Software Process Assurance for Complex Electronics

Richard A. Plastow
Science Applications International Corporation
NASA Glenn Research Center

July 24, 2007

Complex Electronics (CE) are now programmed to perform tasks that were previously handled in software, such as communication protocols. Many of the methods used to develop software bare a close resemblance to CE development. For instance, Field Programmable Gate Arrays (FPGAs) can have over a million logic gates while system-on-chip (SOC) devices can combine a microprocessor, input and output channels, and sometimes an FPGA for programmability. With this increased intricacy, the possibility of "software-like" bugs such as incorrect design, logic, and unexpected interactions within the logic is great.

Since CE devices are obscuring the hardware/software boundary, we propose that mature software methodologies may be utilized with slight modifications in the development of these devices. Software Process Assurance for Complex Electronics (SPACE) is a research project that looks at using standardized S/W Assurance/Engineering practices to provide an assurance framework for development activities. Tools such as checklists, best practices and techniques can be used to detect missing requirements and "bugs" earlier in the development cycle creating a development process for CE that will be more easily maintained, consistent and configurable based on the device used.

Software Process Assurance for Complex Electronics (SPACE)

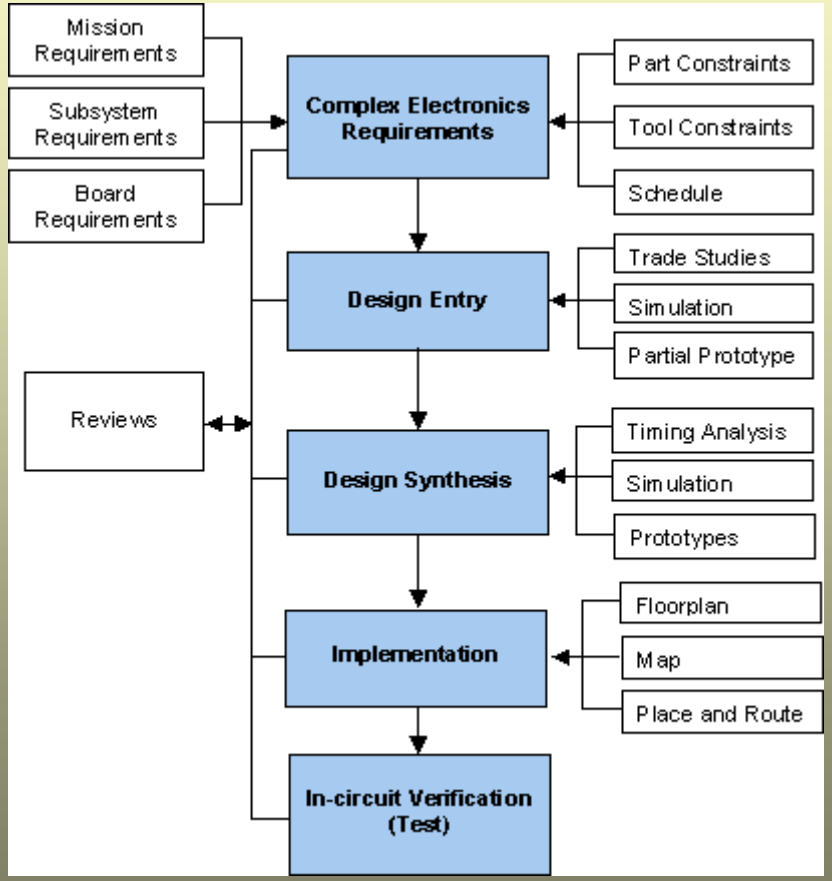
Richard A. Plastow
Science Applications International Corporation
NASA Glenn Research Center

July 24, 2007

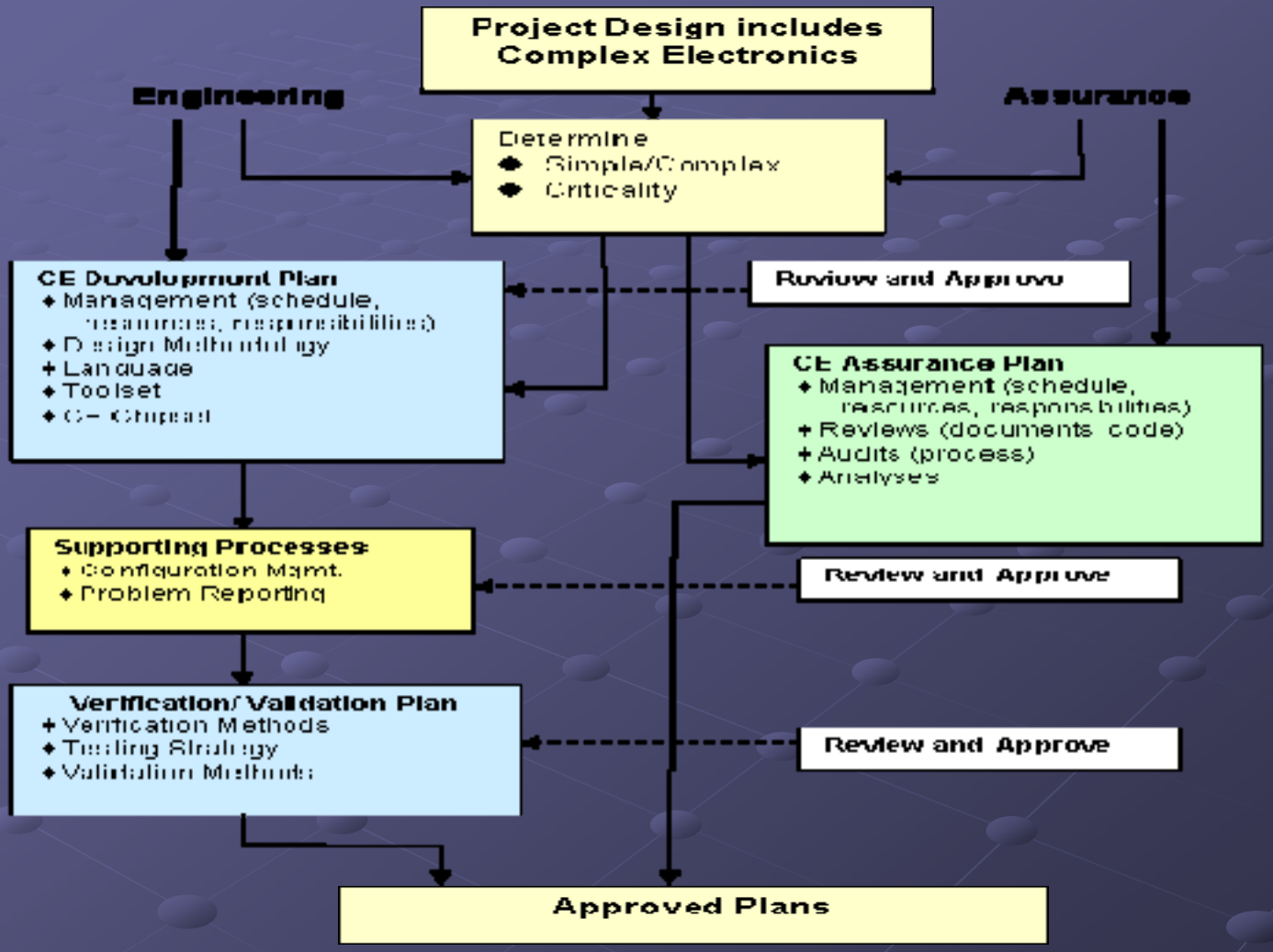
The Quandary

- Complex Electronics (CE) devices can have over one million gates and even a built in microprocessor. These devices are replacing conventional software in many critical applications. How do we assure quality on these devices?

How the Design Process For Complex Electronics should flow



Planning is Where to Start



Simple or Complex?

- **The Federal Aviation Administration (FAA) provides a definition in DO-254, "Design Assurance Guidance for Airborne Electronic Hardware" document. It states *"A hardware item is identified as simple only if a comprehensive combination of deterministic tests and analyses appropriate to the design assurance level can ensure correct functional performance under all foreseeable operating conditions with no anomalous behavior. When an item cannot be classified as simple, it should be classified as complex. An item constructed entirely from simple items may itself be complex."***
- Firmware is not CE**. The most common definition is found in IEEE 610.12-1990: "The combination of hardware device and computer instructions and data that reside as read-only software on that device."**

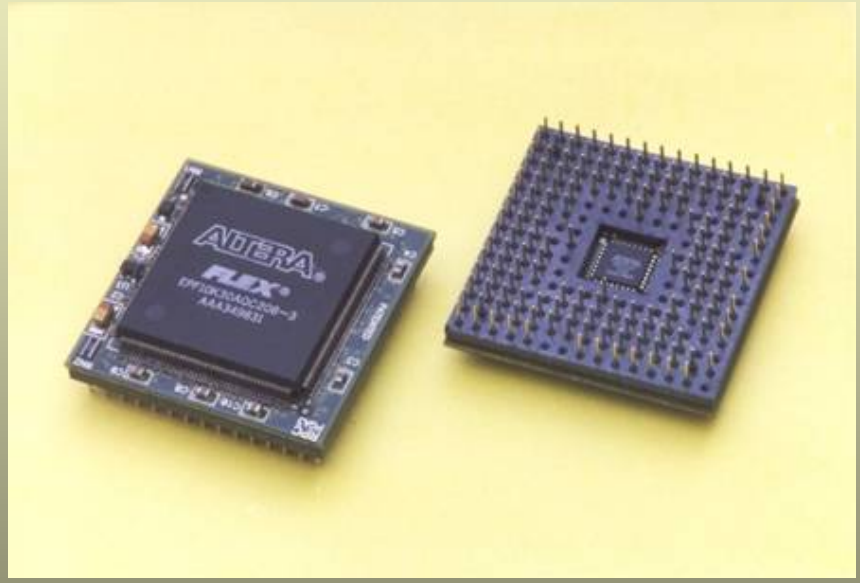
Richard.A.Plastow (SAIC)
Sr. Software Assurance Engineer

What is the Devices Criticality?

Criticality Classification	Criteria
High	1. The complex electronics executes safety-critical functions 2. The complex electronics executes mission-critical functions and is a single point of failure 3. The design is expected to be highly complex 4. The design is expected to have significant risk due to one or more of these factors: i. Unstable requirements ii. Technical concerns with the chosen technology, such as power consumption, design size for the chip, timing, packaging, or operating frequency iii. Highly innovative and untried design approach iv. Highly aggressive schedule v. Inexperience of the design team
Moderate	1. The complex electronics executes mission-critical functions but there is redundancy in the system 2. The design is expected to be moderately complex 3. The design is expected to have moderate risk due to one or more of these factors: i. Some requirements undefined or unstable ii. Somewhat innovative and untried design approach iii. Aggressive schedule iv. Design team contains some inexperienced members
Low	The complex electronics is classified as *Low* if it does not fall into either of the above categories

How do You Start the Process?

- Create an Assurance Plan for your device
  - Can be stand alone or part of the larger assurance plan
  - Plan is based on criticality of device(s)
  - Get concurrence with the plan
  - One plan can cover all devices needed



Assurance Planning is Very Important
Assurance Planning Document

ID	Process Step	Completion Date	QA
1	**Entrance Criteria are met.**		
2	Concur with complex electronics criticality classification (high, moderate, or low). Include in the CEAP (below)		
3	Generate tailored Complex Electronics Assurance Plan (CEAP). This includes steps 4 through 14 below.		
4	Specify purpose and scope of plan.		
5	Determine Applicable and Reference standards and documents.		
6	Define management of the assurance activities, including organization structure, roles and responsibilities, resources, schedule, reporting.		
7	Identify training and tools required for the assurance tasks.		
8	Specify tasks for Requirements phase, including reviews, audits, and analyses.		
13	Specify criteria and tasks for acceptance of complex electronics.		
20	Exit Criteria are met.		

Supporting Processes

- Controlling and monitoring the process used is very important.
  - Configuration Management
  - Audits
  - Selecting the development process
  - Problem reporting / monitoring
  - Risk Management
  - Ad Hoc is NOT a process!

Involve the Correct People

Role	Responsibility
Systems Engineer	• Define the system requirements • Decompose system requirements down to sub-system level • Define system-level testing
Electronics Designer	• Derive requirements for the board or chip level • Design electronics to meet the requirements, using good engineering practices • Implement the design in hardware • Test the hardware; Implement changes
System Safety	• Identify if complex electronics can cause a hazard or are part of a hazard control • Ensure that design errors in complex electronics are considered as a failure mode
CE Process Assurance	• Assess entrance and exit criteria for each life cycle phase • Ensure traceability of the requirements through all levels of development • Analyze the products produced (documents, designs, etc.) against the requirements and the output of the previous phase • Perform white box analysis on CE designs and tests

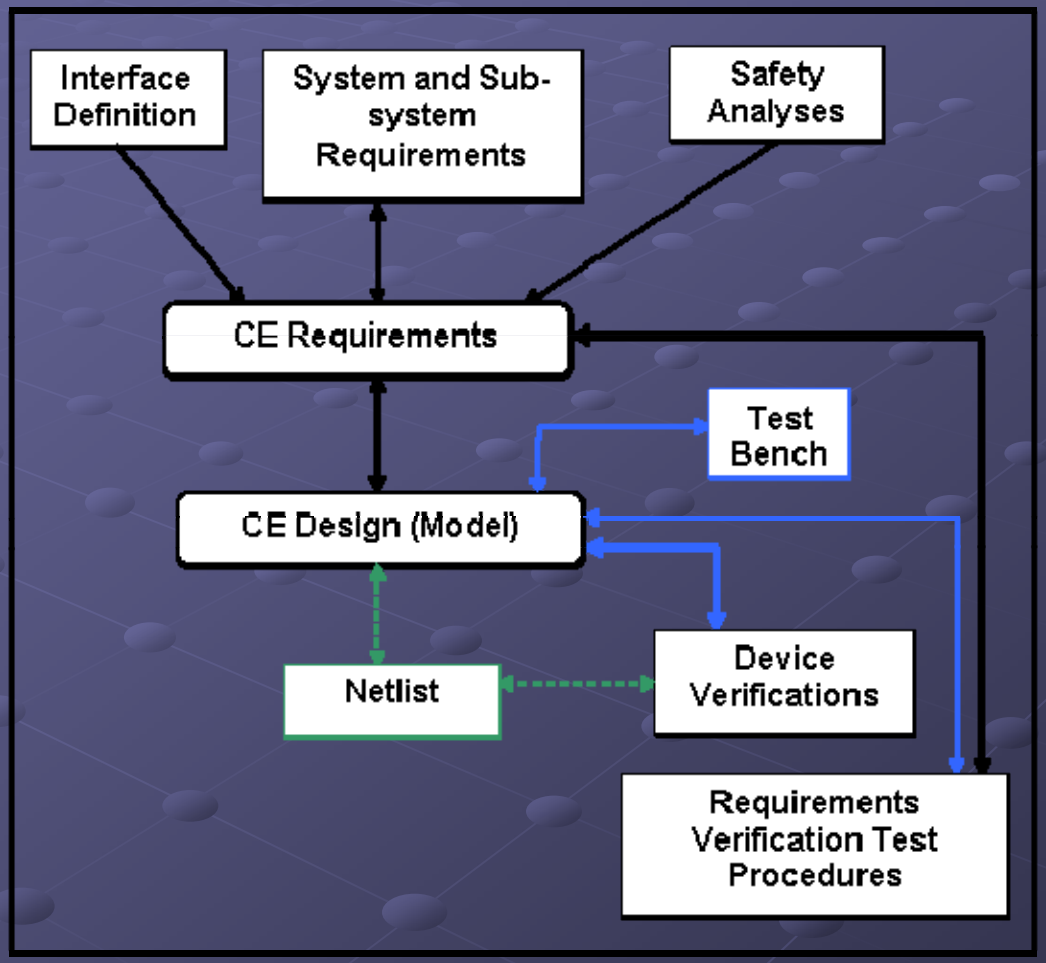
Requirements

- The first step in any design process should be to define and document the requirements and constraints under which the CE must operate. This allows you to think through the issues and document any design decisions and trade-offs.
- Complex Electronics design is often started based on the engineers knowledge of the system, not defined requirements.
- Requirements Reviews
  - Clear
  - Concise
  - Confirmable
  - Traceable
- Interface Control Document verifications
- Signals List

Requirements Review Checklist Snippet

ID	Topic	Criteria	Yes/No/ NA	Comment
2	General	Are the overall system configurations and operating modes defined?		
11	Interfaces	Is the data size and bit order defined?		
12	Signals	For each input signal, is the range of valid data defined?		
13	Signals	For each output signal, is the range of valid data defined? Is a typical value defined?		
20	Initial Conditions	During power-up or reset, do any lines or signals float?		
27	Error Handling	Is the behavior of the CE in response to an external failure or fault specified?		
28	Timing	Has the timing of critical signals been defined?		

Traceability Analysis

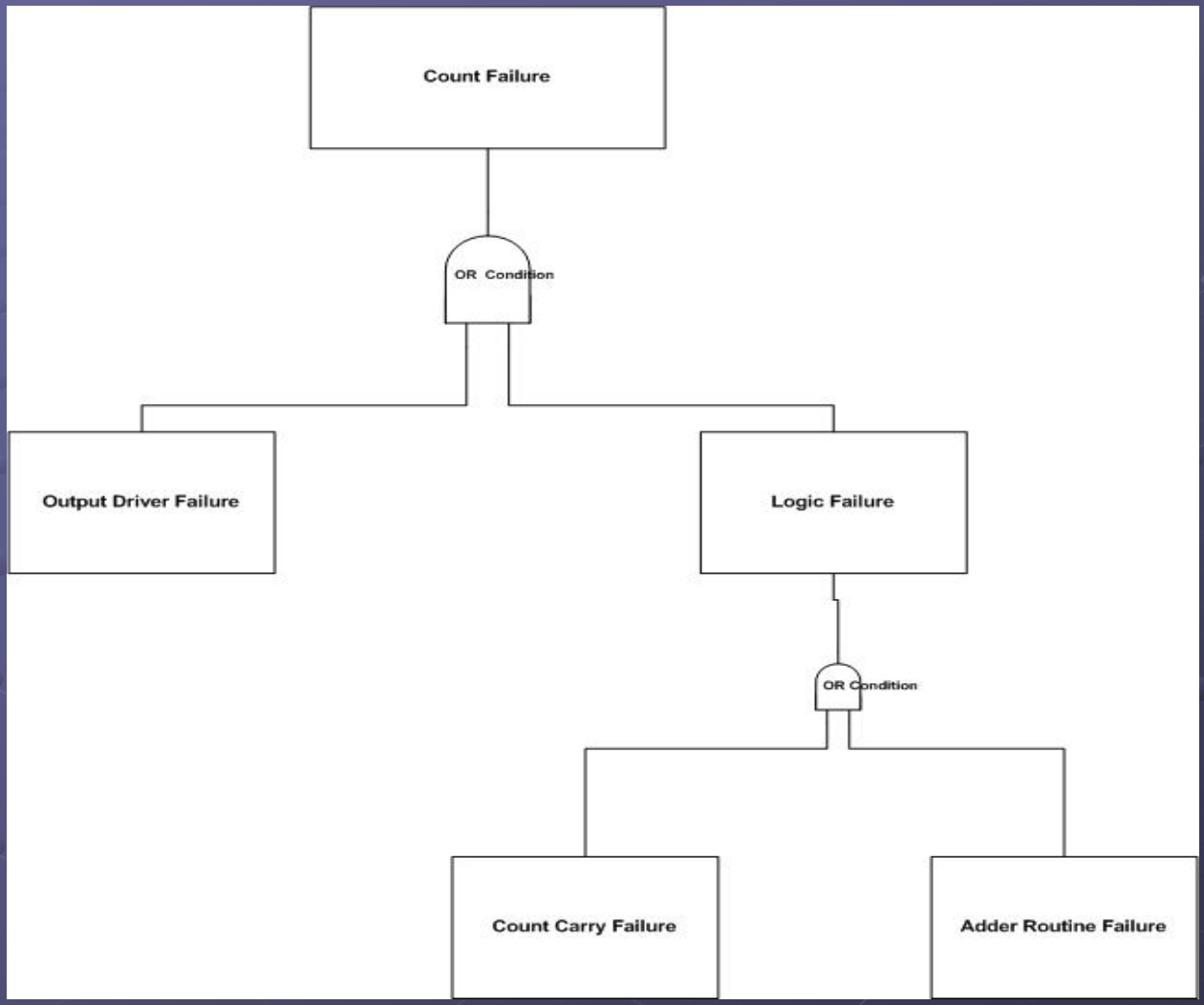


Design

One major difference between CE and software is the aspect of timing and concurrency.

- Design reviews are important.
- Independent engineer should review the design
- Confirm the design supports the requirements
- Use a coding standard
- Have and follow Best Practices

Fault Tree Analysis



Code / Implementation

- Although HDL is not true code, it shares many of the same features and attributes of software. Differences include:
  - During synthesis (compile), the design is mapped to the logic gates of the device.
  - The placement of the logic blocks within the chip, and the routing between blocks, are some of the processes that occur during implementation (link).

Ease of Coding

- Coding Standards, Code Reviews and Best Practices work well on HDLs. They allow:
  - Readability
    - Standard Signal names
    - Names do not change across boundaries
    - Common register names
  - Maintainability
  - Common naming conventions
  - Code reviews
  - Etc….

VHDL Code Example

```
library ieee;
USE ieee.std_logic_1164.all;
ENTITY reg8 IS PORT(
      clk,we:           IN std_logic;
      rdata:              IN std_logic_vector (7 DOWNTO 0);
      Asel, Bsel:    IN std_logic_vector (1 DOWNTO 0);
      Aout,Bout:    OUT std_logic_vector (7 DOWNTO 0) );
END reg8;

ARCHITECTURE one OF reg8 IS<
BEGIN
first: PROCESS (clk, we, rdata, Asel, Bsel)
      TYPE reg_array IS ARRAY(0 TO 3) OF std_logic_vector(7 DOWNTO 0);
      VARIABLE reg:reg_array(7 DOWNTO 0);
      BEGIN
            IF clk'EVENT AND clk='0' THEN
                  IF (we='1') THEN
                        CASE Asel IS
                              WHEN "00" => reg(0):=rdata;
                              WHEN "01" => reg(1):=rdata;
                              WHEN "10" => reg(2):=rdata;
                              WHEN OTHERS => reg(3):=rdata;
                        END CASE;
                  ELSE
                        CASE Asel IS
                            WHEN "00" => Aout<=reg(0);
                            WHEN "01" => Aout<=reg(1);
                            WHEN "10" => Aout<=reg(2);
                            WHEN OTHERS => Aout<=reg(3);
                         END CASE;
                         CASE Bsel IS
                            WHEN "00" => Bout<=reg(0);
                            WHEN "01" => Bout<=reg(1);
                            WHEN "10" => Bout<=reg(2);
                            WHEN OTHERS => Bout<=reg(3);
                         END CASE;
                  END IF;
             END IF;
      END PROCESS first;
END one;
```

Example of Code Review Best Practices

ID	Topic	Criteria	Yes/No/ NA	Comment
5	General	All states in a state machine are defined or invalid states are returned to a known state		
25	Clocks	Do not generate the clock using combinational logic		
27	Signals	Names shall be self-explanatory		
34	Reset	Provides an Asynchronous reset line		
36	Interfaces	All asynchronous inputs are first synchronized before use		
39	Functions	The sensitivity list contains only the signals that should cause the process to be executed		
41	Other	Are unused functions within the IP cores identified? Is the accidental execution of these functions prevented?		

Test

- While complex electronics use test benches and timing models, the idea of a well defined suite of test cases is common in both disciplines. This includes test plans, fault injection and error handling testing and verification.

Test Methodologies

- Best Practices
- Test Plans
  - Tracing to requirements
  - Feasible
  - Cover more than just success
  - Fault Injection
  - Test Verification
- Problem Trend Analysis / Tracking

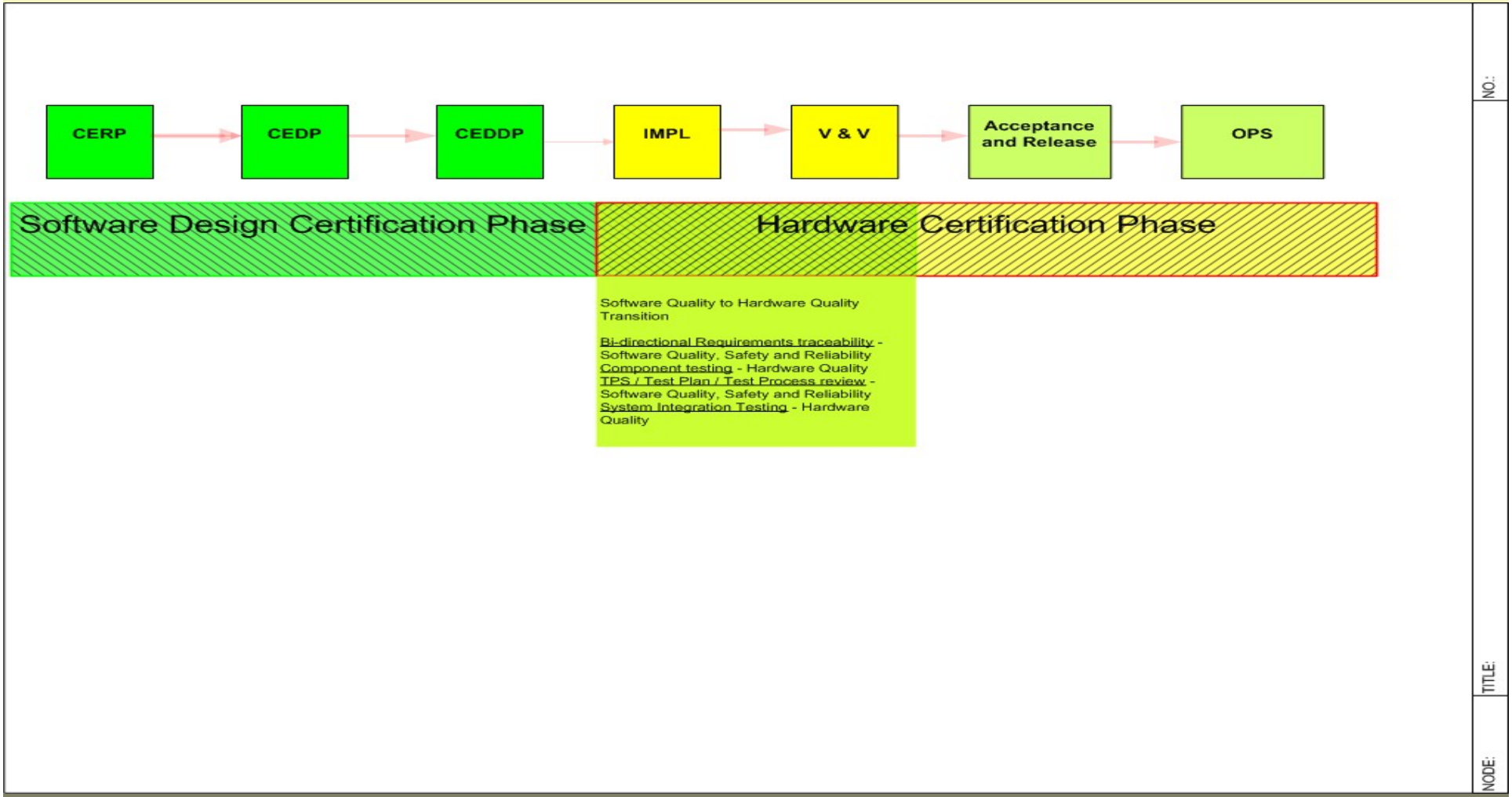
Test Plan Review Checklist

ID	Topic	Criteria	Yes/No/NA	Comment
4	General	Is the functionality of the complex electronics (CE) in off-nominal mode being tested?		
14	Interfaces	Interfaces with COTS IP modules tested		
17	Signals	For each input signal, is the range of valid data tested?		
25	Initial Conditions	Is the state of programmable memory elements upon power-up and after a reset tested?		
27	Error Handling	Have all expected errors or faults been tested to verify they are handled correctly?		
30	Timing	Has the timing of critical signals been tested?		
32	Power/ Electrical	Is the maximum allowable power and current to the CE being verified?		

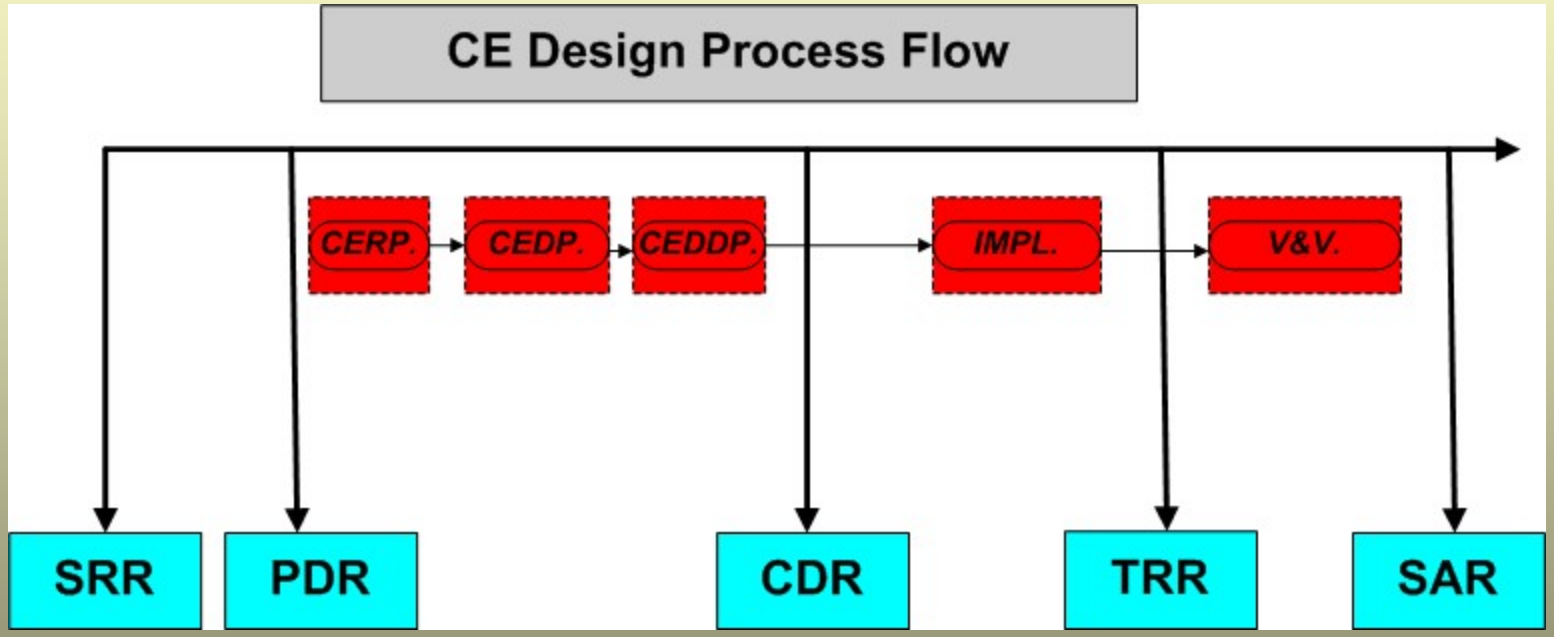
Reality Check

- Many assurance engineers, regardless of their specialty, have little understanding of the complexities of these devices. Any review done will only be to the level of knowledge of the assurance engineer.
- Three courses have been developed
  - Introduction to CE
  - CE Life Cycle
  - Assurance of CE devices
- Techniques and checklists were created to ensure quality.
  - Not all techniques/checklists used on every part.
  - Dependent on criticality and project direction.

Process Flow Example



How Complex Electronics Fits into Review Cycles



Techniques

- Change Impact Analysis
- Decision Tables/Trees
- Design Evaluation
- Design Review
- Failure Mode and Effect Analysis
- Fault Tree Analysis
- Function and Physical Configuration Audits
- Interface Analysis
- Requirements Evaluation
- Requirements Review
- Risk Analysis
- Traceability Analysis

Impact Analysis Defines

- Rational for Change
- Effects on Internal Interfaces
- Effects on External Interfaces
- Effects on Hazards
- Effects on Operations
- Potential for introducing new bugs
- Impact of Change (Minor/Major and why)
- Testing/Verifications needed
- Things to Consider

Checklists

- Planning Phase
- Requirements Phase
- Preliminary Design Phase
- Detailed Design Phase
- Implementation Phase
- Testing Phase
- Operations Phase
- Assurance Planning
- Modifications or Upgrades
- Audits (Functional Configuration, Physical Configuration and In-Process)
- Best Practices (Code Review)
- Testing (Document Review)

Requirements Phase Process Checklist

- Overview
- Entrance Criteria
- Responsible Personnel
- Process Step – Lists Analysis to use/update, documentation(to create, update, etc.), supporting processes needed
- Exit Criteria
- Documentation Status

Conclusion

- Thanks to the NASA Software Assurance Research Program which funded this Research
- Contact Information
  - Richard Plastow
  - 21000 Brookpark Road, MS 50-4
  - Cleveland, OH 44135
  - Richard.A.Plastow@nasa.gov
  - (216) 433-3431