Building Scalable Safety Solutions for Embedded Systems

Daniel Kästner
AbsInt GmbH

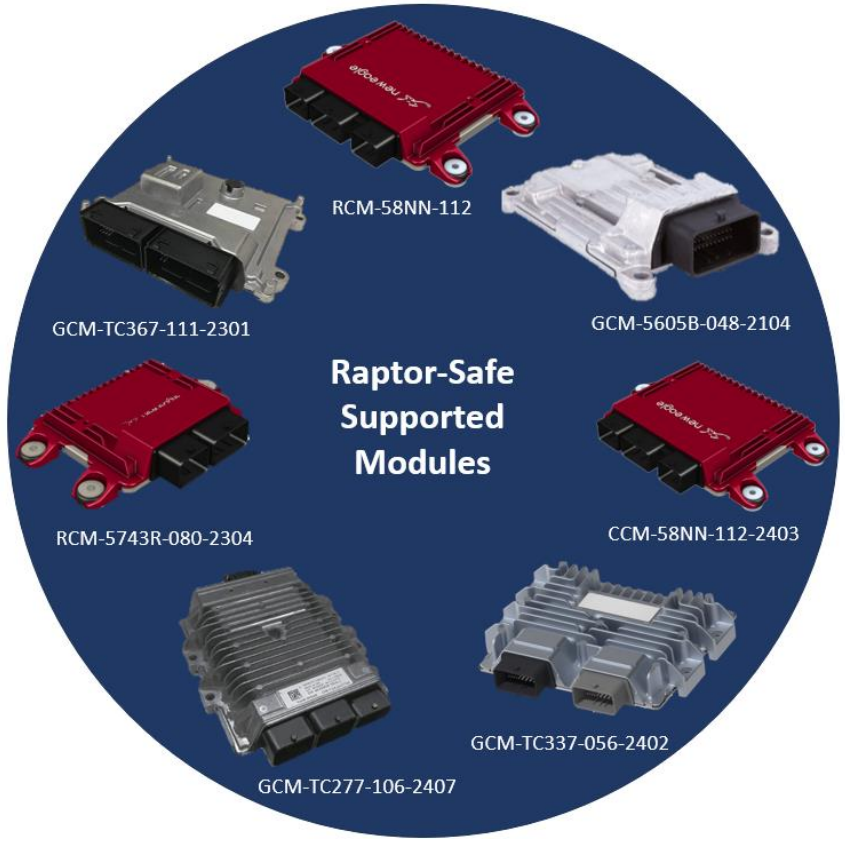
Parker Mosman, Marshal Stewart
New Eagle

2025

AbsInt

Overview of Raptor-Safe

- Easily run advanced static analysis on Raptor-Dev applications
- One license covers every New Eagle ECU
- Stack Check, Time Check, Astrée

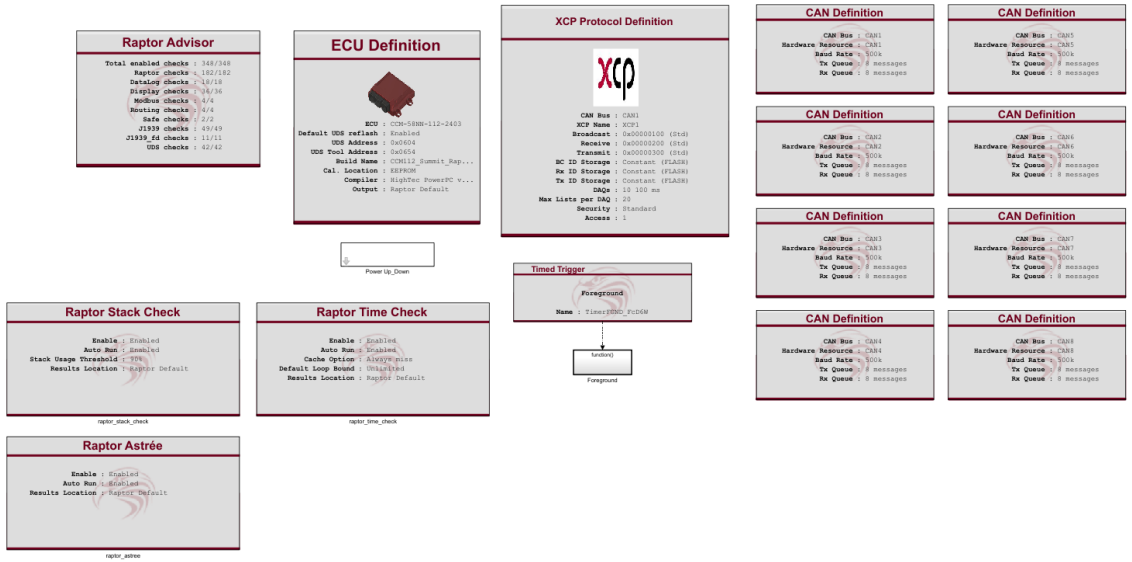


Model-based Software Development

- Application graphically specified by data flow diagrams and/or finite state machines
- Model is software specification and has executable semantics

➢ Automated & integrated development tools:

- automatic target code generation (typically C code)
- automatic simulation
- automatic model-based testing



- BUT: Higher level of abstraction than with programming in C.

➢ What about timing, memory consumption, runtime errors, integration issues?

➢ Automated & integrated development tools:

☞ Static analysis for runtime errors, worst-case stack usage and worst-case execution time (WCET)

Functional Safety

- Demonstration of functional correctness
  - Functional requirements are satisfied
  - ➢ Automated and/or model-based testing
  - ➢ Formal techniques: model checking, theorem proving
- Satisfaction of safety-relevant quality requirements
  - No runtime errors (e.g. division by zero, overflow, invalid pointer access, out-of-bounds array access)
  - Resource usage:
    - Timing requirements (e.g. WCET, WCRT)
    - Memory requirements (e.g. no stack overflow)
  - Robustness / freedom of interference (e.g. no corruption of content, incorrect synchronization, illegal read/write accesses)
  - Compliance with the software architecture, data and control coupling
  - ➢ Insufficient: Tests & Measurements
    - No specific test cases, unclear test end criteria, no full coverage possible
  - ➢ **Static analysis**
    - ⇨ Formal technique (sound): Abstract Interpretation – no defect missed

REQUIRED BY DO-178B / DO-178C / ISO-26262, EN-50128, IEC-61508

REQUIRED BY DO-178B / DO-178C / ISO-26262, EN-50128, IEC-61508

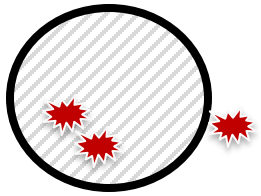
+ Security-relevant
ISO 21434, DO-356A, …

- Code Guideline Checking
- Runtime Error Analysis / Data & Control Flow Analysis / Data and Control Coupling
- Code Metrics
- WCET Analysis
- Stack Usage Analysis

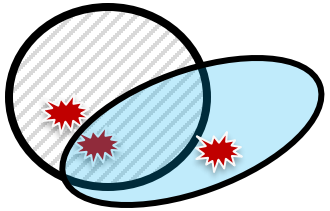
Static Program Analysis

- Computes results only from program structure, without executing the software.
- Categories, depending on analysis depth:
  - Syntax-based: Coding guideline checkers (e.g. MISRA C)
  - Semantics-based

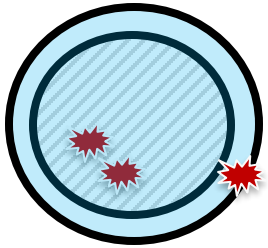
Question: Is there an error in the program?
- False positive: answer wrongly "Yes"
- False negative: answer wrongly "No"

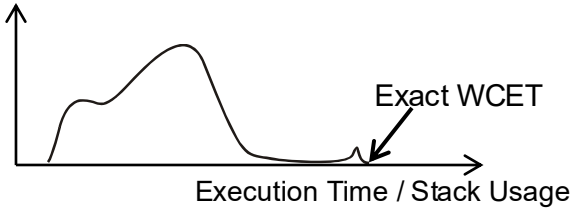


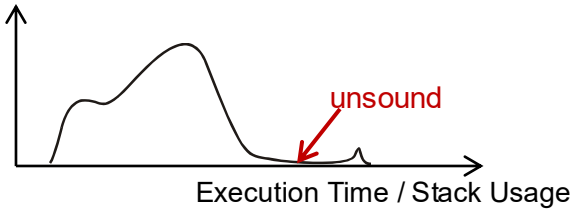
- Unsound: Bug-finders / bug-hunters.
  - False positives: possible
  - False negatives: possible



☑Sound / Abstract Interpretation-based
  - False positives: possible
  - No false negatives ⇨ Soundness
    No defect missed

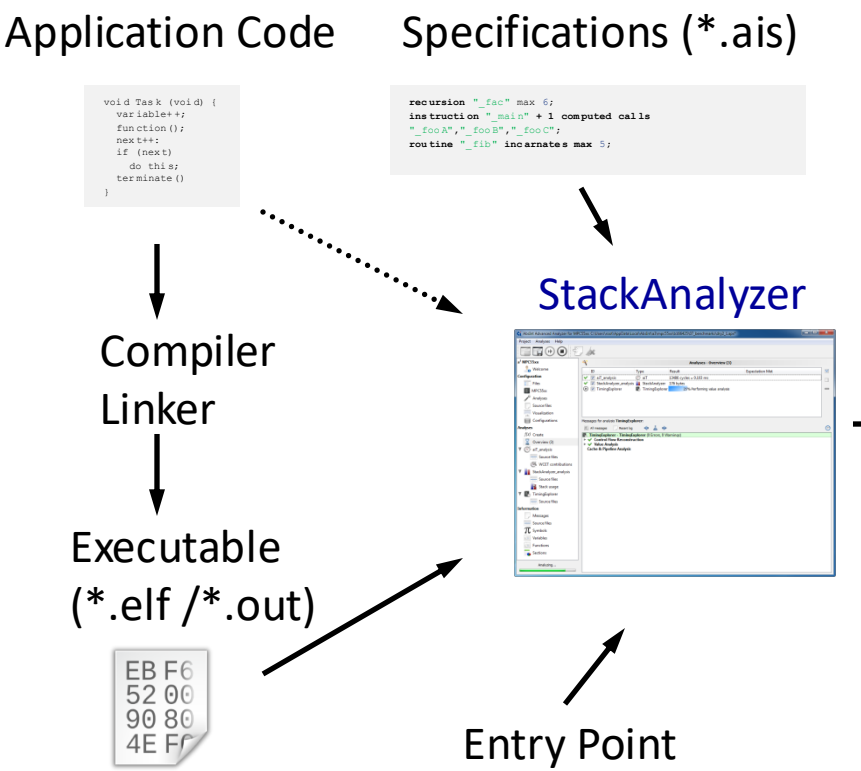




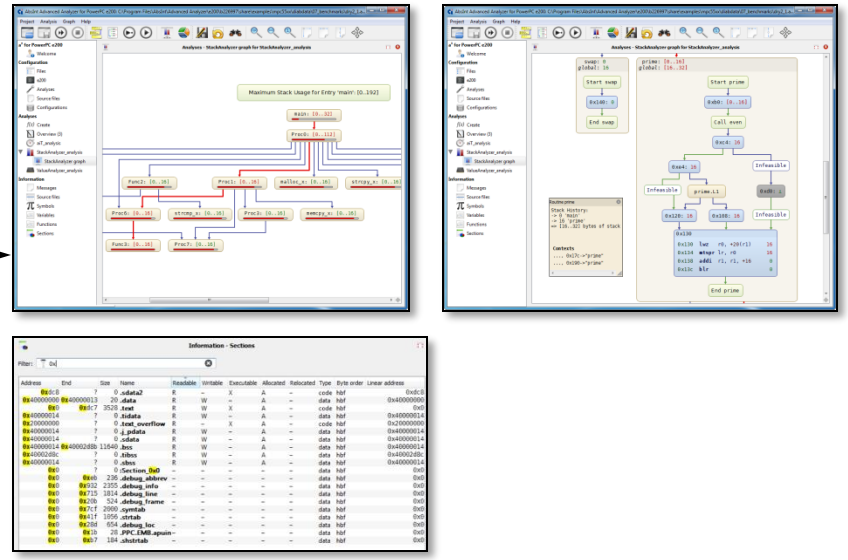


Worst-Case Stack Usage Analysis

- Abstract Interpretation-based static analysis at binary code level
- StackAnalyzer computes safe upper bounds of the stack usage of the tasks in a program for all inputs



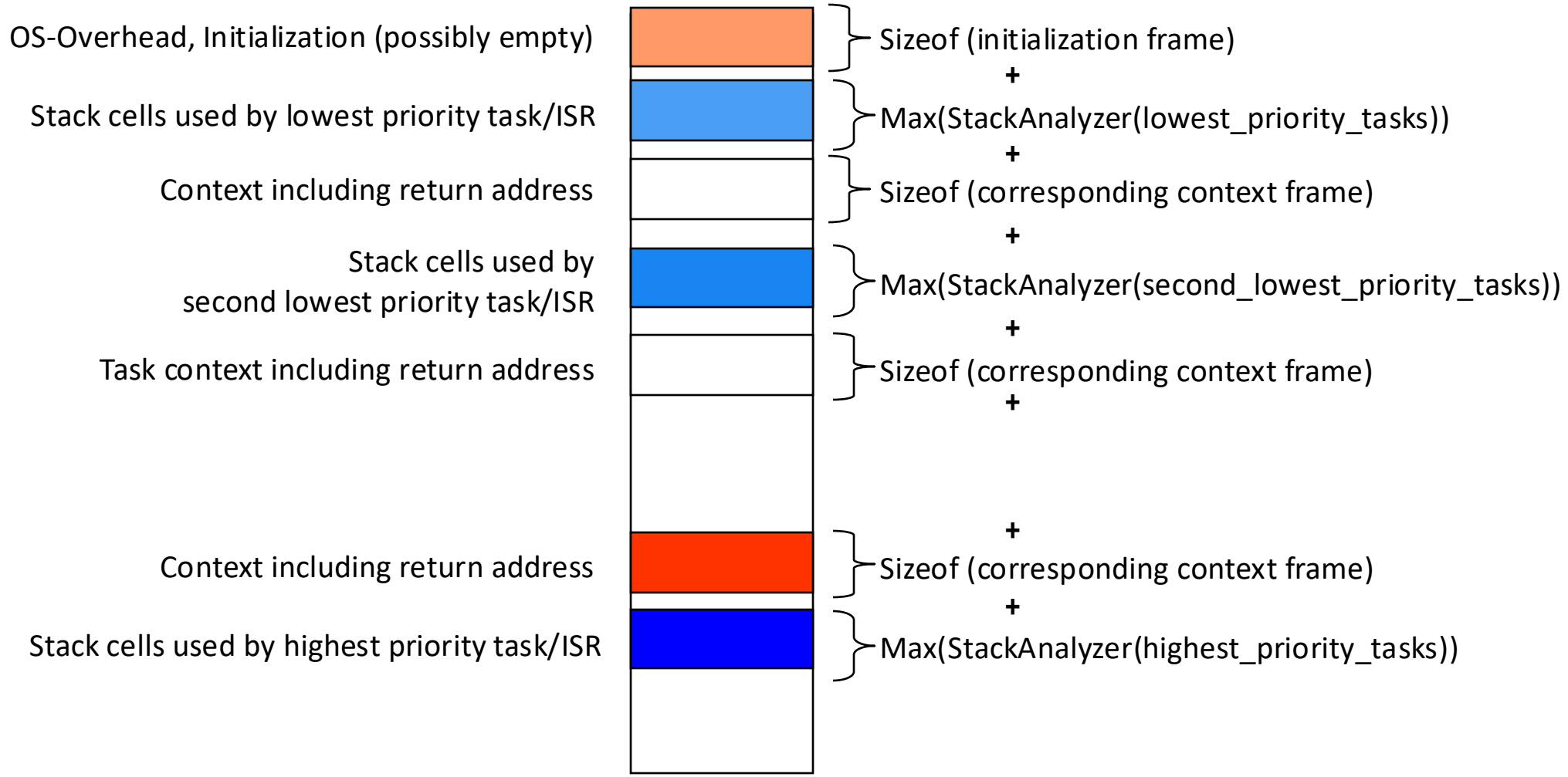
Worst Case Stack Usage
+ Visualization, Documentation



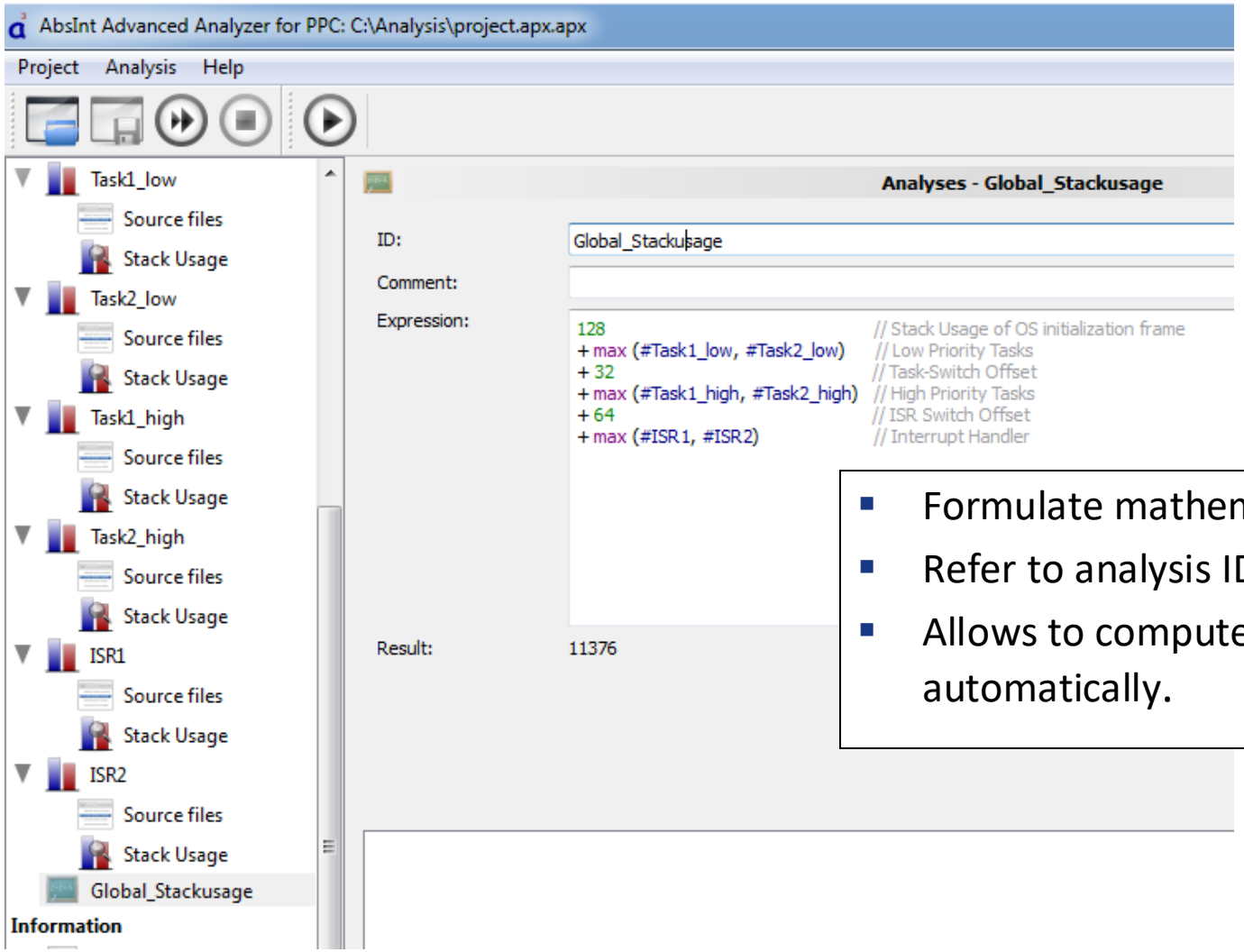
Example Targets:

Am486	C	ARM	
C16x & ST10	T	C28x	
C33x		dsPIC	
ERC32		FR81S	
HCS12		i386DX	
LEON2		LEON3 & LEON4	
M68k & ColdFire	+3	MCS251	
MCS51		MIPS32	
MSP430(x)		Nios II	
PowerPC	+4	Renesas RX	
RISC-V		New: RL78	
S12Z		SuperH	
TriCore		V850 & RH850	
x86	+3	ARC	

Maximum Global Stack Usage



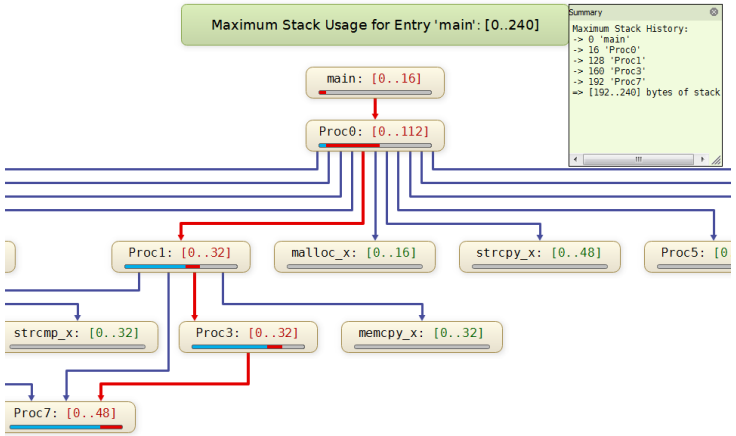
StackAnalyzer Result Combination



- Formulate mathematical expressions
- Refer to analysis IDs
- Allows to compute system stack usage automatically.

StackAnalyzer Benefits

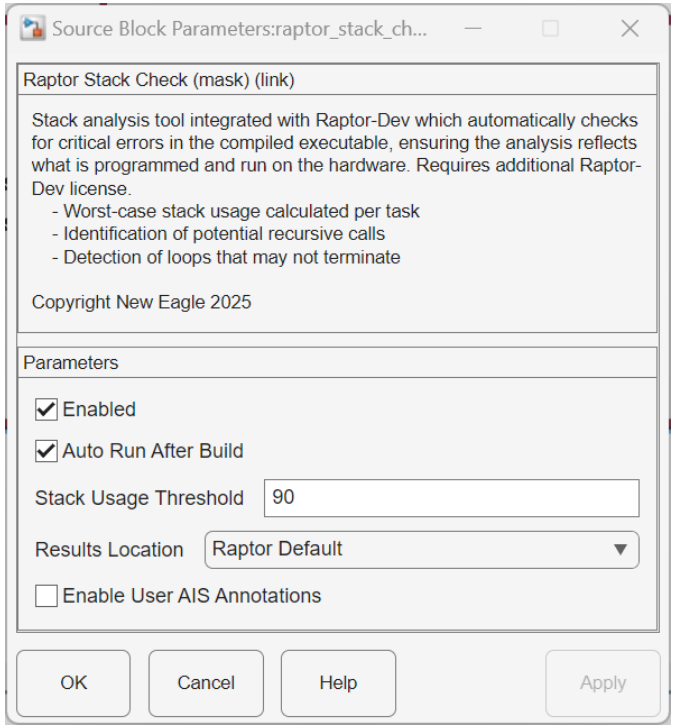
- StackAnalyzer reduces risk of failures in the field: no more stack overflows.
- Computed bounds are valid for all inputs and all execution scenarios.
- Analyses the executable binary
- No debug information needed
- No code instrumentation needed
- Taking into account
  - loops and recursions,
  - inline assembly,
  - object code libraries, and
  - link-time optimizations
- Available for numerous target architectures
- Tool Couplings available, e.g., to Raptor-Dev.

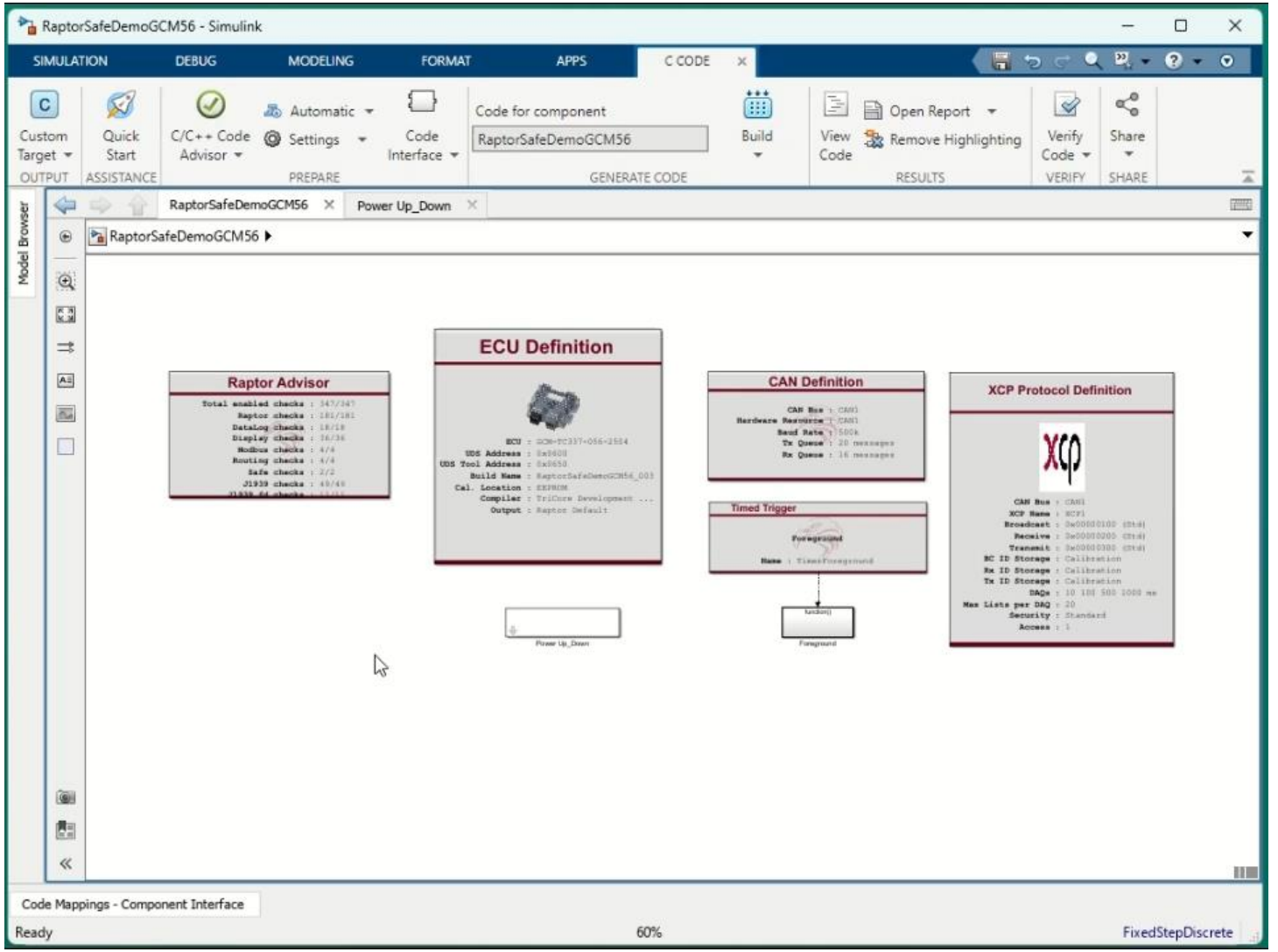


Raptor-Safe Stack Check

Raptor Stack Check

```
          Enable : Enabled
       Auto Run : Enabled
Stack Usage Threshold : 90%
   Results Location : Raptor Default
```





Real-Time Systems

- Correctness depends on their temporal aspects as well as their functional aspects.
- Real-time tasks must terminate within specified time constraints (deadlines).
- Violation of timing constraints
  - is a defect potentially causing a failure
  - degrades availability

7.4.13 An upper estimation of required resources for the embedded software shall be made, including:

a) the execution time; → ➢ Worst-case execution time (WCET)
➢ Worst-case response time (WCRT) ≈ WCET + preemption time + blocking time

b) the storage space; and

EXAMPLE RAM for stacks and heaps, ROM for programme and non-volatile data.

→ ➢ Worst-case stack usage

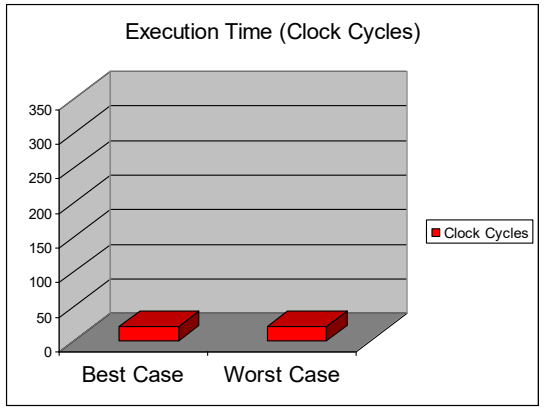
Excerpt from:
ISO/ 26262-6 Road vehicles - Functional safety – Part 6: Product development: Software Level, 2018.

Execution Time Variability

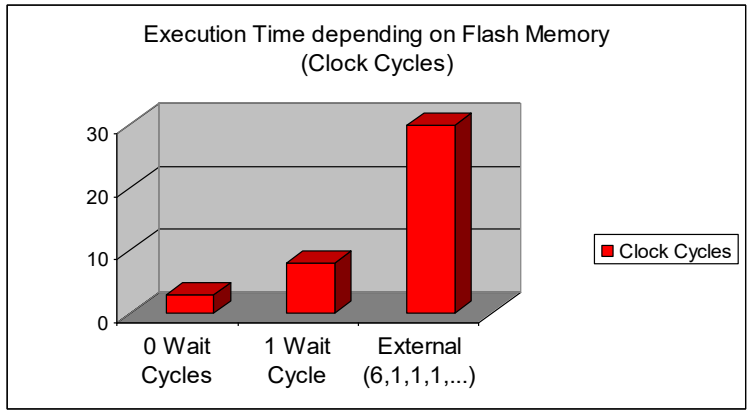
x = a + b;

```
LOAD    r2, _a
LOAD    r1, _b
ADD     r3,r2,r1
```

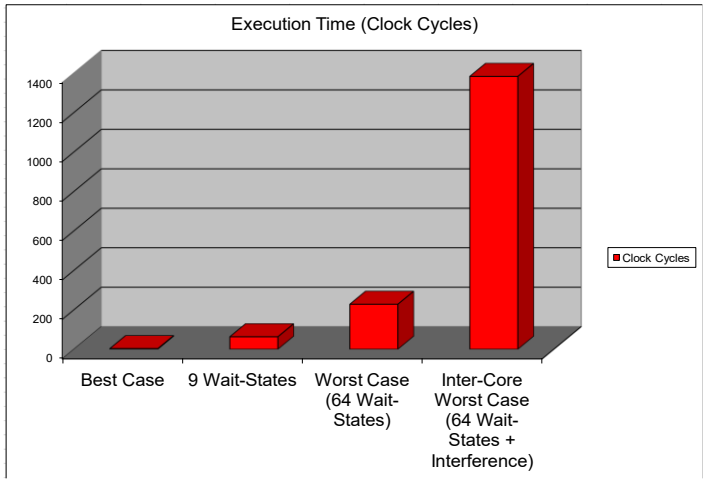
68K (1990)



MPC 5xx (2000)

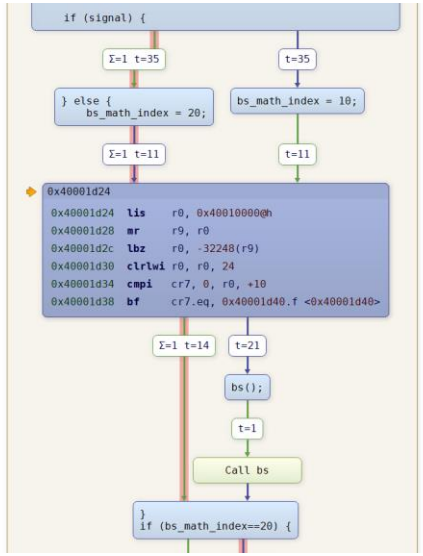


TriCore AURIX 2nd GEN TC397 (2016)



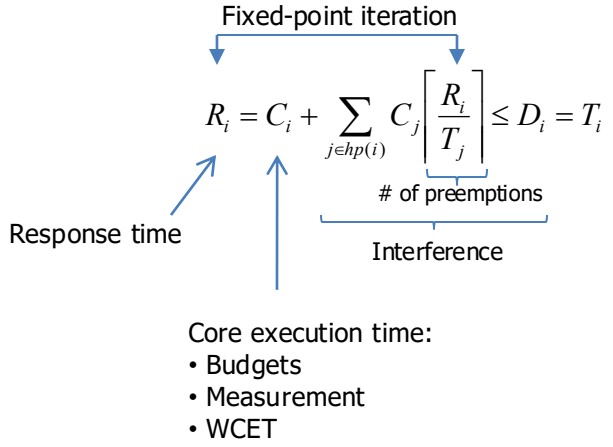
Two Levels of Timing

- Code level
  - Single process, task, ISR
  - Focus on
    - Control flow
    - Processor architecture with pipelines and caches
    - WCET



AbsInt Timing Tools:
aiT
TimeWeaver

- System level
  - Multiple functions or tasks
  - Focus on
    - Integration and scheduling
    - End-to-end timing
    - Worst-Case Response Time (WCRT)



WCRT analysis or simulation tools

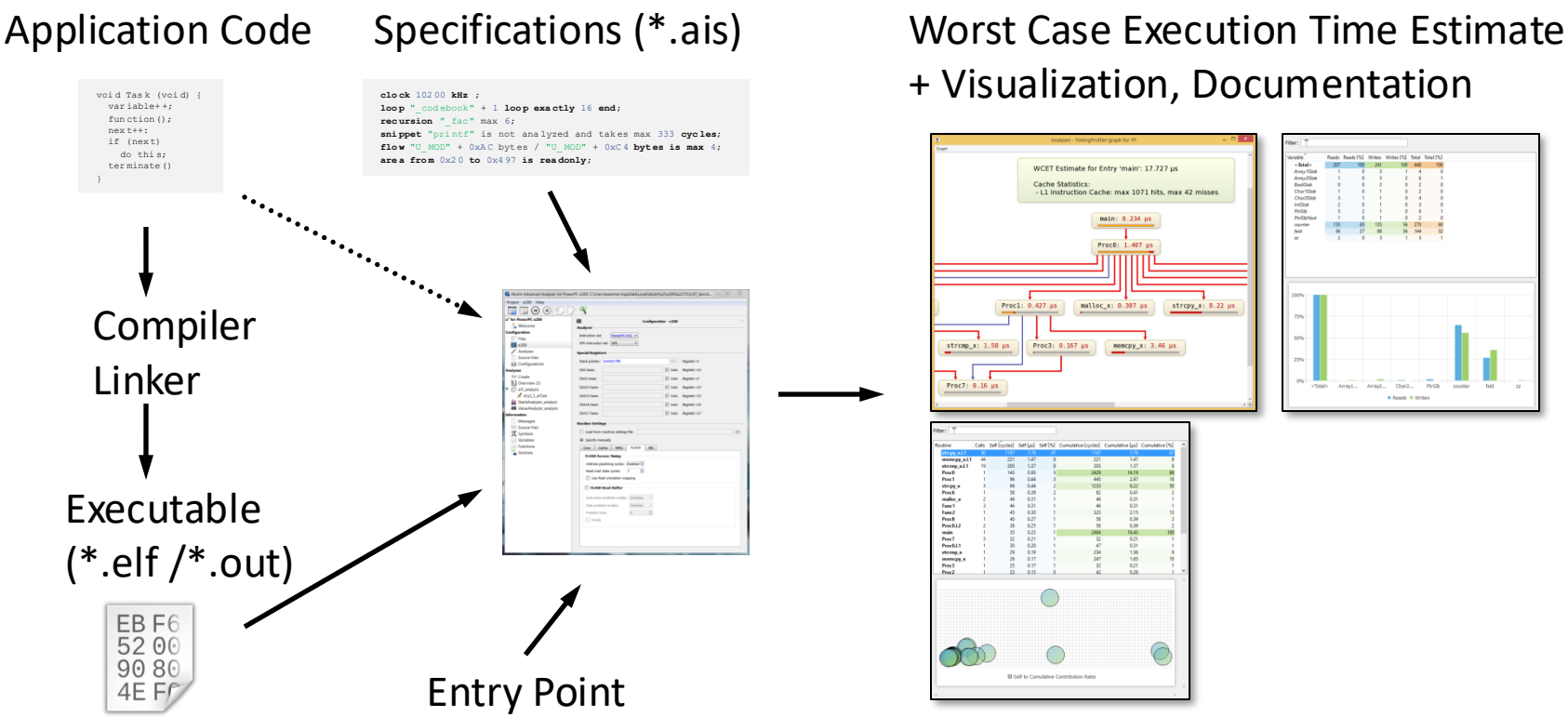
Early-Stage WCET Estimation

- Feedback about worst-case execution time in early stages prevents late-stage design problems

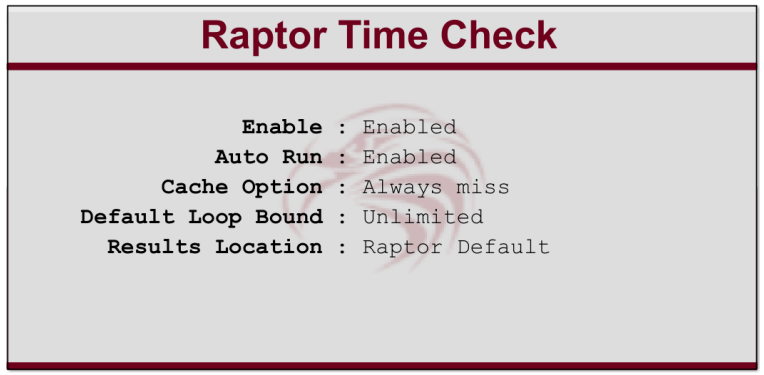
- Assumptions and requirements:
  - Precise microcontroller variant & configuration may not be fixed yet
  - Estimations of the WCET are sufficient
  - Timing information must be available with low computing effort
  - User interaction must be minimal

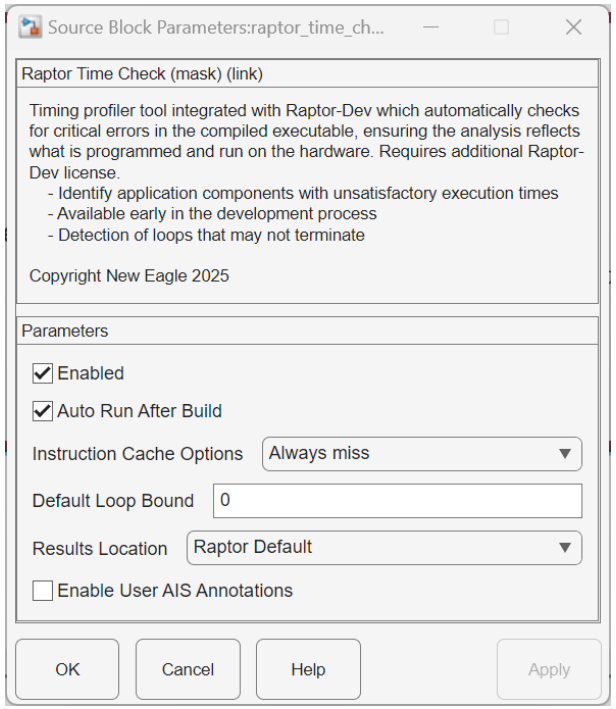
TimingProfiler Workflow

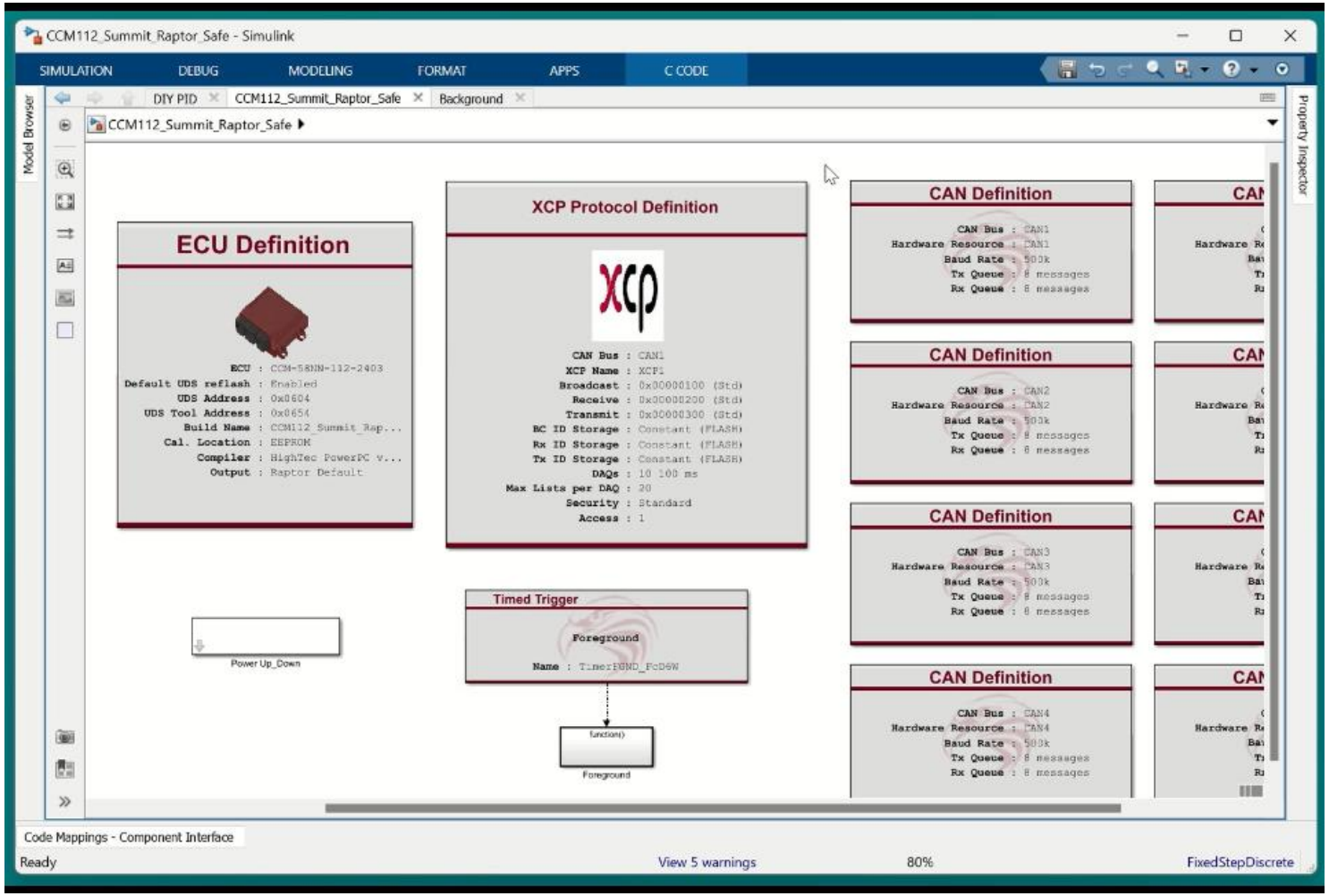
- Global static program analysis: microarchitecture analysis (caches, pipelines, …) + value analysis
- Integer linear programming for path analysis
- ➢ Estimates on the worst-case execution time (WCET)



Raptor-Safe Time Check



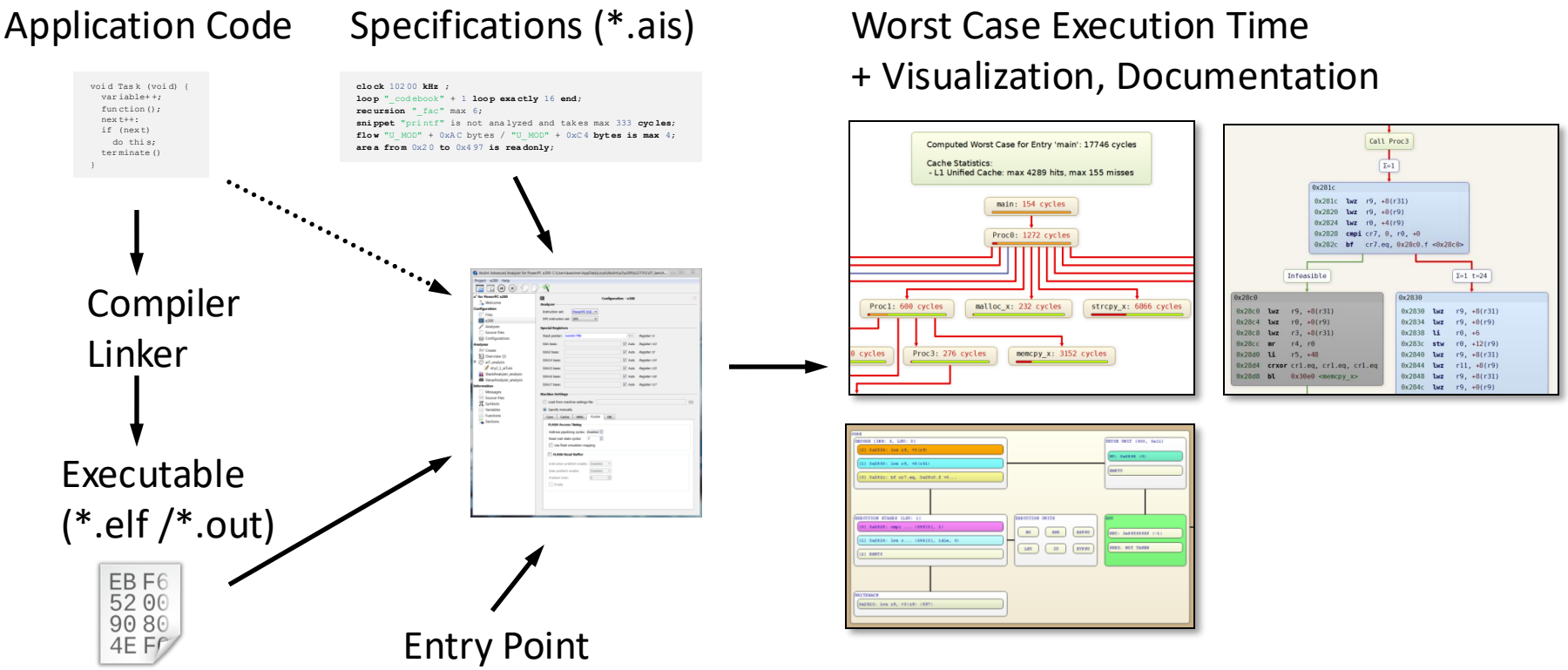




aiT Worst-Case Execution Time Analyzer

- Global static program analysis by Abstract Interpretation: microarchitecture analysis (caches, pipelines, …) + value analysis + path analysis

⇨ Safe and precise WCET bounds for timing-predictable processors



Example Targets:
PowerPC
55xx/56xx/57xx/…
TriCore/AURIX
ARM Cortex
M0/M3/M4/R4F/R5F…
C28x, LEON, V850, …
RISC-V*

Multi-Core Concerns

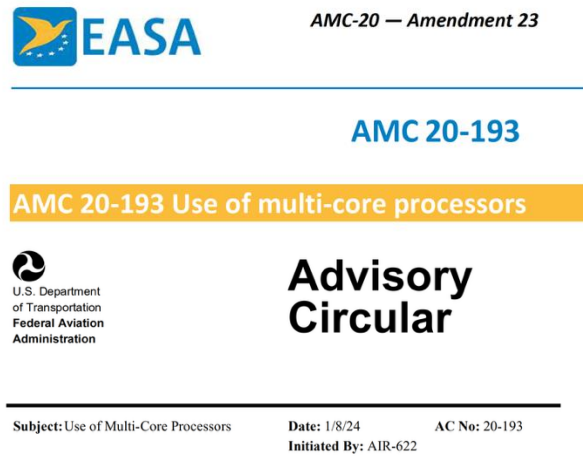
- Contention on shared resources:
  - Memory, caches, external interfaces, prefetch buffers, …
- Typically arbitrated by interconnects / coherency fabric
  - Accesses are subject to delays
  - Maximal delay undocumented and/or unbounded?
  - Transactions might be reordered

⇨ Interference channels have to be identified and mitigated

⇨ Time interference

⇨ Resource interference

⇨ Either robust time and resource partitioning, or WCET has to be determined with all software components on all cores executing in the intended final configuration [CAST32A, AMC20-193]


EASA

AMC-20 — Amendment 23

AMC 20-193

AMC 20-193 Use of multi-core processors

U.S. Department of Transportation
Federal Aviation Administration

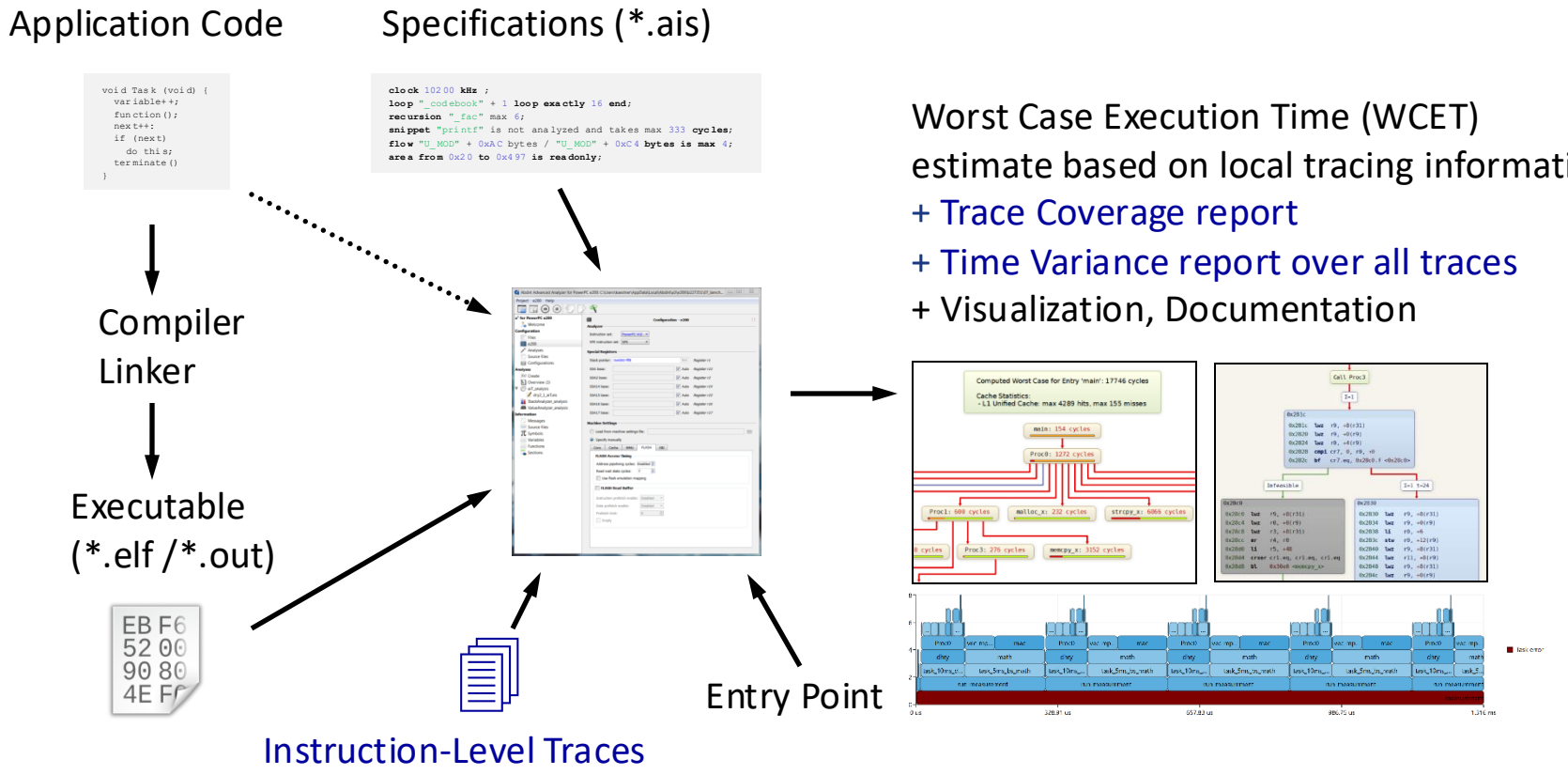
Advisory Circular

Subject: Use of Multi-Core Processors Date: 1/8/24 AC No: 20-193
Initiated By: AIR-622

[AMC 20-193] EASA. AMC-20 – Amendment 23. AMC 20-193. Use of multi-core processors, 2022.
[AC20-193] FAA. Advisory Circular AC No 20-193. Use of Multi-Core Processors, 2024.

TimeWeaver – Non-intrusive Hybrid WCET

- Focus: non-timing-predictable microprocessors
- Combines static value and worst-case path analysis and hardware measurements based on non-intrusive instruction-level tracing

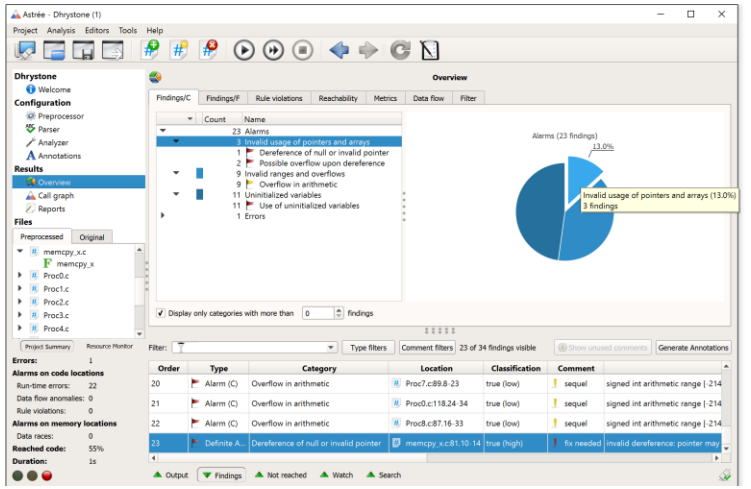


Example Targets:
ARM Cortex R5F/R52/
M7/A53/A72/AXX…
PowerPC QorIQ
P40xx/P50xx…
RISC-V*

Also applicable to:
TriCore/AURIX

Astrée

- Abstract Interpretation-based static runtime error analysis at source code level
- Astrée detects all runtime errors* with few false alarms:
  - Covered defect classes: array index out of bounds, int/float division by 0, invalid pointer dereferences, uninitialized variables, arithmetic overflows, data races, lock/unlock problems, deadlocks, …
  - Ensures C/C++-level memory safety
  - Data and control flow analysis, data and control coupling analysis
  - Non-interference analysis, signal flow analysis, taint analysis for data safety, security and fault propagation, SPECTRE detection
  - \+ User-defined assertions, unreachable code, non-terminating loops, alias analysis
  - \+ Check coding guidelines (MISRA C/C++, Adaptive AUTOSAR C++, CERT C/C++, CWE, ISO TS 17961)
  - \+ Automatic support for ARINC653/OSEK/AUTOSAR OS configurations

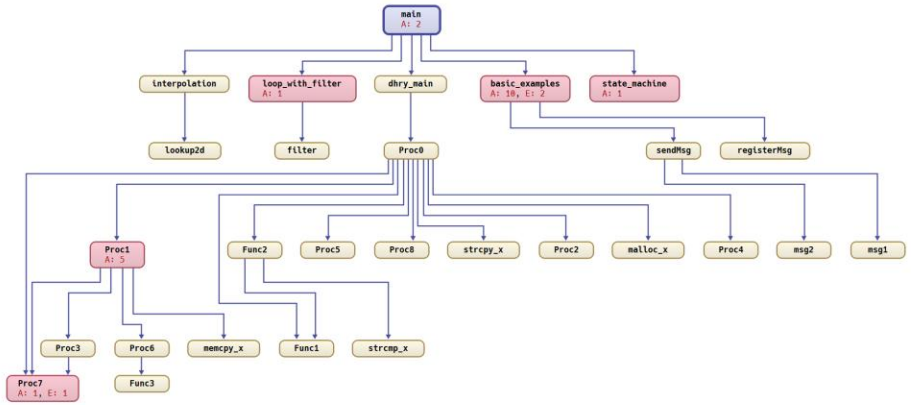


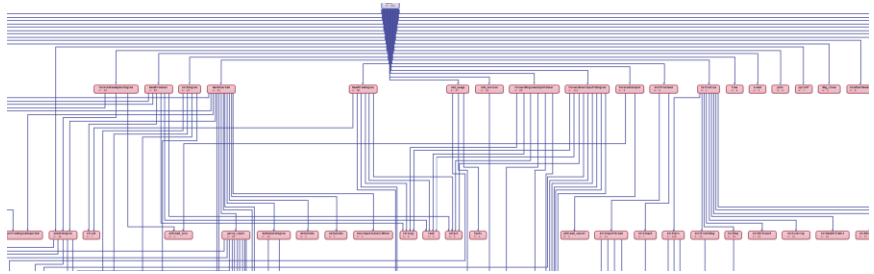
RuleChecker included

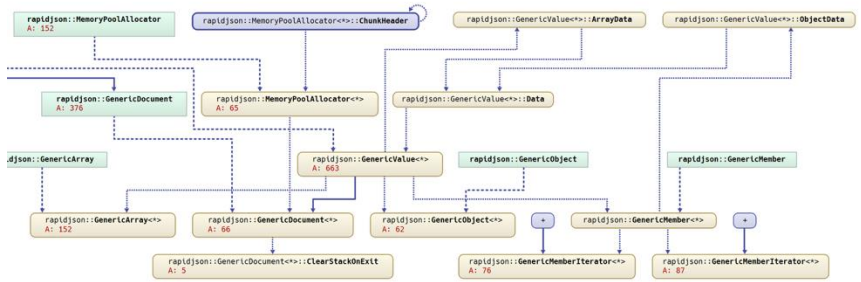
* Defects due to undefined / unspecified behaviors of the programming language

Data and Control Flow Analysis in Astrée

- Control flow analysis
  - Caller/callee relationships between functions
  - Call graph
  - Function calls per concurrent thread
- Data flow analysis
  - List of global/static variables with information about
    - locations/functions/processes performing read/write accesses
    - access properties:
      - Thread-local
      - Shared
      - Subject to data race
- Data and control coupling / Interference analysis
- Soundness: no data/control flow is missed
  - Aware of data and function pointers, task interference, …
  - Freedom of interference can be proven







Non-Interference Analysis

- Astrée provides sound user-configurable taint analysis
  - __ASTREE_taint_add, __ASTREE_taint_add_if, __ASTREE_taint_sink, __ASTREE_taint_alarm
- Signal flow: output signal Y is not influenced by input signal
  - Upcoming law prepared by CARB effectively requires interference analysis of all inputs and outputs
    - Monitoring device needed for all parameters that can affect emissions
    - Omitting OBD only feasible for parameters that cannot affect emissions

13 CA ADC §968.2

CALIFORNIA AIR RESOURCES BOARD

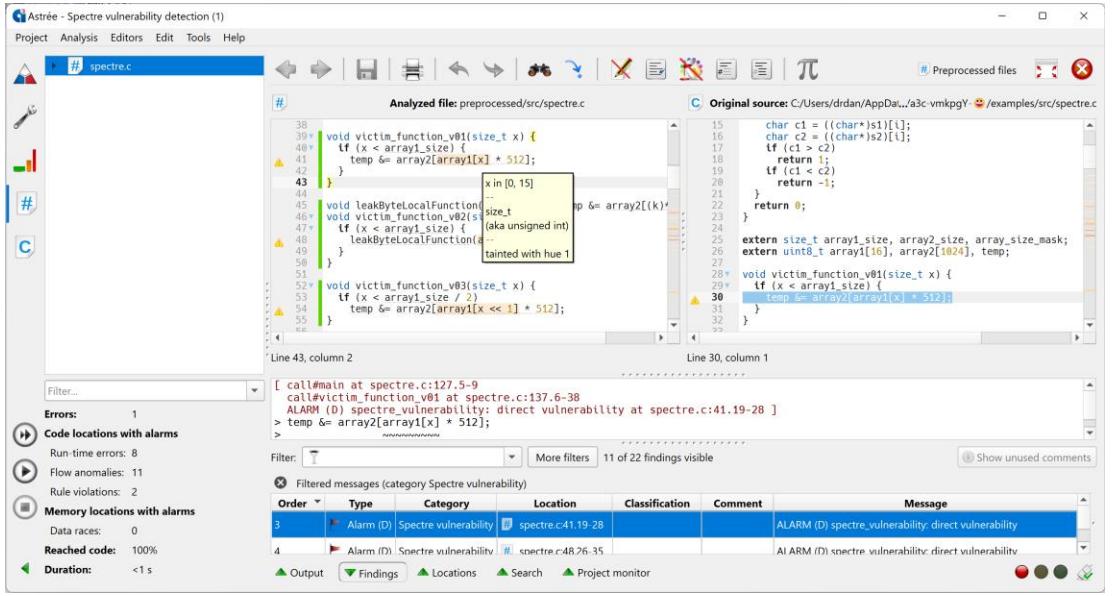
§ 1968.2. Malfunction and Diagnostic System Requirements—2004 and Subsequent Model-Year Passenger Cars, Light-Duty Trucks, and Medium-Duty Vehicles and Engines.

- Freedom of interference between software components at programming language level
  - E.g. components of different SIL/ASIL
  - Data and control coupling
- Cybersecurity properties, e.g., separation between trusted / non-trusted code/data
- Propagation of corrupted values (safety and cybersecurity)

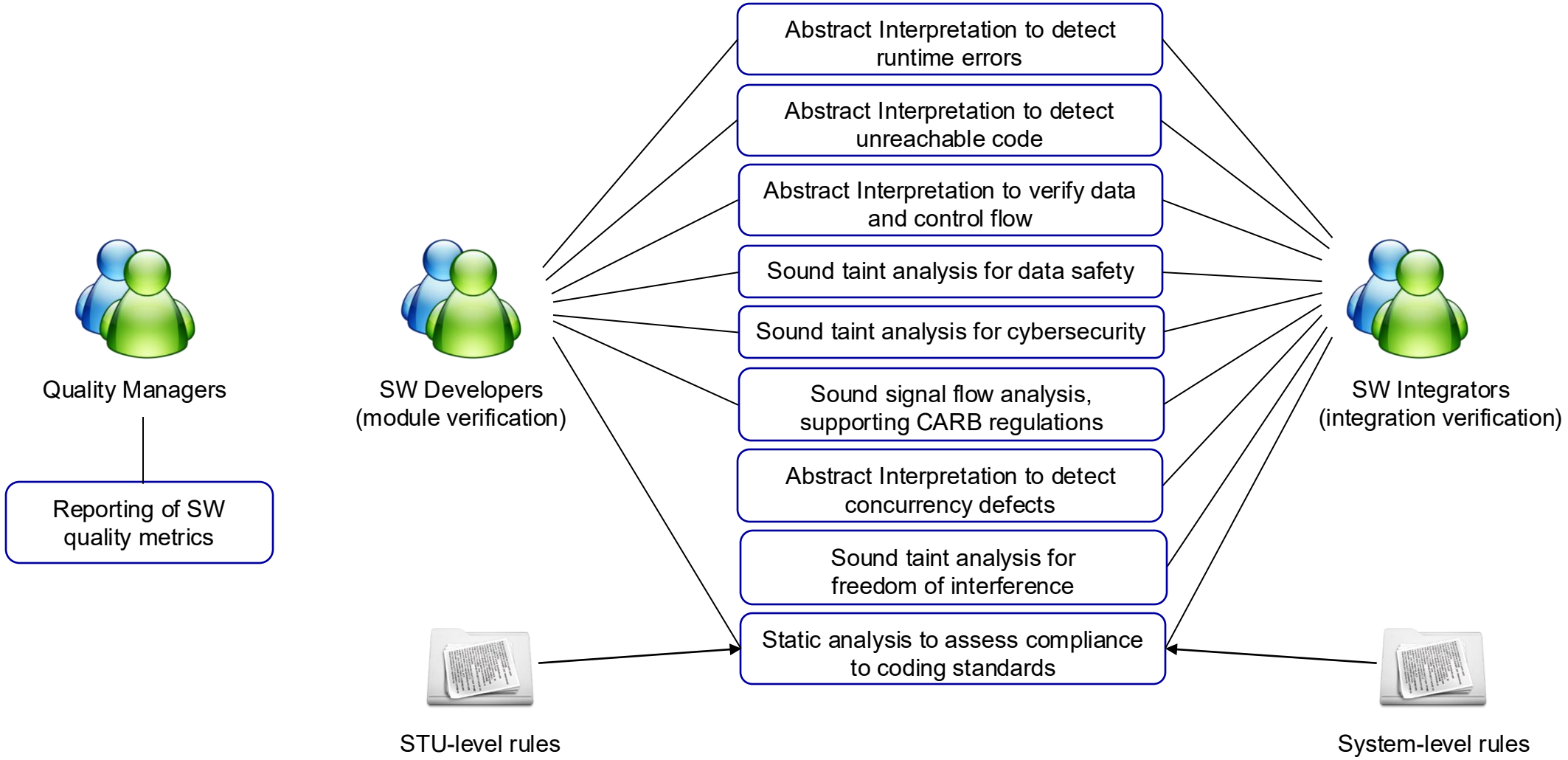
Support for Cybersecurity Analysis

- Many security vulnerabilities due to undefined / unspecified behaviors in the programming language semantics:
  - buffer overflows, invalid pointer accesses, uninitialized memory accesses, data races, etc.
  - Consequences: denial-of-service, code injection, data breach
  - ⇨ Absence can be proven by Astrée

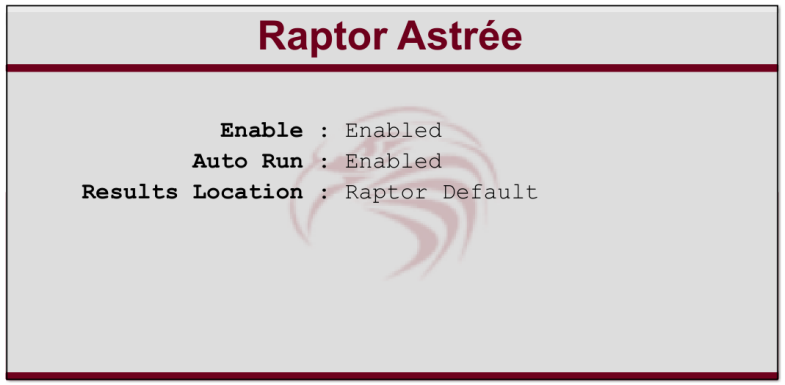
- Astrée provides further sophisticated analysis modules:
  - Checking coding guidelines (RuleChecker included)
  - Data and Control Flow Analysis
  - Taint Analysis
    - Impact analysis (data safety / “fault” propagation)
    - Non-interference analysis (signal flow analysis, freedom of interference)
  - Side channel attacks
    - SPECTRE detection (Spectre V1/V1.1, SplitSpectre)

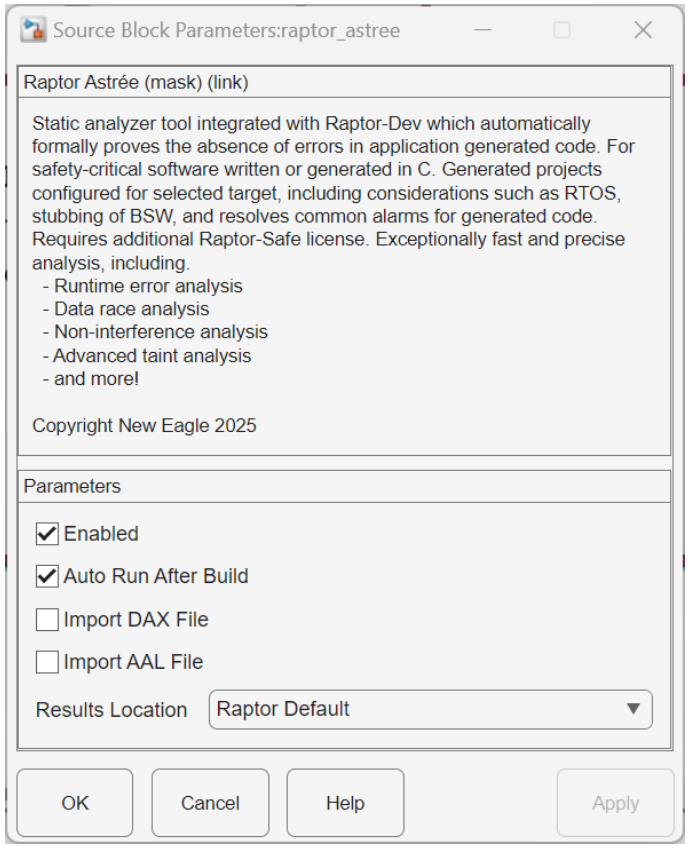


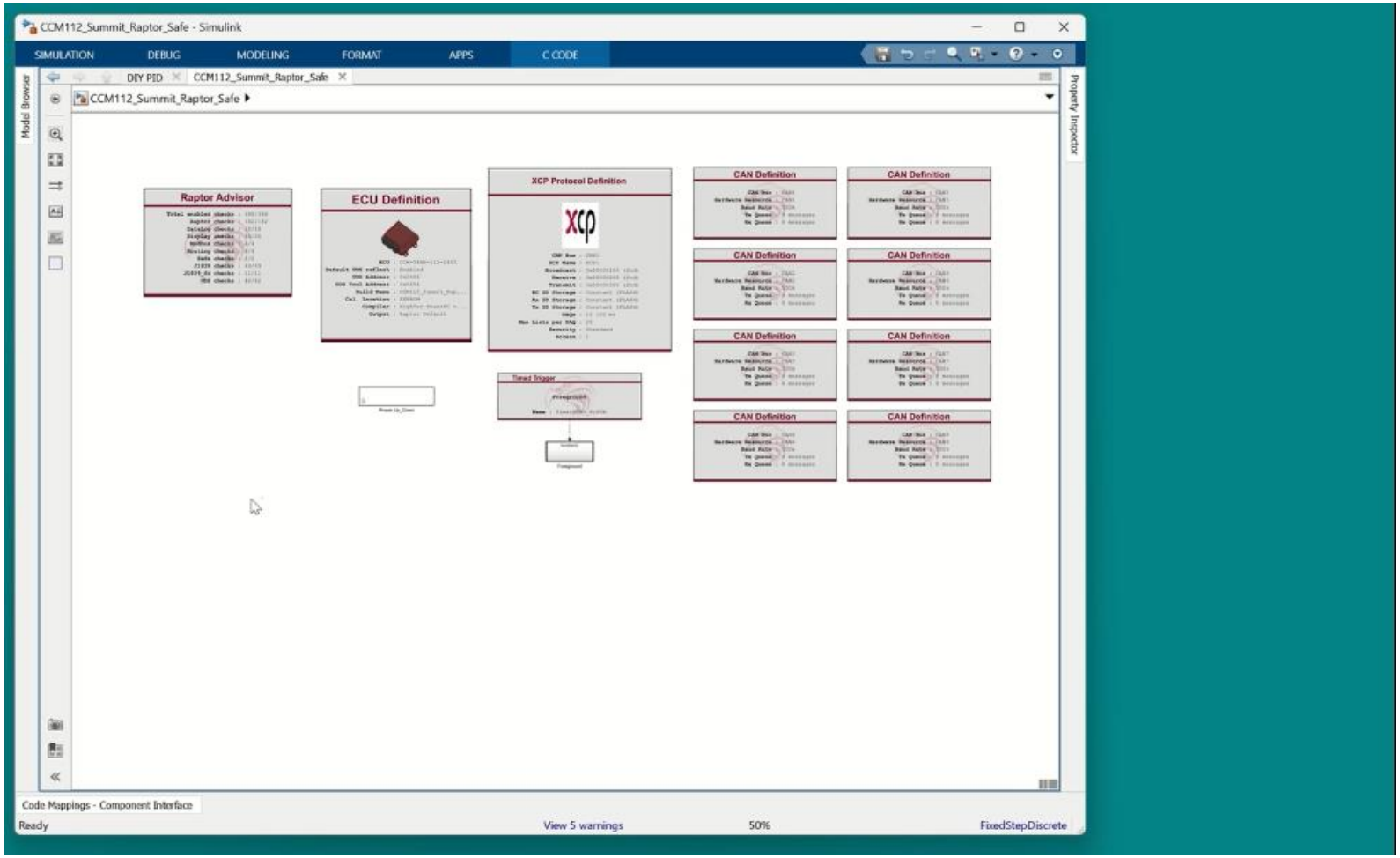
Astrée Use Cases



Raptor Astrée



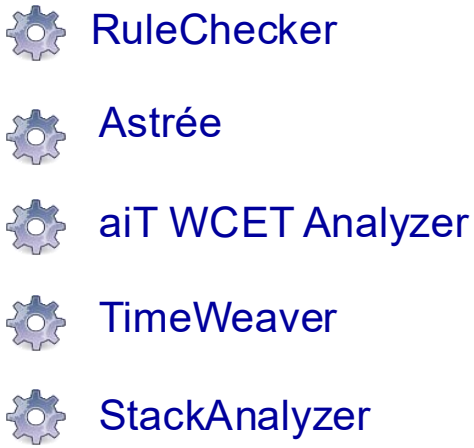




AbsInt

Conclusion

- In safety-critical systems the absence of safety and security hazards has to be demonstrated.
- Static analysis crucial for safety and security
  - Recommended by all safety norms
  - ✓ Checking compliance to coding guidelines
  - ✓ Computing code metrics
  - ✓ Resource analysis: worst-case stack usage, worst-cast execution time
  - ✓ Runtime error analysis by sound static analysis
    - Absence of critical code defects can be proven:
      div/0, buffer overflow, null/dangling pointers, data races, deadlocks, …
    - Data and control flow coupling / Interference analysis
    - Fault avoidance
- Seamless integration into development workflow via
  Raptor-Safe Stack Check, Raptor-Safe Time Check, and Raptor Astrée



email: kaestner@absint.com
http://www.absint.com