ABSTRACT

This article highlights new FAA standards and how they affect developers using Model-Based Design. ARP4754A and DO-178C are the focus, especially DO-331, the Model-Based Design supplement to DO-178C.

INTRODUCTION TO STANDARDS

Few industries place more importance on verification, or prescribe more process guidance, than civil aviation. The FAA and its European equivalent, EASA, provide guidance using standards such as ARP 4754 for aircraft systems and DO-178B for flight software. These standards are often used outside of civil aviation, in whole or in part, for applications including military aircraft and land vehicles. Adoption for UAV programs is rapidly growing because of the FAA’s recent decision to require UAS and OPA certification via FAA Order 8130.34A. UAV systems are heterogeneous, and not restricted just to flight software. Therefore, other standards are used such as DO-254 for hardware and DO-278 for ground software.

However, these standards are more than a decade old and are showing their age. For example, they lack guidance on modern development and verification practices such as Model-Based Design, object-oriented technologies, and formal methods. So the FAA and EASA have worked with aircraft manufacturers, suppliers, and tool vendors including MathWorks to update standards based on modern technologies. Rather than significantly modify the standards, they created technology supplement documents.

TRANSITIONING TO NEW STANDARDS

ARP4754A

ARP4754A addresses the complete aircraft development cycle from requirements through verification. It discusses requirements, integration, and verification for three levels of abstraction: aircraft, systems, and item. An item is defined as a hardware or software element having bounded and well defined interfaces. According to the standard, aircraft requirements are allocated to system requirements, which are then allocated to item requirements.

The fact that ARP4754A addresses allocation of system requirements to hardware and software components is significant to UAV developers, especially suppliers. Some suppliers may have claimed that subsystem development was beyond the scope of ARP4754, even for complex subsystems containing hardware and software, but not anymore. ARP4754A also more clearly refers to DO-178 and DO-254 for item design. In fact, the introductory notes for ARP4754A acknowledge that its working groups coordinated with RTCA special committees to ensure that the terminology and approach being used are consistent with those being developed for the DO-178B update [DO-178C].

DO-178C

Not surprisingly, one of the first changes new in DO-178C is an explicit mention of ARP4754A in Section 2: System Aspects Relating to Software Development, to further align the standards:

This section discusses those aspects of the system life cycle processes necessary to understand the software life cycle processes. System life cycle processes can be found in other industry documents (for example, SAE ARP4754A).
Clarification updates aside, such as the one noted above, DO-178C does not differ significantly from DO-178B, at least at first glance. In fact, a casual reader might miss an item mentioned in Section 1.4: How to Use this Document:

One or more supplements to this document exist and extend the guidance in this document to a specific technique... If a supplement exists for a specific technique, the supplement should be used to add, delete, or otherwise modify objectives, activities, explanatory text, and software life cycle data in this document to address that technique, as defined appropriately in each supplement.

In other words, the standard's big changes are captured in the supplemental documents, specifically those listed in Table 1, such as RTCA DO-331, Model-Based Development and Verification Supplement to DO-178C and DO-278A.

INTRODUCTION TO MODEL-BASED DESIGN

With Model-Based Design, UAV engineers develop and simulate system models comprised of hardware and software using block diagrams and state charts, as shown in Figures 1 and 2. They then automatically generate, deploy, and verify code on their embedded systems. With MATLAB® and Simulink® from MathWorks, you can generate code in C, C++, Verilog, and VHDL languages, enabling implementation on MCU, DSP, FPGA, and ASIC hardware. This lets system, software, and hardware engineers collaborate using the same tools and environment to develop, implement, and verify systems. Developers of complex UAV systems need to leverage all the technologies listed above and be mindful of the governing standards such as those listed in Table 1.

A key benefit of Model-Based Design is early verification. Verification starts as soon as models are created and simulated using tests based on high-level requirements. A common verification workflow is to reuse the simulations tests throughout Model-Based Design as the model transitions from a system model to software model to source code to executable object code using code generators, cross-compilers, and synthesis tools.

An in-the-loop testing strategy is often used as described below and summarized in Table 2:

1. Simulation test cases are derived and run on the model using model-in-the-loop testing (MIL).
2. Source code is verified by compiling and executing it on host computer using software-in-the-loop testing (SIL).
3. Executable object code is verified by cross-compiling and executing it on the embedded processor or an instruction set simulator using processor-in-the-loop testing (PIL).
4. Hardware implementation is verified by synthesizing HDL and executing it on FPGA using FPGA-in-the-loop testing (FIL).
5. Embedded system is verified and validated using original plant model using hardware-in-the-loop testing (HIL).

<table>
<thead>
<tr>
<th>Test</th>
<th>Device under test</th>
<th>Device execution platform</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIL</td>
<td>Design model</td>
<td>Simulink</td>
<td>Verify design</td>
</tr>
<tr>
<td>SIL</td>
<td>Source code</td>
<td>Host computer</td>
<td>Verify source code</td>
</tr>
<tr>
<td>PIL</td>
<td>Executable code</td>
<td>Embedded processor or Instruction set simulator</td>
<td>Verify object code</td>
</tr>
<tr>
<td>FIL</td>
<td>Hardware implementation</td>
<td>FPGA or HDL simulator</td>
<td>Verify hardware implementation</td>
</tr>
</tbody>
</table>
Table 2: Simulation-based test summary

**TRANSITIONING TO NEW STANDARDS USING MODEL-BASED DESIGN**

**ARP4754A**

ARP4754A recommends the use of modeling and simulation for several process integral activities involving requirements capture and requirements validation.

ARP4754A Table 6 recommends (R) analysis, modeling and simulation (tests) for validating requirements at the highest Development Assurance Levels (A and B). For Level C, modeling is listed as one of several recommendations. While ARP4754 made similar recommendations, ARP4754A provides more insight and states that an representative environment model, such as the plant model shown in Figure 1, is an essential part of a system model.

Model use for requirements validation typically uses a model of the environment of a system being developed, which is interfaced to a prototype of a design solution for those requirements. An environment model that is representative of the environment of the system being developed provides a high degree of functional coverage in exercising either a simulated or real system.

Also new with ARP4754A is that a graphical representation or model can be used to capture system requirements. The standard now also notes that a model can be reused later for software and hardware design.

Models used to capture requirements and then directly used to produce embedded code (Software or HDL) come within the scope of DO-178B/ED-128 and DO-254/ED-80 from the time that certification credit is to be taken until the software or hardware is returned to the system processes for system verification.

If you use models to capture requirements, ARP4754A recommends you consider the following:

1. Identify the use of models/modeling
2. Identify the intended tools and their usage during development
3. Define modeling standards and libraries

MathWorks offers additional verification capabilities beyond simulation testing described in Table 2. These including requirement tracing, model standard checking for DO-178C, model-to-code structural equivalence checking, and robustness analysis using formal methods. For UAVs, rigorous verification that includes multiple verification technologies is paramount given their autonomous nature and system complexity. MathWorks provides verification tool qualification kits and workflow guidance regarding the full use of Model-Based Design for DO-178 and related standards.

**DO-178C and DO-331**

A long-standing issue with DO-178B for practitioners of Model-Based Design is the uncertainty in mapping DO-178B objectives to Model-Based Design artifacts. Addressing this mapping was a main goal of the DO-178C Sub-Group (SG-4) focused on Model-Based Design. No single mapping sufficed, so several mappings are provided in DO-331. Some include the concept of a Specification model but most Simulink users do not employ that approach. The other concept of a Design model is a more natural mapping for Simulink users.

Table 3 shows popular mappings of life-cycle data to model usage based on DO-331, Section MB.1.6.3.

<table>
<thead>
<tr>
<th>Process that Generates Life-Cycle Data</th>
<th>Model-Based Design Example 1</th>
<th>Model-Based Design Example 4</th>
<th>Model-Based Design Example 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Requirement and System Design Processes</td>
<td>Requirements allocated to software</td>
<td>Requirements from which the model is developed</td>
<td>Requirements from which the model is developed</td>
</tr>
<tr>
<td>Software Requirement and Software Design Processes</td>
<td>Requirements from which the model is developed</td>
<td>Design model</td>
<td>Design model</td>
</tr>
<tr>
<td>Software Coding Process</td>
<td>Source code</td>
<td>Source code</td>
<td>Source code</td>
</tr>
</tbody>
</table>

Table 3: Example mappings of life-cycle data to model usage.

The essence of this table is the following:

1. A model can be used for design (system and/or software) and should be developed using requirements external to the model (e.g., textual document or requirement database).
2. Source code can be generated directly from the design model (by hand or automatically).
Of course, with 125 pages, DO-331 has a lot more to offer than is shown here, but Table 3 ties together the basic concepts of Model-Based Design for system and software development. Example 5 does this particularly well in that shows that a model used initially for system design is elaborated and reused for software design and code generation.

For example, the controller shown as a component in the system model in Figure 1 and by itself as a software model in Figure 2, is used during system design, reused as an entry point for software design, elaborated during detailed software design (for example by discretizing continuous time blocks and changing double precision data to single precision), and then used as input for embedded code generation. The test cases used for system requirement validation likewise are reused on the model, source code, and executable object code to perform functional testing and collect coverage metrics.

While not advocating for any particular mapping, the approach of using and reusing models for systems and software design along with code generation has provided Simulink and Embedded Coder™ users significant benefits including reduced cost and improved quality. It is nice to see that this same approach is now clearly acknowledged as an acceptable means to certification by the governing standards.

***

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.