Tool Support for Verification of Software Timing and Stack Usage for a DO-178B Level A System

Eng. Daniela Cristina Carta
daniela.carta@embraer.com.br

Dr. Ian Broster
ianb@rapitasystems.com

Eng. Felipe Kamei
felipe.kamei@embraer.com.br

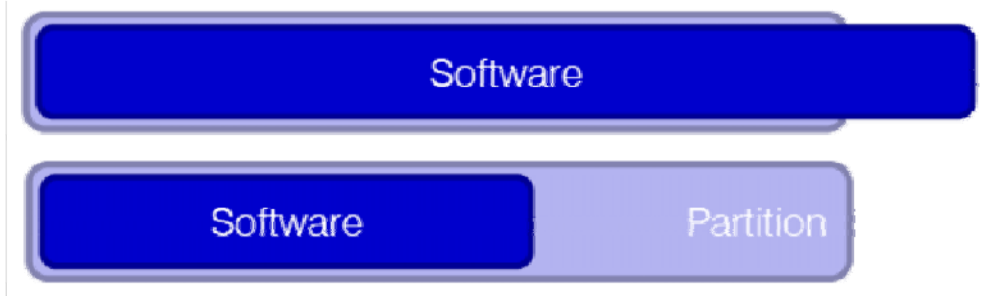
Will Lunniss
wlunniss@rapitasystems.com

Topics

- Importance of determining WCET
- Flight Control System (FCS) software application
- Rapita Verification Suite (RVS) overview
- Method to obtain worst case execution time (WCET) and worst case stack usage
- Results achieved

Timing and Stack

- Compliance with timing and memory requirements
- DO-178B: obtain the worst case timing and the stack usage
- Optimizing the usage of resources, such as CPU usage and stack memory



Compliance to DO-178B

- Section 6.3.4: *“Reviews and Analyses of the Source Code”*
- Tests execution:

- Functional analysis

- Timing analysis
- Stack usage analysis

Tested software features

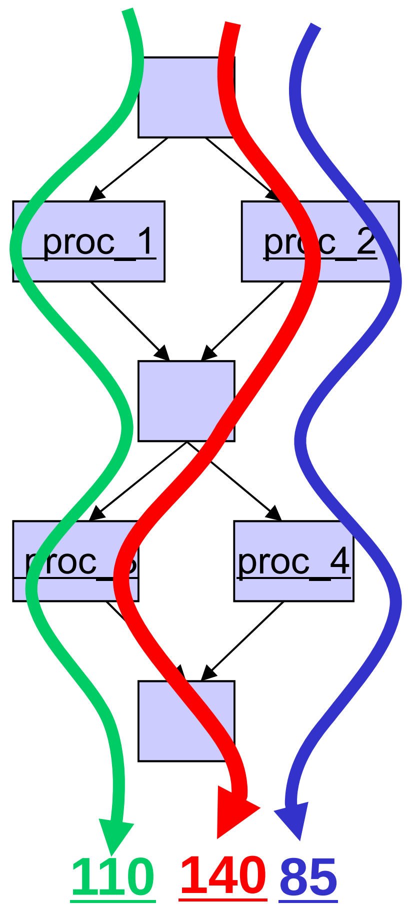
- Embraer R&D project
  - Generate and exercise processes for the development of critical aircraft system and software
- Proof of Concept: Flight Control System (FCS) - Level A software
  - C programming language
  - 73,000 lines of code
  - 7448 PowerPC microcontroller
  - Operating system compliant with avionics standards
- Initial requirements:
  - WCET < 5 ms
  - Stack usage < 20,000 bytes

Rapita Verification Suite

- Rapita Verification Suite (RVS):
  - RapiTime: on-target verification of software timing
  - RapiCover: code coverage
  - RapiSafeStack: stack usage (prototype)
- Automation:
  - Software instrumentation
  - Execution time and stack usage measurement
  - Worst case analysis

Worst Case Execution Time

- Structural code analysis is performed
- Measures time from test cases execution on target
- Determines worst-case path, worst-case execution time and many other metrics



No need of a test case that takes to the worst-case path (reduction of effort)

User opinion

- Allows setting a level of instrumentation suitable for each procedure
- No need of modification on building environment
- Generates code structure and call tree
- Performs measurements in the application running on the real target
- Captures and extracts execution data
- Generates a rich report for user analysis
  - Comparison with resource usage requirements
  - Could support certification argumentation
  - Optimization strategies

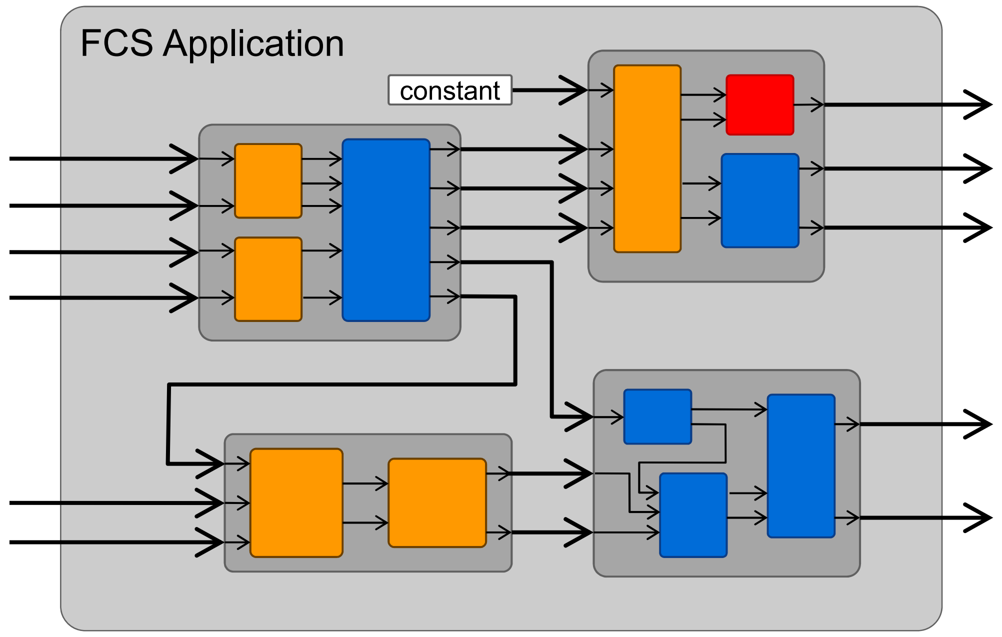
Tool Qualification

- RVS qualification according DO-178B would be necessary
- Will be qualified as a verification tool
  - Using qualification Kit from Rapita Systems
- Set of documentation for qualification activities compliant to DO-178B
- Complemented with tool user activities in the user environment

RVS integration with FCS

- Level of instrumentation
  - Timing analysis: every branch, start and end of functions
  - Stack analysis: start and end of functions
- Timing analysis: 13,000 instrumentation points
- Stack analysis: 152 instrumentation points
- Operating system calls measured end-end (as “black boxes”)

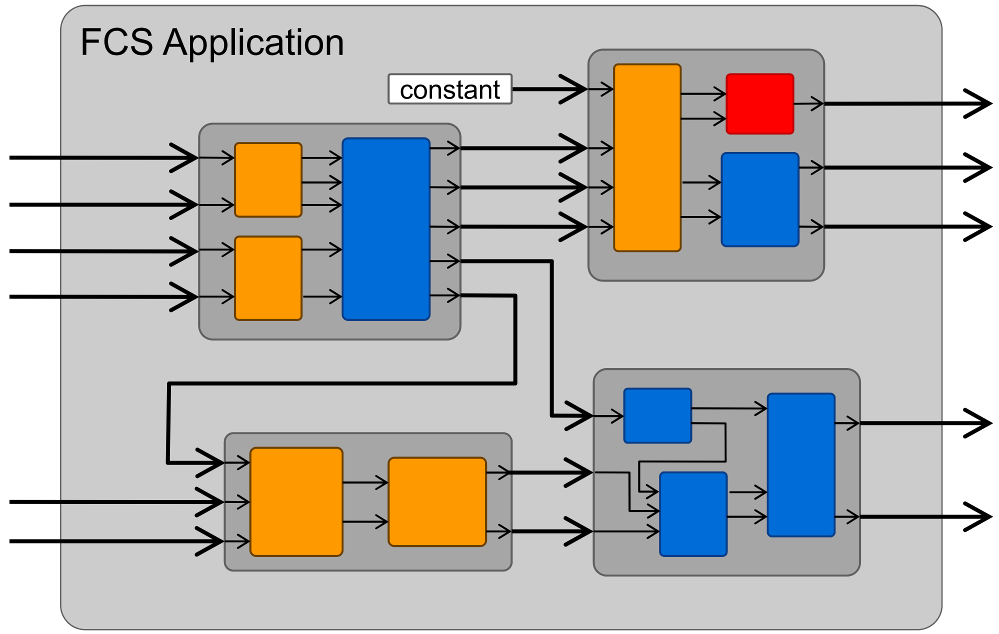
Test Scenarios



🟧 Components not tested according to formal process
🟥 Components not feasible (deactivated code)
🟦 Components tested according to formal process (MC/DC coverage)

- Functional verification and structural coverage analysis performed only internally (component level)
- Formal process: components selected to fully exercise MC/DC
- Verification for the whole application performed only at system level
- Functional test case scenarios for the whole application not available

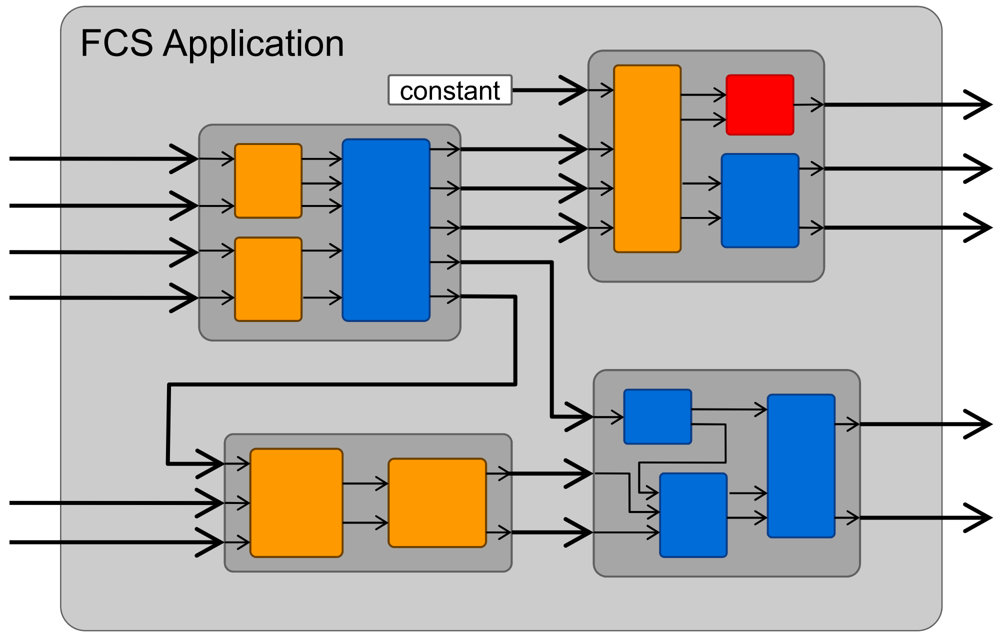
Test Coverage



🟧 Components not tested according to formal process
🟥 Components not feasible (deactivated code)
🟦 Components tested according to formal process (MC/DC coverage)

- Created about 785 test scenarios (500 input parameters)
- Instrumentation points coverage
  - Timing analysis: ≈ 77%
  - Stack analysis: 100%
- Deactivated code: time and stack usage not evaluated

Alternative testing approach

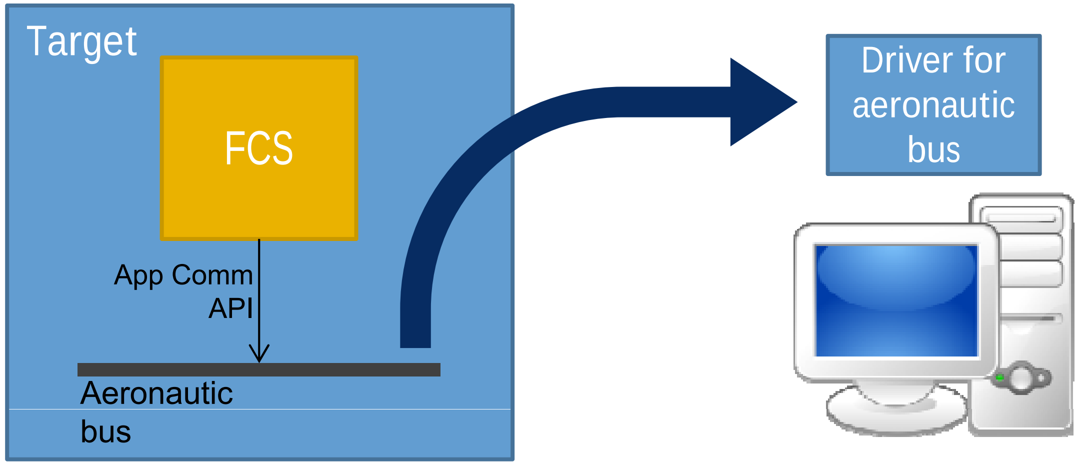


🟧 Components not tested according to formal process
🟥 Components not feasible (deactivated code)
🟦 Components tested according to formal process (MC/DC coverage)

- Alternative approach: exercising application's internal components separately
- Test cases (and drivers) prepared for formal functional verification to be reused (saved effort)
- Considered as standalone software
- Justification is required for analysis performed in modified application

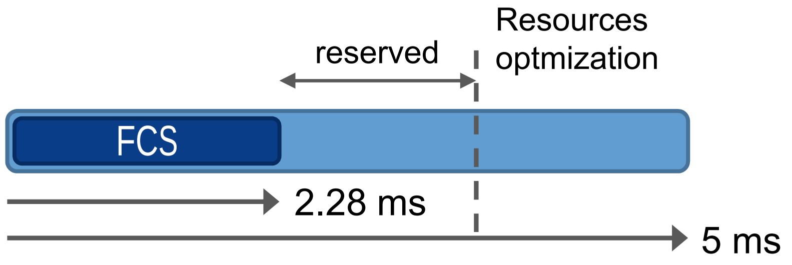
Trace data extraction

- External communication only available through aeronautic bus
- Data recorded in a buffer
- After tests execution buffer is written using communication APIs
- Network configured to route data from target to PC

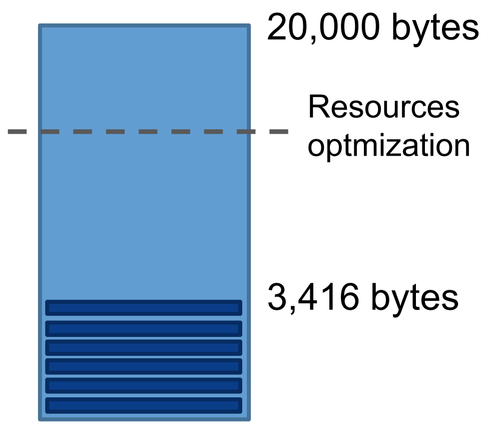


Results

Worst Case Execution Time



Worst Case Stack Usage



Conclusions

- Task can be repeated easily for new analysis
- Method considered efficient
- A more accessible method to extract trace data should be considered earlier in the project
- A test set that exercises most of the code branches is needed
- Analysis can be performed in components separately
  - Different execution conditions must be evaluated, so measurements can be considered accurate
- Both approaches considered acceptable by DERs

FOR THE JOURNEY
EMBRAER