Automated Code Reviews for Fail-Safe Designs

MAPLD’2011
Agenda

• Fail-safe design assurance guidelines (DO-254)
• HDL coding in hardware design life cycle
• Automated HDL code reviews
Fail-Safe Design Assurance Guidelines

- RTCA/DO-254:
  - Design assurance guidelines for Airborne Electronic Hardware (AEH)
  - Accepted by the Federal Aviation Administration (FAA) in 2005
  - The goal of the standard is to ensure that AEH works reliably
  - Design Assurance Levels (DAL) A—E determine hardware design objectives
  - DO-254 projects have special requirements for tools and design flows
DO-254 HW Design Assurance Levels

- **Catastrophic** – failure prevents the safe flight and landing of the aircraft, resulting in many fatalities including the crew.

- **Hazardous** – failure reduces capability of the aircraft to fly, possibly resulting in fatal injuries but not to the crew.

- **Major** – failure reduces flight safety margins and the capability of the aircraft to the point where potential injuries could occur.

- **Minor** – failure does not significantly reduce aircraft safety, resulting in potential discomfort to passengers or crew.

- **No effect** – failure does not affect the operation of the aircraft.
DO-254 HW Design Life Cycle

• Design development activities and supporting processes:

  Planning
  (Section 4)

  Supporting Processes
  • Validation and Verification
  • Configuration Management
  • Process Assurance
  • Certification Liaison

  Hardware Design Process
  (Section 5)

  Section 5.1
  • Req. capture

  Section 5.2
  • Concept. Design

  Section 5.3
  • Detailed Design

  Section 5.4
  • Implement.

  Section 5.5
  • Product transition
DO-254 and HDL Design

- HDL Design fits in the following sections of DO-254 Life Cycle:
  - Creating HDL code (Section 5.3.2)
  - Defining coding standards/policies (Section 6-2.a)
  - Creating code artifacts and documentation (Section 10.3.2)
  - Tracing code to requirements (Section 6.2.1)
  - Managing code versions (Section 7.0)
  - Performing internal reviews and external audits (Section 6.3.3.2)
Creating HDL Code

Section 5.3.2, Detailed Design Process Activities: The detailed design data for the hardware item should be generated based on the requirements and conceptual design data. This may include assembly and interconnection data, component data, HDL, test methods and hardware-software interface data.

- HDL Design is a typical process for Detailed Design phase
- HDL describes functional behavior of the actual hardware
- Text-based entry method requires advanced editors to facilitate design development

HDL Editor

Block Diagram Editor

FSM Editor

www.aldec.com
Creating HDL Code Artifacts

Section 10.3.2, Hardware Design Representation Data is comprised of the set of drawings, documents and specifications used to build the hardware item. Typical hardware design data: conceptual and detailed design data, top-level and assembly drawings, verification and traceability data, hardware acceptance criteria, problem reports, and etc.

- Code and implementation data must be documented
- Automatic HDL Code visualization is essential

Code2Graphics

Export to PDF/HTML
Tracing HDL Code to Requirements

Section 6.2.1, Verification Process Objectives: The objectives of verification process are… Traceability is established between hardware requirements, the implementation, and the verification procedures and results.

- Implementation = HDL Code => **requirements must be linked to**:
  - HDL Code that implements the requirements
  - Verification methods that verify them
  - Verification results

- Linking is usually implemented via HDL Code “tagging” at:
  - Code level (more thorough tagging)
  - Source file level
Managing HDL Code Versions

Section 7.0, Configuration Management Process is intended to provide the ability to consistently replicate the configuration item, regenerate the information if necessary and modify the configuration item in a controlled fashion if modification is necessary.

- Project needs to be managed through the design flow:
  - Version management (SVN, Clearcase, etc)
  - Defect tracking (Mantis, Bugzilla, JIRA, etc)

**Version Management**

**Bug Tracking**
Performing Code Reviews and Audits

Section 6.3.3.2, Design Review: A design review is a method to determine that the design data and implementation satisfy the requirements. Design reviews should be performed as defined in the plan at multiple times during the hardware design life cycle. Examples are conceptual design, detail design and implementation reviews.

- Review session details (minutes, AIs) must be kept as a proof
- DO-254 project reviews:
  - SOI-1: examination of planning documents
  - SOI-2: design audit (after requirements, architecture, coding, and other internal reviews are done)
Defining HDL Coding Standards

Order 8110-105, Section 6-2.a, Verification Process: We must expect that, if they use an HDL, applicants define the coding standards for this language consistent with the system safety objectives, and establish conformance to those guidelines by HDL code reviews.

- HDL code must adhere coding standards defined by team
- Standards could be based on:
  - Industry standards (e.g. STARC, RMM)
  - Standards supplied by a vendor
  - In-house company standards
Challenges in Establishing Standards

- HDL languages are very flexible and enable for creativity in coding
- Certain coding styles and practices lead to:
  - Problems in the downstream design phases
  - In-hardware errors or malfunctions (*safety threats!*)
- What standards to select and deploy?
  - FAA didn't set any formal HDL coding standards
  - Traditional standards are generic (*STARC, RMM*)
  - In-house standards may be impractical
Hardware Design Processes, Order 8110-105, Section 6-2.a: We must expect that, if they use an HDL, applicants define the coding standards for this language consistent with the system safety objectives, and establish conformance to those guidelines by HDL code reviews.

- **Manual vs. automatic reviews:**

  - Manual
  - Automatic
Benefits of Automated Reviews

- Reduces cost of review vs. manual labor!
- Based on comprehensive knowledge base of industry best practices and design expertise
- Code is more consistent and efficient vs. manual review approach
- Reviews can be performed more often – good match with DO-254 philosophy
- Integrated debug environment for efficient design analysis and documentation
Aldec DO-254 HDL Coding Standards

• Aldec DO-254 standard:
  ◇ Based on the feedback from real customers *(DO-254 programs)*
  ◇ Good foundation of guidelines for any design *(Not DAL A/B only)*

• Basic rule groups:
  1. Structure and portability *(Data types, naming, coding, statements...)*
  2. Downstream checks *(Racing, sensitivity lists, clocks & resets, bit widths...)*
  3. Error-prone patterns *(Subprograms, registers, interconnections, hazardous blocks...)*
Group #1: Structure & Portability

- Code structure and readability for efficient reuse, examples:
  - Declare one object per line and always add comments
  - Port description order should follow a pattern
  - Avoid using hard-coded numbers for characteristics that may change
  - Do not use similar identifiers even in different namespaces
  - Use VHDL data types properly
  - All objects declared in the code must be used
  - Global design parameters must be defined in a package
  - ...
Group #1: Rule Sample (Reuse)

Avoid using hard-coded numbers for characteristics that may change

Main message –
Hard-coded number(s) used in the process.

Detail message –
“00000000” should be parameterized.

```vhdl
process (CLOCK, RESET, ENABLE)
begin
  if RESET='0' then
    FIBOUT <= "00000001";
    PREV_FIB <= "00000000";
  elsif falling_edge(CLOCK) and ENABLE='1' then
    PREV_FIB <= FIBOUT;
  case (FIBOUT="00000000") and (PREV_FIB="00000000") is
    when true => FIBOUT <= "00000001";
    when others => FIBOUT <= FIBADD;
  end case;
  else
    null;
  end if;
end process;
```
Group #2: Downstream Stages

- Problems that normally surface later in design flow, examples:
  - Do not use non-synthesizable subset of the HDL language
  - Do not use combinatorial feedbacks (*racing conditions*)
  - Avoid unreachable conditions (*code that will never be executed*)
  - Define all the necessary signals in sensitivity lists (*combinatorial process*)
  - Avoid internally generated clocks unless they are properly isolated
  - Gated clocks can be used only at a top level (*ASIC-specific*)
  - ...

www.aldec.com
Group #2: Rule Sample (Synthesis)

Avoid unintentional latch inference

module calc_sfk( S, AIN, BIN, Y, Z );

input [1:0] S;
input [7:0] AIN, BIN;
output reg [7:0] Y, Z;

always @(S or AIN or BIN) begin
    if (S == 2'b00) begin
        Y = AIN & BIN;
        Z = 0;
    end
    elseif (S == 2'b01) begin
        Y = 0;
        Z = AIN | BIN;
    end
    elseif (S == 2'b10) begin
        Y = 0;
    end
    else begin
        Y = 1;
        Z = 0;
    end
end
endmodule

Main message –
The ‘always’ process description infers latch for 1 signal.

Detail message –
Signal "Z" is not assigned in all cases.
Group #3: Error-Prone Patterns

- Design patterns that are prone to problems and errors, examples:
  - Empty blocks should not be used
  - Do not describe multiple independent conditions in a process
  - Avoid unconnected and misused ports
  - All referenced signals should have drivers
  - Avoid using hazardous synchronization schemes \((\text{process-level})\)
  - Do not locate logic between asynchronous clock domains \((\text{metastable conditions})\)
  - ...

module add_sub( CLK_1, CLK_2, ADD_SUB, ARG_A, ARG_B, RES );
    input CLK_1, CLK_2;
    input ADD_SUB;
    input [BIT_LENGTH-1:0] ARG_A, ARG_B;
    output [BIT_LENGTH-1:0] RES;
    wire [BIT_LENGTH-1:0] res_add, res_sub, trnsmt;
adder adder(
            .A ( ARG_A ),
            .CLK( CLK_1 ),
            .B ( ARG_B ),
            .RES( res_add )
      );
subtractor subtractor(
            .CLK( CLK_1 ),
            .A ( ARG_A ),
            .B ( ARG_B ),
            .RES( res_sub )
      );
div2 div2(
            .CLK( CLK_2 ),
            .ARG( trnsmt ),
            .RES( RES )
      );
assign trnsmt = ( ADD_SUB )? res_add : res_sub;
endmodule

Comb. logic is present on clock domains crossing path!

Violation message –

The "first_stage_ff“ flip-flop obtains data from another clock domain through combinational logic – risk of functional errors caused by propagation of incorrect values/glitches.
Running the Automated Review

• Design review process is governed by the **Flow Manager** in ALINT:

```
Group #1 -> Phase #1 -> Violation DB #1

Criteria #1
```

```
Group #N -> Phase #N -> Violation DB #N
```

```
Criteria #N
```

![Flow Manager](Image)
Analyzing Review Phase Results

• Violation database is generated at each phase:
  ♦ Dedicated Violation Viewer tool is available for violations analysis
Documenting Review Phase Results

- Export to a spreadsheet
  - Document fixed violations (how, what files modified, etc)
  - Document irrelevant violations, if any

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File</td>
<td>Line</td>
<td>Origin</td>
<td>Rule</td>
<td>Violation</td>
</tr>
<tr>
<td>2</td>
<td>gen.x</td>
<td>24</td>
<td>Module “GEN”</td>
<td>ALDEC.2204</td>
<td>Referencing signals before assigning value to them may cause unexpected behavior.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>30</td>
<td></td>
<td>ALDEC.2204</td>
<td>Signal “DOINT[3]” is referenced before assigned.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>31</td>
<td></td>
<td>ALDEC.2204</td>
<td>Signal “DOINT[0]” is referenced before assigned.</td>
</tr>
</tbody>
</table>

- Quality report
  - Access design quality in terms of violated and non-violated rule weights
HDL Coding for Fail-Safe Designs

- As far as HDL Coding is concerned, Aldec tools enable:
  - Implementing structured and repeatable design flow
  - Creating HDL code and managing configurations
  - Defining and checking HDL coding guidelines
  - Creating HDL code artifacts and performing reviews
  - Tracing HDL code to requirements

- Automated HDL code reviews with Aldec ALINT:
  - Executable flows based on predefined or custom coding standards
  - Tools for results analysis and documentation
  - Three preset groups recommended for DO-254 designs