memory (BRAM), digital clock manager (DCM), input/output (I/O) logic, and JTAG. The document covers many of the most common SEU mitigation techniques such as configuration memory protection, user logic protection, module-level protection, and gate-level protection. The document also reports some of the results of various SEU and TID tests performed on Virtex® and Xilinx® XC4000 FPGAs.

A.3.1.3.2 European Space Agency

VHDL Modeling Guidelines [38]

This document defines acceptable practices for designing VHDL models and test benches used by the ESA. The purpose of these requirements is to ensure the models are of high quality so they can be efficiently used and maintained throughout the full life-cycle of a safety critical system. Some of the requirements are to use VHDL93, use the English language, limit the number of characters per line to 80, comment the design in detail within the code, use a defined code header at the beginning, and use assertions. The document suggests avoiding the buffer

mode for the ports of top-level entity and single wait statements, in which a process statement with sensitivity list should be used.

Lessons Learned From FPGA Developments [29]

This document contains information regarding problems encountered and lessons learned in the use of FPGAs involved in satellite missions from the ESA and NASA. This document has also been used by these agencies as an FPGA design guideline. However, the report only focuses on existing once-only programmable devices. The following topics were discussed as lessons learned: (1) transient performance of components not adequately accounted for in the design; (2) little to no documentation from FPGA designers and no established SEU requirements; (3) timing, static timing, clock skew, and low power designs performed with ASICs should be tested with FPGAs; (4) FPGA verification should not be done in isolation by the designer; and (5) specifications should be established to define all relevant system configurations and characteristics to a level allowing FPGA device requirements to be derived.

The document concluded that employing FPGAs for critical use is only recommended when appropriate risk analysis has been performed and when the developer can prove that the selected FPGA will fulfill its task in a given application and environment.

ASIC Design and Manufacturing Requirements [39]

This document presents the requirements for ASIC (not FPGA) design used by ESA. Although FPGA design is not discussed, FPGA and ASIC design share many similarities. The document requires VHDL-based simulation at the architectural level, including the ASIC and other components on the board. Also, during the detailed design, VHDL should be used to simulate the ASIC's functionality. Later, during the prototype testing, the same VHDL test benches should be used. The ASIC's set of specific design requirements include (1) use of asynchronous reset rather than synchronous reset (the ASIC state should be completely deterministic after the reset); (2) ensuring the design is as synchronous as possible throughout the design; (3) mitigation of metastability issues; (4) power minimization using the clock control in the design; (5) addressing the SEU issues; (6) avoidance of floating nodes; (7) avoidance of bus contention; and (8) elimination of unnecessary circuitry.

A.3.1.3.3 National Aeronautics and Space Administration

Independent Verification and Validation: First Year Summary Report for the Programmable Logic Devices Research [63]

This document is related to the SAI Corporation's 2005 study for NASA's Goddard Software IV&V Facility regarding independent verification and validation (IV&V) of defect detection in VHDL based FPGA designs [79, 66]. The document concentrates on verification and validation (V&V) of the VHDL code design, particularly syntax, I/O unknown states, coding style, unnecessary circuitry, dangerous semantics, etc.

The document examines four existing standards for software and hardware V&V: NASA-STD-8739.8 [72], IEEE STD 1012-1998 [80], IEEE STD 1076-2002 [81], and DO-254 [11]. All of these documents mention the need for IV&V with regards to PLDs. The document stated that, "NASA has provided no clear guidance on the software aspects, design and development of PLDs, or how to assure safety, reliability, or quality of these hybrid devices."

Further, the document surveys formal PLD verification techniques, listing the most popular software tools for model checking, emphasizing the model-based verification of the VHDL programs where the design specification is used to test the designer's code. The document also lists most frequent structures in a VHDL design, and lists the VHDL code design practices that need to be avoided.

A Preliminary Practitioner's Guide to Defect Detection in VHDL Based Designs [66]

NASA's preliminary guide suggests an IV&V process for VHDL validation in FPGAs. The process includes artifacts (documents) collection, VHDL standard compliance analysis, pedagogical code examination, design artifact analysis, and final assessment.

Architectural Principles for Safety-Critical Real-Time Applications [55]

This document discusses redundancy management, common mode/cause failures affecting multiple regions, fault avoidance, tolerance, and removal, and exact versus approximate consensus. Stating that critical systems in many industries (such as the aerospace industry) are usually designed from the ground up, it concludes that VHDL and a synthesis methodology should be integrated with formal specification and verification. It was also noted that for safety-critical applications, physical operational hardware faults no longer pose a major threat to dependability, but the dominant threat is now common mode failures, for which no single theory can be applied and for which multidiscipline, multiphase defense is required.

A.3.1.4 Nuclear Industry

Harmonization of the Licensing Process for Digital Instrumentation and Control Systems in Nuclear Power Plants [36]

A Technical Working Group on Nuclear Power Plant Control and Instrumentation (TWG-NPPCI) met in 2001 and 2002 to develop consensus positions on processes for efficiently licensing nuclear power plant safety instrumentation and control systems for a worldwide market. The working group concluded that the present national licensing approaches of digital I&C for nuclear power plants needs to be reduced to a set of documents that can provide the industry with structure and support instead of continually producing and having to refer to more documents.

AP1000 Instrumentation and Controls [41]

Chapter 7 of this document discusses the descriptions and commitments pertaining to the primary instrumentation and control systems of the AP1000 advanced nuclear power plant design. The system uses microprocessor-based distributed digital systems to perform plant protection and control functions and safety monitoring. Digital components for safety systems should be qualified for their intended application by either a 10 CFR Part 50, Appendix B quality assurance program or the item should be dedicated for use in the safety system as defined in 10 CFR Part 21. The NRC-approved EPRI TR-106439, "Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications," (1997) and BTP HICB-18 "Guidance on the Use of Programmable Logic Controllers in Digital Computer-Based Instrumentation and Control Systems." [61, 82]

Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems [42]

This document provides guidance on evaluating the life cycle of safety system software. Figure A-1 identifies the documented evidence that BTP 7-14 indicates should be provided to the NRC by the licensee.

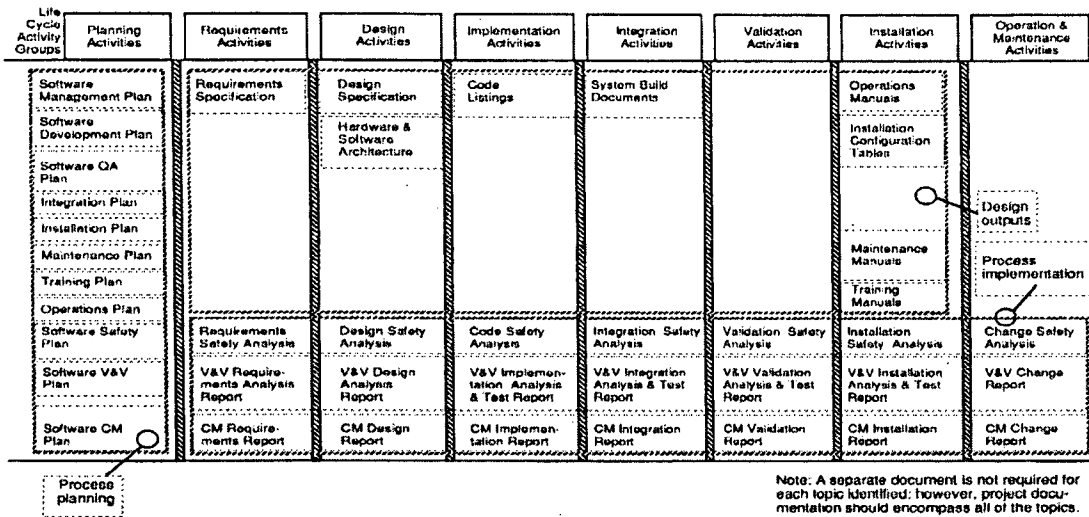


Figure A-1. Documents required throughout the software life cycle [42].

Embedded Digital System Reliability & Safety Analyses (NUREG/GR-0020) [34]

NUREG/GR-0020 discusses the dependability analysis of embedded digital systems as well as metrics that characterize dependability, reliability, availability, and safety. This regulatory research report describes the most common axiomatic models such as Markov models, Petri nets, and fault trees. Each of these models are implemented using commercially available software tools for dependability analysis.

NUREG/GR-0020 also identifies methods that can be used to achieve dependability: defense in depth, redundancy, diversity, and robustness. The most important dependability parameters are failure rate, repair rate, and coverage. The document classifies four types of redundancy: hardware, software, time, and information redundancy. Diversity is classified as human, design, software, functional, equipment, and signal diversity as in NUREG/CR-6303.

In discussing the embedded digital systems reliability and safety analysis, the document recommends that the hardware and software in these systems be analyzed as integral parts of the systems and not separately, as is common practice. However, the document does not address any methods specific to the digital systems, but only suggests that the existing regulatory documents be used for digital systems as well.

A.3.1.5 Automotive Industry

Reliability Considerations for Automotive FPGAs [62]

This Actel paper focuses on the fundamental importance of technology selection and its relationship to overall system reliability relative to the automobile industry. Cause and cure are emphasized. The following topics are discussed: temperature as a primary stress factor in semiconductor failure, neutron-induced soft and firm errors, tamper resistance in automotive FPGAs, and time-dependent dielectric breakdown (TDDDB). Reliability problems frequently encountered by FPGAs are typically due to one of four root causes: (1) the packaging technology, (2) assembly technology, (3) environmental overstress, or (4) electrostatic discharge (ESD). Exposure to high temperature exacerbates these types of problems. Actel also states that antifuse FPGA architectures used for automotive applications are superior in their tolerance to extended temperature exposures (+150°C).

This document discusses SEUs and the fact that it is not possible to shield against high-energy neutrons, so designers should either account for the effects of such neutrons or use neutron-resistant technology. Of the three main FPGA technologies, antifuse, Flash, and SRAM, only antifuse and Flash are immune to the effects of neutron-induced soft and firm errors. SRAM-based products are the least secure of all technologies.

A.3.2 EXISTING STANDARDS

Design Assurance Guidance for Airborne Electronic Hardware (DO-254) [11]

The DO-254 standard establishes assurance guidelines for complex hardware systems that use FPGAs, complex programmable logic devices (CPLD), and ASICs. This standard is concerned with the entire hardware design life cycle—planning, hardware design, validation, verification, configuration management, process assurance, and certification. However, the standard considers FPGAs as purely hardware devices ignoring the fact that FPGA design involves Hardware Design Language (HDL) programming and simulation typical for software systems. Also, the standard does not include any details regarding safe FPGA design practices, acceptance criteria, or licensing procedures that are necessary parts of a regulatory document for I&C safety systems in nuclear power plants.

DO-254 defines five levels of safety criticality from Level A, the most critical, to Level E, not critical. Also, the standard requires the assessment of the hardware safety using the following principles: (1) circuit or component redundancy, (2) separation or electrical isolation between circuits or components, (3) dissimilarity (i.e., diversity) between circuits or components, (4) monitoring of circuit or components, (5) protection or reconfiguration mechanisms, (6) allowed failure rates and probabilities for the circuit and component random failures and latent failures, (7) limitation of usage or installation, and (8) prevention and management of upsets and upset recovery. However, the document does not address the specific failure modes for FPGA-based safety systems.

IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations (IEEE 7-4.3.2) [40]

IEEE 7-4.3.2 supplements IEEE 603 by addressing the use of computers as part of safety systems in nuclear power plants. This standard includes (1) software quality (software tools, V&V and IV&V requirements [IEEE 1012-1998]), software configuration management, and software program risk management); (2) data communication between safety systems and safety to non-safety systems (performance of the safety function shall not be inhibited); and (3) common cause failure criteria (guidance on performing an engineering evaluation of software common-cause failures).

International Standard for Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems—Part 2: Requirements for Electrical/Electronic/Programmable Electronic Safety-Related Systems (IEC 61508-2) [54]

IEC 61508 is a standard that provides a generic approach intended for all industries using electrical/electronic/programmable electronic components to perform safety-related activities. This standard discusses safety requirements and provides a list of specifications that should be met. It also contains a list of safety integrity requirements.

The standard states that using static, dynamic, and failure analysis should reduce the test cases needed and that there should be an estimated rate of failure. Finally, it is stated that it is practically impossible to list all physical failures of complex hardware. One reason given is the difficulty in determining the relationship between failures. Another reason is that there is a

greater contribution of systematic failures in contrast to random failures when complex hardware and software is used. Failures should also be categorized in terms of failures caused by faults, before or during system installation, and failures caused by faults or human errors, after system installation.

IEEE Standard for Software Verification and Validation (IEEE 1012-2004) [80]

IEEE-1012-2004 deals with the software V&V process throughout the entire software life cycle. The V&V process determines whether the developed products for a particular activity, during the software life cycle, conform to the requirements of that activity and whether the software satisfies the intended use. The lifecycle processes include acquisition, supply, development, operation, and maintenance.

The purpose of the standard is to establish a common framework for V&V processes, define V&V tasks, required inputs and required outputs, identify the minimum V&V, and define the content of the software V&V plan (SVVP). The standard provides a detailed description of each of the above processes and underlines specific activities within each process. Each activity is further divided into V&V tasks.

The V&V tasks as well as inputs and outputs are included in an easy-to-follow table that leads the user through the entire V&V process. The table also defines the V&V independence by three parameters: technical independence, managerial independence, and financial independence.

IEEE-1002-2004 is a software-only standard, and it can not be directly applied to V&V process for FPGA-based systems. Even though the top level V&V processes and underlying activities are generic and can be used for FPGAs, the low level tasks are software specific, and not directly applicable to FPGAs.

A.4 DOCUMENT EVALUATION/COMMENTS

It is evident from reviewing the documents discussed in this report that there is no ready-to-use regulatory guidance directly applicable to the FPGA-based safety-critical system design. However, DO-254 does represent a good overall approach for design of hardware-based safety-critical systems. DO-254 considers all phases of the hardware-design life cycle including requirement capture, conceptual design, detailed design, implementation, and production transition. For each of these phases, it describes how to implement V&V, configuration management, process assurance, and certification. It also describes how to apply the assessment and qualification process for the software/hardware tools used during design.

IEEE 7-4.3.2, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," concentrates on the software side of safety-critical system design using digital computers. It can be used as guidance for developing V&V planning, configuration management, requirement traceability, failure modes and effects, and environmental qualifications during the software design. Generally, the software side of the FPGA design more closely resembles the assembly language programming in computers and only recently have higher-level languages been used for FPGA design. The software design in digital computers and microprocessors has almost exclusively involved the use of high-level programming languages and powerful compilers.

Many of the reviewed documents recognize the need for a specific design approach when considering FPGAs for safety-critical systems. This is particularly elaborated in documents from the aerospace community as well as in the transportation and auto industries. This specific design

approach is due to the unique FPGA characteristics that require concurrent and interdependent hardware and software design paths.

The reviewed documents identify a large number of safe FPGA design practices, including hardware and software design. The hardware design practices include issues such as board-level design, FPGA logic design, mitigation of SEEs in FPGAs, programming guidelines, etc. Software design practices include using schematic entry for time-critical designs, avoiding unsafe and ambiguous VHDL and Verilog programming structures, ensuring full coverage during simulation and hardware verification processes, using formal methods to support V&V, etc.

Thus far, the FPGA design methodology used by the manufacturers of safety-critical systems has been based on mainstream FPGA design tools that have not been certified for use for safety-critical designs. The FPGA design verification is performed by exhaustive simulation-based testing. It seems that this approach has been accepted as offering sufficient confidence in a design primarily because such testing involves relatively simple designs with full-simulation coverage capability. However, the growing complexity of FPGA designs in safety-critical systems is likely to require a different design methodology based on more formal verification methods. Furthermore, the mainstream FPGA design tools should be certified for use in developing safety-critical systems.

Recently vendors have started offering FPGA-based systems for I&C safety systems in nuclear plants. From the available documentation, these systems use FPGAs generally for relatively simple tasks. The most frequently employed FPGAs are one-time programmable FPGAs with built-in redundancy for SEE mitigation. Also, some European systems use CPLDs instead of FPGAs, which are appropriate for simple logic subjected to benign environments. SRAM-based reprogrammable FPGAs are widely used in aerospace and military applications where high-density FPGAs and reprogrammability are needed. However, these FPGAs require a different set of mitigation techniques to address SEEs in the configuration and user logic.

A.5 RELEVANT GUIDANCE

When extracting relevant guidance from the existing regulatory and other documents for FPGA-based systems, it is useful to consider two different groupings of information; the first is related to the design assurance guidance applicable to FPGA-based systems and the second involves acceptable FPGA design practices. The first category of information can be extracted from different guidance documents such as DO-254, IEEE 1012, IEEE 603, IEEE 7-4.3.2, and IEC 61508, while the FPGA design practices are covered at different levels and relevance in many conference, journal, and other papers. In this section, these relevant pieces of information are listed in more detail so that the letter report can serve as a framework for the new review guidance document for FPGA-based systems.

A. Design Assurance Guidance

System Safety Assessment

It is necessary for the FPGA-based systems to undergo a methodical system safety assessment. DO-254 offers an acceptable approach where the entire assessment process is divided into three groups:

- Functional Hazard Assessment
- Preliminary System Safety Assessment
- Safety Assessment Process

Also, several documents define a multilevel assurance system and their corresponding failure conditions. While IEC 61508 has only three levels (i.e., low, medium, and high), DO-254 uses a five-level assurance system with the following levels:

- Catastrophic
- Hazardous/Severe
- Major
- Minor
- No Effect

IEEE 1012-2004 uses four software integrity levels based on consequences and mitigation potential.

For failure analysis and control, Annexes A and B of IEC 61508 provide useful information with the measures to avoid and control failures being presented in a comprehensive way.

Design Life Cycle for FPGA-based systems

An appropriate design life cycle for FPGA-based systems should include a planning process, hardware/software design processes, and a number of supporting processes as described in DO-254. The planning process may include specification of the hardware/software design cycle, selection of standards and other regulatory documents, selection of hardware and software design tools, selection of necessary supporting processes, etc. The hardware/software design life cycle should include processes such as design requirements, conceptual design, detailed design, implementation, product transition, product upgrade, maintenance, etc. Supporting processes include validation and verification, configuration management, process assurance, and the life cycle design documentation.

IEC 61508 defines the following safety life cycle activities: requirements specification, validation planning, design and development, integration, operation, maintenance, validation, modification, verification, and functional safety assessment.

IEEE 1012-2004 defines software-specific life cycle processes such as management, acquisition, supply, and development. In this standard, the software is considered a product developed by an outside vendor, and as such used by the system developer. Therefore, the software life cycle processes are not directly applicable to the FPGA systems. The FPGA software (code) development is closely related to the hardware design, and it is usually performed by the same design teams that are designing the hardware platforms to run that FPGA code. On the other hand, software, as defined in IEEE 1012-2004, can be purchased as a final product and then run on a standard of-the-shelf hardware platform.

Validation and Verification

The validation process is defined here as a process that is intended to ensure that the design requirements for the FPGA-based system are correct and complete with respect to the overall system requirements. The verification process is defined as the process that provides assurance that the FPGA-based system meets its design requirements.

In FPGA-based systems, the validation and verification needs to be done for all the processes of the design life cycle related to the system manufacturing, such as design requirements, conceptual design, detailed design, implementation, product upgrade. Independent V&V may be required as well.

The DO-254 guidance defines the V&V process for the hardware systems, including separate activities that may be applied to any of the life cycle design processes. IEEE 1012 also provides the useful concept of V&V activities for every life cycle process with detailed V&V tasks. Since these tasks are related to the software only, they need to be adjusted for use in FPGA-based systems. Also, IEEE 1012, in its Annex C, describes how to implement independent V&V in more detail. There, the independent V&V is defined by three parameters: technical independence, managerial independence, and financial independence. The IEEE 7-4.3.2 standard addresses the V&V process for safety systems in nuclear plant and includes the requirements for independent V&V.

Configuration Management

An FPGA-based system design needs to be supported by a configuration management process to ensure that the system and its parts can be consistently replicated and modified if necessary. DO-254 guidance can be a good reference for establishing configuration management objectives and activities. IEEE 1012-2004 addresses these issues in its software V&V reporting, administrative, and documentation requirements.

Design Life Cycle Documentation

Requirements for the design documentation throughout the design life cycle are specified in many regulatory documents, including DO-254 and IEEE 1012-2004. The main purpose for such documentation is to provide evidence of design assurance and compliance with certification requirements. As defined in DO-254, the documentation needs to be unambiguous, complete, verifiable, consistent, modifiable, and traceable. It should cover the following: hardware/software design plans, design standards and guidance, hardware/software design data, validation and verification data, hardware/software acceptance test criteria, problem reports, configuration management records, process assurance records, etc.

B. Acceptable FPGA Design Practices

Acceptable design practices can be divided into three major groups: system-level design practices, board-level design practices, and FPGA-level design practices. Many of the system-level and the board-level practices for general digital systems are well documented, and they can be applied to FPGA-based systems as well. However, some specific design issues are related specifically to FPGAs that need to be addressed. These issues include power dissipation, simultaneous I/O switching, high-speed line termination, and ground bounce.

The main published sources of information for acceptable FPGA design practices arise from numerous conference and journal papers, FPGA application notes and data sheets, technical reports, white papers, regulatory documents, and internet sites.

Most of the board-level design issues are covered in the application notes of major FPGA manufacturers such as Xilinx, Altera, Actel, and others.

The NASA web site, www.klabs.org, gives substantial useful information related to all major groups of the FPGA design practices. The site offers the most comprehensive coverage of SEU issues in FPGAs because these effects are very critical for space applications.

Acceptable VHDL design practices are covered in several documents listed in Appendix B [29], [38], [39], and [66].

Verification of FPGA-based designs includes a wide range of different methods and techniques such as 100% testability, code coverage, formal methods, insertion-based verification, hardware-software co-simulation. The following documents listed in Appendix B address most of these issues: [22], [28], [30], [37], [63], [76], and [79].

A.6 CONCLUSION

In conclusion, there are several guidance documents such as DO-254 and IEEE-1012-2004 that can be used as the framework for comprehensive guidance on the design of safety-related FPGA-based systems. These documents address the principal relevant issues such as system safety assessment, design life cycle, V&V, configuration management, and documentation requirements. In particular, DO-254 stands as a suitable base upon which comprehensive guidance can be established. However, the life cycle framework offered by DO-254 is restricted to treatment of FPGA-based devices as solely hardware components and does not adequately address the software-like nature of “hard logic” embodied by more complex FPGA designs that cannot be fully tested. Thus, the guidance in DO-254 would need to be enhanced through adoption of software-based system life cycle considerations and activities addressed in other standards. Additionally, several FPGA-specific areas are not sufficiently addressed in any of the guidance documents reviewed and need to be added to the FPGA review guidance. These are related to the acceptable FPGA design practices covering the system level design, the board level design, and the FPGA logic design.

A key need to achieve sufficiently comprehensive guidance arises from the need to tailor the V&V process to account for the specific characteristics of FPGAs. In particular, guidance should cover both the software tools V&V as well as the FPGA design V&V. The inherent complexity of FPGA designs in most cases prohibits 100% code coverage during the code simulation and hardware verification. Therefore, a suitable V&V process is needed for FPGA-based safety related designs. This process should include design tool verification, such as that established in IEEE 7-4.3.2 for example. Also, comprehensive guidance needs to include specific FPGA design practices that will lead to verifiable designs. Some of these practices may include using combinatorial design only, partitioning the design to verifiable pieces, redundancy, error reporting, etc.

APPENDIX B

RESULTS OF SEARCH FOR TECHNICAL STANDARDS RELATED TO FPGA DESIGN

APPENDIX B. RESULTS OF SEARCH FOR TECHNICAL STANDARDS RELATED TO FPGA DESIGN

1. R. Katz (2005, Jul.) "This Is What We Find In This Stuff: A Designer Engineer's View," Presentation at the FY2005 Software/Complex Electronic Hardware Standardization Conference, Norfolk, Virginia, July 26–28, 2005.
http://www.klabs.org/richcontent/Tutorial/MiniCourses/stuff_faa_nasa_2005/index.htm
2. G. Chen, F. Li, M. Kandemir, and I. Demirkiran, "Increasing FPGA Resilience Against Soft Errors Using Task Duplication," ASP-DAC-2005, pp. 924–927.
<http://ieeexplore.ieee.org/iel5/9883/31416/01466490.pdf>
3. John Lach, William H. Mangione-Smith, and Miodrag Potkonjak, "Enhanced FPGA Reliability Through Efficient Run-Time Fault Reconfiguration," *IEEE Transactions on Reliability*, Vol. 49, No. 3, September 2000 pp. 296–304.
<http://ieeexplore.ieee.org/iel5/24/19750/00914546.pdf?arnumber=914546>
4. Chandru Mirchandani, "Using Software Rules To Enhance FPGA Reliability," P226/MAPLD2005, September 2005. http://www.klabs.org/mapld05/presento/226_mirchandani-bof-w.ppt
5. D. Czajkowski, D. Strobel, P. Samudrala, and M. Pagey, "Radiation Hardened, Ultra Low Power, High Performance Space Computer Leveraging COTS Microelectronics With SEE Mitigation," Space Micro Inc. (MAPLD2005/138).
http://www.klabs.org/mapld05/presento/138_czajkowski_bof-m.pdf
6. Howard Bogrow, "The Continued Evolution of Re-Configurable FPGAs for Aerospace and Defense Strategic Applications," Xilinx (MAPLD2005/176).
http://www.klabs.org/mapld05/presento/176_bogrow_p.ppt
7. Carl Carmichael, Brendan Bridgford, and Xilinx, Inc., "A Cost/Benefit Framework for Evaluating Re-Configurable FPGA SEU mitigation Techniques," Xilinx (MAPLD2005/194).
http://www.klabs.org/mapld05/presento/194_bridgford_p.ppt
8. Carl Carmichael, Sana Rezgui, Gary Swift, Jeff George, and Larry Edmonds, "SEE Validation of SEU Mitigation Methods for FPGAs," California Institute of Technology, Jet Propulsion Laboratory, and National Aeronautics and Space Administration (P201-L/MAPLD2005). http://www.klabs.org/mapld05/presento/201_carmichael_bof-l.ppt
9. Sajid Baloch, Tughrul Arslan, and Adrian Stoica, "Design of a 'Single Event Effect' Mitigation Technique for Reconfigurable Architectures," MAPLD 2005, Submission 1024, Session P and L. http://www.klabs.org/mapld05/abstracts/1024_baloch_a.pdf
10. Department of Defense, "Test Method Standard Microcircuits," MIL-STD-883E, December 1996. <http://atlas.web.cern.ch/Atlas/GROUPS/FRONTEND/WWW/RAD/RadWebPage/StandardMeth/milstd~1.pdf>
11. Advisory Circular, "RTCA, Inc., Document RTCA/DO-254, Design Assurance Guidance for Airborne Electronic Hardware," June 2005.
[http://www.airweb.faa.gov/Regulatory_and_Guidance_Library/rgAdvisoryCircular.nsf/7aab5bad14f9417885256a35006d56b0/6d4ae0bf1bde3579862570360055d119/\\$FILE/AC%2020-152.pdf](http://www.airweb.faa.gov/Regulatory_and_Guidance_Library/rgAdvisoryCircular.nsf/7aab5bad14f9417885256a35006d56b0/6d4ae0bf1bde3579862570360055d119/$FILE/AC%2020-152.pdf)
12. Terrence Leier and Robert Haug, "Best Practices in Complex Electronic Hardware Development," Rockwell Collins, July 2005.

13. Thomas Phan, "Special Delegations for Complex Hardware & TSO Software," Presented to 2005 National Software and Complex Electronic Hardware Standardization Conference, Norfolk, Virginia, July 2005.
http://klabs.org/richcontent/conferences/faa_nasa_2005/presentations/tuesday_general.htm
14. Memorandum from Kim Smith, Manager of Small Airplane Directorate at FAA, "Applying Advisory Circular 20-152, 'RTCA, Inc., Document RTCA/DO-254, Design Assurance Guidance for Airborne Electronic Hardware,' to Title 14 Code of Federal Regulations, Part 23 Aircraft; PS-ACD100-2005-50001."
[http://www.airweb.faa.gov/Regulatory_and_Guidance_Library/rgPolicy.nsf/97a612e22b32398d85256b7500496a9a/cf51a956f07b0c208625727c006745ca/\\$FILE/PS-ACE100-2005-50001%20final.pdf](http://www.airweb.faa.gov/Regulatory_and_Guidance_Library/rgPolicy.nsf/97a612e22b32398d85256b7500496a9a/cf51a956f07b0c208625727c006745ca/$FILE/PS-ACE100-2005-50001%20final.pdf)
15. Akira Fukumoto, Toshiba Corporation, "Transition and Current Status of NPP C&I System of BWRs in Japan." http://entrac.iaea.org/I-and-C/TWG_NPP_CI_2005_05/Presentations%5CJapan-Fukumoto.pdf
16. Suresh Srinivasan, Aman Gayasen, N. Vijaykrishnan, M. Kandemir, Y. Xie, and M.J. Irwin, "Improving Soft-Error Tolerance of FPGA Configuration Bits," Department of Computer Science and Engineering, Pennsylvania State University, 2004.
<http://www.cse.psu.edu/~degalaha/paper/jccad.pdf>
17. H. Helstrup, V. Lindenstruth, S. Martens, L. Musa, J. Nystrand, E. Olsen, D. Rohrich, K. Roed, B. Skaali, M. Stockmeier, H. Tilsner, K. Ullaland, and J. Wikne, "Irradiation Tests of the ALTERA SRAM Based FPGA and Fault Tolerant Design Concepts."
<http://lhc-electronics-workshop.web.cern.ch/LHC-electronics-workshop/2003/sessionsPDF/Eleccal/ROED.PDF>
18. Ghazanfar Asadi and Mehdi B. Tahoori, "An Analytical Approach for Soft Error Rate Estimation in Digital Circuits." http://www.ece.neu.edu/groups/trg/index_files/papers/ser/iscas05.pdf
19. Ghazanfar Asadi and Mehdi B. Tahoori, "Soft Error Rate Estimation and Mitigation for SRAM-Based FPGAs," 2005. http://www.ece.neu.edu/groups/trg/index_files/papers/serfpga/fpga05final.pdf
20. Goddard Space Flight Center NASA Advisory, "Application Note on Grounding the MODE Pin in Actel Field Programmable Gate Arrays," November 2002.
http://klabs.org/richcontent/User_Notes/Actel/na-gsfc-2003-02.pdf
21. Goddard Space Flight Center NASA Advisory, "TRST* and the IEEE JTAG 1149.1 Interface," February 2004. <http://klabs.org/richcontent/maplug/notices/na-gsfc-2004-04.pdf>
22. NASA Independent Verification and Validation Facility, *NASA IV&V 2005*.
<http://www.nasa.gov/centers/ivv/about/policyplans.html>
23. Ramin Roosta, "A Comparison of Radiation-Hard and Radiation-Tolerant FPGAs for Space Applications," NASA Electronic Parts and Packaging Program, December 2004.
<http://nepp.nasa.gov/docuploads/3C8F70A3-2452-4336-B70CDF1C1B08F805/JPL%20Rad-Tolerant%20FPGAs%20for%20Space%20Applications.pdf>
24. Commission on Engineering and Technical Systems, "Dedication of Commercial Off-the-Shelf Hardware and Software," Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues (1997).
<http://www.nap.edu/openbook.php?isbn=0309057329&page=71>
25. Actel, "Overview of iRoC Technologies' Report 'Radiation Results of the SER Test of Actel, Xilinx and Altera FPGA Instances,' " 2004.
<http://www.actel.com/documents/OverviewRadResultsIROC.pdf>

26. iRoC Technologies, "Radiation Results of the SER Test of Actel, Xilinx and Altera FPGA Instances," October 2004. <http://www.actel.com/documents/RadResultsIROCreport.pdf>
27. iRoC Technologies, "White Paper on VDSM IC Logic and Memory Signal Integrity and Soft Errors," January 2002.
28. Jerker Hammarberg and Simin Nadjm-Tehrani, "Formal Verification of Fault Tolerance in Safety-Critical Reconfigurable Modules," August 2004. <http://www.ida.liu.se/~rtslab/publications/2005/STTT0152.pdf>
29. Gaisler Research, "Lessons Learned from FPGA Developments," Technical Report, September 2002. http://www.gaisler.com/doc/fpga_001_01-0-2.pdf
30. John Lach, Scott Bingham, Carl Elks, Travis Lenhart, Thuy Nguyen, and Patrick Salaun, "Accessible Formal Verification for Safety-Critical FPGA Design," MAPLD 2005/241. http://klabs.org/mapld05/presento/241_lach_p.ppt
31. Carl Elks and Barry Johnson, "PLD-Based Safety Critical Systems: An Introduction and Survey," Final Technical and Scientific Report, University of Virginia, February 2004.
32. John Lach, Scott Bingham, Travis Lenhart, Thuy Nguyen, and Patrick Salaun, "RAFFIA—Reliable ASIC/FPGA-Based Solutions for I&C Applications," *American Nuclear society International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human Machine Interface Technology*, pp. 1032–37 (2006).
33. John Lach, "Integrated Circuits and Systems Design Methodologies," Research Program Overview. http://www.ee.virginia.edu/graduate/Lach_research_overview.pdf
34. University of Virginia, "Embedded Digital System Reliability and Safety Analyses," NUREG/GR-0020, February 2001.
35. Dr. Wagih Abdel-Kader, "Radiation Induced Effects in Semiconductor Devices," Parts 1–3, South Carolina State University.
36. International Atomic Energy Agency, "Harmonization of the Licensing Process for Digital Instrumentation and Control Systems in Nuclear Power Plants," December 2002. http://www-pub.iaea.org/MTCD/publications/PDF/te_1327_web.pdf
37. L. Harrison and B. Landell, "Design, Test, and Certification Issues for Complex Integrated Circuits," DOT/FAA/AR-95/31, August 1996. <http://klabs.org/richcontent/verification/faa/ar-95-31-ceh.doc>
38. European Space Research and Technology Center, "VHDL Modelling Guidelines," Approved by R. Creasey and R. Coirault, ASIC/001, Issue 1, September 1994. <http://www.eda.org/rassp/vhdl/guidelines/ModelGuide.pdf>
39. European Space Research and Technology Center, "ASIC Design and Manufacturing Requirements," Prepared by S. Habinc and P. Sinander, WDN/PS/700, Issue 2, October 1994. <http://www.eda.org/rassp/vhdl/guidelines/DesignReq.pdf>
40. Institute of Electrical and Electronics Engineers, Inc., IEEE STD 7-4.3.2–2003, "IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations."
41. AP1000 Design Control Document Tier 2, Chapter 7, "Instrumentation and Controls." <http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr1793/chapter7.pdf>

42. Branch Technical Position HICB-14, "Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems," Rev. 5, U.S. Nuclear Regulatory Commission, March 2007.
43. Jonathan Tillack, Lori Kaufman, Karthik Kannan, and Barry Johnson, "Design Standards and Their Application to the Digital Retrofit of Existing Analog Safety-Critical Systems," 2000 Proceedings Annual Reliability and Maintainability Symposium.
<http://ieeexplore.ieee.org/iel5/6628/17683/00816332.pdf>
44. Ray DiSandro and Ray Torok, "Generic Qualification of Digital Components," EPRI, Palo Alto, CA.
45. EPRI, *Generic Qualification of Digital Components for Nuclear Applications: A Low-Cost Approach to Qualifying New Digital I&C Components for Nuclear Plant Use, Especially in Safety Systems*, EPRI-1006958, EPRI, Palo Alto, CA, April 2002.
46. Matthew Chiramal, "Application of Commercial-Grade Digital Equipment in Nuclear Power Plant Safety Systems." <http://ieeexplore.ieee.org/iel5/7654/20915/00969772.pdf>
47. International Atomic Energy Agency, "Managing modernization of nuclear power plant instrumentation and control systems," IAEA-TECDOC-1389, February 2004.
http://www-pub.iaea.org/MTCD/publications/PDF/te_1389_web.pdf
48. Matthew Chiramal, "Regulatory Framework for Digital Instrumentation and Control Systems in Nuclear Power Plants," MIT Workshop on Safety-Critical Software and Safety, February 2001. <http://sunnyday.mit.edu/safety-club/chiramal.rtf>
49. Lawrence Livermore National Laboratory, "Review Templates for Computer-Based Reactor Protection Systems," NUREG/CR-6680, UCRL-ID-139344, August 2000.
http://adamswebsearch2.nrc.gov/idmws/doccontent.dll?library=PU_ADAMS^PBNTAD01&ID=003960624
50. Hluboka nad Vltavou, "CNRA/CSNI Workshop on Licensing and Operating Experience of Computer-Based I&C Systems," NEA/CSNI/R(2002)4, May 2002.
<http://www.nea.fr/html/nsd/docs/2002/csni-r2002-4.pdf>
51. Nihal Kececi and Mohammad Modarres, "Software Development Life Cycle Model to Ensure Software Quality." <http://www.cse.ohio-state.edu/~kirschen/Research/psam-paper2.PDF>
52. M. Hecht and H. Hecht, "Digital Systems Software Requirements Guidelines."
53. "Safety Evaluation by the Office of Nuclear Reactor Regulation Topical Reports 7286-545 and 7286-546," Project #709, December 2001.
http://adamswebsearch2.nrc.gov/idmws/doccontent.dll?library=PU_ADAMS^PBNTAD01&ID=004042634
54. International Standard, IEC 61508-2, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems—Part 2: Requirements for Electrical/Electronic/Programmable Electronic Safety-Related Systems."
55. Jaynarayan Lala and Richard Harper, "Architectural Principles for Safety-Critical Real-Time Applications," Proceeding of IEEE, Vol.82, No. 1, January 1994.
<http://ieeexplore.ieee.org/iel1/5/6554/00259424.pdf?arnumber=259424>
56. Larry Harzstark, "FPGA (Field Programmable Gate Array) Space Qualification Presentation," December 2005. <http://www.aero.org/conferences/mrqw/2005-papers/VII-1%20Harzstark.ppt>

57. Douglas Sheldon, "Integrated Qualification Strategies for FPGAs," December 2005.
<http://www.aero.org/conferences/mrqw/2005-papers/VII-2%20Sheldon.ppt>
58. Michael Wirthlin, Brian Pratt, and Keith Morgan, "The Challenges and Benefits of Partial Mitigation of FPGAs." <http://www.aero.org/conferences/mrqw/2005-papers/VII-3%20Wirthlin.ppt>
59. iRoC Technologies, "Answers to Frequently Asked Questions Regarding iRoC's Testing Methodology for SRAM-Based FPGAs," June 2004.
60. Michael Wirthlin, Eric Johnson, Nathan Rollins, Michael Caffrey, and Paul Graham, "The Reliability of FPGA Circuit Designs in the Presence of Radiation Induced Configuration Upsets," Proceeding of IEEE Symposium on Field-Programmable Custom Computing Machines, 2003.
61. EPRI Working Group on Use of Commercial Digital Equipment in Nuclear Safety Applications with MPR Associates, "Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications," TR-106439, October 1996.
62. Actel White Paper, "Reliability Considerations for Automotive FPGAs," September 2003.
<http://www.actel.com/documents/AutoWP.pdf>
63. Science Applications International Corporation, "Independent Verification and Validation (IV&V) First Year Summary Report for the Programmable Logic Devices Research," SAIC-PLD-0002, ISTO-06-98-133, September 2005.
64. United States Nuclear Regulatory Commission, "Briefing on Digital Instrumentation and Control," November 2006.
65. L. Sterpone and M. Violante, "A Design Flow for Protecting FPGA-Based Systems Against Single Event Upsets," Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005.
<http://ieeexplore.ieee.org/iel5/10366/32969/01544543.pdf?arnumber=1544543>
66. James Cercone, Mike Beims, Richard Grigg, and Jack Horner, "A Preliminary Practitioner's Guide to Defect Detection in VHDL Based Designs," NASA, IV&V Facility, West Virginia.
67. Gaisler Research, "Suitability of reprogrammable FPGAs in space applications," FPGA-002-01, Version 0.4, September 2002. http://www.gaisler.com/doc/fpga_002_01-0-4.pdf
68. Adrian Hilton, Gemma Townson, and Jon Hall, "FPGAs in Critical Hardware/Software Systems," Technical Report No: 2003/01, 2003.
69. Adrian Hilton and Jon Hall, "High-Integrity Interfacing to Programmable Logic with Ada," Ada-Europe International Conference, June 2004.
http://www.praxis-his.com/sparkada/pdfs/hilton_hall_adaeurope.pdf
70. Moore Industries, "IEC 61508 Fact Sheet."
http://www.mooreindustries.com/products/data_sheets/iec_61508.pdf
71. "Functional Safety and IEC 61508—A Basic Guide," November 2002.
<http://www.nepss.org/PSES/IEC61508basicguide.pdf>
72. NASA, "Software Assurance Standard," NASA-STD-8739.8, July 2004.
<http://www.hq.nasa.gov/office/codeq/doctree/87398.pdf>

73. Rod Barto, "Suggestions for FPGA Design Presentation."
http://www.klabs.org/DEI/References/design_guidelines/design_checklists/fpga_design_presentation.ppt
74. Carl Elks, John Lach, and Barry Johnson, "PLD-Based Safety Critical Systems: Task 1 and 2—Assessment, Practices and Design," University of Virginia, November 2006.
75. Carl Elks, Yang Yang Yu, and Barry Johnson, "Quantitative Safety Assessment for Safety-Critical I&C Systems." University of Virginia.
76. Scott Bingham and John Lach, "Accessible Formal Verification for Safety-Critical Hardware Design: The Library Approach RAFFIA 1," *Annual Reliability and Maintainability Symposium*, pp. 29–32 (2006).
77. John Lach, "Dependable Hardware Systems Research Program Overview."
http://www.ee.virginia.edu/graduate/Lach_research_overview.pdf
78. John Lach, Scott Bingham, Carl Elks, Travis Lenhart, Thuy Nguyen, and Patrick Salaun, "Accessible Formal Verification for Safety-Critical FPGA Design," MAPLD 2005/241.
79. Science Applications International Corporation, "Independent Verification and Validation (IV&V) Preliminary Practitioner's Guide to Defect Detection in VHDL Based FPGA Designs," SAIC-PLD-0003, ISTO-06-98-135, September 2005.
80. Institute of Electrical and Electronics Engineers, Inc., IEEE STD 1012–2004, "IEEE Standard for Software Verification and Validation," sponsored by Software Engineering Standards Committee, June 2005.
81. Institute of Electrical and Electronics Engineers, Inc., IEEE STD 1076–2002, "IEEE Standard VHDL Language Reference Manual," sponsored by Design Automation Standards Committee, May 2002.
82. Branch Technical Position HICB-18, "Guidance on the Use of Programmable Logic Controllers in Digital Computer-Based Instrumentation and Control Systems," Rev. 5, U.S. Nuclear Regulatory Commission, March 2007.
83. Interim Defense Standard 00-54, "Requirements for Safety Related Electronic Hardware in Defense Equipment," Issue 1, March 1999. <http://www.dstan.mod.uk/data/00/054/01000100.pdf>

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

NUREG/CR-7006

2. TITLE AND SUBTITLE

Review Guidelines for Field-Programmable Gate Arrays in Nuclear Power Plant Safety Systems

3. DATE REPORT PUBLISHED

MONTH	YEAR
February	2010

4. FIN OR GRANT NUMBER

N6351

5. AUTHOR(S)

M. Bobrek (ORNL), D. Bouldin (University of Tennessee), D. E. Holcomb (ORNL), S. M. Killough (ORNL), S. F. Simth (ORNL), C. Ward (ORNL), R. T. Wood (ORNL)

6. TYPE OF REPORT

Technical

7. PERIOD COVERED (Inclusive Dates)

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)

Oak Ridge National Laboratory	University of Tennessee
P.O. Box 2008	419 Ferris Hall, 1508 Middle Drive
Oak Ridge, TN 37831-6010	Knoxville, TN 37996-5483

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above"; if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)

Division of Engineering
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

M. E. Waterman, NRC Project Manager

11. ABSTRACT (200 words or less)

This report is a compilation of field-programmable gate array (FPGA) design practices that can be used by NRC staff as guidance for reviewing FPGA-based safety systems in nuclear power plants. It can also serve as a basis for developing specific activities supporting the licensing process such as review procedures and acceptance criteria. This document follows on the investigation of existing regulatory documents and standards related to design and review of safety-related FPGA systems. Since the existing regulatory documents are not specific about FPGA design practices, this document also serves as the complement to standards that address general issues related to digital safety systems in nuclear power plants.

FPGA design practices are classified into three major groups: FPGA hardware design practices, FPGA design entry methods, and FPGA design methodologies. Within these major groups, design practices are further classified according to four top-level attributes: reliability, robustness, traceability, and maintainability.

The report focuses on listing and describing FPGA design practices that are potentially unsafe as well as suggesting practices that are acceptable for safety-critical designs. Additionally, the report outlines a design life cycle that could be used by designers and reviewers for FPGA-based safety systems.

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)

FPGA, field-programmable gate array, FPGA guidance

13. AVAILABILITY STATEMENT

unlimited

14. SECURITY CLASSIFICATION

(This Page)

unclassified

(This Report)

unclassified

15. NUMBER OF PAGES

16. PRICE



Federal Recycling Program



UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, DC 20555-0001

OFFICIAL BUSINESS