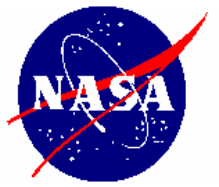


Assurance of Complex Electronics What path do we take?

Abstract

Many of the methods used to develop software bare a close resemblance to Complex Electronics (CE) development. CE are now programmed to perform tasks that were previously handled in software, such as communication protocols. 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 to develop these devices. By using standardized S/W Engineering methods such as checklists, missing requirements and “bugs” can be detected earlier in the development cycle, thus creating a development process for CE that will be easily maintained and configurable based on the device used.

Richard Plastow

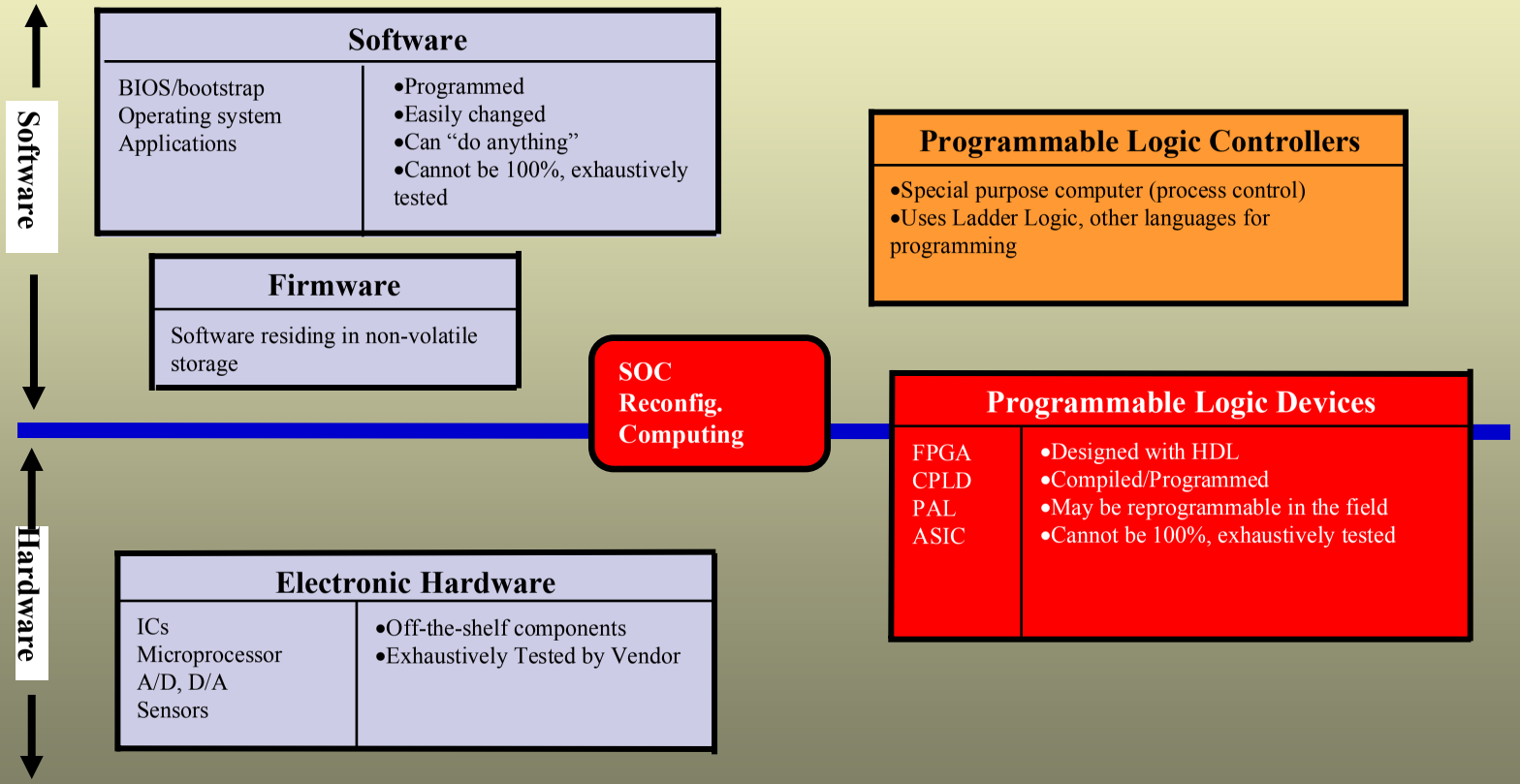
Assurance of Complex Electronics

What path do we take?

The Quandary

- Programmable Logic devices are blurring the hardware / software boundary. It is now common for Complex Electronics (CE) devices to have over one million gates and even a built in microprocessor. These devices are being used to replace software in many critical applications.

Lets take a look



How do they compare?

CE

- Asynchronous
- Costly to change
  - No updates can be done in operation
- No current standards
- Reusable
- Can not be 100% tested

Software

- Synchronous
- Easy to change
  - Patches can be done in operation
- Have defined standards
- Reusable
- Can not be 100% tested

Concerns and Issues

- ASICs and FPGAs have been used to avoid the rigors of the software approval process. (FAA DO-254)
- Complex Electronic devices are designed and programmed by engineers, often without quality assurance oversight or configuration management control of the designs. In addition, the development process may not be well defined or followed.
- High-level languages (e.g. C, C++) are now being used to define complex electronic designs (in whole or in part).
- Complex functionality cannot be completely simulated, nor the resulting chip completely tested.

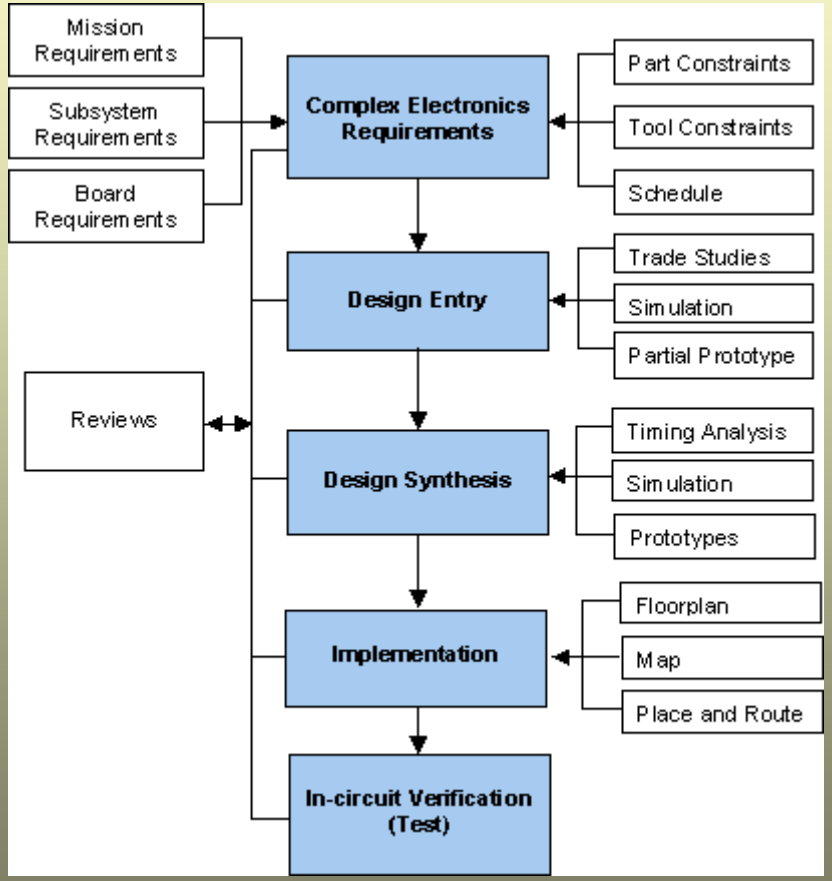
What is to be done?

- Software and Complex Electronics have many things in common.
  - Both have a Quality Assurance program
  - Both Share a common development process
  - Since the Complex Electronics device is a blend, why not use the best of both assurance worlds?

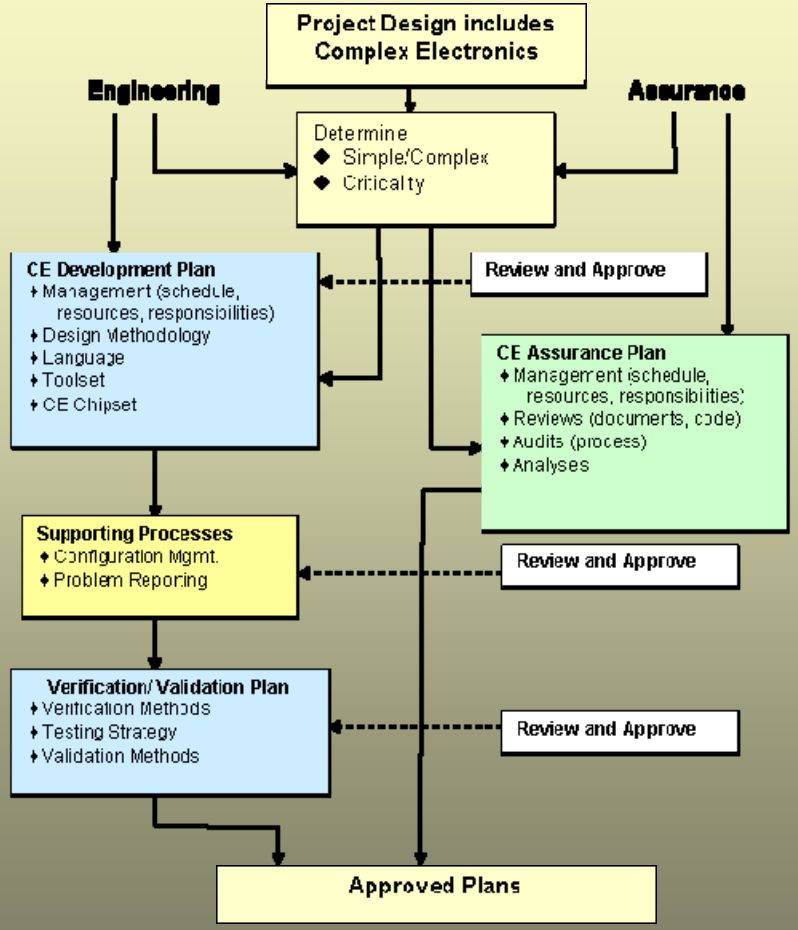
Commonality in Development process

- Software
  - Planning
  - Requirements
  - Design
  - Code
  - Test
  - Operations
- CE
  - Planning
  - Requirements
  - Design Entry / Synthesis
  - Implementation
  - Test (Verification)
  - Operations

How the Design Process For Complex Electronics should flow



Planning is where we should start



Requirements

- In a typical design, the requirements flow down from the system requirements. Development may be by the waterfall, iterative, spiral or other development methodology. Most projects, software and CE, use an iterative approach as they flow through the design process.

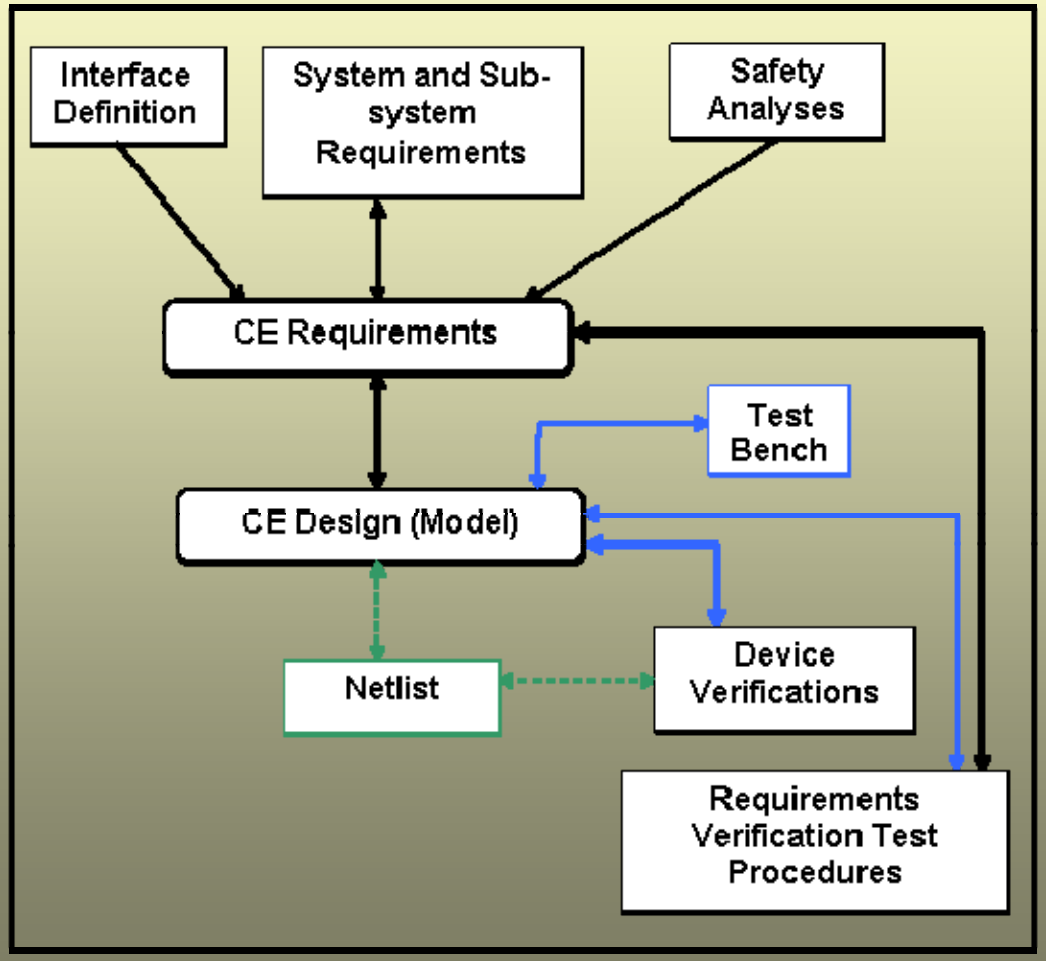
Requirements

- 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.
- Software has a well defined and robust process. While this does not guarantee success, it allows you to find and resolve many issues that may arise.
- Complex Electronics design is often started based on the engineers knowledge of the system, not defined requirements.

An Integrated Assurance Approach

- Requirements Reviews
  - Complete
  - Verifiable
  - Understandable
  - Traceable
- Interface Control Document verifications
- Fit planned hardware

Traceability Analysis



Design

- Whether you are using Uniform Modeling Language (UML), hardware description language (HDL) or some other form, this is where you define the system and it's function. One major difference between CE and software is the aspect of timing and concurrency.
- The basic premise is the same. A good design is expected.

Code / Implementation

- Although HDL is not true code, it shares many of the same features and attributes of software. The differences occur during the “compile and link” functions. 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. This process is loosely comparable to the linking step in software.
- Coding standards, code reviews, and best practices that are used by software work very well on HDL.

Ease of coding

- Coding Standards 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;
```

Test

- While complex electronics use test benches and timing models, the idea of a well define 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
- …..

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.
- Software Assurance Engineers have faced these issues and use many techniques and checklists to insure quality.

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

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)*