METRIC/SI

NASA TECHNICAL HANDBOOK

National Aeronautics and Space Administration

NASA-HDBK-4008
w/CHANGE 1:
REVALIDATED w/
ADMINISTRATIVE/
EDITORIAL CHANGES
2016-01-19

Approved: 2013-12-02

PROGRAMMABLE LOGIC DEVICES (PLD) HANDBOOK

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

DOCUMENT HISTORY LOG

Status	Document Revision	Change Number	Approval Date	Description
Baseline			2013-12-02	Initial Release
		1	2016-01-19	Revalidated w/Administrative/Editorial Changes—This NASA Technical Handbook was reviewed and no technical changes resulted. Administrative changes to conform to the current template and editorial corrections were made.

FOREWORD

This NASA Technical Standard is published by the National Aeronautics and Space Administration (NASA) to provide uniform engineering and technical requirements for processes, procedures, practices, and methods that have been endorsed as standard for NASA programs and projects, including requirements for selection, application, and design criteria of an item.

This NASA Technical Standard is approved for use by NASA Headquarters and NASA Centers and Facilities and may be cited in contract, program, and other Agency documents as a technical requirement. It may also apply to the Jet Propulsion Laboratory and other contractors only to the extent specified or referenced in applicable contracts.

This NASA Technical Handbook establishes programmable logic design engineering guidance. It originated from multiple requests for additional guidance, rationale, resources, references and lessons learned for acquiring, managing, developing, assuring, and maintaining NASA systems.

Requests for information should be submitted via “Feedback” at https://standards.nasa.gov. Requests for changes to this NASA Technical Handbook should be submitted via MSFC Form 4657, Change Request for a NASA Engineering Standard.

Original Signed By **12/2/2013**

Ralph R. Roe, Jr. Approval Date

NASA Chief Engineer

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

SECTION **PAGE**

TABLE OF CONTENTS (Continued)

LIST OF APPENDICES

LIST OF FIGURES

LIST OF TABLES

PROGRAMMABLE LOGIC DEVICES (PLDs) HANDBOOK

1. SCOPE

This NASA Technical Handbook outlines a life cycle as a guideline for planning, designing, verifying and maintaining programmable logic devices (PLDs). Additionally, best practices are recommended for different PLD development phases. This NASA Technical Handbook provides guidance to perform project activities. It covers all aspects of the design cycle from initial planning through release and maintenance. The specific types of PLDs this NASA Technical Handbook addresses are as follows:

- Field-programmable gate array (FPGA).
- Complex programmable logic device (CPLD).

1.1 Purpose

The purpose of this NASA Technical Handbook is to establish PLD design engineering guidance.

The trend toward the increased use of PLDs in aerospace systems requires increased expertise in the design, development, and verification of these systems. Hardware designers are now expected to implement PLD designs that are as complex as traditional microprocessor-based systems that were designed in some cases by large teams of engineers.

The development of successful PLDs requires a coordinated effort. This NASA Technical Handbook contains guidelines that provide for a consistent approach based on best practices for the development of PLDs for flight and ground support systems across NASA Centers. The advancing technology in PLDs has allowed for the implementation of more complex designs in single devices. Many of these devices designed within NASA systems perform critical operations. The guidelines in this NASA Technical Handbook serve to increase confidence in the quality of PLD designs.

1.2 Applicability

This NASA Technical Handbook provides engineering guidance applicable to programmable logic design. It serves as a primer for sound engineering design practice and can be used in its entirety, or in portions thereof, in conjunction with other available design resources.

This NASA Technical Handbook is approved for use by NASA Headquarters and NASA Centers and Facilities. It may also apply to the Jet Propulsion Laboratory or other contractors only to the extent specified or referenced in their contracts.

This NASA Technical Handbook, or portions thereof, may be referenced in contract, program, and other Agency documents for guidance. When it contains procedural or process requirements, they may be cited in contract, program, and other Agency documents.

2. APPLICABLE DOCUMENTS

2.1 General

The documents listed in this section are applicable to the guidance in this NASA Technical Handbook.

2.1.1 The latest issuances of cited documents shall apply unless specific versions are designated.

2.1.2 Non-use of specifically designated versions shall be approved by the responsible Technical Authority.

The applicable documents are accessible at https://standards.nasa.gov or may be obtained directly from the Standards Developing Body or other document distributors. Additional documents and design resources are available at https://nen.nasa.gov/web/pld.

2.2 Government Documents

National Aeronautics and Space Administration (NASA)

NASA-HDBK-8739.23	NASA Complex Electronics Handbook for Assurance Professional
NPR 7123.1	NASA Systems Engineering Processes and Requirements
NPR 8715.3C	NASA General Safety Program Requirements

2.3 Non-Government Documents

None.

2.4 Order of Precedence

This NASA Technical Handbook provides guidance for planning, designing, verifying and maintaining programmable logic devices but does not supersede or waive established Agency requirements/guidance found in other documentation.

3. ACRONYMS AND DEFINITIONS

3.1 Acronyms and Abbreviations

A/D	analog to digital
AID	altered item drawing
ASIC	application specific integrated circuit
CDC	clock domain crossing
CDR	critical design review
CLK	clock
CLKA/B	clock A or B
CLKBUF	clock buffer
CM	configuration management
CMOS	complementary metal oxide semiconductor
COTS	commercial off the shelf
CPLD	complex programmable logic device
CPU	central processing unit
CRC	cyclic redundancy check
CVS	Concurrent Versions System
CxP	Constellation Program
DC	direct current
DDP	detailed design phase
DDR	double data rate
DFF	d-flip-flop
DLL	delay-locked loop
DOORS	Dynamic Object-Oriented Requirements System
EDAC	error detection and correction
EEPROM	electrically erasable programmable read-only memory
EIDP	end item data package
EMI	electromagnetic interference
ESD	electrostatic discharge
ETU	engineering test unit
FET	field effect transistor
FIFO	first in, first out
FPGA	field-programmable gate array
FSM	finite state machine
GIDEP	Government Industry Data Exchange Program
GPU	graphics processing unit
HALE	high altitude long endurance
HCLK	hardwired CLK driver
HCLKBUF	buffer for HCLK
HDL	hardware description language
HSI	hardware/software integration
I/O	input/output
IC	integrated circuit
IP	intellectual property

ITAR	International Traffic in Arms Regulations
JTAG	Joint Test Action Group
LSTTL	Low-power Schottky Transistor Transistor Logic
LVDS	low voltage differential signaling
ms	millisecond
N/C	no connect
NASA	National Aeronautics and Space Administration
ns	nanosecond
OTS	off-the-shelf
PCB	printed circuit board
PCI	peripheral component interconnect
PDP	preliminary design phase
PDR	preliminary design review
PLD	programmable logic device
PLL	phase-locked loop
PnR	place and route
POC	point of contact
POR	power-on reset
PROM	programmable read-only memory
PRP	planning and requirements phase
QA	quality assurance
RDD	revision description document
ROM	read-only memory
RTL	register-transfer level
SDF	Standard Delay Format
SERDES	serializer-deserializer
SET	single event transient
SEU	single event upset
SMA	Safety and Mission Assurance
SOW	statement of work
SRAM	static random access memory
SSO	simultaneously switching output
STA	static timing analysis
SVN	Subversion
TBD	to be determined
TID	total ionizing dose
TMR	triple modular redundancy
TRST	test reset (JTAG test reset signal)
TTL	transistor-transistor logic
UART	universal asynchronous receiver/transmitter
VDD	Version Description Document
VHDL	very high speed integrated circuit (VHSIC) hardware description language (HDL)
VHSIC	very high speed integrated circuit
VIO	voltage input/output
VP	verification phase

3.2 Definitions

Asynchronous: Inputs and/or logic that changes independently of clock changes.

Board: Describes the electronic assembly that contains the PLD. *Note: The board can be purchased or customized for the design.*

Brownout: A condition where the voltage of the power source drops below the nominal operating range because of an excessive current loading.

CLKBUF: A clock buffer type in a MicroSemi FPGA.

Clock Domain: A clock domain is a block of circuitry that operates at a single clock frequency that may differ from the frequency of other blocks on the same chip. A clock domain crossing occurs whenever data is transferred from a flop driven by one clock to a flop driven by another clock.

Cold spare/ing: A spare redundant unit that remains unpowered when inactive. *Note: A cold sparing device is designed to allow powered input signals to be applied to the unpowered device without damage to the device.*

Crosstalk: Any phenomenon by which a signal transmitted on one circuit or channel of a transmission system creates an undesired effect in another circuit or channel. *Note: Undesired capacitive, inductive, or conductive coupling from one circuit, part of a circuit, or channel to another is what usually causes crosstalk.*

Delivering Organization: The organization that is responsible for developing and delivering the PLD design.

Fault Injection: A technique for improving the coverage of a test by introducing faults to test code paths.

Gray Code: An encoding of numbers so that adjacent numbers have a single digit differing by 1.

Hamming Code: A linear code devised for detecting and correcting errors in digital data.

HCLK: A clock buffer type in a MicroSemi FPGA.

HCLKBUF: A clock buffer type in a MicroSemi FPGA.

Hystereis: A phenomenon wherein two (or more) physical quantities bear a relationship that depends on prior history. More specifically, the response Y takes on different values for an increasing input X than for a decreasing X.

Krad (Si): 1000 rad (Si), see also rad (Si).

Metastability: The ability of a digital electronic system to persist for an unbounded time in an unstable equilibrium or metastable state. In metastable states, the circuit may be unable to settle into a stable '0' or '1' logic level within the time required for proper circuit operation. As a result, the circuit can act in unpredictable ways, and may lead to a system failure.

Primitives: The physical hardware components that exist in the PLD. These primitives will be connected together in a specific way on your PLD in order to fulfill your design as specified. For FPGAs, the most basic primitives are flip-flops and lookup tables.

Rad (Si): The quantity of any type of ionizing radiation that will impart 100 ergs of energy per gram of silicon.

$R_{term:}$ Value of the termination resistor.

Synthesis: A process that starts from a high level of logic abstraction (typically Verilog or Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL)) and automatically creates a lower level of logic abstraction using a library containing primitives.

Test Bench: To simulate a design, the designer needs both the design under test or unit under test and the stimulus provided by the test bench. *Note: A test bench is a Hardware Description Language (HDL) code that allows the designer to provide a documented, repeatable set of stimuli that is portable across different simulators. A test bench can be as simple as a file with clock and input data or a more complicated file that includes error checking, file input and output, and conditional testing. Verilog designers sometimes refer to a Verilog test fixture. "Test bench" and "test fixture" are used synonymously throughout this NASA Technical Handbook.*

Test Procedure: A document that delineates the test steps required to verify full compliance of the PLD implementation with the requirements.

Tool: Describes the PLD software design package used to generate the PLD design.

T_{PD}: Refers to the propagation delay time of a device.

$T_{REM:}$ Refers to the synchronous clear preset or removal time.

V_{CC}: Refers to the power supply pin for a bipolar junction transistor integrated circuit (IC).

V_{DD}: Refers to the power supply pin for Complementary Metal Oxide Semiconductor (CMOS) ICs.

$V_{IH:}$ Refers to the voltage threshold for input high.

Verification: The determination that the end product meets all of the specified

requirements.

Z_{driver}: Output impedance of the driving circuit.

Z_{trace}: Characteristic impedance of a circuit board trace.

4. LIFE CYCLE AND DEVICE DEVELOPMENT

Life cycle development for the PLD development process consists of a series of phases with entry and exit criteria. This section describes this series of development phases and the products developed in each phase. Depending on the criticality and risk to the project, two or more phases can be combined into a single phase. Additionally, device development does not typically consist of a single pass through each phase. The phases iterate based on the needs of the design.

This section provides a typical template for the development life cycle. The remainder of this NASA Technical Handbook uses this development cycle for describing what products to complete in each section.

4.1 Life-Cycle Definitions

The sections that follow describe the PLD life-cycle phases. See also figure 1, PLD Life-Cycle Development Phases.

4.1.1 Planning and Requirements Phase

Development planning is essential for early identification of all the development needs and risks. It also provides the PLD design team with a clear view of the near- and long-term objectives and goals. The planning and requirements phase (PRP) occurs when requirements have been allocated to the board where the PLD resides. High-level system requirements flow down and the design team uses them to generate the PLD requirements.

The planning phase includes documenting a development plan that all developers follow. The plan documents the systematic approach used to manage, design, develop, verify, document, and review all PLDs delivered. The design team can customize the level of detail to the complexity of the design and project. Appendix C provides more details on customization.

During development of the PLD, the design team reviews the requirements and/or the development plan and updates it as required. Section 5 describes this phase in greater detail.

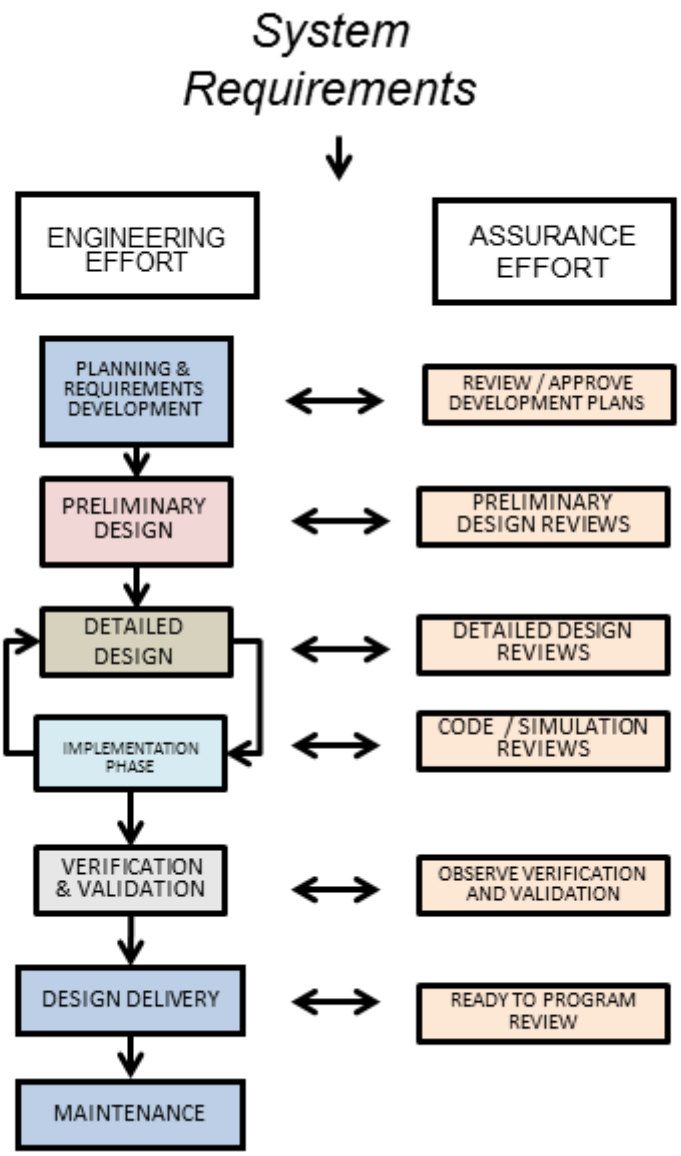

Figure 1—PLD Life-Cycle Development Phases

4.1.2 Preliminary Design Phase

During the preliminary design phase (PDP), the design team generates a top-level design based on the device requirements. This top-level design includes block diagrams and data flow diagrams that serve as the design architecture. Design trade studies are completed during this phase. This phase is complete when the stakeholders identified in the PRP approve the preliminary design. Section 6 describes this phase in greater detail.

4.1.3 Detailed Design Phase

The detailed design phase (DDP) begins after approval of the preliminary design. During the DDP, the preliminary design is used to generate a more detailed design based on the requirements and architecture. This phase iterates as needed with the design implementation phase to further modify and/or expand until it meets all requirements. This phase is complete when the stakeholders identified in the PRP approve the documented detailed design. Section 7 describes this phase in greater detail.

4.1.4 Design Implementation Phase

During the design implementation phase, design capture (such as coding in a hardware description language (HDL)) is completed. The design team performs design simulation, synthesis, place and route (PnR), and timing analyses. The design may be tested on a development board and/or hardware resembling the final flight system. This phase iterates with the DDP as needed. Section 8 describes this phase in greater detail.

4.1.5 Verification Phase

Verification determines that the end product meets all of the specified requirements whereas validation determines the requirements are correct. Validation is beyond the scope of this document since it often requires a higher level of integration. Verification activities include reviews, tests, and simulations in accordance with the plan developed in the PRP. Formal acceptance of the final design occurs at the conclusion of the verification phase (VP). Section 9 describes this phase in greater detail.

4.1.6 Delivery Phase

After successful verification, the design team releases the completed design for integration into the next higher-level assembly (e.g., board, box, etc.). Section 10 describes this phase in greater detail.

4.1.7 Maintenance Phase

The development team continues to provide troubleshooting support for issues deemed unsolvable by the user of the design, and investigates and tracks potential design anomalies to completion. Maintenance support covers any issues reported after the delivery. Section 11 describes this phase in greater detail.

5. PLANNING AND REQUIREMENTS PHASE

Numerous projects do project planning as part of the system planning. All items that are identified in the PRP can be done separately or as part of the larger planning.

The output of the PRP for a PLD can include documents, such as:

a. A development plan.

b. A requirements document.

c. A verification plan.

The design team applies customization to each of the documents depending on the project. Appendix C shows a proposed PLD classification to aid in determining an appropriate customization.

All planning and requirements documentation and development will need to be coordinated with Project Management, System Engineering, and the Chief Engineer. The use of NPR 7123.1, NASA Systems Engineering Processes and Requirements, continues to serve as the governing requirements for system development.

5.1 Roles and Responsibilities

The scope of work for each role on the development team varies with the size and complexity of the development. A development team member may perform multiple roles for a simple development. Conversely, multiple individuals might be required to share the responsibilities for a single role for a large development. Work with the line management to set the initial roles and responsibilities for the development effort. This can then be worked into the efforts defined by the project.

A PLD development effort typically includes the following roles and responsibilities:

a. PLD Team Lead.

(1) Co-develop the development plan with the technical development lead.

(2) Manage the development plan.

(3) Manage budget and schedule.

(4) Collect development metrics and provide status to project management.

(5) Manage procurement contracts.

b. PLD Technical Development Lead.

(1) Co-develop the development plan with the PLD team lead.

(2) Provide single point of contact (POC) for technical interactions with external stakeholders.

(3) Oversee the technical aspect of the overall development and the end delivery of the design (including design analyses/trade studies, identifying intellectual property (IP) core usage, etc.).

(4) Develop, negotiate, and maintain the PLD design requirements.

(5) Select design and configuration management (CM) tools when necessary.

(6) Develop/select the revision control process and associated tools(s).

(7) Owner of the PLD design specification.

(8) Manage design tools (maintenance support, license files, tool availability).

(9) Ensure adherence to the required design process.

(10) Responsible for the overall quality and delivery of the end product.

(11) Assign responsibilities to members of the development team.

c. PLD Configuration Lead/Librarian.

(1) Manage revision control.

(2) Maintain the design repository/library.

(3) Build and release (official and unofficial) design into the test venue and for delivery.

(4) Assemble the end item data package (EIDP) of the official releases.

d. Verification Lead.

(1) Responsible for the development of the PLD verification plan, including defining test (hardware and software) requirements, verification items for simulation and pass/fail criteria for each verification item.

(2) Support the test lead on the PLD hardware verification plan development and hardware testing.

(3) Oversee and ensure the completion of the verification activities.

(4) Archive the verification results.

(5) Generate test reports.

(6) Contribute to development of functional coverage checks.

e. Design Engineer.

(1) Design the assigned design modules.

(2) Generate pertinent documentation for the assigned design modules.

(3) Perform incremental testing.

(4) Contribute to the development of functional coverage checks.

f. Possible Customer/Stakeholders

(1) Board Design Lead: Owner of the board where the PLD resides. The board design lead works with the technical development lead on defining the PLD requirements, ensures compatible interface between the board and PLD throughout the design phases, and coordinates the overall verification effort for the board.

(2) Test Lead: Responsible for leading the effort on completing the PLD verification at the board level and providing a single POC for reporting the testing status to the PLD development team. The test lead is the source of the testability requirements (software, test points, fault injection, etc.).

(3) Test Equipment Development Lead: Responsible for developing and negotiating the functional requirements for the test equipment with the technical development lead.

(4) Safety and Mission Assurance (SMA) personnel: Serves as an independent assessor. NASA-HDBK-8739.23, NASA Complex Electronics Handbook for Assurance Professional, documents the SMA roles and responsibilities, and they are NOT duplicated in this document. In addition for safety-critical projects, SMA personnel will also use NPR 8715.3C, NASA General Safety Program Requirements, for applicability to PLDs.

(5) System Safety Engineer.

(6) Users: People who operate the PLD in its intended applications. Depending on the design, users may include:

A. Software developers.

B. Subsystem engineers.

C. System engineers.

D. Test engineers.

A possible role-based organization chart of the development team is shown in figure 2, Example Development Team Role-Based Organization Chart, follows.

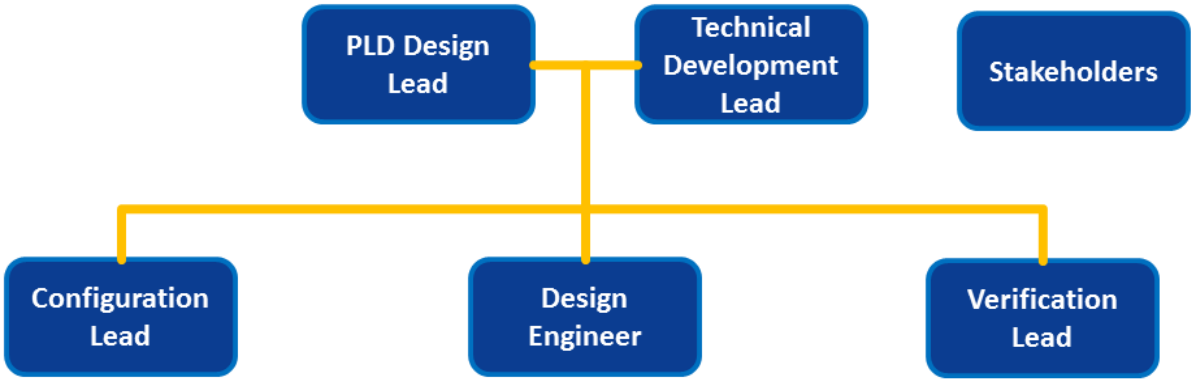

Figure 2—Example Development Team Role-Based Organization Chart

In addition to the development team, the corresponding delivery organization also plays an integral role in the development process. It is the delivery organization's responsibility to:

a. Ensure the quality of its products through defining and enforcing local design and monitoring processes.

b. Support the assembling of review boards.

The line organization can delegate this authority to the PLD design lead.

5.2 The Development Plan

The PLD design process includes documenting a development plan(s) for the developers to follow. This plan(s) documents the systematic approach used to manage, design, verify, document, and review all PLDs delivered. Development planning involves defining several items that have long-term implications on the project. These can include the following:

a. Development tools.

b. Metrics measurement.

c. Risk management.

d. System safety considerations

e. Configuration management.

f. Training.

g. Schedule.

h. Bug tracking.

i. Project deliverables.

j. Review planning.

k. Roles and Responsibilities.

The design team can customize the development plan depending on the complexity and project requirements. All items listed above do not necessarily require a separate detailed plan. The team notes each item, the related activities, and issues that affect the cost and schedule. Many times these plans can be a part of an overall project development plan**.**

5.2.1 Development Tools

The technical development lead determines the development tools and version to be used. This includes at least the following:

a. Determine a specific list of tools with the following information:

(1) Manufacturer.

(2) Tool.

(3) Version.

(4) Function.

(5) Repository for tool archival.

b. Determine the rationale for tool upgrades. It is a best practice not to upgrade the tools without sufficient justification (e.g., re-characterization of the targeted device and/or a bug fix that would affect the functionality of the programmed PLD). When an issue warrants a tool update, revise the development plan and notify the development team.

c. Determine how tool alerts, such as Government Industry Data Exchange Program (GIDEPs) or manufacturer notices, will be communicated to the design team. This could come from other discipline areas, such as assurance or parts engineering. This is also applicable to the PLD itself, not only the programming tool.

d. Determine a data back-up/recovery plan.

5.2.2 Metrics Measurement

Advances in today’s technology have allowed PLD capacity to become increasingly dense. Therefore, PLD designs have also become proportionally more complex. There is a need for projects and programs to be able to measure the progression of the PLD design toward the final end product. Metrics measurement allows project management to track PLD design through design, implementation, verification, and integration.

The objectives of the measurement process are as follows:

a. To provide visibility into the design progress.

b. To complete the development per the agreed-upon schedule and budget.

c. To identify programmatic risks.

d. Monitor accepted and rejected design change requests.

The PLD team lead would be responsible for collecting project metrics, storing them in the project folder, analyzing the measurement data and reporting them periodically. The report may also contain a summary statement of the metrics pointing out any measurements trending towards a limit that may indicate a pending problem. Project metrics for each PLD can include:

a. Coverage metrics (code and functional).

b. Design effort (man-hours) – planned versus actual.

c. Budget – planned versus actual.

d. Requirements stability—number of requirements, number of requirements changes, and number of items to be determined (TBDs).

e. Device resource utilization in percentage of capacity per phase (e.g., C-cells, S-cells, PLD inputs/outputs (I/Os), static random access memory (SRAM) memory, timing, etc.)—actual.

f. Requirements verification progress (procedures or test benches developed and verified).

g. Number of problem reports and review item discrepancies—open, closed, and withdrawn.

h. Number of reviews—planned versus actual.

i. Risks and mitigations.

5.2.3 Risk Management

Depending on the size and project requirements of the PLD development, the development team will want to identify and manage risks. The technical development lead identifies and manages the risks. Pay special attention to areas such as the following:

a. Device resource utilization.

b. Timing.

c. Power consumption.

d. External interfaces.

The project’s risk management team owns these items; the PLD team does not hold them.

5.2.4 System Safety Considerations

Safety-critical applications impose requirements on a project's system safety team to develop key documents, including a safety data package and a hazard report. For such safety-critical PLD designs, the PLD team could be tasked with supporting the safety team to provide information that they can use to document hazards and their mitigation. Hazard identification could cover safety elements:

a. Controlled by the PLD.

b. Monitored by the PLD.

c. Caused by the PLD.

d. Mitigated by the PLD.

NASA requires Safety and Mission Assurance (SMA) to validate the verification of safety requirements. This can include the witnessing of tests. This validation typically happens at levels higher than the PLD, such as at box or subsystem level. For those cases where validation is applied down at the PLD level, PLD verification planning needs to accommodate key participants such as SMA.

5.2.5 Configuration Management and Revision Control

The project or Center defines formal CM for control of released items. At lower levels, the PLD team or line organization typically establishes a revision control system to manage the changes to and archiving of working documents and design files. A controlled item is anything submitted to the revision control or CM system. A typical system needs to incorporate capabilities including the following:

a. Comprising an established tool or proven process.

b. Method(s) for identifying/marking a controlled item.

c. The process for modifying a controlled item.

d. The process established for user access and privileges.

e. Institutional IT Security measures necessary to secure controlled items for unauthorized access.

The PLD team follows established processes for revision control. If none exist, then they work with their line organization to establish revision control. The PLD team, in conjunction with the project, if necessary, determines the minimum document and design file types to be controlled. A set of revision-controlled items typically includes the following:

a. Design environment (e.g. *.ADB file for MicroSemi devices).

b. Design and test files.

c. Deliverables.

At a minimum, any time the design team programs the design into a physical device, it also puts the controlled items into the revision control system. It is best practice to keep all files used in design, simulation, and hardware testing under revision control. This is especially useful in diagnosing problems where the design needs to revert to a prior test configuration.

5.2.5.1 Design Environment

The revision control goal for the design environment is to ensure that the design environment is documented, controlled, and reproducible. These revision control items include the following:

a. Design tools such as those used for simulation, synthesis, PnR, and PLD programming. If the design is not text- or HDL-based, such as Matlab or schematic, keep the design entry tool under revision control.

b. Setup, project files, and any other file that can dictate the behavior of the design tool.

c. Script files and makefiles.

5.2.5.2 Design and Test Files

All files used in design, simulation and hardware testing are kept under revision control. This set of revision-controlled items typically includes the following:

a. Design documents (design specifications, the verification plan, test procedures, etc.).

b. Files needed for producing the design.

c. Internal or external IP that is used for the design.

d. Simulation test benches and environment.

e. Models used in simulation.

f. Test software and test scripts used during PLD hardware testing.

g. Information obtained from reviews (e.g., peer reviews, action items, defects found, etc.).

h. All results from verification efforts including test results. Any known or open issues related to this design.

i. All programming files (fuse files, bit files, etc.).

j. Bug reports.

It is a best practice to adopt one of the common software revision control tools (e.g., Concurrent Versions System (CVS) or Subversion (SVN)) to manage the files. In addition to keeping the test software and scripts under revision control, the design team records the script, software, and hardware version information in the test log and archives it along with the test artifacts. The design team reconstructs the test environment when troubleshooting a problem using these items.

5.2.5.3 Deliverables

The development plan also defines how the delivered design is uniquely identified within the version control system. This includes the identifying files used for programming the design, the programmed device, and information about when to use the programmed device in a higher assembly. Section 10 contains a list of delivery items.

5.2.6 Training

Training may be required to ensure that all members of the development team maintain a core skill set. The skills can include, but are not limited to:

a. Digital design (basic design, I/O standard, I/O thresholds, etc.).

b. Design capture (VHDL, Verilog, etc.).

c. Design analysis (worst-case analysis, signal integrity analysis, etc.).

d. Radiation effects.

e. Design tools (synthesis, PnR, etc.).

f. Verification (simulation techniques, test techniques, etc.).

g. CM tools or revision control tools.

The PLD team identifies the needed training and records the costs and schedule to implement it in the development plan. The team also itemizes training costs in the development budget.

5.2.7 Schedule

The PLD team lead creates and maintains the schedule for the PLD life cycle, which includes, but is not limited to:

a. All the tasks and work products required to implement this plan. Include planning through the verification effort.

b. All receivables (products that are delivered to the PLD team) that affect the ability to implement the development (e.g., tools, IP cores, development hardware, etc.).

c. The assigned resource and individual responsible for each task/work product.

d. All project level reviews (preliminary design review (PDR), critical design review (CDR), code reviews, etc.).

e. Dependency links with the upper-level project schedule tasks that ensure the development activities occur at the required times.

f. Upon approval, the schedule is baselined with an initial cost estimate.

g. Deliverables.

h. Training.

i. Monitor progress against schedule periodically.

j. Identify issues that affect the PLD development team's ability to adhere to the schedule.

k. Revise schedule as problems are discovered or new requirements added.

5.2.8 Bug Tracking

Bugs are errors in the design or implementation. "Bug tracking" tools simplify the process of capturing and tracking bugs and the exact logic and test case versions required to recreate them. This tool is useful for tracking project-level metrics related to defects found versus defects fixed. It is a best practice for the PLD development team to have a tool for tracking bugs, issues, or potential design changes. Bug tracking information can include:

a. A description of the error, issue, and/or potential design change.

b. The PLD version.

c. The severity of the error, issue and/or potential design change.

d. The status of the error or issue (open, resolved, in progress, use as is).

e. The test configuration.

f. The individual assigned to resolve the error or issue, or to implement the design change.

g. The required completion date/release date.

h. A closure statement identifying the resolution and the source version number with the change.

i. Any impacts to safety.

A best practice for bug tracking is to select a tool that is searchable, sortable, and provides statistics for a number of open, resolved items. For larger projects, this simplifies the gathering of metrics and the sorting of large quantities of issues.

5.2.9 Product Deliverables

Every PLD delivery includes at least the products itemized in section 10. The PLD team lead identifies the expected deliverable(s) for this unique development. If there are intermediate developmental deliveries, they are identified and documented in the development plan.

All product deliverables are included in the schedule.

5.2.10 Review Planning

The development plan defines the appropriate types of reviews that will occur and the appropriate stakeholders that can participate for each review. For purposes of this document, a broad range of possible reviews and participants is provided. The reviews will be customized per the complexity of the design.

The reviews ensure that the design meets all of its requirements for its intended application early in the development process to prevent costly debug and rework. Benefits of reviews include the following:

a. The identification of issues.

b. The clarification of requirements.

c. The sharing of knowledge (technical knowledge, lessons learned, etc.).

d. The generation and satisfaction of exit criteria prior to proceeding to the next phase.

Reviews listed in table 1, List of Possible Reviews, are internal to the PLD development team and separate from the project milestone reviews. Depending on the project, the schedule of the reviews may be planned to coincide with project-level reviews. The reviews and list of involved participants can be customized based upon the complexity of the design.

Table 1—List of Possible Reviews

Phase	Possible Reviews	Possible Participants
Planning and Requirements	Planning Document(s)	Project, Line Management, Stakeholders, PLD Technical Development Lead
	Requirements Review	Project, Stakeholders, PLD Technical Development Lead
Preliminary Design	Architecture Review	Stakeholders, PLD Technical Development Lead, Subject Matter Experts as appropriate.
	Preliminary Design Review	Stakeholders, PLD Technical Development Lead, Subject Matter Experts as appropriate.
Detailed Design	Design Specification Review	Stakeholders, PLD Technical Development Lead, Subject Matter Experts as appropriate.
	Detailed Design Review	Stakeholders, PLD Design Team, Subject Matter Experts as appropriate.
Implementation	Code Review(s)	PLD Design Team, subject matter experts
	Synthesis Review	PLD Design Team, subject matter experts
	PnR Review	PLD Design Team, subject

Phase	Possible Reviews	Possible Participants
		matter experts
Verification Phase	Verification Review	Stakeholders, PLD Technical Development Lead, Verification Lead
Delivery Review	Ready to Program Review	Stakeholders, PLD Technical Development Lead, Verification Lead

5.2.11 Design Review Checklist

Design review checklists are often used for reviews to facilitate the design review process. Depending on the focus of the review, some review checklists can have more details than others can. Appendix B contains sample checklists.

5.3 Requirements

Requirements are allocated to each electronic assembly and flowed down to the board and ultimately the PLD. PLD requirements are negotiated between the board design lead and the technical development lead. The requirements development process iterates as needed. Analyze and review allocated requirements during each iteration. Any issues and questions that are found are discussed and resolved. As a result of this review and discussion, the allocated requirements are refined, clarified, and new requirements derived as needed. For complex PLD designs with many requirements, tracking such requirements can be aided by using a formal requirements database tool such as the Dynamic Object-Oriented Requirements System (DOORS).

5.3.1 Requirements Document

The technical development lead generates a PLD requirements document to capture all the requirements that flowed down. The technical development lead also provides requirements traceability and derived requirements. Once baselined, this document provides traceability for the implementation, and establishes the guidelines for the test and verification steps. Most PLD requirements flow down from the board-level requirements, and can include:

a. Functions to be implemented from the requirements. Block diagrams are appropriate for upper-level function only.

b. Performance (speed, critical timing, throughput).

c. Interface requirements (signal levels, timing, interface-specific data formats).

d. Environmental constraints (thermal, radiation level at part, mission duration).

e. Testability requirements (Joint Test Action Group (JTAG), board scan, software, observable internal points).

f. Redundancy requirements.

g. IT Security (any security requirements on the fielded device, such as mitigation of reverse engineering or unauthorized access).

h. Safety-critical requirements.

For safety-critical requirements, a unique identification can be used. The identification will be used by SMA to identify the appropriate level of verification for the requirements that are safety critical.

For smaller projects, the PLD requirements can be documented in the requirements at either the board or box level and identify the PLD requirements as such.

5.3.2 Requirements Traceability

A traceability matrix provides for top-down and bottom-up requirements tracing. Top-down tracing, such as change impact analysis, verifies that requirements are implemented, and assesses the impact of a system-level requirement change. Bottom-up tracing, such as defect impact analysis, ensures that only required features are present and assesses the impact on the higher level system of any defect found at a low level.

5.3.3 Requirements Maintenance

The technical development lead maintains the requirements document. This includes a process for new or changing requirements that involves the following activities:

a. Analyze the requirements change to determine schedule and cost impacts.

b. In the event of a conflict, assemble the appropriate people to work the issue until impacts and resolutions are determined.

c. Analyze the requirements change for technical impact.

d. Analyze the requirements change for risk impact, including safety.

See also section 5.2.3 for project risks.

5.3.4 Safety Considerations

Safety considerations are typically translated to requirements at the PLD level, such as redundancy, interlocks, upset rates, etc. Appendix E contains additional design considerations. For safety-critical designs, use design criteria 5 found in table 5, PLD Classification, of Appendix C.

5.4 Verification Plan

The verification lead generates the verification plan for verification activities, which includes methods, environments, and criteria. The verification plan documents the plan to verify that the design meets all requirements, is subject to the review of the project team, and includes details regarding how the verification team will record, address, and track to closure the results of verification activities. Verification occurs to the lowest level possible. PLD design verification typically comprises simulations and testing in a hardware environment. Under the formal list of verification methods, simulation falls under analysis.

Independent verification is recommended for larger and more complex projects or safety-critical applications, because it supports a more objective interpretation of the requirements. The advantage here is a more complete assessment of the design.

The verification plan typically includes the following:

a. Formal methods of verification, from a systems engineering standpoint, typically include:

   (1) Inspection.

   (2) Analysis (e.g., simulation, etc.).

   (3) Demonstration.

   (4) Test (preferred verification method).

b. Environment of verification (e.g., hardware test configurations; the simulation approach, including test bench descriptions; identification of additional verification, such as negative testing or simulations).

c. Identify verification method for each requirement. This is necessary for requirements traceability.

d. Test descriptions, including:

   (1) Test identifier.

   (2) Requirements addressed by the test case.

   (3) Prerequisite conditions.

   (4) Test input.

   (5) Instructions for conducting procedure.

(6) Expected test results, assumptions, and constraints.

(7) Criteria for evaluating results.

(8) Requirements traceability.

(9) Identification of test configuration.

5.4.1 Verification of Intellectual Property Cores

The use of IP cores (acquired, inherited, reused) does not eliminate the need to verify the core. See section 6.3 for more information on the use of IP. Considerations for the verification of IP cores include:

a. Verify that the designer has reviewed the latest IP core data sheet specifications, application notes, revision notes, test benches, addendums or errata notices for core modifications, updates, license agreement, and any vendor documentation describing the extent of vendor verification of their IP cores.

b. When using heritage design or design elements, verify that the IP has been demonstrated to meet all requirements in the new intended application.

c. Verify that the verification (simulation and hardware test) strategy is robust enough to maximize coverage of functional or operational scenarios and configuration (e.g., “test as you fly,” “fly as you test”).

d. Verify project requirements on radiation have been implemented in the IP core design. In other words, verify that the IP core can be synthesized to incorporate any necessary radiation-mitigation strategies, such as triple mode redundancy (TMR) or, for the case of a hard core, comes implemented with appropriate radiation-mitigation strategies.

e. For externally obtained IP, seek vendor-created test bench files/examples for verification.

5.4.2 Tool Verification

The purpose of tool verification is to identify errors that the tool may inject in the design. While it is possible to verify some tools, it is costly. A more effective method to mitigate the risk is through extensive PLD verification using accepted industry-standard tools. It is a best practice for the designer to read all errata that came with the tools and be part of the user community so that errors/workarounds can be mitigated/used early.

5.4.3 Verification Matrix

The verification matrix maintains traceability from the requirements through the verification of all the requirements. The matrix shows how the requirements are described/designed in the design specification and where in the verification plan the requirements are tested. Finally, it traces the requirements into the verification documentation (test procedure, test bench, analysis report, etc.) that identifies how the requirement is verified. Table 2, Sample Verification Matrix, below shows an example.

Table 2—Sample Verification Matrix

Requirement	Design Specification	Verification Method (I,A,D,T)	Verification Plan	Test Procedure/ Test Bench/ Analysis Report
1.5.7 abc	Section 2.3.5	T	Section 1.3	TPR-123

6. PRELIMINARY DESIGN PHASE

During the PDP, the design team generates a top-level design to describe the overall architecture of the design. For small projects, the design team may combine the preliminary design in the DDP. For these projects, the system-level design, the subsystem design, and the assembly design may incorporate the preliminary design.

6.1 Entry Criteria

The PDP is the design phase that begins the design cycle. The top-level design is generated in the PDP. With the design of the top level, the items from the planning phase can be updated and changes identified. The PDP can be entered when the following criteria are met:

a. Development plan is baselined.

b. PLD requirements are written, but may not be baselined.

This list does not include all elements of the planning phase. It allows for work to proceed while additional planning is in progress.

6.2 Design Specification (Preliminary Version)

The preliminary version of the design specification is prepared, reviewed, and baselined. The specification can include the following items:

a. A board-level block diagram depicting PLD internal and external interfaces.

b. A top-level block diagram of the PLD architecture.

c. Data flow diagram(s).

d. Selected PLDs part type(s) with preliminary utilization estimates.

e. A clock distribution description with frequency information.

f. A reset distribution description.

g. Radiation effects mitigations (e.g., scrubbing, TMR, etc.).

h. Board-level considerations (e.g., signal integrity, power integrity, power consumption, etc.).

i. A description of all hardware and software interfaces.

j. Re-configurability requirements (if applicable).

If the PLD interfaces to elements outside the board, then the design team includes a system-level block diagram.

6.3 Interface Definition (Preliminary Version)

The interface definition defines the interface between multiple interfaces (e.g., multiple pieces of hardware, hardware and software, avionics and software, etc.). This document need not be unique to the PLD. It can be part of a larger document.

The preliminary version of the interface specification is prepared, reviewed, and baselined. The interface specification can include the following items:

a. The register interface.

b. Bit definitions of registers.

c. All interface signals.

d. Memory map.

e. Protocols (e.g., RS-422, PCI, 1553, etc.).

f. Pinout.

6.4 Use of Intellectual Property (In-House or Out-of-House)

Each development organization documents the approach to meeting certification requirements for any computing system's hardware elements that include any off-the-shelf (OTS) components (e.g., a previously designed "heritage" circuit board assembly, an application-specific integrated circuit **(**ASIC) used in another design, IP procured or obtained from another source.)

Each developer of PLDs that include non-development items (i.e., design elements that are reused from another application or that are procured or obtained from a source outside their developmental control, such as IP) ensures that the inclusion of such non-development items meets all the requirements of the PLD design in its intended application. This assessment extends to the overall circuit design as well.

When purchasing IP, the licensing agreements are reviewed to make sure they extend through the life of the project. This includes the maintenance phase. Additionally, purchased IP may not

include the source code or the detailed core design, possibly making testing of the core very difficult. When the source code is not available, the project has to analyze the risk of using that core. The core will have to be verified without the ability to review the source code completely. There may also be limited access to testing the core. The project team weighs the risk versus the benefits of that core.

When purchasing IP, allow your Center's legal team to review the licensing agreement. Licensing agreement negotiations can be lengthy, impacting the schedule. Also, ensure that the agreement extends through the life of the project, including the maintenance phase.

6.5 Exit Criteria

The PDP is the first phase of the design cycle. The PDP is complete when the following criteria have been met:

a. The preliminary design is complete.

b. A preliminary version of the design specification is baselined.

c. The interface definition(s) are baselined.

d. A list of proposed IP (heritage and new) that will be used in the design is identified.

e. Reviews are completed as defined in the development plan.

f. The planning phase documents (see section 5) are baselined, including:

(1) Requirements.

(2) Development plan.

(3) Verification plan.

7. DETAILED DESIGN PHASE

The DDP is entered upon completion of the PDP. The DDP generates the design specification, but not the design files used to implement the PLD. Detailed design information, such as timing diagrams, detailed block diagrams, is developed during this design phase. The information developed in this phase is used for capturing the design during the design implementation phase.

7.1 Entry Criteria

The DDP details the PLD design information as well as generating the complete design. The DDP can be entered when the following criteria are met:

a. The preliminary design specification is baselined.

b. The interface definition is baselined.

c. The PDP and associated reviews are successfully completed.

d. The planning phase documents (see section 5) are baselined, including :

(1) Requirements.

(2) Development plan.

(3) Verification plan.

7.2 Coding Standard

It is recommended that the PLD team utilize a common HDL coding style. Design organizations may consider the use of common coding standards across multiple projects, in order to facilitate effective reviews and design insight.

The following items help make a design reviewable:

a. Standard coding style.

b. Documentation that is consistent with the design.

c. Current, regularly updated documents.

d. Use of appropriate comments for the design.

e. Use of configuration control for the design.

Examples of the types of HDL standards that may be defined include, but are not limited to:

a. Use of naming conventions to allow recognition of the function of signals by their name.

b. Use of the header of the HDL design to capture nomenclature (see Appendix A).

c. Use of modular design to ease testability, readability, and simulation.

d. Use of editor tools (tool editor software, Ultra-Edit, use of spaces versus tabs, etc.).

e. Use of self-checking/documenting test-benches.

f. Use of proper code documentation, particularly inline documentation, which later helps if a personnel change is made in the middle of the project.

g. Documentation of each procedure or function.

h. Use of inline comments to explain thoroughly the applicability of assumptions and rationale used to achieve the design.

i. Use of a consistent design language across all designs, agreed upon at the project/branch level.

7.3 Design Standard/Best Practices

This section describes a list of items that are design specific. The PLD designer reviews all the items to decide which are applicable and which are not.

7.3.1 Synchronous Design

It is recommended as a best design practice to use synchronous design methods. The behavior of a synchronous design is predictable/deterministic and more tolerant to race conditions, hazards (dynamic or static), or glitches introduced by environmental conditions. Fully synchronous designs can also be more easily migrated or reused across PLD device types. The use of non-synchronous design methods is rarely appropriate, and their use has to be sufficiently justified, because they usually require more effort to validate or verify in the design.

Synchronous design techniques include the following:

a. Transition all signals on the clock edge.

b. Avoid using a gated clock (a clock that is driven from logic, not a clock buffer).

c. Handle carefully signals that cross clock domains. There are many techniques, such as double registers; metastable resistant registers; first in, first outs (FIFOs); etc. The design identifies the best method for the particular implementation.

d. Synchronize asynchronous inputs to system clock in one location.

e. Minimize the number of clock domains.

f. Utilize synchronously de-asserted rests.

7.3.2 Clock Design

When following synchronous design methodology, a PLD best practice is to use low-skew clock buffers on clock and reset nets. Use of low-skew clock buffers simplifies timing analysis and avoids race conditions (e.g., hold-time violations) by enabling more simultaneous (synchronized) clock arrival times. This ensures more deterministic functional behavior. Therefore, when sequentially adjacent registers are clocked on a common edge, use low-skew clock resources. It is acceptable to design with routed clocks, and this can often result in a reduction of power consumption or an effective increase in the number of clocks available. However, precise skew-tolerant design techniques and analysis have to be used. In addition, routing clock signals over long distances inside the PLD to the inputs of clock buffers using regular routing resources may make the signals vulnerable to crosstalk from nearby signals, possibly resulting in errant behavior. To avoid this, ensure that PLD clock input pin or logic that generates the clock is close to clock buffer input.

7.3.2.1 Chip-to-Chip Timing Strategy

Many analysis tools are useful for analyzing logic within a single chip, but few are effective at analyzing system or chip-to-chip timing. In addition, while the worst-case behavior of the clock-to-out of the source chip is easily analyzed using “minimum” or “best case” timing parameters, the hold time of the destination chip may need to be analyzed assuming a slow path for the clock and a fast path for the data for the same calculation. A strategy to consider would be to assign or allocate delays to each signal leaving one device and being received at the next. These

allocations can be used as constraints for synthesis and timing analysis to ensure that the system will meet timing once integrated.

In general, rigorous analysis of the chip-to-chip interface delays and clock skew is performed to validate the destination capture scheme. Analysis considerations include the following:

a. Trace delays from board-level connections (or other medium connectors).

b. Output (source chip) and input (destination chip) buffer delays.

c. Delays from input buffer routing to internal structures.

d. Clock tree latency.

e. Signal skew.

f. I/O signal integrity (e.g., transmission line effects, ground bounce, and cross talk).

The optimum criteria for passing is that all worst-case setup and hold times are always satisfied.

7.3.2.2 Delay-Locked Loops (DLLs) and Phase-Locked Loops (PLLs)

Consider the analysis required before using the DLLs and PLLs. DLLs and PLLs can have many useful functions in digital systems, but they have some requirements that have to be satisfied:

a. Check that the worst-case frequencies (both slowest and fastest) are compatible with the circuits; often the acceptable ranges are very limited.

b. When these circuits clock finite state machines (FSMs) or other sequential logic, verify that the DLL/PLL starts up, stabilizes, and locks up properly and reliably to ensure safe circuit operation.

c. Check the worst-case performance when the DLL or PLL encounters a single event upset (SEU). This can result in a change of programming of the DLL or PLL, which is sometimes subtle, or a change in mode.

(1) Safe operation of the system has to be ensured during these off-nominal conditions.

(2) An SEU or single event transient (SET) can cause the DLL or PLL to unlock or glitch and, consequently, make the entire circuit that is within the clock tree unstable or inoperable, necessitating a reset.

d. Account of DLL/PLL clock jitter and stability.

e. Power supply and decoupling strategies on the circuit board for the DLL/PLL circuit.

Use of internal DLL/PLL PLD circuitry has to include careful analysis of a project's radiation requirements and available radiation test data on the PLD's DLL/PLL circuitry, which is typically softer than the other elements within the PLD.

7.3.2.3 Crossing Clock Domains

FPGA designs with multiple clock domains are now common. Perform clock domain crossing (CDC) analysis. Based on analysis of the clock trees, identify all signals crossing clock domains and determine the need for metastable state resolution. When de-metastability logic is used, it needs to be evaluated for its correctness and effectiveness. This is especially important for reset signals used in each clock domain to ensure proper operation for all possible sequences of reset removal between clock domains. Additionally, ensure that the latency involved in signal synchronization is tolerable to the system.

7.3.2.4 Opposite-Edge Clocking

Consider duty cycle in timing analysis for designs that use both clock edges. With justification, document any designs that pass data from one edge of a clock to the other. Analyze such circuits using the worst-case duty cycle for each phase. Often designers assume a 50 percent duty cycle, which may not be the case. Sources of duty cycle distortion include oscillator characteristics, with 50 +/- 10 percent duty cycles being common, including uneven delays through logic gates and buffers, etc. Unless required to meet timing, avoid using opposite-edge clocking as it complicates the timing analysis for the design.

7.3.2.5 Metastability

Ensure that proper synchronizers are used for each asynchronous signal to guard against metastability. Often designers will use two series D-flip-flops (DFFs). While this is a common and acceptable topology, be aware that for high-speed signals, the failure rate of this synchronizer can be non-negligible. Analysis has to be done for situations that may require a third DFF to be put in the series. The following are notable conditions to address:

a. The metastability analysis includes nominal through extreme temperature and voltage values.

b. Ensure that there is margin in these circuits as they are impractical to test and verify.

c. Also, note that for ASICs, different flip-flop macros may have significantly different metastable parameters. This can also be a consideration in PLDs.

7.3.2.6 Latches

A best practice is to avoid the use of latches because they complicate the timing analysis of a design. Furthermore, PnR tools do not satisfactorily analyze timing paths with latches. It is not uncommon that a latch can be replaced in a design by a DFF. Therefore, the recommended approach is to replace latches with DFFs.

Latches can also be unintentionally inferred (added by the tool during synthesis), for example by not defining all output states. Run a register report after PnR to confirm that there are no latches present.

7.3.3 Finite State Machine Design

FSMs have to be designed with care for critical applications. Verify the implementation at levels of abstraction appropriate for each circuit. For example, for certain critical sections of an HDL-based design, examination of the tool-generated netlist via a schematic viewer may be required to ensure that a reliable circuit has been implemented. There are very few designs with a single independent state machine. Most designs have several, if not many, interconnected state machines. Any correction or recovery algorithm (an FPGA or card-level reset, for example) would need to take into account all of the interconnected state machines and be thoroughly analyzed and tested in order to verify proper recovery. Note that the correction method may even reside at a higher level such as at the subsystem or box level.

7.3.3.1 State Machine Transitions and Default States

Analyze and verify that state machines start in a known state and transition through all states as intended. The analysis also needs to include, especially for critical state machines, considerations for off-nominal events that could cause faults that interrupt nominal state transitions. Credible faults could occur because of such hazard events as a disturbance on the power bus, an electrostatic discharge (ESD) event, a radiation-induced upset, etc. Note that hazard events are not synchronized to the system clock and the logic network is not guaranteed to be glitch free.

Any critical state machine has to be robust under all credible faults. In general, analysis of the combinational circuits that implement the next-state logic and their inputs to the registers making up the state register is needed. In particular, for any of these schemes, it has to be determined whether the circuit implementations are static hazard free and, if not, whether an erroneous transition to a state (or set of states) can occur. It may be very difficult for a user to guarantee deterministic behavior of the state machine if an upset occurs. This could also be the case for very large or complex state machines.

7.3.3.2 State Machine Transition and Fault Recovery

The PLD design team addresses FSM lockup at a higher level and analyzes FSM outputs. In general, each FSM is analyzed to make sure that the system can detect a locked up FSM and return it to a known state in a timely and safe manner. Recovery options can include an automated system recovery or an external event (watchdog reset, commanded reset, or power cycling), if acceptable to the system design. For mission-critical applications, the FSM outputs have external protection, such as requiring flight software to “arm” the FSM outputs. Implementation of a reset from an external resource (e.g., watchdog timer) can simplify fault recovery analysis in that the reset forces the entire design associated with that reset into a known state. Though the externally generated reset would be the preferred approach, if there were no system-level watchdog available or it does not adequately address an FSM’s response to a fault, an alternative method could be to implement an onboard watchdog that operates off separate clock and reset resources.

A further recommendation would be to incorporate off-nominal events into the simulation environment to assess system response, such as verifying a watchdog reset. This assessment may be extremely valuable given that this functionality may be impossible to test in a hardware test

environment.

7.3.3.3 State Encoding

The use of a state vector encoding scheme that meets requirements is another recommended best practice. The choice of state vector encoding is one that factors in radiation effects, timing constraints, and criticality of operation. Often, designers allow the synthesis tool to choose the encoding scheme that is optimal for timing constraints. However, the designer still reviews the synthesis tool's choice factoring in radiation effects and criticality of operation. Following are some factors to consider:

a. In PLDs, if using TMR with triplicated registers, upsets are more common from SETs than SEUs. Therefore, combinatorial logic poses a greater vulnerability. One-hot encoding uses more flip-flops, but less combinatorial logic to encode the states and becomes a robust high-speed option.

b. With one-hot encoding, single bit errors are detectable. If the implementation decodes all bits in the state, the implementation can recover from an invalid state. However, when one-hot encoded state machines experience an upset, it is likely that two state bits will become "hot" and activate two parts of the design that are not normally activated simultaneously. The designer determines whether this situation could cause any damage.

c. With binary coded state machines, detecting illegal states and transitions requires the use of additional logic that increases susceptibility to radiation effects.

d. Binary encoding of states that uses all 2^n states does not guarantee the FSM will not lock up if the FSM interfaces with external logic. In this situation, an SEU could disrupt the normal sequence of operation.

e. Generally, the upset rates between different state machine types are very small even with error detection and recovery (safing) implemented. As such, avoid over-engineering FSMs. More specifically, do not assume that "safing" FSMs individually is better than using one hot and/or binary FSM encodings without safing. Whether to safe an FSM or not depends on the system level consequences of an FSM being upset and what recovery actions are needed.

7.3.3.4 State Machine Synthesis

The PLD design team analyzes the synthesis reports and synthesis tool outputs. Common things to check include the following:

a. Recognized state machines.

b. Lockup states.

c. Outputs from the state machine that can glitch.

d. Unintended register replication.

e. Implementation of the desired and specified style (sometimes the synthesis tool substitutes one type of state machine for another).

Additionally, some logic synthesis tools generate “safe” state machines. Use of this feature is not recommended because it typically increases the use of combinatorial logic, which increases the SET susceptibility. If this feature is used, examine the generated design carefully as it could sometimes add an excessive number of gates to the implementation. Other times, resets are generated on the opposite edge of the clock resulting in tight timing for the removal of clears that are not visible to the designer. Note that when using enumerated states in VHDL, not all physical states are covered (only enumerated states are covered). Hence, the “others” clause only refers to states in the enumerated type and not the physical realization. The HDL does not identify one-hot or binary or gray-coded implementation or which registers have been replicated. This is not detectable at the black box simulation level or by Boolean equations for logical equivalence.

To ensure that the state encoding is not altered by the synthesis tool, synthesis directives such as “syn_preserve” (from Synplicity) can be used to ensure that the synthesis tool does not change the state encoding.

7.3.4 Resets

It is important to understand the purpose of reset implementations. For example, different approaches are recommended depending on if the reset signal is used to initialize finite state machines as opposed to protecting external resources (e.g., contention on a tri-state bus, electrically erasable programmable read-only memory **(**EEPROM) protection, relay activation, pyrotechnic initiation).

Every flip-flop in a design is reset to a known state by the global power-on reset (POR) signal. Failure to reset all registers in a design can cause mismatches between simulation and actual device. Each flip-flop has either an asynchronous preset or a reset signal. If the flip-flop needs both a reset and preset condition, one has to be changed to a synchronous signal. If they are both asynchronous, they can nullify each other since the timing relationship between them is undetermined. It is important to make sure that the design specification clearly defines all register states at POR for bits that are directly controlling circuit functions (on- or off-chip) without first being initialized by software.

Synchronous resets can typically be analyzed with just static timing analysis (STA) as far as the PLD is concerned but require board, box and/or system analysis to determine if indeterminate PLD outputs are acceptable until the synchronous reset is clocked through. Synchronous reset signals have to be routed on low-skew global routing networks. This is vitally important to ensure that all synchronous elements in a design are reset at exactly the same time. System reset is typically active low at the board level because at power on, the board starts at 0 volts so nothing needs to be done to assert the reset. Inside the PLD, the active state of the reset (high or low) is dependent on the technology being used. It is best to check the device technology prior to setting the reset active state. A recommended system reset is asynchronously asserted, and synchronously de-asserted, with the minimum pulse width required. An analysis is done to

determine the time that the reset is to be asserted and de-asserted. This is design dependent, and is done for the following reasons:

a. Ensures that the PLD is held in reset at power up even if the clock is not stable.

b. Ensures that the PLD cannot come out of reset in the absence of a clock.

c. Ensures that the internal reset signal is held for a sufficient amount of time to propagate throughout the chip. This does not allow a glitch on the reset line to cause an internal race condition.

Refer to Appendix F for more information on resets.

7.3.4.1 Implement Synchronous De-Asserted Reset.

It is recommended that the team implement synchronous de-asserted reset using a global buffer, if available. For asynchronous presets and clears, there are two basic parameters that have to be met. First, there is a pulse width requirement that has to be guaranteed. Second, removing the preset or clear from a device asynchronously to the clock may result in metastable states in the sequential circuit. This parameter is frequently called the removal time (asynchronous (clear and preset) or removal time) and is denoted as t_{REM}. Unfortunately, many data sheets do not specify the removal time. Use a synchronously de-asserted reset to ensure that the removal time requirement is met.

7.3.4.2 Generate a Reset Tree Diagram

Drawing a tree of all the reset sources, buffers, and domains is often helpful in ensuring that the reset logic is well defined. Often there are multiple forms of reset from system resets, software resets, watchdog timers, etc., and having a detailed tree diagram shows the relationships between them. Ensure that proper synchronization is made when required. Additionally, if the reset needs to be activated quickly, the tree helps to ensure that the logic and delays are well understood. Examples include when there is a need to protect non-volatile memories from false writes, or other circuits from initiating one-time events, such as firing pyrotechnics or explosives.

7.3.5 Device Inputs/Outputs

The PLD designer reviews/analyzes each I/O’s dynamic and static characteristics, such as timing, logic threshold, direct current (DC) voltage characteristics, and I/O standard, with the board designer to take into account the manufacturer’s recommendations to ensure proper operation of the devices.

7.3.5.1 Simultaneous Switching Outputs (SSOs)

Adhere to the vendor’s recommendations for handling SSOs. There may be limits to the number of output pins that can switch at one time. Sometimes the manufacturer specifies these limits in a data sheet, describes them in an application note, and/or leaves them to the discretion of the designer. Ground/V_{DD} bounce can be a serious issue that can dynamically affect input switching

thresholds, decrease system noise margins for fast-switching devices with large pin counts, and lower noise margins. It is also important to note that for many devices, the number of SSOs can affect the propagation delay time (T_{PD}).

Care and planning is also important for pin assignments. Pin assignments that seem organized with all the data bits on a bus lined up in a row have been notorious for causing both ground bounce problems on the printed circuit board (PCB) and routing problems inside PLDs. Consider using power integrity tools, as they provide a means to accurately predict the effects of SSOs on a given design. Note the considerations that follow for simultaneous switching outputs, noise immunity, quiet design principles, and minimizing bounce issues:

a. Use the lowest possible I/O slew rate and drive strength the design timing will support.

b. Distribute SSO signals across different power banks instead of grouping them together. Refer to the device datasheet for recommendations on allowable SSO signaling per ground pin.

c. Control the number of SSOs through sequencing. For example, determine whether address or data bus bits all need to switch at the same time.

d. For some families, programming “unused” outputs improves internal grounding or power integrity of the I/O ring, if they are terminated to the power or ground sources on the board.

e. When PLD output drive is not sufficient, particularly for large memory arrays or long lines, use external buffers, being careful to adhere to proper PCB design techniques.

f. Do not use sockets for flight applications.

g. Choose input thresholds with maximum noise margin.

h. Choose input options with higher voltage threshold to mitigate the effects of ground bounce.

i. Keep sensitive signals (i.e., clocks, resets) physically away from pins that can cause ground bounce (i.e., high frequency switching pins, pins with fast (short) rise times and address/data buses).

j. Assign clocks to pins that are close to ground pins.

k. Driving test data through the Joint Test Action Group (JTAG) test interface, especially over multiple parts, can induce SSO data patterns, particularly with large data buses, for example, switching patterns from FFFFFFFF to 00000000. Though this may be an artificial failure or an artifact of the test, this can damage or potentially overstress hardware through a loss of control.

l. Test cabling, particularly for vibration, thermal/vacuum, and electromagnetic interference (EMI) tests, presents different conditions for normal bench testing or systems application. Design for the worst-case over the entire project flow.

m. If applicable, consider the use of lower voltage I/O standards. PLDs often have lower voltage I/O standards available. Lower voltage I/Os have lower transient currents that can reduce SSO.

7.3.6 Radiation Effects on Reconfigurable PLDs

The use of reconfigurable PLDs is sometimes desired because it allows in-flight reconfiguration, usually has more on-chip resources, and can operate at higher speed than the one-time programmable PLD. The issue with using reconfigurable PLD in space flight or high altitude vehicles (e.g., High Altitude Long Endurance (HALE)) is its susceptibility to radiation-induced errors. As a result, additional logic is usually needed to reduce such susceptibility and improve the device reliability for space applications.

Reconfigurable PLDs contain internal memory for configuring the logic blocks within the PLD to perform the required logic functions. The internal configuration memory is not always radiation hardened and can be upset by radiation events. When the configuration memory is altered through a radiation event, the impacted logic can be permanently changed unless the PLD configuration memory is reloaded or scrubbed. The upset rate of configuration memory depends on the PLD technology and the radiation environment.

7.3.6.1 Radiation Mitigation Techniques

The simplest form of protection used on PLD configuration memory is open-loop scrubbing. This consists of using a separate radiation hardened or tolerant device to continuously overwrite the configuration memory of the PLD with configuration data from a known and reliable source. This method of protection is easy to implement and can usually correct the upset memory very quickly. However, its effectiveness depends on the radiation upset rate of the PLD configuration port. It only works if the PLD configuration port stays healthy. Without a feedback mechanism, it can sometimes be difficult to determine whether the configuration port is receiving the data sent by the scrubbing logic. As a result, this method of protecting PLD configuration memory is usually used along with other protection schemes, such as configuration memory read back or periodic reconfiguration of the entire PLD.

Another technique is to read back and verify the configuration memory. The content of the memory can be verified by either comparing it to that from a reliable source or checking the associated cyclic redundancy check (CRC) signature against the expected value. Once the configuration memory is deemed corrupted, it can be overwritten with data from the reliable source. The correction can be done only for the corrupted memory blocks or all the configuration memory blocks, depending how sophisticated the readback/scrubbing controller design is. Unlike the scrubbing logic, the readback logic can be implemented inside the reconfigurable PLD itself as long as the health of the readback logic can be monitored by an external radiation hardened or tolerant circuit or PLD. This method of configuration memory protection is more reliable than

open-loop scrubbing. Any corruption to the configuration data or the configuration port can always be detected. The drawback with this approach is that response time can be slow. Once the corruption is detected, another mechanism, such as open-loop scrubbing or full reconfiguration, has to be deployed to correct the problem. This is expected to take more time than using open-loop scrubbing.

7.3.6.2 Redundancy

TMR is also used for mitigating radiation induced upsets in PLDs. The scheme employs three copies of the circuit to be protected by TMR and a voting logic. The correct output of the circuit is determined using majority rule. Feedback logic is sometimes used to correct the errors in the corrupted copy of the circuit. For reconfigurable PLDs, TMR is usually not built into the PLD. The designer may need to implement TMR or other methods to mitigate the potential problems caused by radiation-induced upsets.

Designers need to be aware that there are different TMR methods including Local TMR, Global TMR, and Distributed TMR. Each has its advantages and disadvantages in terms of protection afforded versus impacts on device or circuit resource. It is not sufficient to rely solely on manufacturer’s literature on radiation effects mitigation. For the purposes of this document, it is recommended that the PLD designer (or design team representative) also consult appropriate radiation effects engineering resources to determine the optimum method for the intended application.

TMR can be implemented manually by the designer or by using design tools. Once TMR is implemented, the designer ensures that the PLD synthesis tool does not reduce the redundant circuits from the TMR design while optimizing the design.

7.3.7 In-Flight Reconfiguration

In-flight reconfiguration is used very carefully if the PLD is performing system-critical functions. Improperly configured PLD can cause the device to stop functioning in-flight. Systems designed to support in-flight PLD reconfiguration have to have a fail-safe mode to restore the system to the last working state in the event of a reconfiguration error. The fail-safe mode allows the PLD to be reconfigured using the last known reliable configuration file stored locally in the system, from an external location on the spacecraft, or even from the ground.

7.3.8 Designing for Testability

Consider adding signals to facilitate design development and debugging. Debugging designs in the lab comes with many constraints. One of them is visibility of signals. Be sure to allocate test points or spare pins for diagnostic purposes. There are many steps that a designer can take to address these constraints, including the examples that follow:

a. JTAG interface. PLDs often come with JTAG interfaces that can be used to probe internal signals. Such interfaces can come in very handy, but often come with frequency limitations and limit the number of signals that can be viewed simultaneously. Be sure to accommodate the PLD signals associated with the JTAG interface.

b. Debug mux. A multitude of internal signals can be brought to a multiplexer whose output connects to PLD outputs. The multiplexer select signals can be driven by PLD inputs or by other suitable means. This approach can be used when a PLD's JTAG interface has limitations that pose a problem.

c. Heartbeat output. A quick and easy method of checking if a PLD is "functional" is to generate a heartbeat output signal that pulses periodically when certain critical events occur. The utility of such a signal depends greatly on how the designer chooses to generate the output.

d. Ensure there is a way to disable debug outputs for flight. Signals that are clocks or switch states can be a source of unwanted EMI.

Regardless of testability features implemented, it is important that the implementation be analyzed sufficiently in the final flight design to ensure no impact on in-flight system functionality or reliability. Ideally, all testability features are disabled for flight.

7.4 Design Specification

During the DDP, detailed design information is generated based on the preliminary design. Detailed timing, state, and block diagrams are generated. The PLD specification containing the detailed design is prepared, reviewed, and released. The specification contains enough details to allow an engineer to implement and verify the design. Updates to the specification include the following:

a. Block diagram of the PLD architecture, including:

(1) Clock generation, distribution, frequencies, etc.

(2) Reset generation, distribution, etc.

(3) State machines, state diagrams, and state tables.

b. A functional description of the device and a detailed description of the individual blocks, including IP definition and usage. The mathematical functions that are required will include the following:

(1) Method of calculation, including the name of the theorem (e.g., whose method of a fast Fourier transform or which Reed Solomon implementation is being used).

(2) Precision.

(3) Accuracy.

(4) Performance.

c. List of targeted devices to be used, including prototyping versus engineering units versus flight units.

d. SEU mitigation implementation, including scrubbing, TMR, etc.

e. A description of all interfaces, including board-level implementation constraints (critical pins for board routing, proximity to other devices, input slew-rate limitations, etc.).

f. Software interface, including:

(1) Memory map.

(2) Register definitions.

(3) Operations constraints.

g. Timing information, including waveforms for critical interfaces.

h. Re-configurability, if applicable.

7.4.1 Clock Tree Diagram

Draw a board-level diagram showing the clock trees for the circuit that includes PLLs, DLLs, clock buffers, clock dividers, and all chips that use the clock. See the example provided by figure 3, Sample Clock Tree Diagram, that follows.

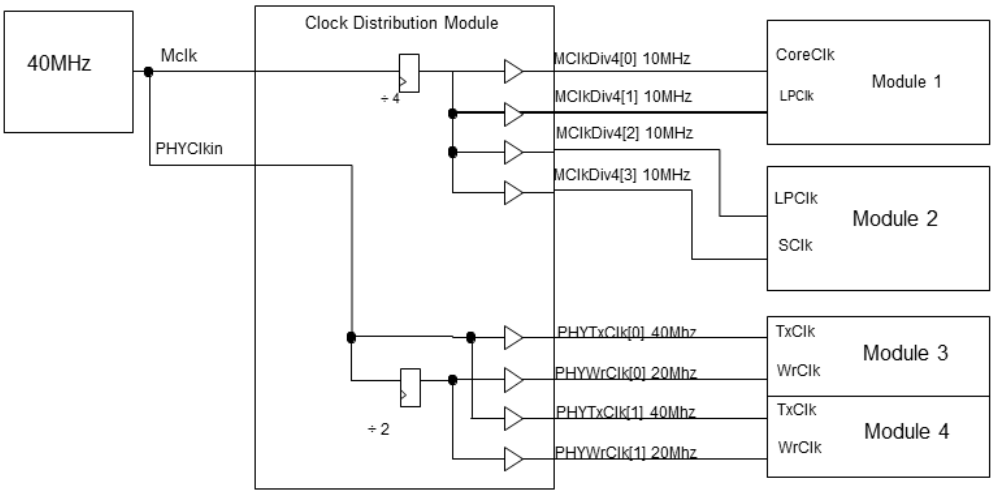

Figure 3—Sample Clock Tree Diagram

7.5 Board and PLD Interface Considerations

In order for a PLD to operate correctly, it is important to review and ensure the correctness of the interface between the PLD and the board on which it resides. The following list provides some key areas to review for the PLD and board interface:

a. Review the connections of vendor-specific special pins. Pins such as TRST, VPUMP, and VREF for PLDs from MicroSemi and HSWSPEN, INIT_B and PROB_B for PLDs from Xilinx all have to be terminated properly for the PLD to operate correctly. Review the PLD datasheet for the requirements about how the special pins are connected, and verify that the corresponding connections are correct on the board.

b. Review output loads and fan-outs. Make sure that the PLD I/O buffer driving strength is compatible with the loading (e.g., high capacitance or non-logic loads).

c. Review I/O standard compatibility. Make sure that the correct logic threshold is selected for the PLD I/O buffer when mixed logic families are used.

d. Review signals coming from and going to a different power domain. Make sure the cold-sparing and power sequencing requirements are met. Review circuitry that may be powered independently of the PLD, including cold-sparing applications, and make sure that adequate protection circuitry is implemented.

e. Review circuit board signal integrity analysis. Ensure that the signal overshoots and undershoots do not violate PLD manufacturer’s absolute maximum ratings for I/O signals. Using low slew rate for the PLD outputs and terminating the board signals are two effective methods for reducing ringing levels for board level signals.

f. Review selections of pull-up and pull-down in the PLD I/O buffers and on the circuit board. Make sure that the selections for the pull-ups and pull-downs inside the PLDs are consistent with those implemented on the circuit board.

g. Plan the design with testing in mind and incorporate the resources needed to facilitate it:

(1) Consider observability as the design is implemented.

(2) Consider approaches for debugging the circuit while the part is on the (breadboard/engineering test unit (ETU)/flight) board.

Appendix F details additional board-level considerations.

7.6 Design Test Coverage

The designer identifies what design features or functions outside the formal requirements verification process need to be tested. The purpose is to mitigate any risks that may develop as a result of functions that may not be fully tested during requirements verification.

7.7 Software Development for an Embedded Processor

PLDs may contain one or more embedded processors, such as microcontrollers, central processing units (CPUs), graphics processing units (GPUs), and/or digital signal processors. These embedded processors execute software ranging from a simple series of instructions to an operating system running applications, which is separately developed from the PLD design code. This NASA Technical Handbook does not cover the development, verification, and validation of software for such embedded processors. All software will be covered by software requirements in NPR 7150.2, NASA Software Requirements.

7.8 Exit Criteria

The DDP is the final phase of the design cycle. Upon completion of the DDP, the following criteria are met:

a. The design specification is reviewed and baselined.

b. The interface definition(s) are baselined.

c. The detailed design (from the design specification) is completed.

d. Reviews are completed as defined in the development plan.

8. DESIGN IMPLEMENTATION PHASE

The design implementation phase is the phase when the design is implemented, coded, and reviewed. This occurs after the detailed design is completed. The generated result will be the PLD programming file(s) that will be used in the verification aspect. During the design implementation phase, the verification team may begin to implement preliminary versions of the

verification items. These can include test procedures, test scripts, test benches, etc. In addition to generation of the programming files, the design undergoes several timing analyses, including STA and worst-case timing analysis.

8.1 Entry Criteria

The design implementation phase can be entered upon completion of the DDP.

8.2 Implementation

The implementation of the design follows the detailed design and involves:

a. Transferring the design from the design specification to the HDL code (coding).

b. Incremental testing: Simulations at both the module and chip levels.

c. Reviewing the design files.

The sections of this phase are not followed in linear form. These phases are iterated by the designer and the team, as appropriate.

8.2.1 Implementation of the Design

The implementation of the design occurs when the design engineer takes the design specification, the coding standard, and the design standard and implements the design in the specified technique (VHDL, Verilog, schematic entry, etc.). The designer follows the coding standard setup by the project and applicable sections of the design standard (section 8.4).

8.2.2 Incremental Testing

As part of the coding, the designer typically performs incremental testing (generally in the form of simulation) to test the operation of the code. Incremental tests begin at the lowest level and continue up through various levels of code. For purchased IP, it is best first to perform a test on the IP to confirm its functionality prior to using it in the design.

8.2.3 Reviewing the Design Files

As the design is developed, the PLD team lead organizes and runs reviews on the implementation (code, drawings, etc.) per the development plan. For example, the purpose of a code review is not just to check the stylistic portions of the code, but to ensure that the design is complete and meets the requirements specified. Issues found at the design file level are much easier and take fewer resources to fix than those found in various stages of verification. Prior to any review, the design is deposited in the appropriate revision control system.

8.3 Verification Preparation

The verification team begins development of the test preparation (test benches, test scripts, test procedures) used for verification and independent testing of the device. This effort can occur in parallel to the development of the design. Section 9 defines the verification items.

8.4 Synthesis and Timing Analysis

This section describes a list of items that are design specific. The PLD designer reviews all the items to decide which are applicable and which are not.

8.4.1 Synthesis and Place and Route

8.4.1.1 Synthesize the Design

When synthesizing the design, consider the following items:

a. Synthesis is performed for both prototype and flight parts to ensure that the design is implementable on the final flight part.

b. Use constraint files and script files, rather than the tool's graphical user interface, to assist with self-documenting design.

c. Review output files and logs to verify that the design was not altered by the synthesis tool.

(1) Search through the synthesis report for unexpected resources/primitives macros such as:

A. Flip-flops without sets or clears, indicating circuitry that will not be reset on POR or reset command.

B. Flip-flops with both sets and clears, indicating possible asynchronous design techniques (the absence of set/clear flip-flops does not indicate the absence of asynchronous design techniques).

C. Unintended latches. For intended latches, the timing has to be checked manually.

D. Opposite-edge flip-flops (e.g., falling edge flip-flops in a predominantly rising edge design) that could place constraints on clock symmetry and be more difficult to analyze with the STA tool.

(2) Search for high fan-out resources. These resources may need to be distributed via spare global clock network or other high fan-out/low-skew buffers.

(3) Review and verify that the synthesis tool properly encoded all FSMs.

(4) Verify that de-metastability buffers are not replicated.

d. Understand all warnings and notes. Disposition all warnings.

e. Timing analysis in the synthesis phase can be used as an indication of long timing paths and of predicted timing. However, accurate timing analysis can only be achieved from a STA that is performed after PnR.

8.4.1.2 Register Replication

Synchronizing circuits for asynchronous inputs and CDCs are checked to ensure key flip-flops in the source and destination clock domains have not been replicated. Replicated flip-flops in synchronizers can result in intermittent functional failures. Synthesis constraints are added to these flip-flops to prevent replications and preserve the structural design of each synchronizer.

8.4.1.3 Transfer Netlist to Vendor-Specific Place and Route Tool

To transfer a netlist to a vendor specific PnR tool consider the following:

a. Set timing constraints. Document and archive constraints files for reproducibility and review.

(1) Ensure that false paths/multi-cycle paths are valid.

(2) Consider timing requirements for devices that interface with the PLD.

b. Set up the PnR tool for:

(1) Part type:

A. Package.

B. Speed grade.

(2) Temp range:

A. Military range is suggested to ensure sufficient timing margin.

B. The tools assume that temperature is the device junction temperature and not the case temperature of the board's thermal control surfaces.

(3) Voltages (core, I/O).

(4) Radiation level.

(5) Device-specific settings.

c. Fix pin assignments and properties:

(1) I/O standard, threshold, slew rate, output loading, etc.

(2) Verify I/O compatibility with the board designer.

(3) It is preferred to accomplish this with a vendor-specific constraints or configuration file.

d. Run PnR.

e. Export min-typ-max Standard Delay Format (SDF) files for simulation:

(1) Min/max delays are contained within this file, ranging from the best case to the worst case.

(2) Use loading for each pin by reviewing schematics and specifications for each interfacing part.

8.4.1.4 Post-Route Verification

In PnR verification, consider the following:

a. Review all logs from vendor tools for errors, warnings, and notes. Disposition all warnings.

b. Timing analysis:

(1) Use the vendor's STA tool or an approved independent STA tool that can perform an equivalent analysis using SDF files generated by the vendor tool.

(2) Check maximum timing with procured speed grade.

(3) Check minimum timing using the fastest speed grade, in case the PLD vendor delivers a faster PLD (from their current inventory).

(4) Include delays to/from pads on the board.

(5) Consider clock source and delays.

(6) Include loading on outputs.

(7) Get min/max data for any device interfacing with the PLD.

(8) Analyze both best case and worst case timing to reveal any setup/hold time violations.

(9) A minimum 10 percent timing margin is suggested.

(10) Carefully scrutinize logic crossing clock domains, and the symmetry requirements of clocks when both edges are used.

(11) Slew rates (especially clock slew).

c. Review and verify pin report assignments.

d. Run back-annotated simulations (minimum, typical, and maximum). Verify that timing and functionality are both met.

8.4.2 Timing Analysis

Timing analyses showing positive margins are performed to ensure the PLD will operate properly through all environmental conditions it may experience. PLD timing requirements are derived from all of the PLD's interface components to ensure timing compatibility. Constraints are applied as needed to the synthesis and/or PnR software to meet all timing requirements. The environmental factors that affect timing are as follows:

a. Temperature.

b. Voltage.

c. Radiation—total ionizing dose (TID).

d. Process (speed grade, best/worst).

e. Aging.

PLD vendors try to provide timing estimates, which are based on user-specified environmental values or ranges, that envelope the effects of all of these factors. The effects of some factors (such as radiation and aging) are harder to predict than others. Therefore, positive timing margins (margin on top of margin), such as 20 percent at initial PnR and 10 percent for the final, are preferred. For example, prolonged radiation exposure can slow down some circuit elements and/or speed up others within the same device. In addition, logic upsets due to transient radiation events are addressed, but not as part of timing analyses.

Timing analyses may need to be run with more than one set of environmental conditions and resolved into an overall set of minimum and maximum values that envelope the values from various environmental conditions. The number of environmental condition combinations analyzed is often referred to as the number of corners analyzed. An example of a two-corner analysis is shown in the table 3, Example of Two-Corner Analysis, below.

Table 3–Example of Two-Corner Analysis

Environmental factor	Condition	
	Worst/Max	Best/Min
Temperature	125 °C	-55 °C
Voltage	3.0V and 1.35V	3.6V and 1.65V
Radiation	100 krads	0 krads
Process (speed grade)	STD	-1
Process (best/worst)	Worst	Best

The types of timing analyses needed depend on the functional types of inputs, outputs, and internally generated signals in the PLD. The types of timing analyses are as follows:

a. Clock frequency.

b. Reset pulse widths.

c. Asynchronous input pulse widths.

d. Synchronizer metastability settling time.

e. Synchronous input setup and hold times.

f. Combinatorial input pulse widths.

g. Synchronous output delays.

h. Combinatorial output delays.

i. Clock domain crossings.

Typically, several types of timing analyses are needed to show that a PLD design will perform properly. For example, a PLD design with a peripheral component interconnect (PCI) interface would need clock frequency, synchronous input setup and hold times, and synchronous clock-to-output timing analyses to show compliance to PCI interface timing requirements.

8.4.2.1 Clock Frequency Timing Analysis

STA software is used to calculate the maximum clock frequency for each clock domain that uses dedicated, skew-safe clock circuitry. This is typically the first timing check designers perform to see if their design implementation works at the required clock frequency(ies). Specify a frequency for each clock domain in the PLD to constrain the PnR software so that the required frequency(ies) is achieved. Specifying frequency constraints enables the STA software to report pass/fail status.

For clock domains that pass signals between both edges of the clock, account for duty cycle variations. Note that rising versus falling edge delay differences inside the PLD contribute to duty cycle variations.

If the maximum clock frequency reported by the STA software is not fast enough, review the slowest paths and consider using a faster speed grade PLD, redesigning the logic and/or determining if multi-cycle clock constraints can be applied. Slow paths sometimes consist of signals that functionally have two or more clocks cycles to settle. For these paths, consider adding multi-cycle timing constraints.

Avoid clock domains that do not use dedicated skew-safe clock circuitry because using them may require more manual effort (analysis and/or structural design modifications) to preclude internal setup and hold timing violations.

8.4.2.2 Reset Pulse Width Timing Analysis

Analyze resets used to drive asynchronous clear or preset inputs to sequential elements (typically flip-flops) to meet the minimum low and high pulse widths needed by the routed PLD.

Resets that assert and negate asynchronously to the PLD’s clock domain(s), such as power-on resets, are often synchronized before being distributed throughout the PLD. Resets are often asynchronously asserted and synchronously negated, so that all sequential elements are initialized as soon as possible but released from reset on the same clock event. Analyze synchronized resets, as well as synchronously generated resets, like synchronous inputs to

sequential elements even though these drive the asynchronous inputs. This will sufficiently negate the synchronized reset before the next clock event so that all sequential elements come out of reset together.

8.4.2.3 Asynchronous Input Pulse Width Timing Analysis

Analyze asynchronous inputs that synchronize to clock domains inside the PLD to ensure the minimum low and high pulse widths needed by the synchronizing circuit(s) in the routed PLD are less than the minimum low and high pulse widths of the input.

The minimum low and high pulse width of a “standard” two flip-flip synchronizer (shift register) is not just its clock period. The minimum low and high pulse width is the sum of the flip-flip’s clock period, setup time and hold time and the rise versus fall time variances of the input’s path to the first flip-flop.

Some asynchronous inputs are gated by other inputs or signals in the PLD and do not need to be synchronized on every clock edge. It may be better to design and analyze these as gated synchronous inputs.

8.4.2.4 Synchronizer Metastability Settling Time Timing Analysis

Flip-flops used to capture asynchronous signals may have longer clock-to-output settling times when setup or hold times are violated due to metastability settling in the flip-flop. Although it is possible to have a flip-flop’s output oscillate indefinitely due to metastability, it is unlikely the feedback paths in the flip-flop will be perfectly matched to sustain this.

The typical synchronizing circuit consists of two or three flip-flops connected as a shift register. For each synchronizer, ensure the path from the first flip-flop to the second has sufficient timing margin (slack) to account for metastability settling. Having margin is typically not an issue if there is no combinatorial logic between these flip-flops and these flip-flops are usually placed close together. It may be worth adding a maximum path delay constraint for each synchronizer from the first flip-flop to the second that is less than the clock period to help ensure margin for metastability settling.

8.4.2.5 Synchronous Input Setup and Hold Time Timing Analysis

STA software can be used to calculate the minimum setup and hold times needed for each synchronous input pin. Setup and hold constraints can be specified to guide the PnR software and enable the STA software to report pass/fail status.

Some synchronous inputs are gated by other inputs or signals in the PLD and do not need to meet setup and hold timing on every clock edge. For these types of inputs, it may be worthwhile to constrain the setup and/or hold times to values larger than the clock period, as appropriate. This is analogous to multi-cycle timing constraints.

8.4.2.6 Combinatorial Input Pulse Width Timing Analysis

Analyze combinatorial inputs used to drive outputs without being clocked to ensure the rise versus fall time variances through the combinatorial logic do not exceed the input's minimum pulse widths. Otherwise the input pulse could be missed and not reach the output or significantly distort the output pulse shape. This could cause glitches that produce unwanted outputs. It is recommended that outputs be clocked.

8.4.2.7 Synchronous Output Delay Timing Analysis

For designs where the output delay relative to the clock matters, STA software can be used to calculate the minimum and maximum clock-to-output delays. Minimum and maximum clock-to-output delay constraints can be specified to guide the place-and-route software and enable the STA software to report pass/fail status.

For designs where the output delay relative to the clock does not matter, this is documented and no constraints are applied.

A best practice is to register outputs through an I/O register. This produces more predictable output timing and delays.

8.4.2.8 Combinatorial Output Delay Timing Analysis

For designs where the output delay relative to its input(s) matters, STA software is used to calculate the minimum and maximum input to output delays. Constraints for minimum and maximum input to output delays are specified to guide the PnR software and enable the STA software to report pass/fail status.

For designs where the output delay relative to its inputs does not matter, this is documented and no constraints are applied.

8.4.2.9 Clock Domain Crossing Timing Analysis

The timing analysis shows that each CDC works properly and includes dispositions for each signal involved in the CDC. The clock domains have either fixed or (more often) varying clock edge timing relationships.

Synchronous CDCs may be able to take advantage of guaranteed timing relationships between the source and destination clock domains and use STA to verify setup and hold times for their control signals.

Asynchronous CDCs have to work with all timing relationships (leading, lagging and co-incident) between the source and destination clock edges. Asynchronous CDC control signals are analyzed using asynchronous pulse width timing analysis.

Setup and hold timing analysis are used for the data signals of CDCs. More specifically, the analysis shows that data presented on the appropriate source clock edge meets the setup and hold

times needed by the appropriate destination clock edge. CDCs may be designed to hold data for multiple clocks before and/or after control signal transitions to facilitate meeting these setup and hold times.

8.5 Baseline Work Products

Design is baselined in the appropriate configuration control system. This includes design source files, simulation test benches, script files used during simulation, synthesis, and PnR. The goal is to keep everything required for design implementation and verification under configuration control so that the design can be reliably reproduced.

8.6 Generate the Programming File

This phase includes generating the PLD design per the design. The design may not be the complete design but may be the completed section(s) per the plan. It includes the process to synthesize, PnR, and generate the programming files for the design.

Prior to generating the program file, the following occurs:

a. Design implemented in HDL is baselined.

b. Simulations are completed.

c. Reviews per the development plan are completed.

8.6.1 Programming File(s) Build Considerations

A build procedure is used to document the steps involved with generating the programming files(s) to ensure that versions of all source files and tools are identified. If required, it can serve as an as-run procedure. In addition, the procedure can contain SMA witness or acceptance. The procedure serves as a standard method to generate the programming file(s) that maintain a detailed list of what source files are used. By tracking the versions of the source files, changes and modifications are easily tracked through revision control. The procedure may contain the following information:

a. A mechanism to get all source files from the revision control repository. This can be either a procedure to check out the latest/specified version or a script to perform the operation.

b. When using a script, it is a best practice to record the version number of each file in a text file. This maintains the name and version of each of the source files.

c. The detailed information about the software used to perform the build (such as version).

d. Location for the repository where the programming file and released EIDP are stored.

e. An executable procedure (where special considerations are required) that describes how to conduct the build and how to verify it was successful.

f. Identify the need for involvement, including witnessing of operations, by appropriate quality assurance (QA) personnel as required by the project.

g. Record of the checksum(s)/CRC(s) of the program file(s).

h. How to disposition any anomalies in the build.

i. Procedure for baselining or archiving the as-run programming procedure and data.

8.6.2 Implementing the Design

Implementing the design is the process by which the build (or implementation) procedure is used to generate the programming file(s) that are required. A separate build procedure is used for each PLD that is generated. When implementing the design, the design files are first to be placed in the appropriate CM system. The procedure is then run with the appropriate level of SMA required by the project. The resulting programming files are then put in the appropriate revision control repository.

8.7 Exit Criteria

Upon completion of the implementation phase, the following criteria (from the development plan) are to be met:

a. The design implementation is complete.

b. Reviews are completed as defined in the development plan:

(1) Back-annotated simulation and regression simulation suites were performed for formal design releases.

(2) Worst-case analysis of the design was performed to ensure sufficient internal and external I/O timing margins.

c. All build files that were needed to generate the implementation file for programming the PLD should be complete. These include, but are not limited to, the following:

(1) Timing Constraints.

(2) Synthesis.

(3) PnR.

(4) Timing Analysis.

(5) Programming File.

d. All files are baselined in the appropriate revision control system.

9. VERIFICATION

The verification of the design is the process where the design is checked to make sure that it meets the specified requirements. Verification is generally a formal process in which verifications are monitored by QA as they are being performed. See section 5 for a description of verification planning.

The verification begins with a discussion of the test process and test procedures. Test procedures have to be completed prior to verification, but are described here because they logically fall into the verification phase.

9.1 Entry Criteria

Prior to beginning the verification process, complete the following procedures:

a. All test procedures and code used in the verification are to be maintained in version control.

b. A successful build of the device is maintained under version control.

c. A programming procedure is baselined.

9.2 Test Process

The test process involves using the baselined test procedures and SMA (when applicable) to verify the requirements.

9.2.1 Test Procedure

A test procedure is a document that delineates the test steps required to verify full compliance of the PLD implementation to the requirements. It is to be written clearly, so that it can be handed to an independent tester. The test procedure may include topics such as:

a. Test identifier (name or number to uniquely identify each test, for example, the universal asynchronous receiver/transmitter (UART) test).

b. Requirements addressed by the test case.

c. Test configuration(s) (recorded information about the test setup, such as software version number, PLD version number, etc.).

d. Prerequisite conditions (what needs to be completed prior to the test, e.g., ESD certification).

e. Test input (things required as input to the test, e.g., test scripts).

f. Instructions for conducting the test.

g. Assumptions, constraints, and expected test results, including pass/fail criteria for evaluating results.

9.2.2 Quality Assurance Provisions

The use of QA in the test process depends on the project requirements.

a. For all projects designated as safety-critical or flight projects, the programming and test of the programmable device is witnessed or monitored by QA personnel, in accordance with the project documentation.

b. For non-flight projects, the project may choose to provide independent personnel to monitor the activities, filling the role normally filled by QA for flight projects. Any discrepancies encountered are handled according to the project error tracking process. Section 10.1 describes the programming test procedure in detail.

9.3 Simulation

Simulations can be used to unit test functionality of the PLD design prior to formal testing on the board. The amount of testing required is dependent on the functionality. Simulation for verification may not always be appropriate. For example, the simulation of the digital interface of an analog-to-digital (A/D) converter does not verify the operation of the A/D converter. The verification of the A/D conversion circuit can only take place on the hardware. However, it is advantageous to simulate the operation to ensure that the appropriate signals are operating as expected.

Simulation for formal verification can be used for a number of functionalities. Examples of these would include:

a. Off-nominal operation.

b. Hardware failures.

c. Basic command operation.

d. Internal operation of the PLD.

It is a best practice to simulate all functionality of the PLD prior to running on the actual hardware even if formal verification is not gained.

When using simulation for formal verification, a self-checking test bench is the best method. The test bench that is used for formal verification is reviewed during the review process. The test bench review is set up to ensure appropriate operation and requirements verification.

9.3.1 Test Bench Development

Test benches are developed in order to facilitate simulation testing. The test benches generate the test vectors that are used to drive the circuit being simulated.

When developing test benches, an independent PLD test bench developer is preferred, but is not mandatory. Observe the following guidelines:

a. Assign a test number to the test cases for tracking purposes.

b. Use self-checking/documenting test benches.

c. Analyze code coverage of simulation and test vectors.

d. Automate tests using scripts for repeatability and unattended runs.

e. Place test bench components under revision control.

Check the simulation results for correctness. The following items help in the review of the results:

a. Review waveforms for internal and external signals for proper functionality.

b. Compare simulation results with requirements, especially interfaces. This can include collaborating with other team members to ensure compliance of interfaces.

c. Disposition all warnings and errors reported by simulation tools. Document the dispositions.

9.3.2 Modeling of External Components

External components on the circuit board interfacing to the PLD are modeled. While there may be others, some examples of models are as follows:

a. Electrically EEPROM flash memory.

b. 1553 chip.

c. Memory.

d. PCI.

Specific features to include in a model would be the following:

a. Timing from device data sheets (setup and hold checking).

b. Handshake protocols.

c. Error checking.

d. Fault injection.

It is a best practice to obtain models from the vendor so that the appropriate delays are correct. Some simulation tools come with models. Check with the tool vendor for their models.

9.3.3 Simulation Results

The results of the simulation are reviewed for correct operation. The tester reviews the following:

a. Waveforms for sanity check.

b. I/O waveform/timing with the board design lead or other PLD designers as appropriate.

c. All warnings and errors reported by simulator:

(1) Understand why they are there.

(2) Document any decision to ignore them.

9.3.4 Code Coverage

When textual design entry method is used, code coverage is used to assess the thoroughness of the simulation cases. Code coverage analysis is performed for statement coverage, toggle coverage, branch coverage, and expression coverage. All of these coverage analyses can uncover deficiencies in the design simulation. Coverage analysis is used frequently while implementing the design to assess the completeness of the simulation test benches.

Code coverage analysis of the simulation test benches is performed for each formal release. Simulation code coverage analysis provides a useful measure on the thoroughness of the simulation test benches. A design is only released after it has been thoroughly simulated. Though it is desirable to have a high simulation coverage, setting up a hard coverage requirement is not practical. Achieving high simulation coverage can sometimes be very laborious and time consuming and not be supported by the project schedule.

Instead of imposing requirements on the code coverage level, the designers disposition all the coverage deficiencies flagged by the code coverage tool. Reviewing coverage deficiencies helps the designer understand the cause of the deficiencies and determine how they need to be addressed. In the disposition, the designer documents the rationale for the coverage deficiency and risk associated with not addressing this deficiency. The dispositions are released along with the coverage report as part of the final EIDP of the PLD design. The disposition for the coverage deficiencies and the coverage report provide a metric for the quality of the design simulation. They are treated as part of the design documentation and released along with the design.

9.3.5 Back-Annotated Simulations

Back-annotated simulation allows the designer to simulate the synthesized, PnR design using the timing delay models extracted for the worst-case and best-case operating conditions. The designer bases the design simulated during the back-annotated simulation on the final HDL net list produced by the PnR tool and models of the logic primitives specific to the targeted PLD device. The back-annotated simulation is performed using maximum, typical, and minimum

timing models. PLD design is verified with back-annotated simulation before each formal release. By running a post PnR structural simulation with unit delay (without the timing information), the designer can verify that the design is accurate. The value of a unit delay post PnR structural simulation is that it may execute significantly faster.

Passing the back-annotated simulation proves that the design is not altered by the synthesis and PnR process. It also validates the timing constraints used during the synthesis and PnR process when the back-annotated simulation is performed using the best and worst timing delay models.

Since running back-annotated simulation can be very time consuming, the designer may not want to re-run every test in back-annotated mode. The designer needs to decide which parts of the simulation need to be re-run using back-annotated design.

9.3.6 Defect and Anomaly Reporting

If a failure occurs during testing, the designer investigates the failure. This investigation needs to include root-cause determination, which is necessary to ensure that a proposed solution resolves the failure. The designer may need to generate a new PLD version and perform regression testing. The amount of regression testing depends on the level of functionality where the failure occurred. Automated, self-checking test benches will help simplify regression testing. The verification plan clearly defines the details and corrective actions of test failures.

The designer documents any anomalies that occur during formal testing (qualification and production acceptance).

The designer may initiate replacement or reprogramming of a device due to a test failure. The designer will establish a new design baseline prior to programming a configuration-controlled device with a new design. Section 10.1 discusses programming practices.

9.4 Independent Peer Verification

Independent verification is an approach used to provide higher confidence in the outcome of the verification process, by avoiding the inadvertent masking of design errors that can sometimes occur if the same personnel are responsible for both design and the design verification. A classic example is the misinterpretation of a requirement: If both are designed, tested, and analyzed by the same person with the same misinterpretation, the design can pass the verification, but still not meet the requirement. The use of independent personnel for verification is highly recommended in cases where sufficient personnel, cost, and schedule are available. There is additional overhead for maintaining separate personnel for design and verification that may be prohibitive for some projects; however, without this measure, those projects will have to decide whether to accept an increased risk.

As with all verification activities, independent verification can occur at any point throughout the life cycle, including simulation, analysis, developmental tests (e.g., “breadboard,” “engineering model,” etc.), or integration tests with software or higher level systems. The design team may

also utilize independent verification for items such as IP or other non-developmental items used as part of their design approach.

In those cases where the outcome of independent verification is a required step in the developer's process, the design team has to identify those constraints early in the development to allow for adequate planning and schedule.

9.4.1 Functional Coverage

The verification team verifies the design team's work products from all phases of the life cycle against the same set of requirements used by the design team. In order for an independent team to begin verification, they have to use the same set of requirements used by the design team. In addition, the verification team has to have access to all the design products (e.g., HDL, constraint files, etc.), as well as documentation from the design team, including traceability reports, full descriptions of any derived requirements, or design features that were incorporated. For testing, the independent team generates test cases and procedures per the verification plan and verifies the functionality based on the requirements.

For simulation, the independent team may utilize test benches provided by the developers, primarily as a way to begin to understand how the design operates but also generates its own test benches to use.

For formal verification activities, the independent team:

a. Utilizes revision-controlled design products (e.g., VHDL, design description documents, traceability reports, etc.).

b. Controls all verification products (such as test procedures, scripts, etc.) in accordance with the projects revision control requirements.

c. Documents any issues per the bug tracking tools and issues formal verification reports.

d. Provides coverage reports as an output of the verification process.

9.4.2 Communication (Independent Peer Verification)

Communication between the design and verification teams is of paramount importance for independent peer verification. The term "independent" is not to be interpreted as a ban to communication, including (where appropriate) direct contact and discussion about the design, and any verification issues found. A successful outcome is achieved through a thorough understanding of the work of each team.

Traditionally, design activities and independent verification activities could be divided into separate phases, where the design team handed off the design to the verification team, and then a verification report was provided back to the design team. The design team could then address any required changes and iterate through the verification team until all issues were resolved. In

this scenario, written documentation, such as design and version description documents and verification reports (see section 9.5), are the primary formal means of communication.

However, a more iterative model utilizes ongoing development (including correction of issues previously found by verification) concurrently with verification activities. Verification activities can span multiple iterations of the design cycle that leads ultimately to the final product. This can result in multiple versions, with differing issues, and different iterations of changes. In this scenario, communication is of even greater importance. Ultimately, both the design and verification teams have to converge upon one final design supported by successful verification results.

A “bug tracker” tool (usually electronic) is one valuable way of achieving communication between the design and verification teams, particularly in cases where the two activities are active in parallel. The bug tracker provides not only communication, but assists in traceability of changes, and development of the Version Description Document (VDD). See section 5.2.8 for additional information about bug tracker tools.

9.5 Verification Review

The developer and/or an independent party, as defined in the development plan, can perform the verification review. The verification process ensures that the PLD meets all the requirements for development and design of the project.

The verification team reviews the PLD against the known criteria (usually in the form of a checklist and a traceability matrix) and issues a verification report.

The function coverage check involves the tractability matrix.

During the verification review, the developer and/or independent reviewer reviews the tractability matrix to ensure that the testing covered all functions.

The verification report provides an overview of the verification results. It includes the following items:

a. Overall success/failure evaluation of the verification.

b. Any known deficiencies, limitations, or constraints.

c. Any results of any corrective actions that were assigned during the verification process.

d. Impacts of any items listed.

e. Verification environment including impacts.

f. Verification results:

   (1) Project unique identifier of test and procedure.

   (2) Details of results with traceability.

(3) Problems/anomalies encountered.

(4) Deviations from test cases.

g. Verification log:

(1) Date and time of the verification.

(2) Verification environment, hardware and software configuration.

(3) Identification of individuals who performed the verification.

A verification report does not necessarily mean a separate document. The report can be an as-run test procedure, provided the team records all pertinent information in the test procedure.

9.6 Regression Testing

Regression testing is required for changes made to verified code. During the test phase, a regression plan is to be developed that will indicate what testing needs to occur as changes are made. A basic form of a regression plan can be a regression matrix, as seen in table 4, Regression Testing, below.

Table 4—Regression Testing

	Test Procedure 1	Test Procedure 2	Test Bench 1	Test Bench 2
Function 1		X	X	
Function 2	X	X		
Function 3				X

The table above shows that a change in a particular function requires a re-test of the appropriate test procedures or test benches.

9.7 Hardware Verification

It is desirable to have the test environment for hardware verification resemble the final configuration to achieve a higher level of confidence in the test results. This can include the use of hardware emulators, simulation software, flight hardware, qualification hardware, certification hardware, certification software, and flight software.

Devices used for the final deliverable article are preferred for the final verification of a PLD design. For cases in which the final deliverable is a one-time programmable PLD and there are budget and schedule constraints, verification may be performed on a prototype device prior to the final deliverable. In these cases, it is recommended that the prototype device closely resemble the final part.

In addition to the choice of prototyping devices, it is also important to perform hardware verification using high fidelity test interfaces and stimuli. If simulators produce the test stimuli, it is important to have the behavior of the simulator verified against the final interfaces and document any non-compliance.

When the PLD design contains a software interface, it is recommended that the PLD be tested with the final software, such as the flight software. Though test software is sometimes sufficient for verifying the PLD software interface, it typically does not operate the PLD the same way as the final software. Test results obtained using test software have lower fidelity than those from testing using the final software.

9.8 Exit Criteria

Projects that have multiple releases (i.e., projects that have re-programmable flight units or projects with engineering releases) iterate through the verification process.

Verification is complete when the following items are complete:

a. All as-run test procedures and test reports are released.

b. All review documents are released.

c. All documents that have changed are released.

d. EIDP is released.

e. Anomalies are documented and dispositioned.

f. All requirements for this release are verified

10. DELIVERY

Once the verification team completes the verification of the PLD, the design is delivered for board or assembly level testing. Delivery of the device requires that related design documentation and data products be completed and released under CM control. This includes, but is not limited to, items identified as follows:

a. Design Specification: The version that describes the as-built design.

b. As-built Design Files: The set of files that enables other designers to troubleshoot the design or to recreate the design when needed. Following is a list of as-built design files released when the design is delivered:

(1) Pre-Programmed Verification Matrix: Record that shows how the design was verified prior to programming.

(2) Waivers and Problem Reports: Waivers and problem reports generated against the delivered design.

(3) Design Source Files: Includes all the source code for the as-built design, the test-bench source files, and the simulation models.

(4) Synthesis Project and Script Files: Includes all the files the synthesis tool uses to produce the as-built synthesis net list.

(5) Synthesis Constraints File: Includes both the timing and physical constraints files for the as-built design.

(6) Synthesis Transcript/Log File: Includes the synthesis transcript/log file for the as-built design. Additional documentation is provided for the disposition of all the synthesis warnings.

(7) PnR Project and Script Files: Includes all the design database and binary files produced during the PnR process. The goal is to have all the files necessary to regenerate the fuse/configuration/programming file when needed.

(8) PnR Constraints File: Provides both the timing and physical constraints files for the as-built design.

(9) PnR Transcript/Log File: Includes the PnR transcript/log file generated for the as-built design. Additional documentation is included for the disposition of the PnR warnings.

(10) PLD Programming File: The "fuse/configuration/programming" file for the design. Necessary precautions are required to preserve the integrity of the programming file when transferring the programming file from one computer platform to another.

(11) Design Review Reports: Includes review reports for all design reviews held for the delivered design and evidence supporting the closure of the action items from those reviews.

(12) PLD Design Drawing: The drawing that documents the PLD vendor information, PLD vendor part number, tool used for developing the design (including licensing information), and the revision and/or the last modification date of all the files listed above. All the design files referenced in this drawing are released under this drawing number as supporting documents.

(13) Revision Description Document (RDD): After the initial release, tracks the revision history of the design.

(14) Post-Programmed Verification Matrix: The verification record for any testing done on the programmed device.

c. Verification Plan.

d. Other Documentation (such as a drawing or readme file): Contains the following information:

(1) Location of the repository for related design analyses.

(2) Location of the repository for the design review reports and action items.

e. The RDD contains, at a minimum, the following information:

(1) The version and last modified date of the released files.

(2) The revision control version(s).

(3) All changes made since the previous releases need to contain bug numbers from the bug tracking system.

(4) Data integrity checks (checksums).

(5) Workarounds for any issues.

In addition to the above files and documents, the team also releases a drawing or equivalent documentation along with the design to document how to label the programmed device and how it can be clearly linked to its source design files.

10.1 PLD Programming

A programming and, when required, post-programming test procedure is developed for each design. The design team can capture the programming procedure in a document or in an altered item drawing (AID). As a minimum, the procedure contains the following information:

a. Name, version, and checksum information of the programming file.

b. Location for the repository where the programming file and released EIDP are stored.

c. An executable procedure (where special considerations are required), such as installing the device into the programming hardware, loading the design files into the programmer, configuring the programmer for the specific device, ensuring appropriate ESD, programming the device, etc.

d. Identify the need for involvement, including witnessing of operations, by appropriate QA personnel, as required by the project.

e. Verify the expected checksum against that reported by the programmer based on the fuse information loaded from the programming file. Verify the expected checksum against that reported by the programmer based on the fuse information read back from the programmed device.

f. How the programmed device needs to be marked and labeled. This can be a reference to the AID or equivalent document.

g. Post-programming test requirements and procedure, when applicable.

h. A minimum post-programming requirement would be to verify the checksum of the program once the device is programmed.

i. How to disposition devices rejected by the programmer or not passing the post-programming test.

j. Procedure for archiving the as-run programming procedure and data

10.2 PLD Marking

Once programmed, a PLD becomes an application-specific part. The manufacturer's part number no longer identifies it. The design team assigns a unique identifier or a project part number to the programmed device. Typically, instructions for programming and marking a programmed PLD are captured in a design-specific AID and the drawing number, serial number, and revision information for the AID becomes the part number marked on the programmed PLD.

No PLD is programmed until the design is released under CM control. This includes all the data products required by the EIDP, such as the design source files, synthesis, and PnR transcripts, and, most importantly, the programming files. In addition, the team also releases an AID or equivalent document to link the EIDP revision information and design specific information, such as fuse checksum or design date code, to the marking or part number of the programmed device.

It is very helpful if the PLD is designed with a software readable identifier or label. This is usually implemented by storing the identifier or label, including the design revision information, in a software accessible register. When such an implementation exists, it is also important to document software accessible identification in the AID or equivalent document.

11. MAINTENANCE

PLD maintenance considerations are different depending on the design's place in its life cycle. This section describes designs as either active or retired. The determination of the category of a design is based on the life cycle plan developed by the delivering organization. The rules used to make this determination are based on factors like the delivering organization's lifetime guarantee on the design and whether an active project is currently using the design.

11.1 Design Libraries

It is recommended that the delivering organization establish a design library (revision control or CM) to store designs. Maintaining an active design in the library is a job focused on protecting the pedigree of the design from the modifications performed by projects. The delivering organization is responsible for ensuring that the design has been verified with the appropriate level of rigor before submitting the design to the library.

It is recommended that projects refrain from inheriting designs directly from each other and instead go through the design library. This ensures that the organization can play its role in protecting the pedigree of the design. Direct project-to-project inheritance of designs bypasses this important role of the organization.

The delivering organization is typically not able to make design changes without project funding. Therefore, track the desired design changes with designs in the library. They can be implemented and verified when project funding becomes available and the new revision of the design can be accepted back into the library.

11.2 Active Designs

Active designs are those that the organization has released under configuration control and used on active projects. Other projects have to be able to retrieve the design from a library under CM and immediately put to use. This includes the following:

a. Its specification is kept up to date.

b. Problems reported against the design have been documented.

c. Its tool chain is maintained in working order.

11.3 Post-Delivery Anomalies

Even after a thorough verification process, customers using the designed part may report anomalies and changes to the requirements. When this occurs, there are two options: Fix the anomaly, or use the design as-is. Fixing an anomaly may be the preferred solution. However, the complexity associated with making a change involves a risk trade.

Making a change to a design reintroduces that design into the design process and sets it back to an appropriate design phase. The design modification has to progress through the same processes with the same rigor until a new final design can be delivered to the customer. Along with the design process, the project team considers schedule and budget effects.

Design problems considered to be use-as-is are documented as a design idiosyncrasy in the appropriate level of documentation. Capturing the idiosyncrasy preserves intent behind the design and provides a path for product improvement. At a minimum, the design specification is updated to document the as-built behavior and the idiosyncrasy is clearly identified.

11.4 Retired Designs

Eventually the project designates a design as no longer active, and it is retired. The requirements for active designs no longer apply, but the organization keeps all retired designs available for reference and study. As part of the life-cycle plan, it defines a format for storage of these designs. The selected format focuses on longevity because designs may be stored for a very long time. When a design is originally released as active, a copy of the design in this designated

format has to accompany it. This way, as the design ages and is considered retired, the design knowledge is stored in an electronic format which is backed-up.

12. RELEASE PROCESS

There can be multiple levels of the release process, initial, intermediate (or engineering), and final.

For the initial release, the developer is moving the PLD logic code into revision control for the first time. This includes information such as version, tools required, special instructions, etc.

The intermediate or engineering releases are interim releases that add functionality or correct issues. As the intermediate releases are saved, an active note of the changes with rationale for the changes are documented and maintained. Other items in the notes include issues found and testing completed. This allows multiple developers/users to understand the changes and a single developer to remember why the changes were made. The version number increases each time the release is changed and checked back into the revision control tool. A developer may group multiple changes into an intermediate release. Note: Some engineering releases may be used in preliminary hardware/software integration (HSI) testing. These releases are usually accompanied with the VDD or drawing.

The final release is the release provided to the customer or stakeholder as the final product. This release has completed verification as defined in the development plan and is accompanied with a released EIDP.

Some projects use multiple revision/configuration repositories. The development team may have a separate repository than the final project or Center repository.

13. OUT-OF-HOUSE CONSIDERATIONS

This section provides guidelines aimed at ensuring the quality of PLD designs from out-of-house contracts and procurements. These guidelines can be customized based on the criticality or class of the mission, complexity of the application, or maturity of the design (new, minor modifications, re-use).

13.1 Contract Statement of Work (SOW)

If procuring a board/system that includes PLD(s), contract SOWs can include the following:

a. The vendor provides NASA insight into the PLD design, development, and test activities, including monitoring verification adequacy; trade study data; auditing the PLD design and development process; and participation in PLD reviews and technical interchange meetings.

b. The vendor documents NASA’s insight into PLD activities/processes in the PLD development plan.

c. The vendor provides PLD design data package(s) for each PLD at point(s) defined in the PLD development plan.

d. NASA performs an independent assessment of all PLD designs against the project's specified design guidelines using the information in the PLD design data packages.

e. The vendor reviews all NASA PLD design findings and recommendations.

f. The vendor resolves all action items with NASA.

13.2 The Flight PLD Design Review Process

As illustrated in figure 4, Flight Project PLD Design Review Process, the optimum management of flight project PLD development efforts encompass five distinct phases, as follows:

1. Issue contract requirements for vendors to deliver a PLD development plan that describes the vendor's process for developing flight PLD applications, and deliver the PLD design data packages for each PLD, which contains all the required data elements to perform an independent review of the design.
2. Review and acceptance of the vendor's submitted PLD development plan.
3. Participate in agreed-to reviews to assess vendor's compliance to the approved PLD development plan.
4. Review and acceptance of the PLD design data.
5. Continue review and assessment of flight PLDs throughout the integration and test phases.

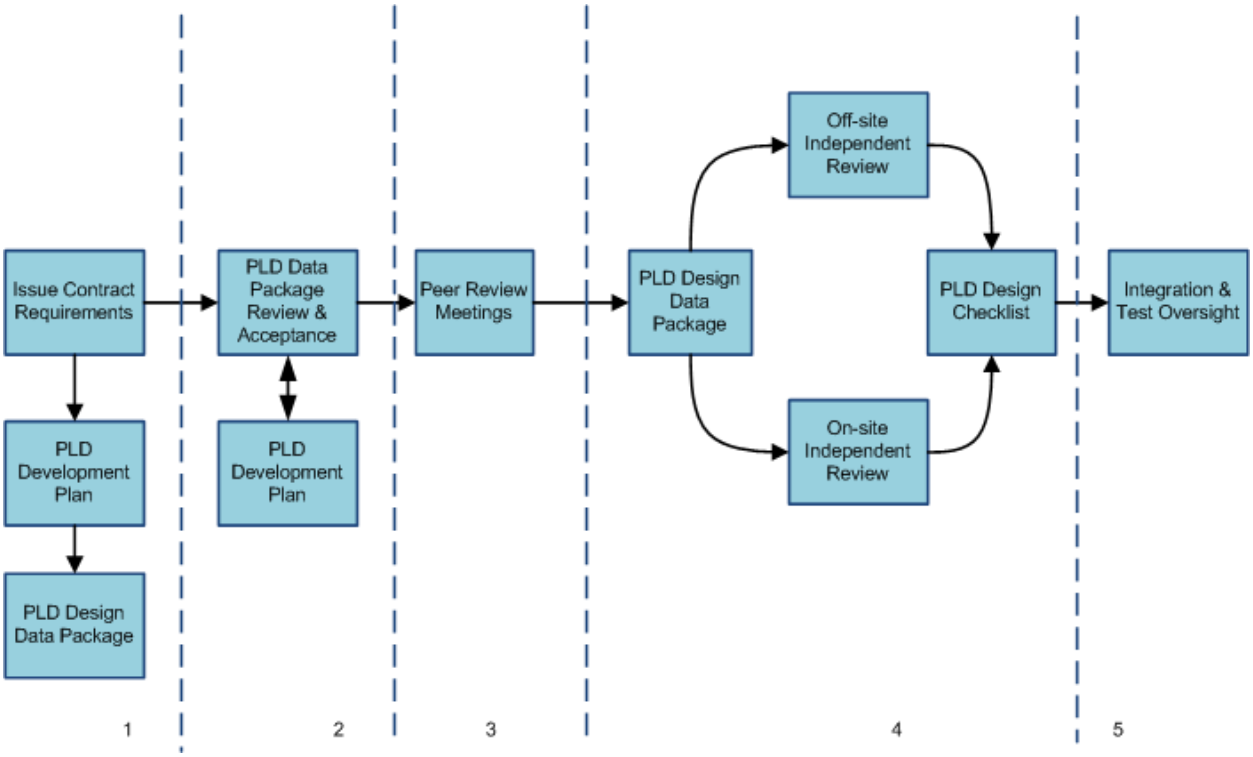

Figure 4—Flight Project PLD Design Review Process

The following sections detail each of the above phases.

13.2.1 Issuance of Contract Requirements

This initial step is to ensure that all vendors providing hardware for the instruments and spacecraft are contractually obligated to deliver a PLD development plan (see section 5.2) describing the vendor's methodologies for developing flight PLD solutions, and the complete set of artifacts that allow for an independent review of all flight PLD projects developed by the vendor. These two sets of requirements are depicted in the subsections that follow.

13.2.2 The PLD Design Data Package

A PLD design data package is required for each PLD design to allow independent NASA review and assessment of vendor PLD designs.

The PLD design data package is submitted for each PLD developed by the vendor for the Flight Project.

The PLD design data package includes the same information defined in section 10. The PLD design data package has to contain all source code and design files indicated in section 10. Be sure that purchased IP licensing agreements allow NASA to view all source code, have access to all design files, and remain valid during the entire maintenance cycle.

13.2.3 PLD Development Plan Review and Acceptance

Once the contract is in place, the vendor is required to submit its PLD development plan for NASA's review and approval.

Several iterations may be required until the final version of the vendor's PLD development plan is accepted by NASA. It is also possible that the plan is modified over the lifetime of the project to adjust to the reality of the system development. Every modification of the plan requires formal acceptance by NASA.

13.2.4 Vendor-Internal Peer Review Participation

In accordance with the vendor process described in the approved PLD development plan, NASA representatives may participate in peer review meetings. The PLD reviewer representing NASA is expected to receive the review materials (presentations, requirements, specifications, test descriptions, source code, etc.) at least a week before the scheduled date of the meeting. The reviewer reads all the material in preparation for the meeting. After the meeting, the reviewer prepares a summary report for the flight project management as a record of his/her participation in the meeting.

The PLD reviewer fulfills two roles in the peer review meetings. First, the reviewer provides expertise to support the review team in assessing the state of the project, making suggestions for corrections and/or improvements. Second, the reviewer assesses whether the vendor is following the process described in the approved PLD development plan.

13.2.5 Independent Review and Acceptance of PLD Design Data Package

At a point(s) established in the approved PLD development plan, the vendor supplies NASA with a PLD design data package. For simple revisions to PLDs, one review may be sufficient after PnR, timing analysis, and signal integrity analysis have been completed. Typically, a minimum of two reviews are warranted, one concurrent with the beginning of non-flight testing and one prior to commitment of the final design.

The two acceptable formats for the independent review are off-site independent review and on-site independent review.

13.2.5.1 Off-Site Independent Review

A PLD subject matter expert chosen by the project performs an independent assessment of the vendor submitted PLD design data.

The vendor reviews and assesses all NASA PLD design findings and recommendations. The vendor notifies NASA of those instances where corrective action was not taken on specific PLD design findings and recommendations.

It is recommended that the reviewer utilize a PLD design checklist to perform the analysis of the design (one suggestion is included in Appendix B). The checklist can also be provided to the vendors as a guide for them to prepare their designs for independent review.

The independent review is complete when all findings have been dispositioned/resolved or, if not, risks are identified and bounded. The reviewer submits a report to the appropriate flight project manager with all the responses to any findings and notifications and the final version of the checklist with comments and responses.

13.2.5.2 On-Site Independent Review Board

An alternative method to independently review a PLD design data package is to assemble an independent review board at the vendor's facility. A PLD subject matter expert chosen by the flight project leads the review board. The vendor provides the complete PLD design data package the least two weeks prior to the review board meeting.

The lead reviewer and other review board members assess the PLD design data package prior to the review board meeting. At the meeting, concerns and recommendations are discussed and action items are assigned. The lead reviewer is responsible for taking action items and follow-up on their resolution.

The vendor reviews and assesses all independent review board design findings and recommendations. The vendor notifies NASA of those instances where they decided not to take corrective action on specific PLD design findings and recommendations.

The work of the independent review board is complete when the lead reviewer is satisfied with the submitted design and submits a report to the project.

13.2.6 Integration and Verification Oversight

Design errors and/or manufacturing defects may appear during stress testing and/or integration phases, be it at the board, box, instrument or spacecraft levels. The PLD review team participates in delta reviews to resolve all technical problems.

APPENDIX A

SUGGESTED SOURCE CODE HEADER AND FOOTER TEMPLATE

Header:

```
//*****************************************************************************
// File Name:
//
// PLD Name:
//
// Project Name:
//
// HDL Standard:
//
//
// THE TECHNICAL DATA IN THIS DOCUMENT IS (OR IS NOT) CONTROLLED UNDER THE U.S.
// EXPORT REGULATIONS, RELEASE TO FOREIGN PERSONS MAY REQUIRE AN
// EXPORT AUTHORIZATION.
//
//
// Notes:
// This section includes what the module is intended to do and other
// related design information
//
// Change log is located at the end of the file
//
//*****************************************************************************
```

Footer:

```
//*****************************************************************************
//
// Change Log:
//
// Date:  mm/dd/yy  Name:  FirstName LastName
//
// Description:
// Use “Created” for “Description” for the initial release.
// The name field includes first name and last name.
//
//
// Date:  mm/dd/yy  Name:  FirstName LastName
//
// Description:
// Describe the changed made for the next revision
//
//
//*****************************************************************************
```

Rationale:

The information contained in the header stays static throughout the development, but the footer is expected to change over time. In order to avoid changing the line number of the code, it is recommended to include the change log in the footer.

APPENDIX B

SAMPLE PEER REVIEW CHECKLISTS

A PLD design checklist is a helpful tool to assist both the designer and subsequent reviewers. The checklist can be used during the design process to ensure all aspects are covered that can then be reviewed during design reviews. A sample design checklist follows. The files that may be needed in the design reviews are as follows:

a. PLD design specification.

b. PLD verification matrix.

c. Synthesis script/project file.

d. Synthesis constraints file.

e. Preliminary synthesis log/transcript file.

f. PnR script/project file/database file(s).

g. PnR constraints file.

h. PnR log/transcript file.

i. Worst-case timing analysis report.

j. Fuse or bit file.

k. Design source files.

l. Test bench files.

m. Board schematic drawing and net list.

n. Data sheet for components connected directly to the PLD.

o. Other script files used during design, simulation, synthesis, and PnR.

p. Action item status summary from earlier reviews (gating and peer reviews).

q. A readme file describing the simulation environment and setup to help reviewers rerun the simulation when needed.

B.1 Design Specification

	Review Item	Rationale	Reviewer Comments/Feedback
1.1	Is the design specification using required format? If not, list the deficiencies.	In order to facilitate the review process, it is desirable to have a template for the design specification.	
1.2	Is the design specification complete? Does it have sufficient details for design implementation and verification? If not, identify the deficiencies.	The design specification is complete enough to start a code walk-through. The peer review is delayed if the design specification is deemed immature. The design specification provides sufficient information about the design to allow the peer review reviewer to perform an effective review of the HDL source code.	
1.3	Check the design specification against the board schematic/netlist. List I/Os with internal and external pullups. Identify all the I/O standards used in this design. Are the internal and external terminations compatible? Do the pin assignments agree? Is the correct I/O standard used for the intended use?	Need to double check the interfaces between the PLD and the board. Need to ensure that the physical requirements, such as I/O standard, pin assignment and I/O type, imposed by the board designs, are met.	

B.2 Design Source Code

	Review Item	Rationale	Reviewer Comments/Feedback
2.1	Does each design module have required header/trailer (when applicable)? Is the design, HDL source code (or equivalent) well commented? Can one understand the flow of the design using only the comments? Is there an International Traffic in Arms Regulations (ITAR) warning included in all source files, if it is required on the project?	The code is to be well commented to make it readable and inheritable by others. See Appendix A for recommended header/trailer template.	
2.2	Is the HDL code (or equivalent) easy to follow? Identify improvements for better readability.	The code has to be readable to the reviewer to have a meaningful review. The statements need to be short and concise.	
2.3	Is there a consistent coding style in the source code? Is there a consistent signal naming convention?	Regardless of the coding style adopted, it is always easier to follow the code when the style is consistent. This is also true with signal names.	
2.4	Does the HDL source code contain excessively long lines? Is there proper indentation in the source file for readability? Are the indentations consistent (same number of spaces, no mixing tabs with spaces)?	Excessive long lines and inconsistent indentation can make the HDL source difficult to read.	

	Review Item	Rationale	Reviewer Comments/Feedback
2.5	Are there any recommended coding/implementation optimizations?	The designer can sometimes overlook simple things. The reviewer can sometimes spot design optimizations not seen by the designer. Reasonably optimized code can be reviewed more easily.	

B.3 Design

	Review Item	Rationale	Reviewer Comments/Feedback
3.1	Is the design matching its specification? Identify the inconsistencies if there are any.	The design specification and the design need to be consistent.	
3.2	Does the HDL use any constants/parameter in the design? Are there any constants/parameters requiring adjustments between different design instances (engineering model versus flight model)? If so, how are they adjusted to ensure the proper values assigned for a specific design instance?	If there are any parameters used for customizing the design for different design instances, an autonomous process, such as the use of scripts, is developed to ensure that the parameters are set correctly for the targeted application.	
3.3	Are there any global definitions/variables, and “ifdef’s” in the source files? If so, are they set up correctly for the intended applications?	Need to make sure that there is a reliable process to set up these variables correctly during the build process.	

	Review Item	Rationale	Reviewer Comments/Feedback
3.4	How many clock domains are there? How are the clocks buffered? What are their duty cycles and phase relationships? How many signals cross clock domains? Are the signals pulsed or leveled signals? How are the signals buffered/de-meta-stabilized, for both signals going from fast to slow and slow to fast clock domains? How are input signals buffered?	Clocks need to be buffered using global buffers. Need to pay more attention to signals crossing clock domains. Need to understand how the pulsed signals are de-metastabilized when the signals are going from a fast to a slow clock domain.	
3.5	How many resets are there? How are they buffered? How are they de-asserted? When the reset is used in multiple clock domains, is it de-asserted synchronously to the clock from the domain?	Need to use high fan-out buffers for resets. Need to make sure each reset used in the design is de-asserted synchronously.	
3.6	How many FSMs are in the design? Can the logic flow in the FSM be understood easily? Are all the FSM outputs registered?	The FSM code is clear enough for review. FSM outputs need to be registered to avoid potential glitches caused by state transition.	

	Review Item	Rationale	Reviewer Comments/Feedback
3.7	Are all the states used in the FSMs defined? Do the default conditions in the FSM direct the FSM to a valid safe state? Are there any states polling for status? If so, is there potential for them to become lock-up states?	Need to identify potential lock-up states and look for undefined states in the FSM. FSM designs contain a default condition. Need to ensure that the default conditions are capable of directing the FSM back to a safe state in the event of an SEU.	
3.8	Are there any vendor/device specific primitives used in the design? If so, why are they used?	Using vendor/device specific primitives makes the design less portable.	
3.9	Are the vendor SSO-related recommendations followed? If not, is there power integrity analysis data available to show acceptable noise level on the ground and power planes?	SSO causes ground bounce and reduce noise margin. Need to identify these signals in advance so that proper measure can be taken to alleviate its impact.	
3.10	Are there any derived or gated clocks? Are both edges of a clock used in the circuit? If so to either, how are they used? Are there any potential timing issues with the way they are used?	Derived and gated clocks tend to create asynchronous designs. It is sometimes hard to weed out timing problems with asynchronous designs. Need to pay extra attention to the STA when both edges of a clock are used in a design.	
3.11	Are there any potential race conditions? (This is hard to tell without deep design penetration.)	Race conditions cause the design to behave unpredictably.	

	Review Item	Rationale	Reviewer Comments/Feedback
3.12	Is there any register using only synchronous reset? If so, what is risk of having an undefined state for that register? For registers driving output signals, what is done to ensure a deterministic state for these output signals if synchronous reset is used?	Need to make sure that the outputs of the PLD have deterministic state during POR. Using synchronous reset can create brief uncertain in the outputs.	
3.13	Do all the “case” (or equivalent) structures have a default state? List all cases with a default state.	Need to make sure the case structure has all possible cases covered.	
3.14	Are there any IP or inherited design modules in the design? If so, are all inputs to the top level IP or inherited modules either connected to a signal source or terminated? Are all the unused outputs from the top level IP or inherited modules left unconnected?	IP or inherited design modules are expected to have unused functions, inputs and outputs. Unused inputs at the top level need to be properly terminated and unused outputs need to be left unconnected.	
3.15	Are register clears always using an explicit action? If not so, described how they are cleared.	Do not want to clear any registers inadvertently.	
3.16	Are control bits that activate different functional behavior independently controllable without knowledge of prior bit states?	Do not want read/modify/write.	

	Review Item	Rationale	Reviewer Comments/Feedback
3.17	Are all software writeable registers readable?	Do not want write-only register. Need to be able to verify the content of register after write access.	
3.18	Is there any feed-through network (input pin tied directly to output pin without going through any flip-flops)?	Hard to control the timing of feed-through network.	

B.4 Implementation

	Review Item	Rationale	Reviewer Comments/Feedback
4.1	Does the preliminary synthesis report have warnings? Can they be removed easily?	Too many warnings tend to cover up real problems.	
4.2	Review the synthesis and PnR constraint files. What are the driving factors behind these constraints?	Need to understand if the design is constrained adequately. Over or under constraining the design can lead to unreliable product.	
4.3	Are there any tool-specific synthesis directives embedded in the HDL source code? If so, why are they needed?	Need to understand why the synthesis directives are used.	
4.4	Could the design be altered by the synthesis tool in any way during the synthesis process?	Need to make sure that the synthesis tool does not unintentionally change the FSM design, replicate registers or logic blocks, etc.	

	Review Item	Rationale	Reviewer Comments/Feedback
4.5	Is the synthesis script/project set up correctly?	Need to review the synthesis directives and settings. As a minimum, review the following information: - Device technology. - Part number. - Package. - Speed grade. - FSM encoding control*. - FSM optimization control*. - Resource sharing setting. - Fan-out limit. * All the FSM-related synthesis directives need to be TURNED OFF. This allows the design entered by the designer to stay unaltered during the synthesis process. In Synplicity, for example, the symbolic_fsm_compiler has to be set to zero to preserve the FSM encoding scheme selected by the designer.	
4.6	If “ifdef” is used in the source code, is the synthesis environment set up for the right “ifdef” conditions?	“ifdef” can change the as-built design. Need to make sure the right variables are defined in synthesis.	

	Review Item	Rationale	Reviewer Comments/Feedback
4.7	What device/package/speed grade/operating voltages are the project set up for? Are the parameters correct?	Need to review the PnR directives and settings. Need to review (when applicable): - Device family. - Package type. - Speed grade. - Die voltage. - I/O voltage.	
4.8	Identify the parameters for fuse/bit file generation. Are they set up correctly?	Need to review the fuse/bit file generation settings. For MicroSemi, need to review the selection of the following options (when applicable) for the fuse file: - Are clamping diodes for unused I/O pins disabled? - What programming algorithm is selected? - Is JTAG reset pull-up resistor used? - Is the global set fuse used? - Is Antifuse security feature enabled? (This feature is not desired.) For Xilinx, need to review the following parameters for Bitgen: - Setting for HswapenPin. - Setting for UnusedPin. - Is DONE pin driven after completion of configuration? - Is PERSIST set if configuration readback is used?	

	Review Item	Rationale	Reviewer Comments/Feedback
4.9	What operating environment is the PnR tool set up for? Is it consistent with the expected operating environment?	The environment has to be set up correctly for the tool to perform the STA correctly. Need to review set up on: - Operating temperature range. - Operating voltage range.	
4.10	What are the flip-flop, combinatorial logic, I/O and timing margins? Are they adequate?	Need to ensure the design margins are meeting the requirements defined by the design organization.	
4.11	Is the I/O standard and threshold level consistent with the components shown in the schematic?	Need to make sure that the signal levels specified by the PLD are compatible with those specified by the devices interfacing to the PLD. For example, is a 5V CMOS output from the PLD driving a device using 3.3V CMOS I/O? This might work logically, but can create an electrical stress condition to the device.	

B.5 Verification

	Review Item	Rationale	Reviewer Comments/Feedback
5.1	How are the test cases organized? What is the fidelity of the simulation models? Is the test bench organization support effective verification review?	The design of the test bench affects the effectiveness of the verification effort. It uses high fidelity simulation model supplied by vendor when possible. Constraints of the simulation models need to be noted in the verification matrix. The organization of the simulation tests need to support the organization of the verification to facilitate the verification review process.	
5.2	Are there any complex algorithms in the design? If so, how are they verified? Is there any Matlab type simulation done on the algorithm? If so, what has been done to reconcile the Matlab simulation results with the simulation results?	It is sometimes impossible to simulate complex algorithm adequately in HDL simulation. However, it is likely to achieve reasonable confidence level of the design when comparing the limited HDL simulation results with those produced with Matlab or other modeling tool.	
5.3	Are self-derived requirements from the design specification covered in the verification?	It is important to make sure that the verification addresses the self-derived requirements documented in the design specification.	
5.4	What is the maturity of the verification matrix? Has the verification matrix been reviewed?	The maturity of the verification matrix can help gauge the maturity of the design.	
5.5	What is the code coverage of the existing simulation test bench?	Uncovered statement, toggle, transition, etc. need to be dispositioned.	

	Review Item	Rationale	Reviewer Comments/Feedback
5.6	Are there any functions not testable in hardware? Can anything be done to improve the testability of these functions in hardware?	Need to pay special attention to functions that can only be tested in simulation.	
5.7	Are all the test benches self-checking? If not, what is done to make sure that the test results are consistently verified?	Self-checking test benches makes sure that the simulation results are consistently verified. Visual inspection of the waveforms does not always detect unexpected test results.	

B.6 Radiation Tolerant FPGA (RTAX)

	Review Item	Rationale	Reviewer Comments/Feedback
6.1	Are internal probe points limited to 2 per tile?		
6.2	Is embedded FIFO controller used in the design?	Embedded FIFO controller is not used because it is not rad-hardened in RTAX.	
6.3	Are single-ended clock inputs only assigned to the P-side of a P/N clock input pair?	According toMicroSemi: "…when CLKBUF (HCLKBUF) is used with a single-ended I/O standard, it must be tied to the P-pad.…of the hardware clock (HCLK) package pin. In this case, the HCLK N-pad can be used for user signals."	

APPENDIX C

CUSTOMIZING

This appendix provides a framework to assist projects in the customization process. It uses a risk-based assessment and safety-criticality factors. This process cannot be applied to safety-critical or mission-critical projects. The following tables calculate a project classification score and then use this score to determine applicable parts of this document to the given project.

a. The first step is to generate a classification score. Table 5, PLD Classification, provides a list of criteria.

b. Each item (row) gives a score from 1-5 based on a number of factors.

c. Average the score from table 5 (rounding up).

d. Table 6, PLD Development Customization Recommendations, uses the classification score from table 5.

e. Use table 6 as a starting point for customizing.

Table 5—PLD Classification

PLD Classification Factor	1	2	3	4	5	Score
Resourcing						
Project cost	<500k	$500k-1M	$1M-2M	$2M-20M	$20M and up	
Organizational Complexity						
Development Location	Single Branch	Single Center	2 Centers	3 Centers	4+ Centers	
Customers	Self	Low number of users	Within Center	Outside Center	External to NASA	
Developers Experience	Staff experienced	Most staff experienced	Half of staff experienced	Few staff experienced	Staff not experienced	
Technical Complexity						
Test Requirements	No testing required	Minimum testing	Standard testing required	Integrated testing	Major testing effort required	
Board	Commercial Off The Shelf (COTS) Evaluation Board	Proven custom	Modified custom	New custom	Flight boards	
Operational Software	No software	Familiar to project	Software has to be purchased and learned	Some software has to be developed	Full custom software has to be developed	
Integration of Deliverables	Standalone	Some integration	Integrated	Highly integrated	Fully integrated	
Implementation Risk	Well proven, known to	Proven with some Center	Proven, but new to Center	Partially proven	Pioneering	

PLD Classification Factor	1	2	3	4	5	Score
	Center	experience				
Safety Critical	Projects identified as safety-critical and/or mission-critical uses column 5 for every row in this table.					N/A
Classification = Simple Average (score / 9)						

Key Definitions:

- **Board –** The board on which the PLD is run.
- **Customer** – Location of the customers.
- **Developers Experience** – Examines the experience of the development team.
- **Development Team Location –** Defines the locale of the development team.
- **Implementation Risk** – An indication of how familiar the team or Center is with this type of implementation or technology.
- **Integration** – An indication of the types of devices with which the PLD interfaces.
- **Integration of Deliverable** – The PLD is always on a board with some additional circuitry. This is meant to distinguish the level of integration with other boards, functions, software, etc.
- **N/A** – Not applicable.
- **Operational Software** – The software that interfaces to the PLD. It can be internal on an embedded processor or and external interface (i.e., custom command handler).
- **Project cost –** The total cost of the project (not just the PLD).
- **Software Tools** – Determines the complexity of software that needs to be integrated.
- **Test Requirements** – Defines the testing requirements. Small development or research project do not have formal verification requirements.

Table 6—PLD Development Customization Recommendations

Classification	1	2	3	4+
Planning Documentation				
PLD Development Plan	Schedule and Budget	Schedule and Budget	Yes	Yes
Requirements	Simple	Simple	Yes	Yes
Bug Tracking	N/A	N/A	Simple	Yes
Verification Plan	N/A	N/A	Yes	Yes
Planning Review	N/A	N/A	Yes	Yes
Requirements Review	**N/A**	Informal	Yes	Yes
Preliminary Design Phase				
Design Specification	Simple Block Diagrams	Simple Block Diagrams	Yes	Yes
ICD	N/A	N/A	Yes	Yes
Architecture Review	N/A	N/A	Recommended	Yes
PDR	N/A	Informal	Recommended	Yes
Detailed Design Phase				
Design Specification	State Diagrams	State Diagrams	Yes	Yes
Design Practices (Section 7.3)	Review	Review	Yes	Yes
Detailed Design Review	N/A	Recommended	Yes	Yes
Design Specification Review	N/A	N/A	Recommended	Yes
Implementation Phase				
Build Procedure	N/A	N/A	Yes	Yes
Code Reviews	Recommended	Recommended	Yes	Yes
Revision Control	Basic	Basic	Full	Full
Design Practices (Section 8.4)	N/A	N/A	Yes	Yes
Synthesis Review	N/A	N/A	Yes	Yes
PnR Review	N/A	N/A	Yes	Yes
Verification Phase				
Test Procedures	N/A	N/A	Yes	Yes
Verification	Unit Testing	Unit Testing	Full	Full
Independent Peer Verification	N/A	N/A	Informal	Yes
Verification Review	N/A	N/A	Yes	Yes
Delivery Phase				
Programming Procedure	N/A	N/A	Yes	Yes
Revision Control	Basic	Basic	Full	Full
Formal Release (in CM)	Recommended	Recommended	Yes	Yes
Documentation	Simple Report	Simple Report	Full	Full
RDD	N/A	N/A	Yes	Yes
Delivery Review	N/A	N/A	Yes	Yes

KEY:

- **Basic** – Revision Control is a simple tool that tracks versions of items. Complicated merges and branches are not applicable.
- **Full** – Per this NASA Technical Handbook.
- **Informal** – A discussion with colleagues that is not formally documented or tracked. It could be a tabletop review or informal discussion.
- **N/A** – Not applicable.
- **Preferred** – This is a recommended item, but in a tight project, could be combined in the larger project planning.
- **Recommended** – A practice that is high priority.
- **Simple** – Implementation might involve a spreadsheet or an email.
- **Unit Testing** – Simple module-level simulations or testing.
- **Yes** – A practice that is highly recommended.

APPENDIX D

LIFE-CYCLE PRODUCT GUIDANCE

Table 7, Guidance for PLD Life-Cycle Products at Various Reviews, below summarizes the current guidance for PLD life cycle products and their maturity level at various life cycle reviews. The chart serves as guidance only. Final decisions are made at the project level.

Table 7—Guidance for PLD Life-Cycle Products at Various Reviews

Product	SRR	SDR	PDR	CDR	TRR	SAR
Development Plan		P	B	U	U	
Schedule	D	P	B	U	U	
Requirements Specification	D	P	B	U	U	
Verification Plan		D	P	B	U	
Design Specification	D	P	P	B	U	
ICD		D	P	B		
Test Procedures					B	
Test Reports						F

Key:

- **D** – Draft
- **P** – Preliminary
- **B** – Baselined
- **U** – Updated
- **F** – Final

APPENDIX E

SAFETY CRITICAL

E.1 Safety-Critical Considerations

The following bulleted list provides design considerations (highlighted in bold) that the PLD requirements document needs to address. The PLD design team considers these items when creating the requirements and implementing the design. For example, in a safety-critical design, the design team ensures that a requirement is in the form of: "The PLD device shall provide the status of inhibits used to control hazards." This was taken from the first bullet/design consideration.

- **Inhibit Status.** Computing systems provide the status of inhibits used to control hazards.
- **Hazardous Function Control.** Computing systems provide hazardous function control where the inadvertent activation, deactivation, or proper control by the function could result in an identified critical or catastrophic hazard.
- **Safe Initialization.** Computing systems are initialized to a known, safe state. Circuitry interfaced to the PLD takes into consideration the transient nature of inputs/outputs during power-up/power-down conditions.
- **State Transition.** Computing systems transitions safely between all predefined known states.
- **Orderly Shutdown.** Computing systems that implement termination of safety-critical functions perform orderly, controlled shutdowns of those functions to known, safe states.
- **Off-nominal Power.** Safety-critical computing systems establish a safe or powered-down state when self-monitoring functions detect off-nominal power conditions.
- **Operator Overrides.** Computing system overrides require at least two independent actions by the operator.
- **Command Sequence.** Where execution of commands out of sequence can cause a hazard, the computing system rejects commands received out of sequence.
- **Anomaly Recovery.** Computing systems establish a predefined safe state prior to the operational time predicted to cause a critical failure, following detection of predetermined indications of incorrect or incomplete processing.

E.2 Checklist for Safety-Critical Design Specification

The following checklist items are suggested for the development and documentation of PLD designs used in safety-critical applications. Not all items are appropriate to all designs.

a. Demonstrate completeness of system state requirements including the following:

(1) Design starts in a known safe state. Interlocks are initialized or checked to be operational at system startup, temporarily overriding interlocks.

(2) All signals are properly initialized upon startup.

(3) The maximum time for the PLD to have deterministic behavior/outputs is specified.

(4) Expected response to all documented hazards for all safety-critical functions is documented, demonstrated, and tested (as possible).

b. Demonstrate completeness of requirements for events that trigger state changes including:

(1) Robustness criteria:

A. Every state has a behavior (transition) defined for every possible input.

B. Every state has a behavior (transition) defined in case there is no input for a given period of time (a timeout).

(2) Non-determinism criterion—the behavior of the state machine is deterministic.

(3) Value and timing assumptions:

A. All incoming values are checked and a response specified in the event of an out-of-range or unexpected value.

B. All inputs are fully bound in time, and the proper behavior specified in case the limits are violated or an expected input does not arrive.

c. Demonstrate output specification completeness including:

(1) Upon exceeding the duration limit of a hazardous action sequence, the state machine cancels the sequence automatically, returns to a safe state, and informs the operator.

d. Demonstrate completeness of the specification of transitions between states.

APPENDIX F

BOARD LEVEL DESIGN CONSIDERATIONS

The topics in this appendix focus on considerations for the PCB design. They may serve as discussions but are not necessarily applicable to the PLD design.

F.1 Board Reset Consideration

The following subsections address items of consideration when resetting the design board.

F.1.1 Reset Logic Circuit

Provide sufficient noise margin, adequate slew rate, and glitch filtering. Transient effects have to be considered on the reset circuit. For the application of power, the output of the POR or reset circuit is ideally a solid logic level and glitch-free. This requires the POR circuitry to be designed using logic elements that operate correctly at the low ramping up voltages seen during power up. This ensures that the POR signal is valid at the earliest time possible in the power-up-down sequence of events. Inrush currents to timing capacitors are not to exceed the maximum for that capacitor type. Rise times to logic gates, if used as a comparator, are not to exceed the input's specifications. Gates with hysteresis inputs are often used for this purpose. Note that even with that type of input, output glitches may occur and several stages of logic gates are not to be used to dampen them. The most robust solutions often utilize a comparator with hysteresis.

Another transient factor to consider is the rise time of the flight power supply, both best and worst cases. These often differ substantially from laboratory supplies and may be non-monotonic or have substantial overshoot and ringing. Note that flight power supplies are often slew-rate limited to minimize conducted emissions on the power bus. The time constant of the supply may exceed that of the POR circuit.

For discharge, ensure that there is a low impedance path for timing capacitor discharge and that the inputs of logic gates are protected. Most CMOS inputs, but not all, have ESD diodes from the input to the supply rail. Discharging a large capacitance through that input may damage it.

Also, consider the requirements and response of the circuit to momentary disruptions on the power bus. While many circuits may recover or be recoverable from a power-on reset, this is not true for all circuits. One such example is non-volatile, erasable memories, which need to be carefully protected.

For reset circuits, many implementations have “asynchronous application, synchronous removal” of the reset circuit. However, for many devices, in particular many programmable devices, the inputs can be blocked or ignored during the power-on transient. This may be because of the need for charge pumps to start or configurations to be loaded and then released. For devices with synchronized inputs, the clock oscillators have to start, perhaps taking many tens of milliseconds (ms), before the reset can be applied. Outputs of these devices are to be handled at the same system level as the reset, which may appear to work on the schematic or in the HDL design files, but is ignored by the real circuits. See figure 5, Recommended Power-On Reset Implementation, that follows.

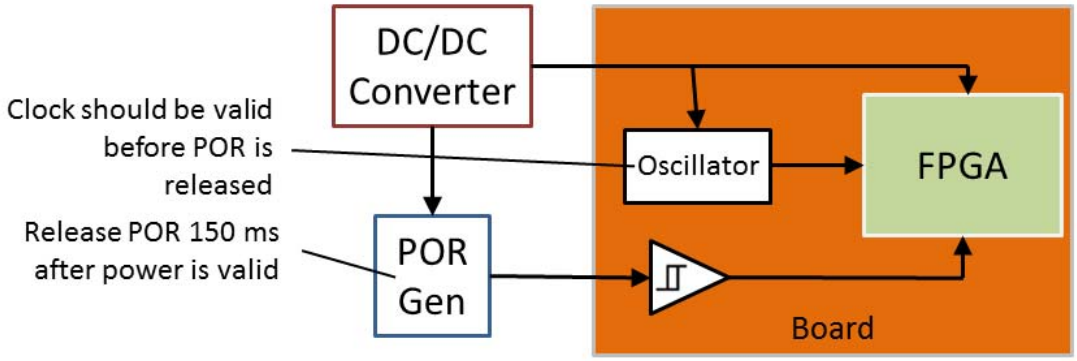

Figure 5—Recommended Power-On Reset Implementation

Steady state or DC effects are also important. Check the leakage currents of timing capacitors and logic gates, as the amount of leakage current times the resistance of the timing resistor may result in a voltage drop that eliminates all noise margins.

F.1.2 Ensure Appropriate Component Start-up Time

Allowing for appropriate start time is critical. It is important to ensure that the guaranteed reset time is sufficient for all circuits in the system in order to avoid premature release of the POR signal, which may result in an indeterminate state. The following illustrates some factors to consider when determining start-up time:

a. For many PLDs, start-up includes the build-up of voltage in the charge pump to charge internal capacitances, a delay, and then a release of outputs.

b. Other PLDs require a sequence of resets for proper loading and release, with many circuits having internal power-on reset circuits. The timing of all these resets have to be analyzed for best- and worst-case behavior.

c. Some standard components on digital logic boards (e.g., crystal clock oscillators) can have a substantial startup time.

d. Components, such as PLDs and crystal clock oscillators, may have start-up times that are a function of the rise time of the power supply. This behavior is often inadequately specified or not specified at all.

F.1.3 Protect Mission-Critical PLD Signals During POR

Many logic elements do not follow their truth tables as the power supply ramps up. Therefore, the POR signal has to act as a gate (via external circuitry), blocking false signals during the power supply rise time transient and then releasing after all circuits are stable. On the other side, when the power comes down, the POR circuit may need to be asserted early, ensuring that critical circuits are safe before the logic elements lose control as the voltage drops. Devices that often need protection are pyrotechnic and explosive initiators, EEPROMs, flash memories, etc. Note that some devices, such as microcontrollers, have internal flash memories; therefore, it is recommended that the team evaluate all components and system interfaces for necessary protection by the POR signals.

F.2 Special Pins

This section contains general information on special pins for all PLDs.

F.2.1 Properly Terminate Configuration Pins

A common problem identified during design reviews is the improper termination of special pins. For every device, the design team carefully reviews the data sheet and design schematics to determine whether each special pin is properly terminated. Termination of many of these special pins cannot be verified by test.

Ensure that each configuration pin is carefully checked against the latest data sheet. Some pins have very high internal pull-up resistors that can be compromised by high-speed signals on the board level. In addition, some configuration pins can naturally just happen to float to the desired state with nominal operation observed. Beware of special pins, such as programming pins, that are required to be terminated appropriately for flight.

Different devices have different pins and there is no overarching, as a general rule, other than that each pin is checked.

F.2.2 Unused Inputs

Do not leave unused inputs (pins that are defined as input) floating. In general, all devices need to have properly terminated inputs. For normal CMOS devices, this is a requirement. Certain programmable devices, such as FPGAs, often take care of unused pins via software, exploiting the programmable nature of the microcircuit. However, the limitations for each pin have to be thoroughly considered. For example, in MicroSemi SX and SX-S, clock inputs, such as HCLK or the global routed clocks, do not have an output stage—they are special purpose. They have to be terminated by the user. Failure to do so can result in large unintended currents that could cause device damage.

Depending on the device, the manufacturer may reserve pins labeled as “N/C” (no connect) for internal purposes. Check each pin carefully according to the specification and clarify with the manufacturer if necessary about the risk involved with terminating them on the board.

F.2.3 Follow the Manufacturer’s Recommendations for Test Interface

Many devices have custom test interfaces that have to be handled on a case-by-case basis. Since they hook up to test equipment, following the manufacturer's instructions is recommended. For example, MicroSemi SX-S device test pins need to be series terminated.

F.2.4 Disable the Debug Interface for Flight Configuration

If PLD I/O are used to implement a debug interface for development, make sure that the inputs are safely terminated or driven and that outputs are not toggling in the final flight configuration, causing unnecessary EMI and noise.

F.3 Device Inputs/Outputs

The following subsections describe device inputs and outputs.

F.3.1 Signal Termination

Ensure that output signals are terminated properly. For single-ended signals, start by using termination resistor values equal to the trace impedance minus the output impedance of the driver ($R_{term} = Z_{trace} - Z_{driver}$); then perform signal integrity analysis to optimize the termination resistor values.

a. Address edge-sensitive signals, such as clock output signals, with special care to ensure that there is a smooth transition through the threshold. For loaded clocks, perhaps traveling over long runs, reflections may often result in non-monotonic transitions causing false or double clocking. Note that this may happen on the “inactive” edge. Similarly, overshoot and ringing can also cause false clocking, particularly on the transition to ground. Unterminated nets could result in ringing that is a source of EMI, even when they do not contribute to logic failures.

b. Ensure good signal quality as damage to I/Os may happen. Most manufacturers have tight limits on how far outside the rail a signal may travel, sometimes coupled with maximum time outside of the recommended limits.

c. Plan on termination resistors in advance to support signal integrity analysis efforts. The signal integrity analysis may show that they can be eliminated. However, if they are required, adding them later could require additional time in layout, debugging, rework, and/or costly board respin. Measurements can also be used to supplement this effort.

d. Review schematics for proper terminations on interfaces such as the PCI interface.

e. Signal Termination: Some modern logic devices provide internal source termination at specific values. Board designers can consider the source termination options of the FPGA, and design traces accordingly. In addition, input terminations can be considered as well, especially with Low Voltage Differential Signal (LVDS) terminations or double data rate (DDR) interfaces.

f. Be aware of part-specific default operation. For example, differential interfaces, such as RS-422 and LVDS standards, require that receivers whose inputs are open, output a logic '1' to the system. Thus, systems need to be designed so that such a logic '1' state does not cause an unintended lock up of the system.

F.3.2 Tri-State Bus Considerations

A recommended practice is to avoid contention and floating. Bus contention wastes power, needlessly generates noise, and stresses components. Consider the following actions:

a. Avoid contention when actively driving tri-state buses. Have a guaranteed off-time between drivers on the bus in the worst-case. A clock cycle between tri-stating one driver and enabling another may be sufficient but a thorough timing analysis is necessary. Be sure to consider timing parameters that have to be added together. For example, the tri-state time of an external SRAM's output enable that is controlled by a PLD's state machine would be the sum of the time cut off out of the PLD + the travel time on the board + the SRAM's tri-state time.

b. Do not allow the bus to float for a long time or have slow transition times, as this increases power and noise and may negatively affect reliability.

c. Consider parking the bus when not in use (drive to 1's or 0's) instead of using pull-up /-down resistors. Some PLDs have a "keeper" I/O standard that does this.

d. When parking a bus and still using pull-up/-down resistors, ensure that the bus is fully driven to the parked state before it is tri-stated to avoid ringing.

e. For portability, infer a tri-state buffer in register-transfer level (RTL) design files instead of instantiating a device-specific tri-state buffer.

f. Be sure to consider the power up, reset, or the configuration cycle for the device to make sure that outputs are not floating or contending. One method to mitigate contention is to gate FPGA-sourced control signals with board-level POR using discrete CMOS logic gates to disable tri-state drivers in all devices on the bus.

F.3.3 Input Transition Times

It is important to examine input slew rate. Some high-speed devices have very stringent restrictions on input transition times. Failure to meet the requirements may result in oscillations (figure 6, Glitches Due to Input Slew Rate Violations), multiple clocking, degradation, or damage.

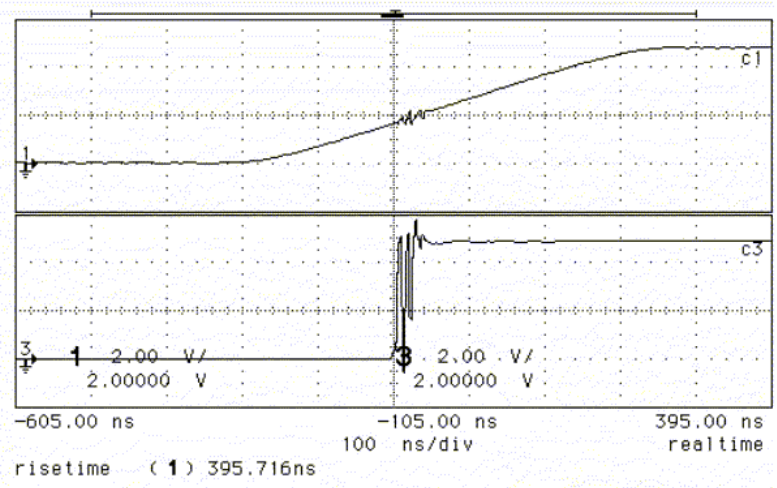

Figure 6—Glitches Due to Input Slew Rate Violations

Simple pull-up or pull-down resistors, with transition times in the hundreds of nanoseconds (ns), may be too slow. Take appropriate precautions if older digital logic families are used that may have outputs that are not compatible (e.g., too slow) with high-speed devices.

F.3.4 Avoid Shorting Outputs Together

Shorting outputs together is sometimes done to increase drive on the board. This has to be avoided since it may damage components if the switching speeds are not matched, and it can be difficult or impractical to test this redundant topology. If this needs to be done, consider using an external buffer or splitting the loads between two or more nets, each driven by a single output.

F.3.5 Mixed I/O Standards

Examine voltage thresholds, DC compatibility, and noise margins. When mixing devices from multiple families, even from the same manufacturer, care has to be taken to ensure the devices are operated within specifications and there is sufficient noise margin. This may be problematic when substituting parts for either upgrading circuit performance or dealing with obsolescence issues.

For inputs, many CMOS technology devices advertise “transistor-transistor logic (TTL) compatible” inputs. However, these inputs may in fact differ rather significantly from their TTL counterparts. The first major difference for many but not all devices is the impedance presented to the interface when power is removed from the device. For example, when radiation-hardened CMOS latches were substituted for soft 54LS373 latches in the Galileo attitude control

computer's memory units, block redundancy circuits failed since the sneak path through the radiation-hardened inputs' ESD protection diodes was not accounted for when power was removed.

Another related difference is the maximum voltage that can be applied. Some bipolar devices are useful for reliable level shifting from higher voltages to lower ones; CMOS replacement devices will forward bias to the protection diodes resulting in unintended current flows and possible damage or circuit failure. Finally, many CMOS inputs have logic thresholds that are not truly TTL compatible. That is, the TTL voltage threshold for input high (VIH) specification is often not met, with VIH (max) values of 2.2V, 2.4V, and sometimes 2.5V being specified whereas true TTL devices have a threshold defined by two diode drops, typically in the range of 1.2V to 1.4V.

TTL outputs are only guaranteed to drive to VOH = 2.4V so there may be little or even negative noise margins present in these situations. The switching point difference can also lead to circuit failure, depending on the signal integrity. Often TTL outputs, when switching, exhibit non-monotonic-behavior in the waveform, particularly with heavy and/or long loads. While the behavior is often at a high enough voltage so that TTL devices operate correctly, the often higher VIH of CMOS devices may result in multiple clocking. Pull-up resistors can restore adequate DC noise margins in these situations if given enough time to settle, which may be quite a while for this passive circuit. Note, however, that TTL to CMOS clock interfaces designed in this fashion often fail logically since the CMOS input may see multiple transitions resulting in double clocking.

CMOS output stages can also be problematic, and subtle device characteristics can cause errors. It is strongly recommended to check all specifications carefully. For example, many CMOS devices when driving loads are specified at only very low current levels for high or logic '1' signals. However, TTL inputs take substantial currents and do not present the high impedance seen by CMOS field effect transistor (FET) inputs and the output may be dragged down. For output loads that are a mix of CMOS and TTL inputs, they often have to be split to guarantee the high voltage needed for the CMOS inputs. This is typically 70 percent of VDD, and the high current needed for TTL inputs, with the lower VIH of 2.0V. Another factor to consider is the structure of the output stage in the CMOS device. For example, some devices will not swing all the way to the high rail and are voltage limited. This may result in some totem-pole current if the p-channel FET in the next input stage is not cut off. Some devices, even with a 5V I/O supply like an RT54SX series, only drive outputs to the core voltage of 3.3V, making this CMOS output incompatible with 5V CMOS inputs on the same board.

Components currently have many supply voltages, including 1.5V, 1.8V, 2.5, 3.3V, and 5.0V. There is also an abundance of I/O standards with the newest devices being programmable, so their characteristics are not obvious or known from viewing a board schematic. Therefore, I/O compatibility has to be carefully verified, particularly when substituting "new and improved" devices or alternate devices.

F.3.6 Power Switching and Cold Sparing

Examine all power-up and power-down modes or transitions. For either redundancy or power savings, care has to be taken when designing a system with blocks that are independently powered. Many CMOS devices present low impedance when powered down through either the intrinsic or the ESD protection diodes. Other devices, with cold sparing inputs, may have high input impedance that is suitable for operation. For PLDs, selecting 3.3V PCI compatibility, as one example, can result in a “cold sparing” device no longer being high impedance since a clamping diode is enabled. While many bipolar devices are compatible with cold sparing architectures, some devices have a sneak path to V_{CC} through the output (figure 7, Sneak Path in Some LSTTL from Output to V_{CC}). Be sure to consider test setup, not just the flight configuration. For example, consider whether a piece of test equipment needs to be powered up and down co-incidentally with the flight unit.

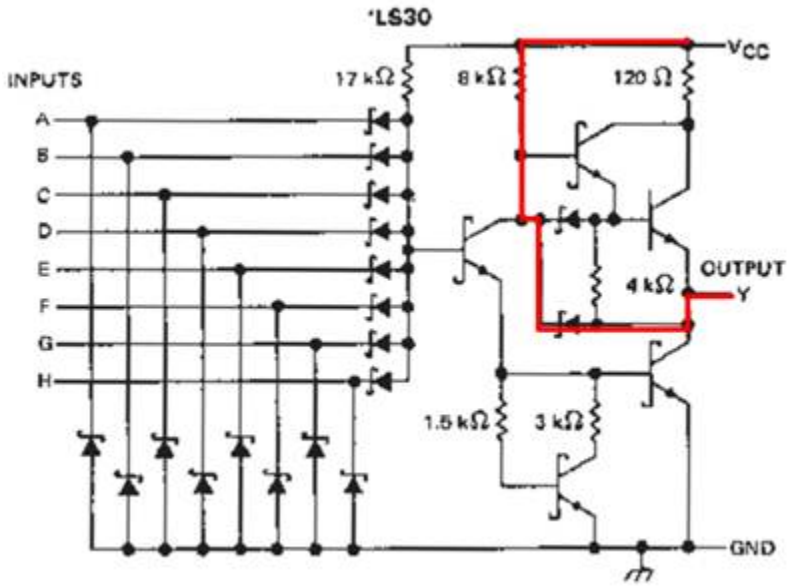

Figure 7—Sneak Path in Some LSTTL from Output to V_{CC}

F.4 Power Related Design Considerations

The following sub-sections address power-related design considerations.

F.4.1 Supply Sequencing

Follow device family datasheets to ensure proper power-up and -down sequencing. Power sequencing requirements may differ between flight and prototype devices. Many of the newer technology devices require two or more power supplies. Often these are divided into supplies to power the core of a logic device and a second supply to operate the I/O cells. Additional supplies may be needed for PLLs and DLLs, special I/O standards, or various bias supplies, such as external charge pumps. The supplies have to meet all of the DC standards as well as ripple characteristics, particularly for circuits such as PLLs. The sequence that power is supplied to a single device in certain cases, can affect circuit behavior, performance, and reliability. For

certain devices, such as SX-S series devices, if the I/O supply is brought up before the logic core, a large inrush current may be present; this would not be the case if the order of the supplies were reversed. For certain devices, incorrect power sequencing can result in overstress or damage. Often the requirements for sequencing are in application notes. When parts that require sequencing are present, they need to be flagged, and the design has to incorporate circuit protection, as required.

F.4.2 Signals Into Unpowered CMOS I/Os

Analyze design for sneak paths between I/O that interfaces powered and unpowered devices. The power supply sequencing between interfacing ICs, either on the same or separate boards, has to be considered. Many ICs, particularly CMOS, present a low impedance to the system when powered off. Most of these ICs require that the power supply be brought up prior to the application of signals on either the inputs or the outputs (many PLD outputs also have inputs active in the general purpose I/O modules). Some programmable ICs cannot be analyzed by inspection of the schematic; the particular I/O configuration has to be reviewed. For instance, some I/O modules provide for cold sparing; that is, they present high impedance to the system when powered off. That same I/O, configured differently, may have clamp diodes switched in while powered off for PCI compatibility. The design details are needed to do a worst-case analysis.

F.4.3 Startup Voltage Rise Time

Perform voltage supply rise time measurement on actual hardware and verify that the results meet PLD requirements. Startup current transients are common in many devices. The size of the current can be a function of time between power cycles, temperature, ramp rate of the supply, radiation exposure history, power supply sequencing, etc. These currents can be large for certain devices, often as high as several amps. Ensure the power supply system does not limit current to steady state levels because insufficient current during the startup sequence can result in a failure to properly initialize, power device shutdown, recycling in an infinite loop, or a system lockup. Similarly, some parts have restrictions on minimum and maximum power supply rise times. Failure to meet these levels may result in circuit degradation or damage.

F.4.4 Bypassing and Distribution

Follow device datasheets and/or device application notes for proper decoupling. Perform a power integrity analysis at the frequencies of operation. Logic devices can be rather large, consisting of billions of gates. Synchronous design techniques, high operating frequencies, and large I/O counts can result in a challenge to the power distribution and conditioning system. Application notes contain most of the manufacturer’s supply details. Follow these rules unless a power integrity analysis and testing of the system for worst-case conditions reveal an alternate course. Worst-case test patterns can be exploited to ensure high-fidelity power and then replaced with the flight application. JTAG interfaces may also be used with care given that the JTAG test patterns do not violate design limits, such as SSOs. Consider the aging effects of time and radiation.

F.5 Non-Volatile Memories

The following sub-sections address interfacing to non-volatile memories (EEPROM, flash, etc.).

F.5.1 Protection During Power-Up/Down Transitions

The system designer considers the power-down warning and provides enough bulk capacitance to complete a write access of non-volatile memories, if needed. In cases where non-volatile memory protection is required, the PLD requirements specify the mitigations performed by the PLD.

Indeterminate signals sourced from the PLD are a common problem for erasable non-volatile memories. The analysis and test has to examine all of the signals for proper and safe operation during power-up, power-down, and brown-out transients. Note that the real power supply and its bounded characteristics has to be used, not laboratory supplies that most likely have substantially different characteristics. Some devices have a reset pin to help protect against inadvertent writes. The design, analysis, and test/evaluation of this circuit under all conditions are critical for maintaining the integrity of the non-volatile memory contents. Consider circuit operation if the power is shut down during either a planned or unexpected write cycle. The design needs to ensure the proper completion of write cycles to protect the contents of the non-volatile memory. The write cycle often includes the time for not only the bus operation to complete, but also for writing to the internal part, which can take about 10 ms. Another related consideration is the unexpected application of a system reset signal. Shutdown states are entered to help ensure that write cycles are fully completed and properly shut down, with the critical signals put in a safe mode.

F.5.2 Analysis of Damage During Write Cycles

Implement a mechanism to detect corrupted writes. The technology of the non-volatile memory has to be carefully considered if the memory is to be written in flight. Some of these devices, such as EEPROMs, use high voltage to write the cell. If struck by a heavy ion with high voltage applied, the failure mode needs to be analyzed and dealt with appropriately. Thus, writing in flight has to be considered a high-risk operation.

F.5.3 Cycle Count

Design an interface to maximize useful life of memory. Many non-volatile erasable memories have limited number of access cycles. Each device has to be treated on a case-by-case basis with system lifetime and radiation factored in. For example, the 128k x 8 Hitachi die, for example, has a lifetime write specification limit of 104 cycles in byte mode with 105 cycles in page mode. The write mechanism for this device utilizes an 8-byte subpage as the smallest unit that can be written. Therefore, writing the same memory space one byte at a time is more stressful than page writes since entire subpages have to first be fetched and then re-written.

F.5.4 Transients and Noise

Treat control signals to non-volatile memories as critical by minimizing them. It is critical that the signals interfacing with non-volatile memories be clean, that system noise be kept to a minimum, and that they always meet all specifications. In this case, signals include not only logic signals but also power and ground connections; therefore, robust bypassing needs to be used. Noise glitches on EEPROMs, for example, can cause false write cycles to be generated, resulting in inadvertent altering of the device's contents.

F.5.5 Reliability

Design interfaces to implement the required error detection and correction (EDAC). For reprogrammable PLDs that are configured, it is a best practice to maintain a non-changeable memory like programmable read-only memory (PROM) that can be switched on if the configuration memory fails.

The required reliability of the non-volatile, erasable memory device is highly dependent on its application. If the device operates as part of a large memory array, then some bit failures and even page failures can be tolerated either by error correction techniques or by error detection and mapping the failed segment out of service.

Applications such as boot read-only memory (ROM) require error free operation for the CPU. Another example is when the PLD is accessing memory. For single bit failures, a Hamming code may suffice, although that may be awkward for serial PROMs. Some failure modes of non-volatile memory devices may result in a bit oscillating or not providing a valid logic level; in this case, an EDAC device may or may not correct the single bit error, depending on the logic design of the EDAC device being used and whether or not it is static hazard free. In any event, the devices employed, combined with the architecture of the particular system, have to ensure that there are no lockup states from any credible failures. Credible failures include any single bit error and an inadvertent corruption of a non-permanent memory's contents.

Other forms of redundancy may be required, such as TMR, with switchable spares. Some options include the ability to switch in alternate devices, the use of permanent memory such as PROM, or the use of storage buffers to replace erasable non-volatile memory functions, using operational overhead to manage the risk. For example, if a configuration memory device for a PLD fails, a storage buffer and CPU may configure the PLD using a different loading mode, assuming that, of course, the PLD is not needed to run the computer. In general, for critical applications, permanent memories, such as PROM, are to be used to ensure that the spacecraft or other system cannot be permanently lost. This can take the form of boot and safe-hold code for a processor or a basic operating configuration for a PLD.

F.5.6 Refreshing and Reloading

When considering whether to refresh or reload a device, factor the device's specified data retention against mission life. Another consideration is the guaranteed storage time of the device versus mission length. Each device has to be analyzed on a case-by-case basis. Ten years is a frequent specification for the retention of memory contents; however, system lifetimes of several decades is not uncommon.

Refreshing can be risky. The usefulness of it has to be verified with the manufacturer's assistance, to ensure a guarantee of storage integrity, particularly in the radiation environment. When the device is refreshed, it may be susceptible to damage in the space environment by heavy ions. Other errors can occur that damage the contents, such as a computer crash, brown out, or the unexpected removal of power due to a bus fault or a spacecraft entering a safe mode. In addition, each write cycle takes away from the operational lifetime of the component.

F.6 Logic Design

The following sub-sections discuss noise immunity in the design and considerations for off-nominal events. Consider the items listed during the design cycle.

F.6.1 Noise Immunity and Quiet Designs

If applicable to the design, consider the following board design activities to ensure adequate and robust noise immunity:

a. Choose differential signals, particularly for connections between boards. Newer logic devices are directly supporting differential standards. Additionally, high-speed, lower power differential devices support standards, such as LVDS, are now qualified.

b. Serializer-Deserializer (SERDES) components/cores can cut down the number of lines, reducing noise, and, therefore, increase the noise immunity of the system.

c. Use hysteresis buffers to clean up noisy inputs.

d. Inputs that are "TTL compatible" often have specifications and real thresholds that are not TTL compatible, particularly for V_{IH}. Use level shifters as needed.

e. Outputs, particularly from some CMOS families, may not be able to drive TTL loads to a valid logic '1' with sufficient noise immunity. Calculate worst-case currents and voltage output versus worst-case input thresholds. Use level shifters as needed.

f. Make sure PLD I/O standards are compatible with external interfaces.

g. Adequate bypass capacitance for several decades of noise frequency on the voltage input/output (VIO) pins greatly reduces ground bounce and noise problems.

h. Refer to the vendor application notes and datasheets for additional board design requirements.

F.6.2 Defensive Design and Designing for Off-Nominal Events

It is advisable to consider credible but unplanned events. Often many of these situations can be economically handled with some planning. Following are a few sample activities to consider:

a. Perform limit and validity checking. The system needs to respond in a reasonable fashion to unexpected input states. For data passed from one source to another, simple bounds checks can detect and cause appropriate action for many off-nominal conditions, such as a disconnected source. This may result in all Fs being returned on a data bus. For floating point numbers, determine whether the input is in a valid format. A minimum criterion is that any credible input does not damage hardware and prevent recovery. Assume that the probability of software failure is 100 percent.

b. Provide fail-safe interfaces. Analyze the performance and safety of the circuits if a wire breaks in a connector, for each wire. For power, use multiple wires such that if any one wire breaks the remaining set can carry the load (and be sure to test this redundancy). For signals, consider on-board terminations that pull floating signals into a safe and operational state. This can also provide protection if the board or subsystem is powered with a connector not hooked up, perhaps by test error. Avoid putting signals such as power and ground on adjacent pins, as a short can take out the system.

c. Consider lockup states. If interfacing to a device that could potentially lock up, determine a mechanism to detect and recover from such lock up state.

d. Protect PLD pins. Avoid having PLD outputs directly driving cables or massive capacitive loads.

e. Ground the PLD lid.

Follow manufacturer’s recommendation for grounding the PLD lid. The lid may need grounding to ensure that there is no buildup of charges. Doing this prevents ESD events.

APPENDIX G

GUIDANCE

A.1 Purpose and/or Scope

The purpose of this appendix is to provide guidance and is made available in the reference documents listed below.

A.2 Reference Documents

National Aeronautics Space Administration (NASA)

NPD 2810.1	NASA Information Security Policy
NPR 7150.2	NASA Software Engineering Requirements

Goddard Space Flight Center (GSFC)

300-PG-8730.0.1	GSFC Assurance Activities for Digital Electronics for Spacecraft, Instruments, and Launch Vehicles
500-PG-8700.2.7	GSFC Design of Space Flight Field-Programmable Gate Arrays
500-PG-8700.2.8	GSFC Field-Programmable Gate Array (FPGA) Development Methodology

Jet Propulsion Laboratory (JPL)

IOM 3455-09-023	JPL Design Review Checklist
DocID 68532	JPL FPGA/ASIC Hardware Development

Marshall Space Flight Center (MSFC)

MSFC-STD-3663	Marshall Space Flight Center (MSFC) Standard for Configurable Logic Device Developments