Example: Design tool with formal background

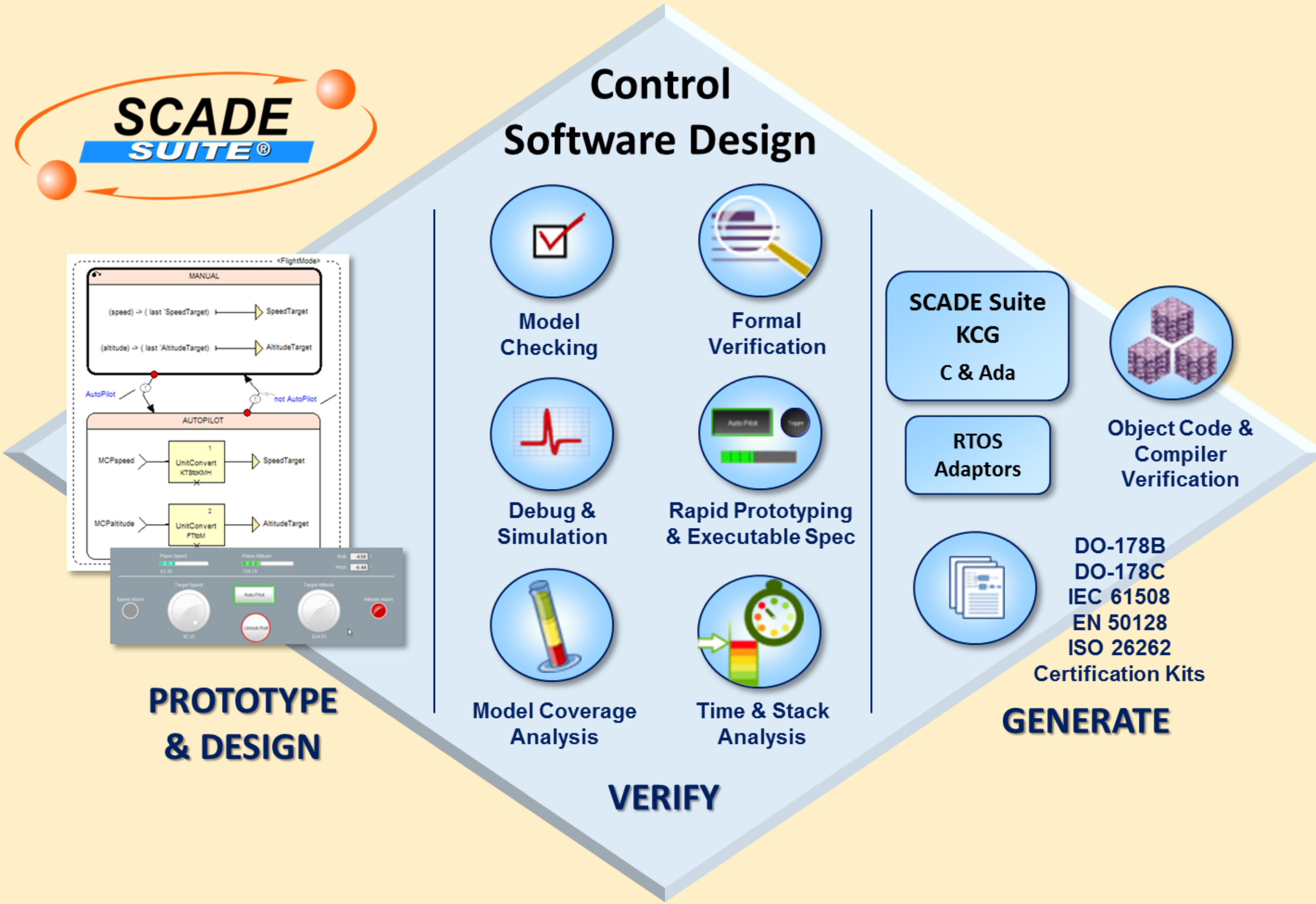
SCADE Suite

Safety Critical Applications
Development Environment

Esterel Technologies
(part of ANSYS)

Development of
embedded, real time,
safety-critical software
with formal methods
and integrated verification

Overview


SCADE SUITE®
Control
Software Design
Model
Checking
Formal
Verification
Debug &
Simulation
Rapid Prototyping
& Executable Spec
Model Coverage
Analysis
Time & Stack
Analysis
SCADE Suite
KCG
C & Ada
RTOS
Adaptors
Object Code &
Compiler
Verification
DO-178B
DO-178C
IEC 61508
EN 50128
ISO 26262
Certification Kits
PROTOTYPE
& DESIGN
VERIFY
GENERATE

Applications of SCADE

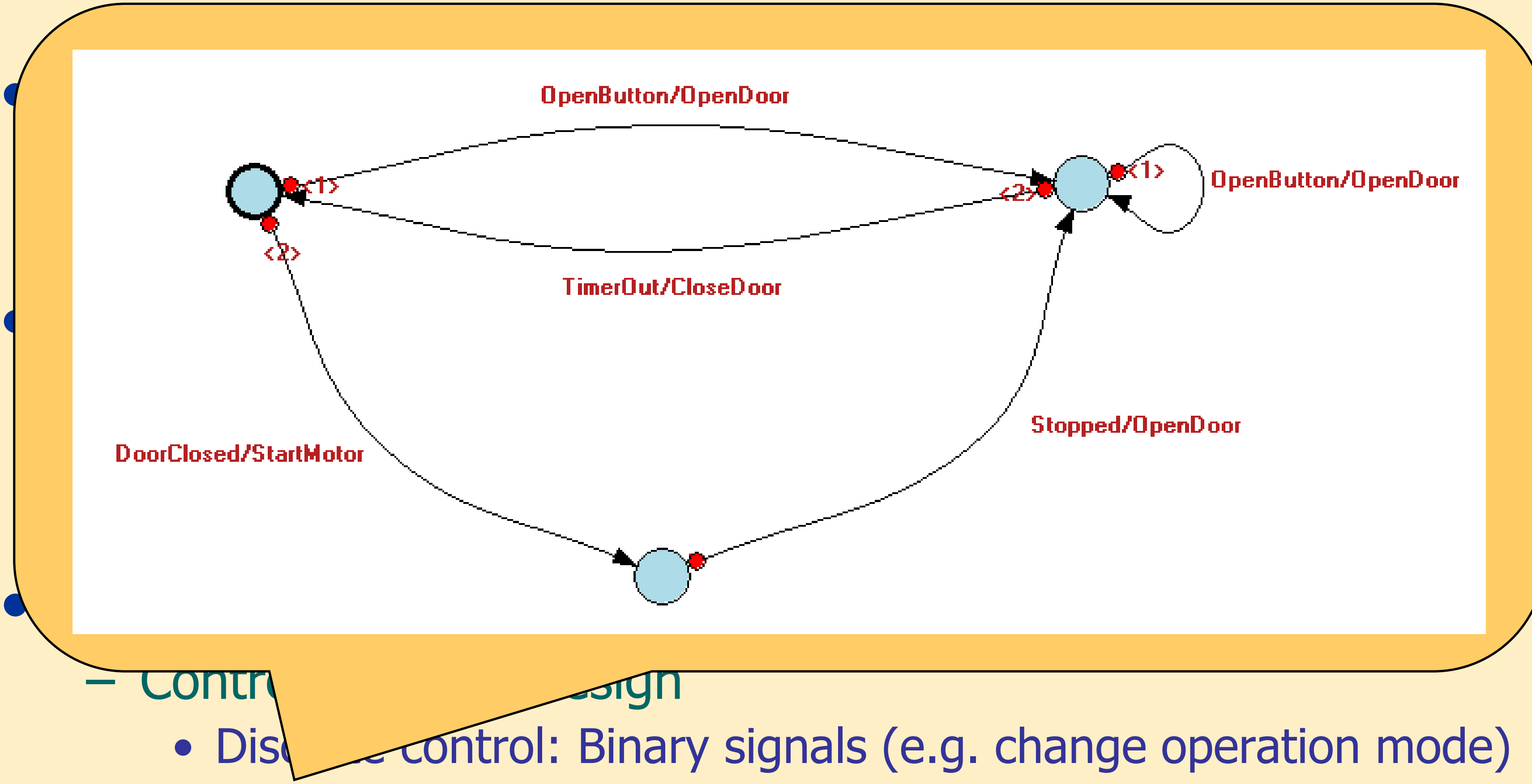
Includes source code developed in SCADE:

- Airbus A380, Airbus A340
- Boeing 787
- Dassault's Falcon 7X
- Ariane 5
- M51
- Eurocopter
- Z8 Helicopter
- Audi A6, A8
- PSA 407, PSA 607
- BMW, Honda Motocycles
- … and many more

Application domain: Embedded controllers

- Controlling/affecting physical processes
  - Sensors, actuators, control loop
  - Human-machine interface
- Behvior of controllers: Cyclic
  - Read sensor data –> Process –>
    Instruct actuators –> Read sensor data…
  - Time-driven, event-driven or polling
- Design approaches:
  - Control-oriented design
    - Discrete control: Binary signals (e.g. change operation mode)
    - Finite-state automata (states, events, actions)
  - Data-oriented design
    - Continuous control: Signal processing (diff. equations)
    - Data-flow network (processor components, data paths)

Application domain: Embedded controllers

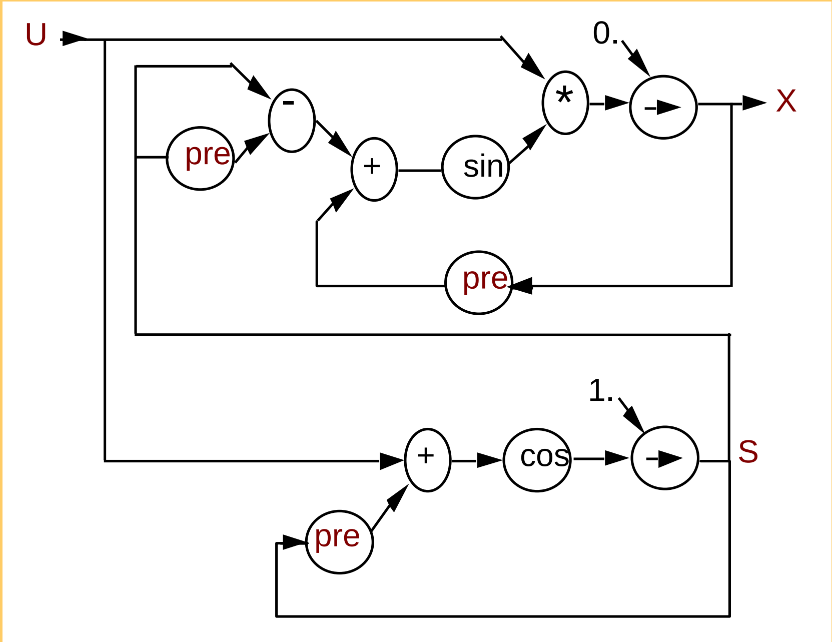


– Contr … esign
  - Dis… control: Binary signals (e.g. change operation mode)
  - Finite-state automata (states, events, actions)

– Data-oriented design
  - Continuous control: Signal processing (diff. equations)
  - Data-flow network (processor components, data paths)

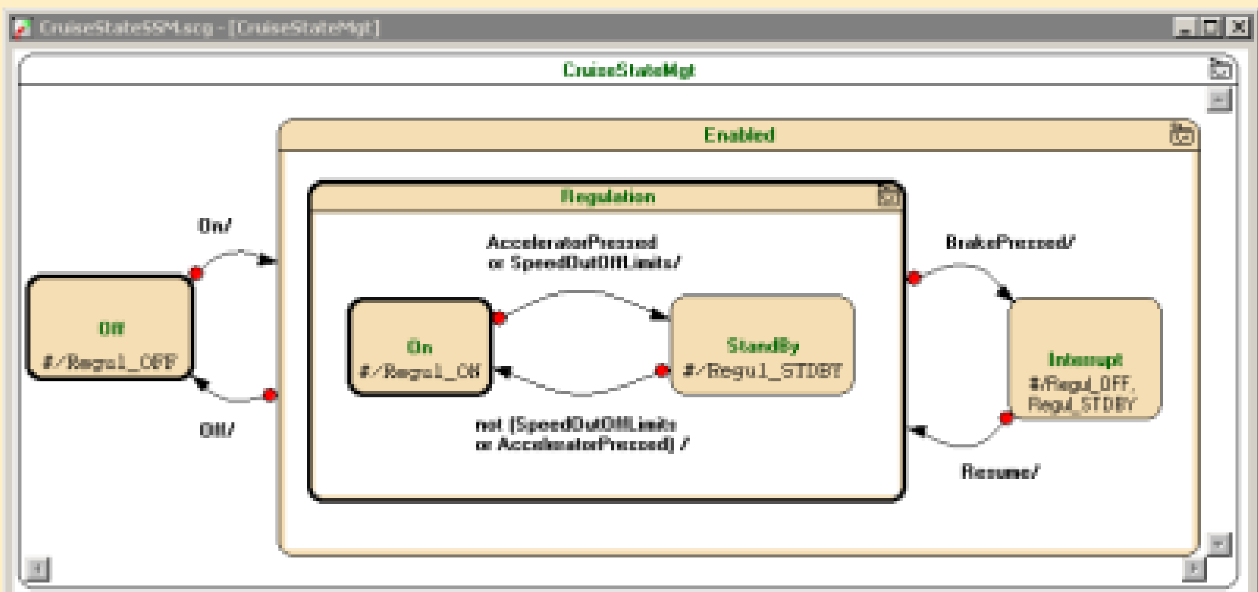
Application domain: Embedded controllers

- Controlling/affecting physical processes
  - Sensors, actuators, control loop
  - Human-machine interfa
- Behavior of controller
  - Read sensor data –>
    Instruct actuators –>
  - Time-driven, event-dr
- Design approaches:
  - Control-oriented des
    - Discrete control:
    - Finite-state aut ata
  - Data-oriented design
    - Continuous control: Signal processing (diff. equations)
    - Data-flow network (processor components, data paths)



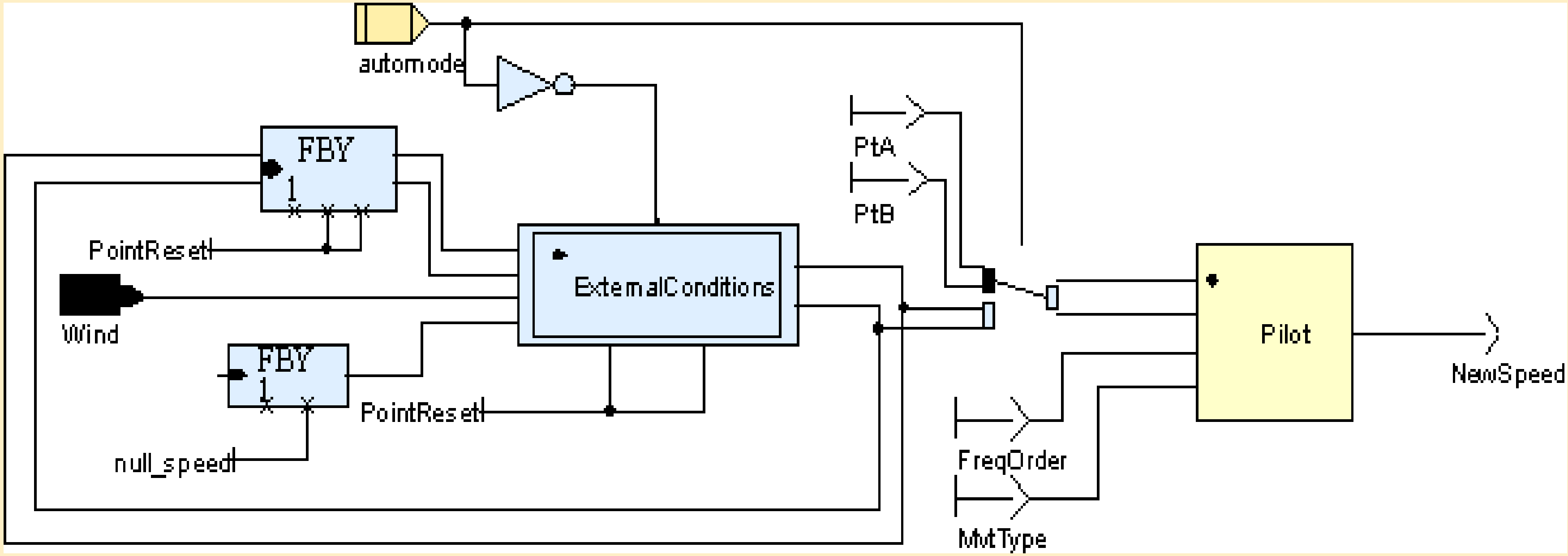
Formalism: Safe state machine

- States, restricted state hierarchy
  - No concurrent regions, transitions between hierarchy levels
- Restricted set of modeling elements
  - E.g. no history state
- Deterministic behavior (with proper restrictions)



Formalism: Data-flow diagrams

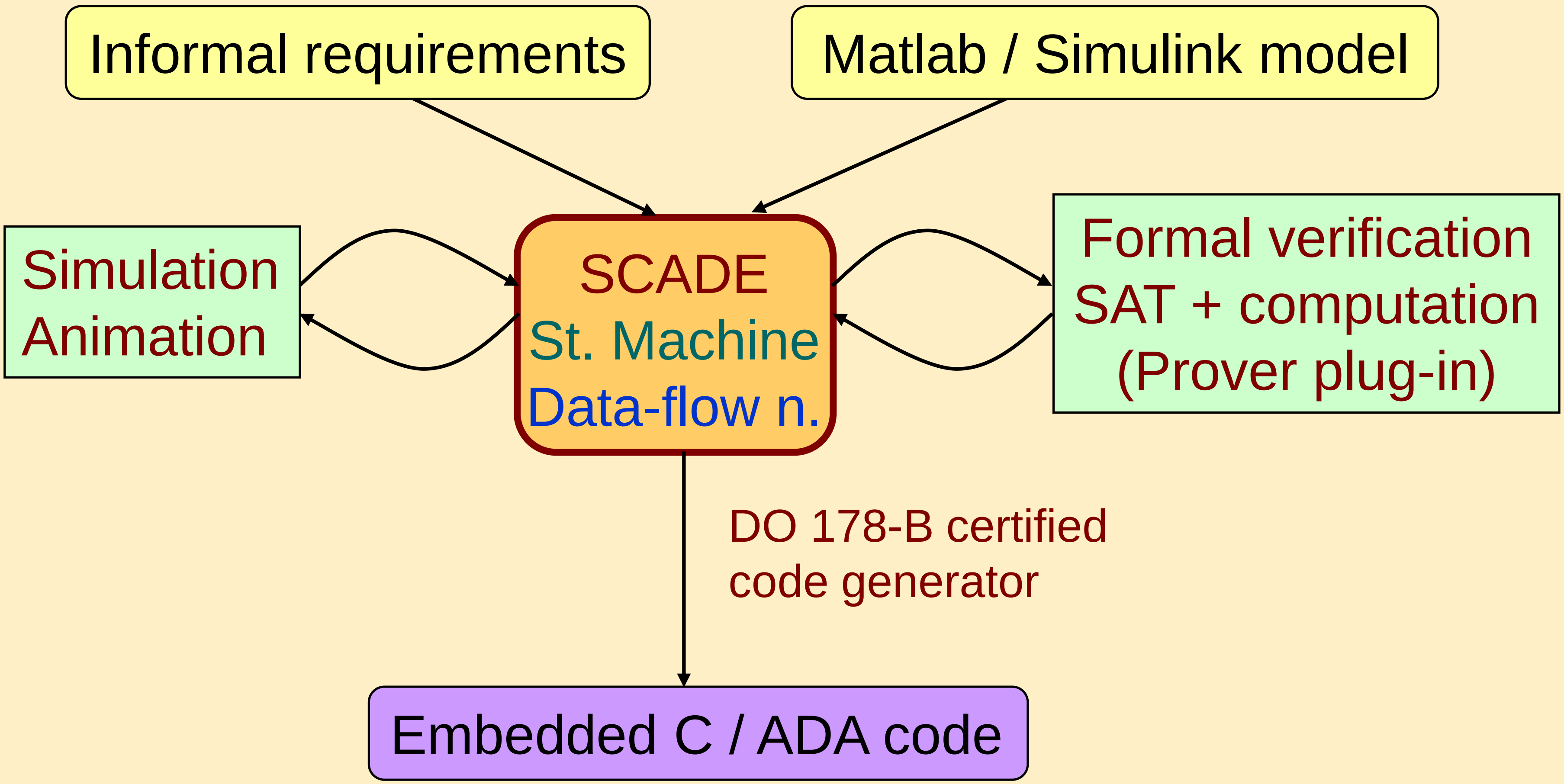
- Function blocks are elements of computation
- Directed arcs denote the direction of data flow
- Inputs are sampled periodically
- Outputs are computed and validated cyclically



Development of the SCADE language

- Formal language
  - Basis: the synchronous Lustre language (Univ. Grenoble, 1983)
  - More that 20 years of preliminary research
- Parallel definition of the SCADE language and its mathematically precise semantics
  - Interpretation of a SCADE model is independent of the tools
- Safety in the main focus since the beginning
  - The language was defined in cooperation with industrial partners (later users) and approval authorities

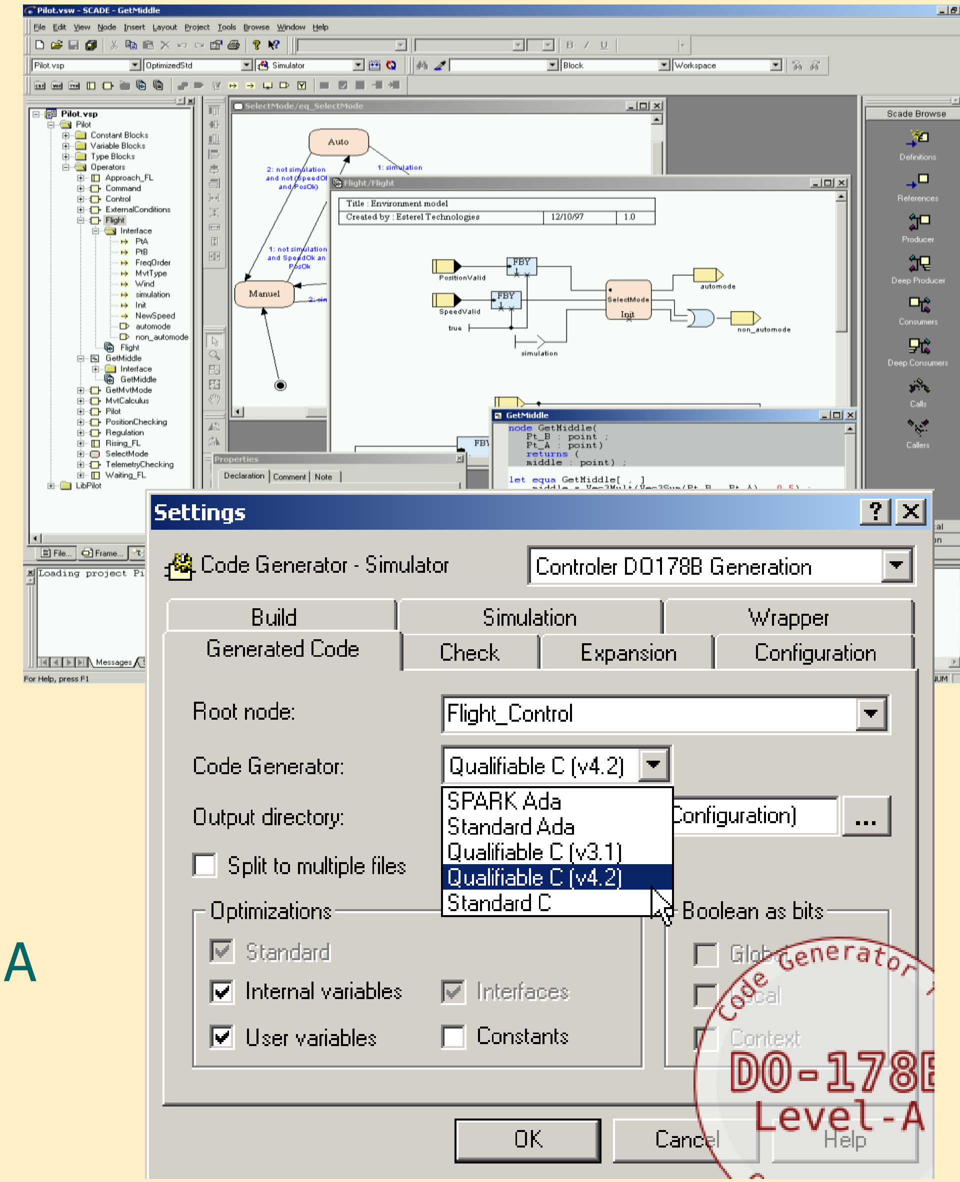
Support of the design process



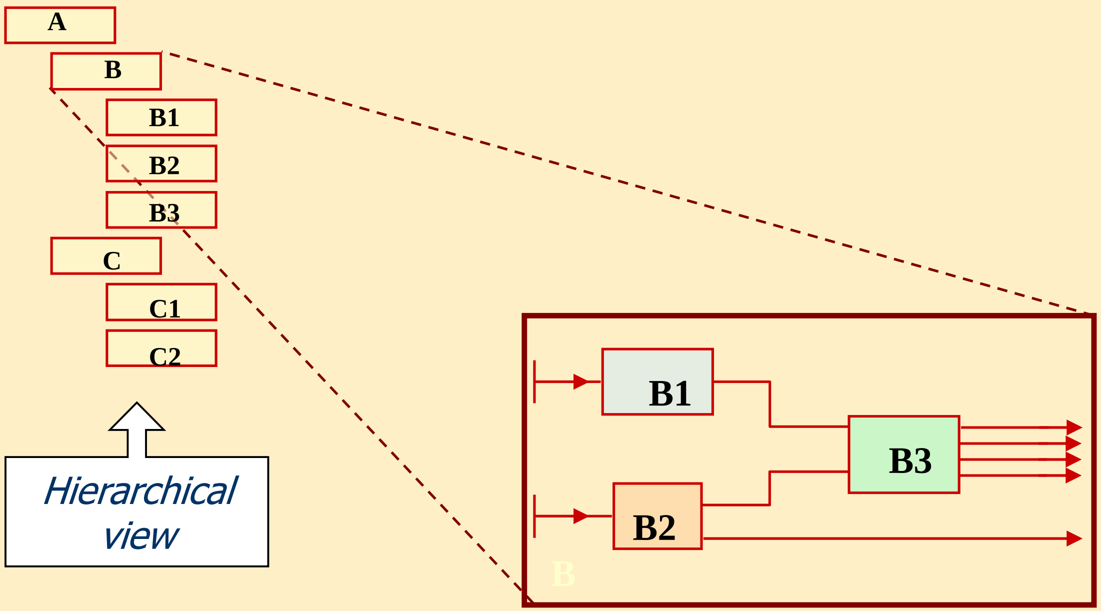
Preserving semantics + certified code generator and compiler
$\Rightarrow$ modul/unit testing is not necessary (Airbus: 50%)

The SCADE Suite tool

- Graphical editor
  - Data-flow diagrams
  - Safe state machines
- Static analysis
- Simulation
  - Interactive and batch mode
  - Testing / debug functions
- Formal verification
  - Checking of properties
- Code generation
  - Ada & C
  - Qualified C: DO-178B Level A or MISRA conformance



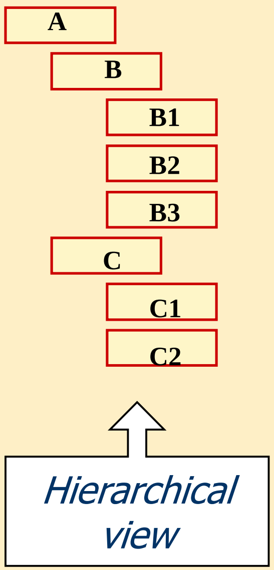
SCADE model: different views on demand

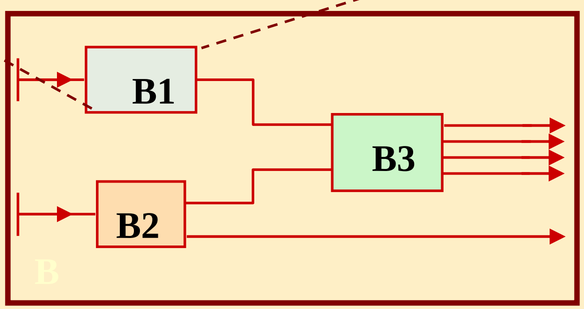


SCADE model: different views on demand

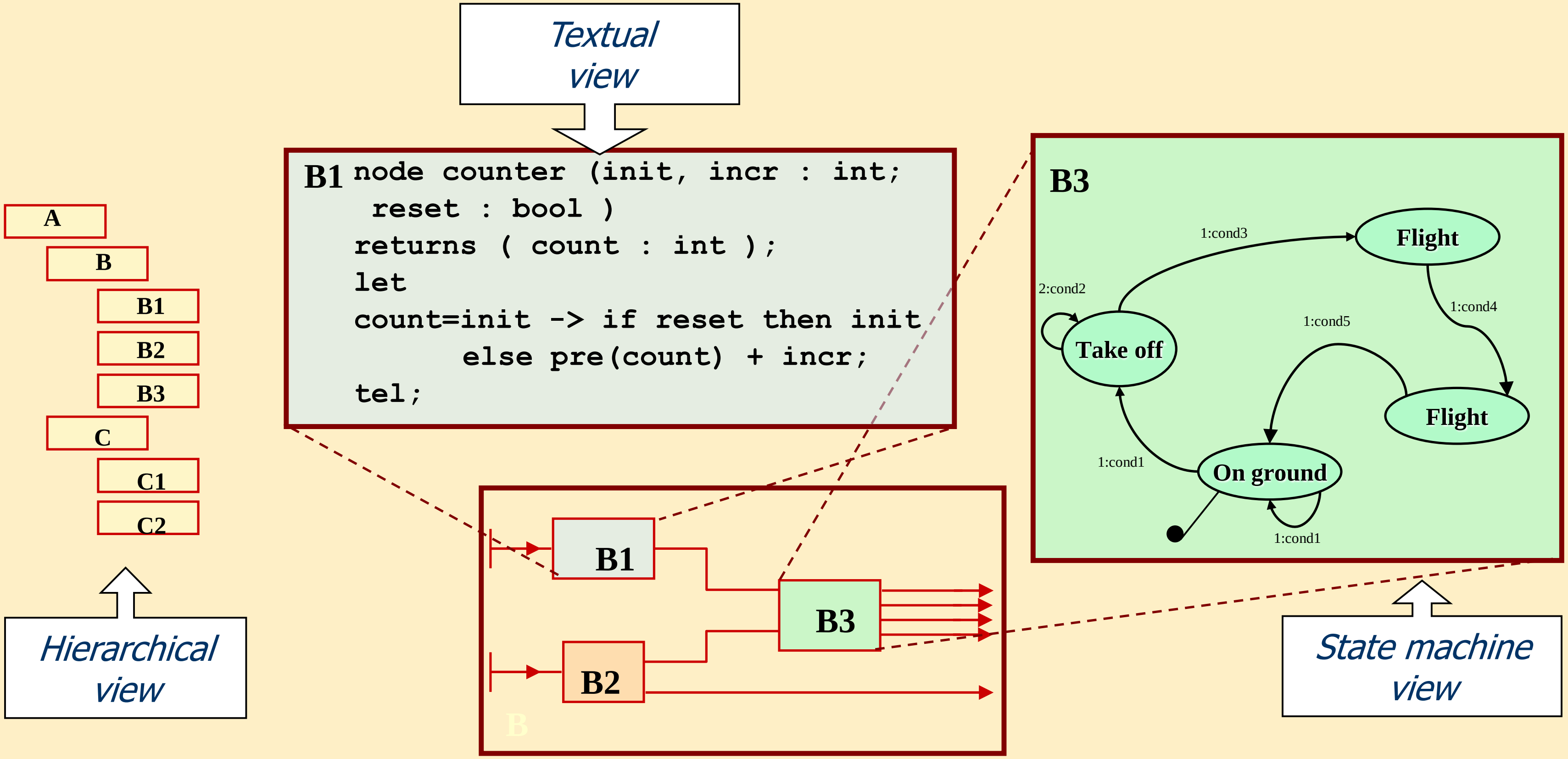
Textual view

B1
```
node counter (init, incr : int;
 reset : bool )
returns ( count : int );
let
count=init -> if reset then init
      else pre(count) + incr;
tel;
```

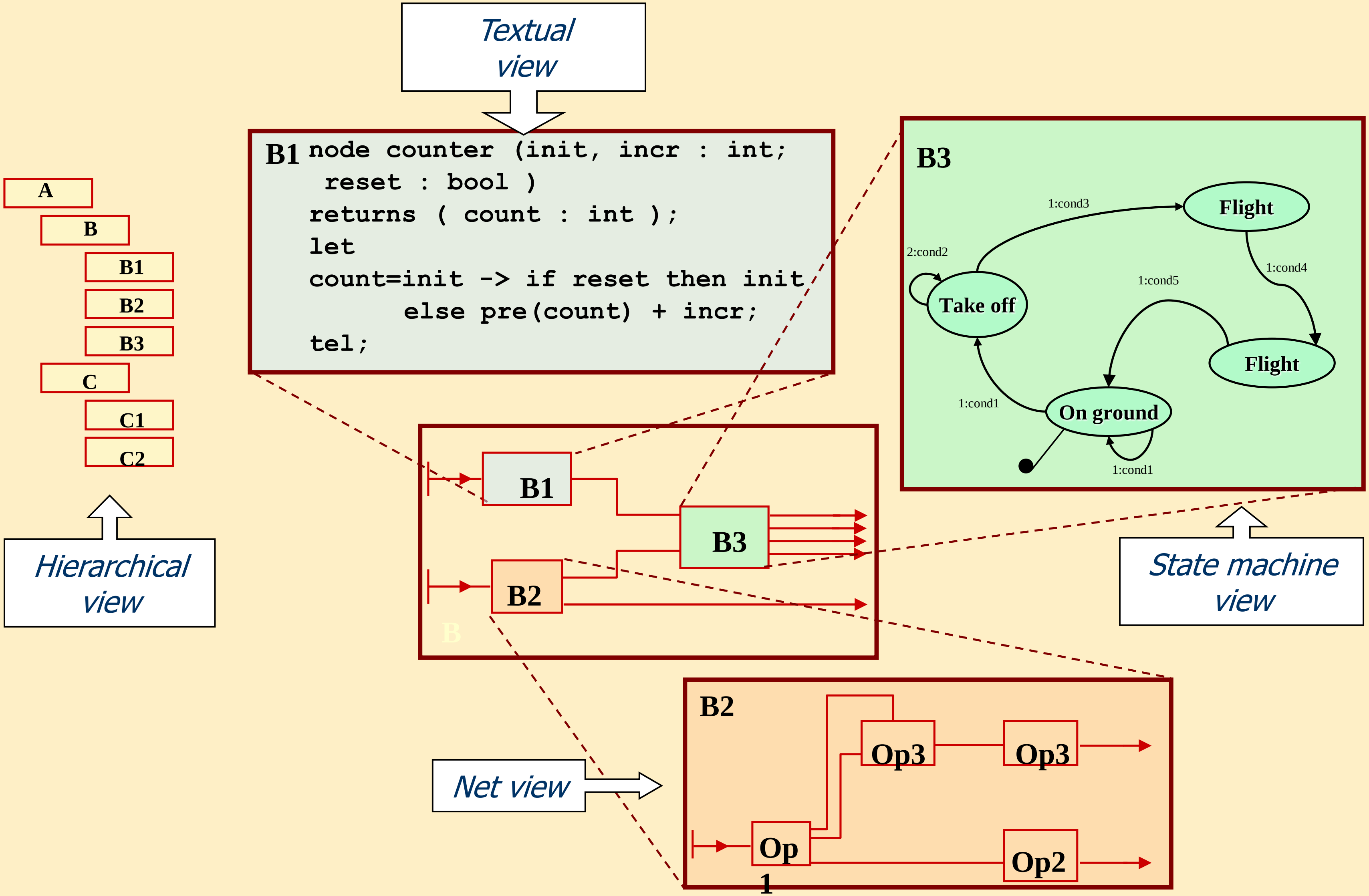




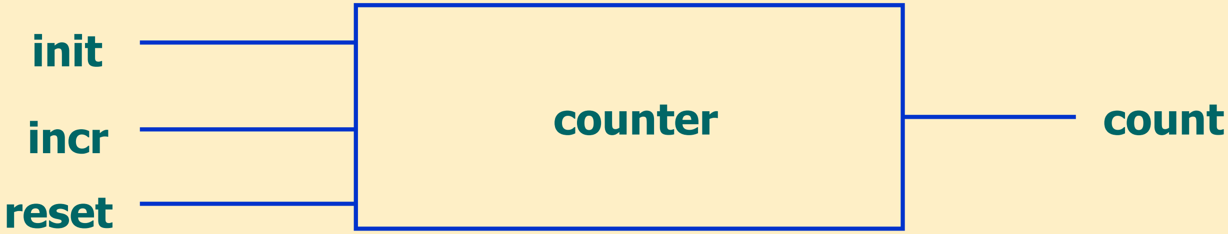
SCADE model: different views on demand



SCADE model: different views on demand



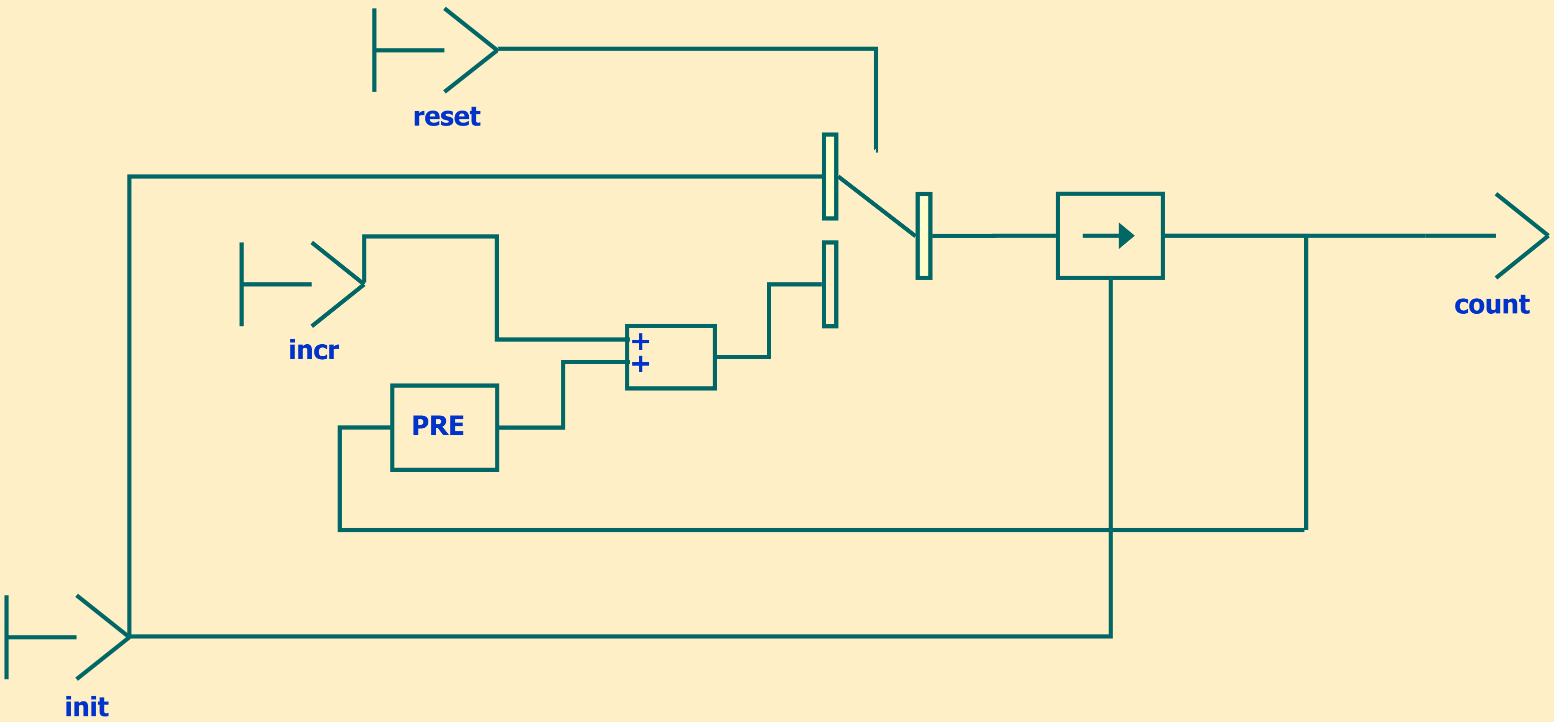
Example: Textual component



Textual description of behavior:

```
node counter (init, incr : int; reset : bool)
  returns (count : int);
let equa eq_counter [,]
   count = init -> if reset then init
                   else pre(count) + incr;
tel;
```

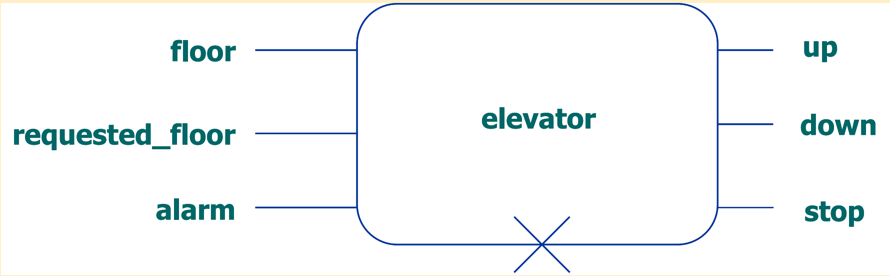
Example: Graphical component (block diagram)



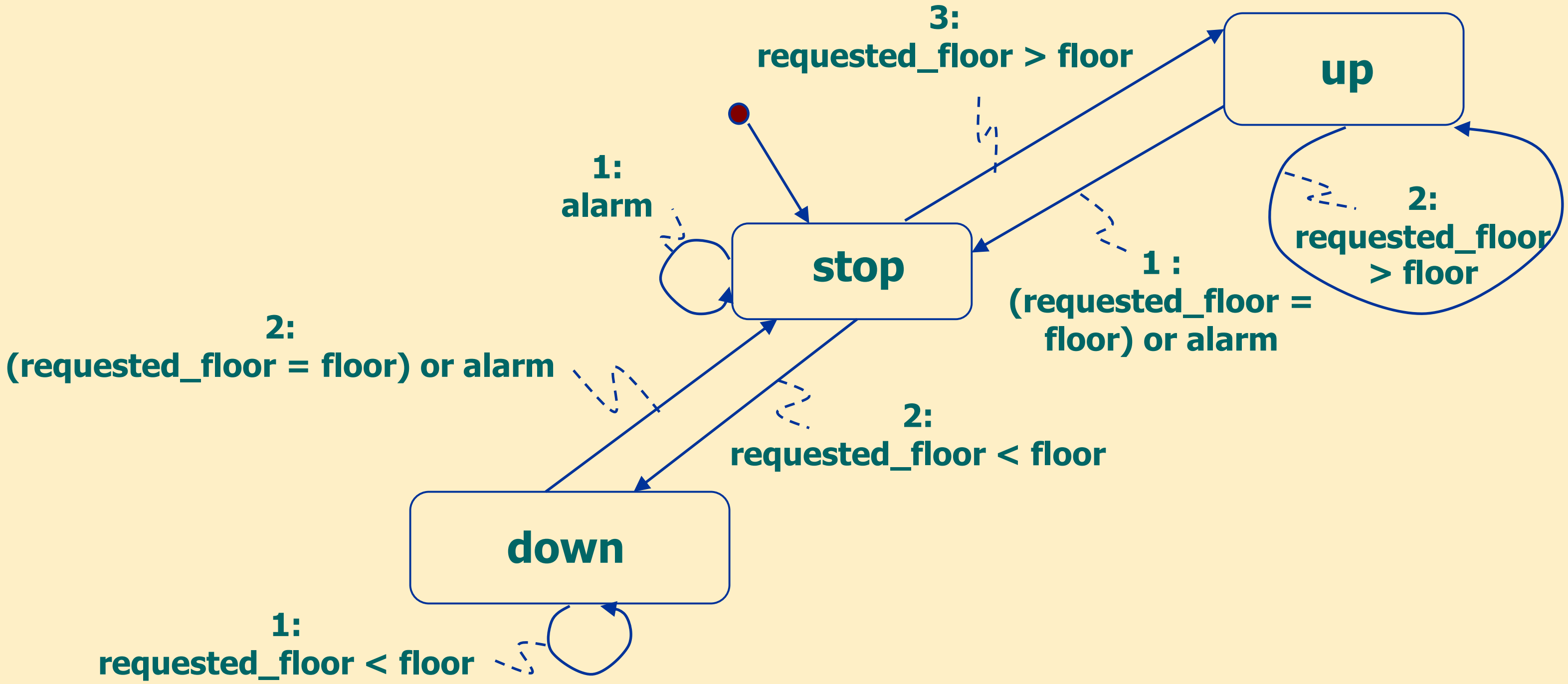
Equivalent textual description:

```
count = init -> if reset then init
          else pre(count) + incr;
```

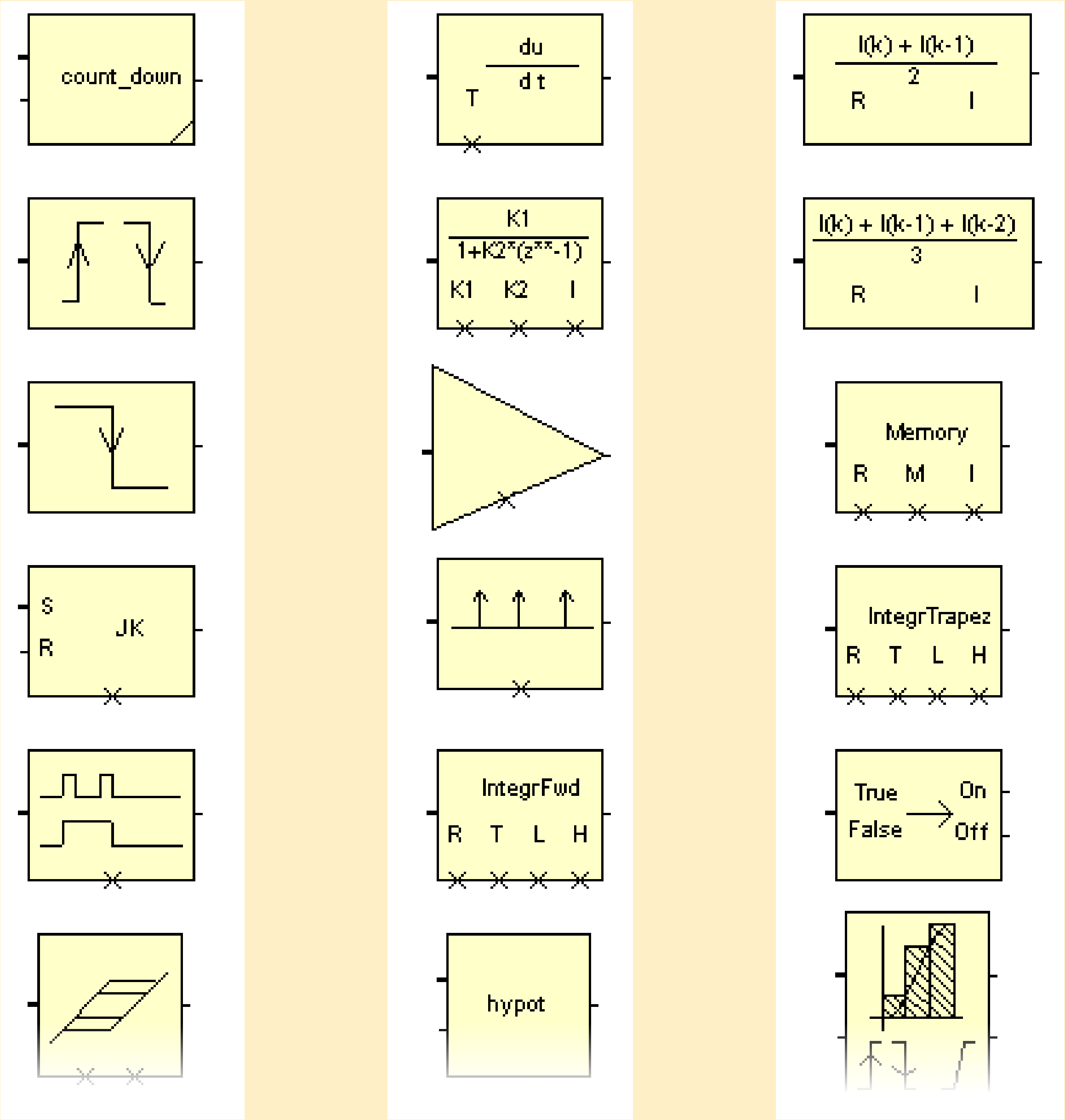
Example: State machine component



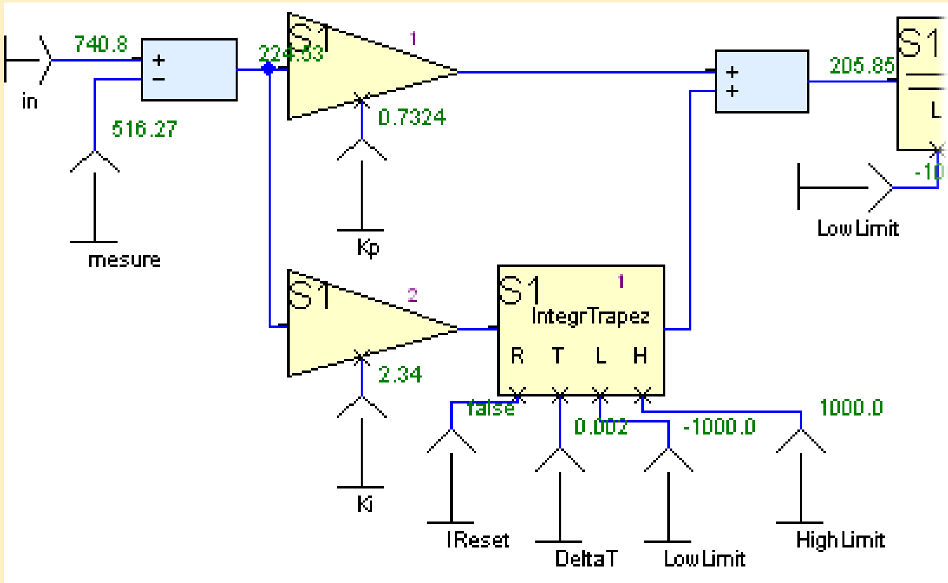
State machine of the behavior (Moore):



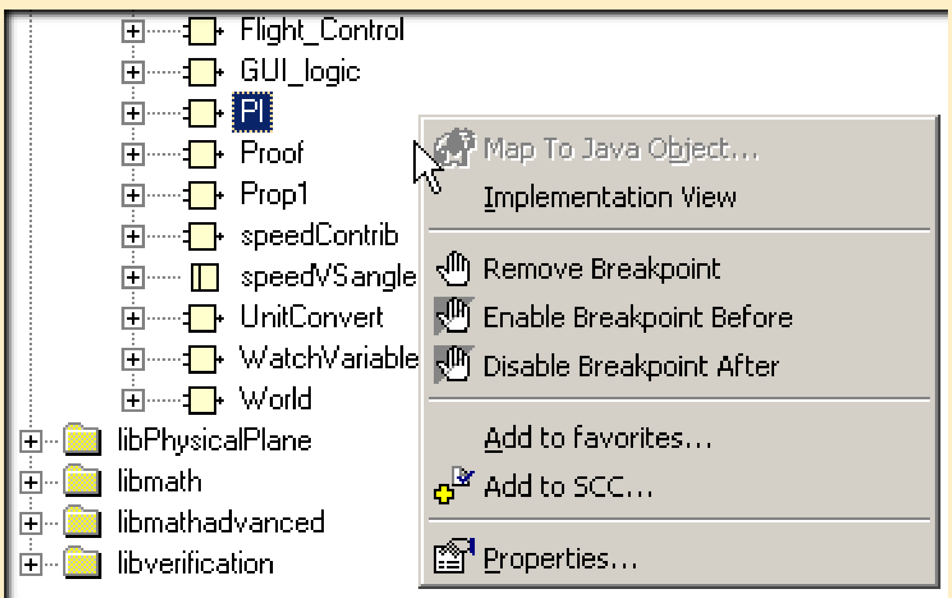
SCADE block library

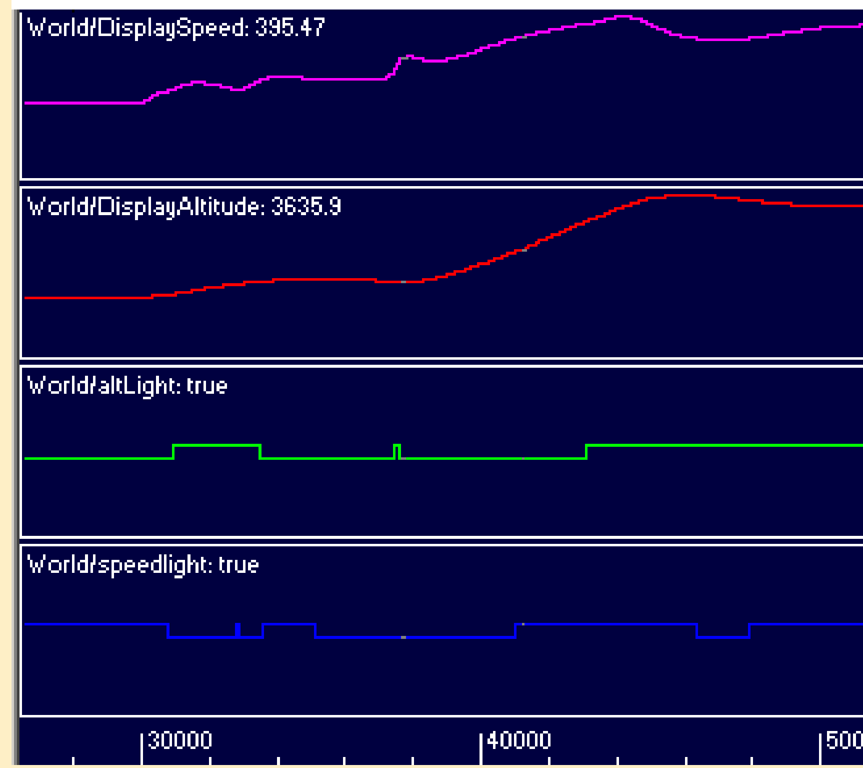


Analysis techniques: Simulation



Variable	Value
World/GUI_logic 1/SpeedSetPoint	740.8
World/GUI_logic 1/AltSetPoint	1097.3
World/Flight_Control 1/Control/elevatorCmd	1.556
World/Flight_Control 1/Control/throttleCmd	1.0
World/Flight_Control 1/Control/speedSensor	516.27
World/Flight_Control 1/Control/altSensor	165.45

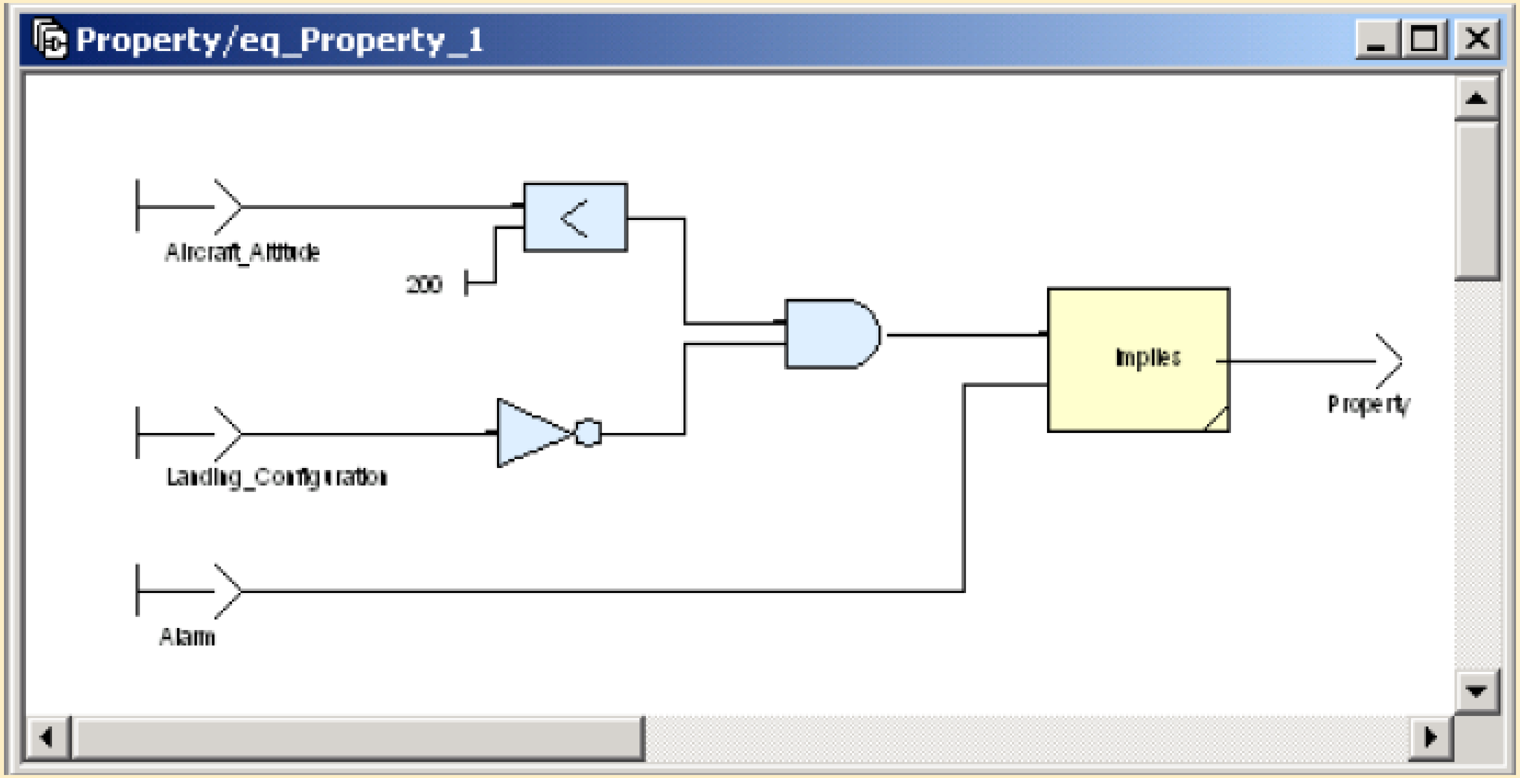




Analysis techniques: Formal theorem prover engine

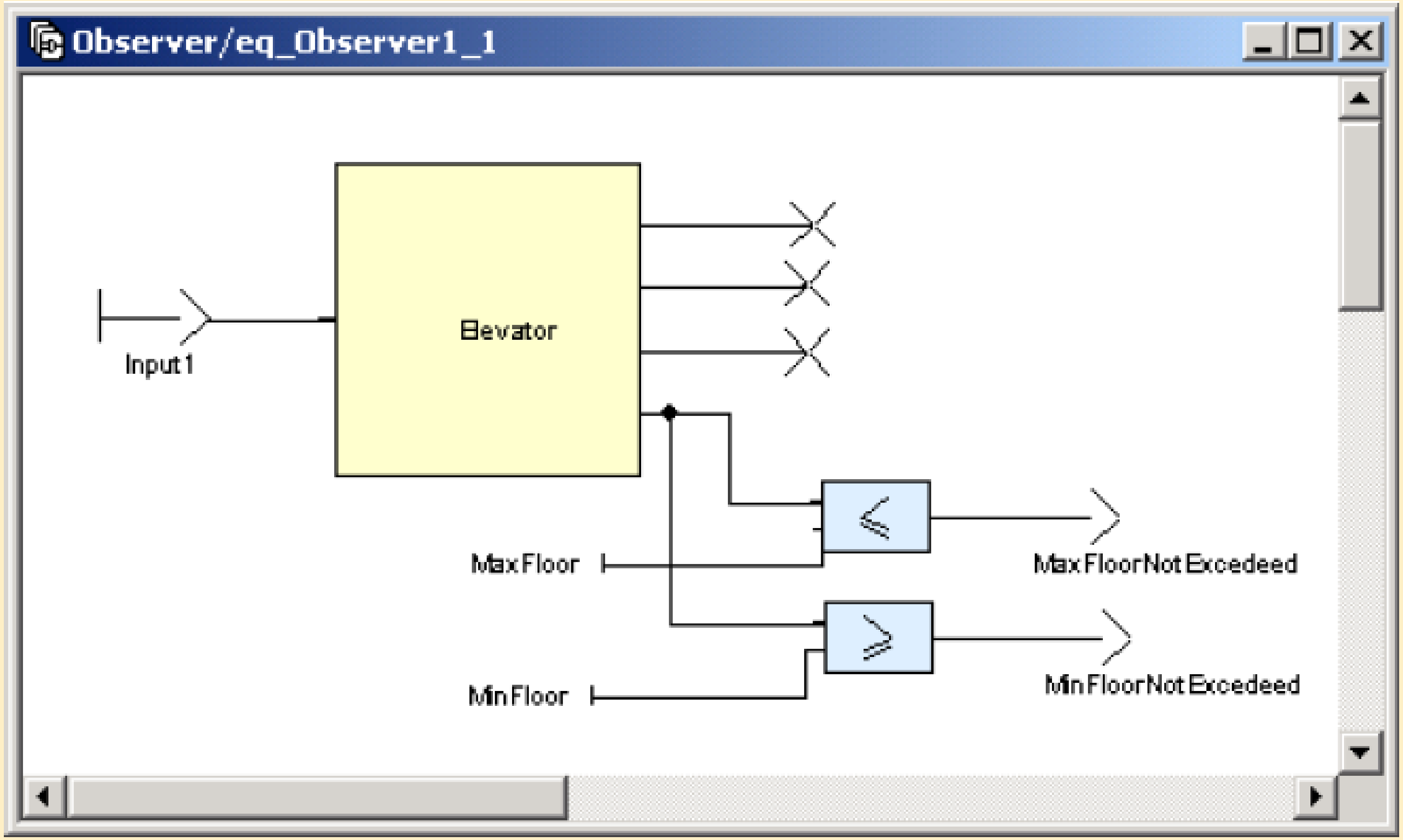
- Definition of properties: Property component (property node); output is either true or false
  - Describes correct behavior, e.g.

    Aircraft_Altitude < 200 and not Landing_Configuration implies Alarm



Analysis techniques: Formal theorem prover engine

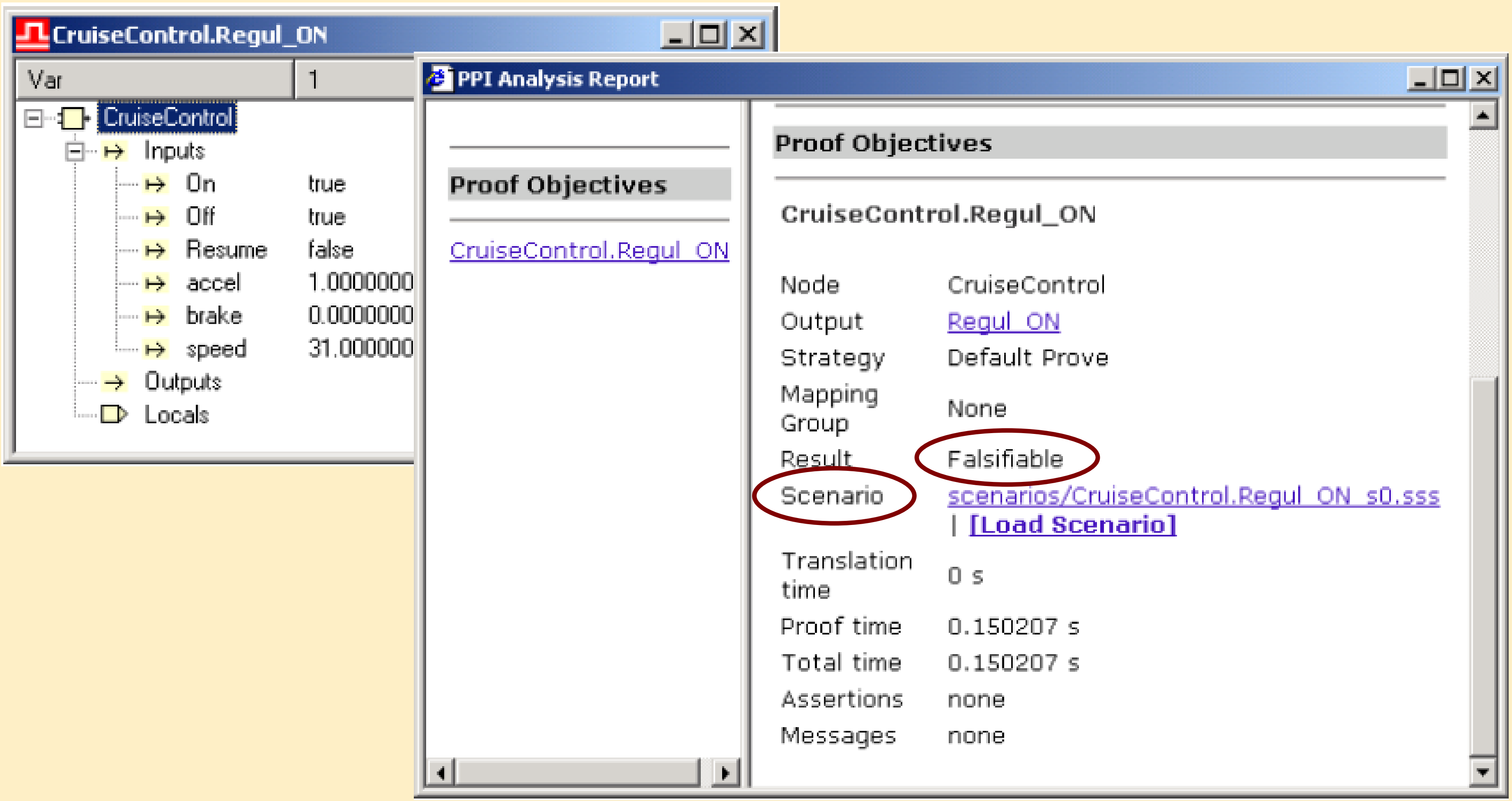
- Integration of property nodes into the design:
  Inserted as observer components (observer nodes)



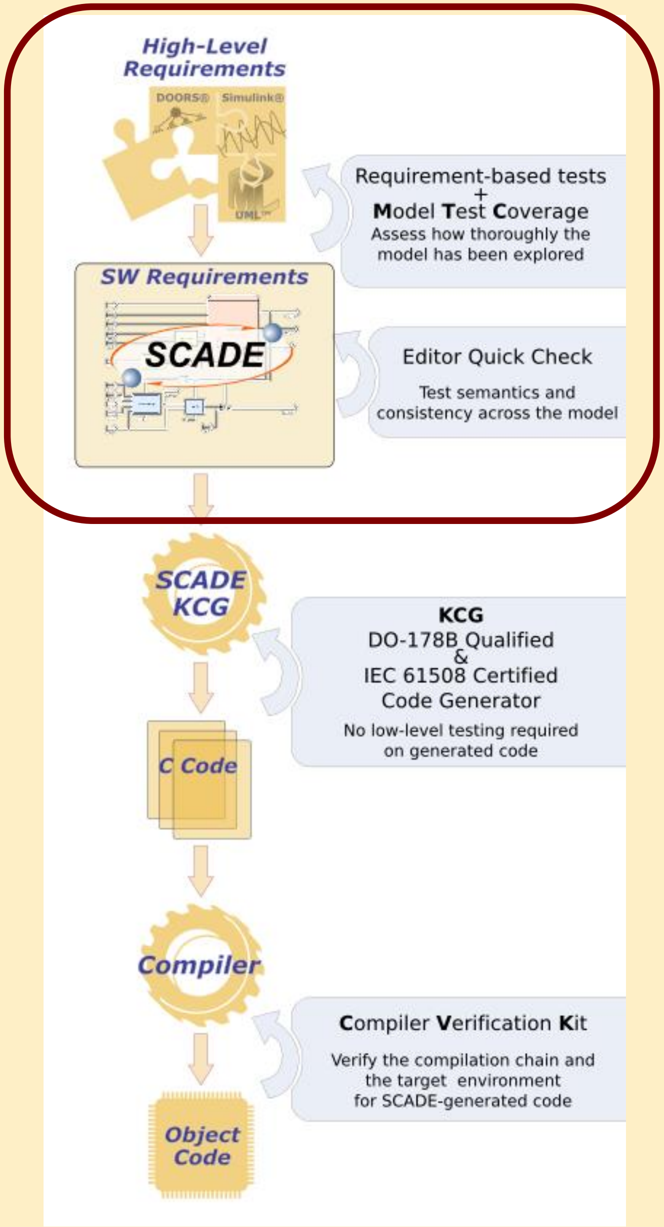
Analysis techniques: Formal theorem prover engine

To be proven: Output of observer component is always true

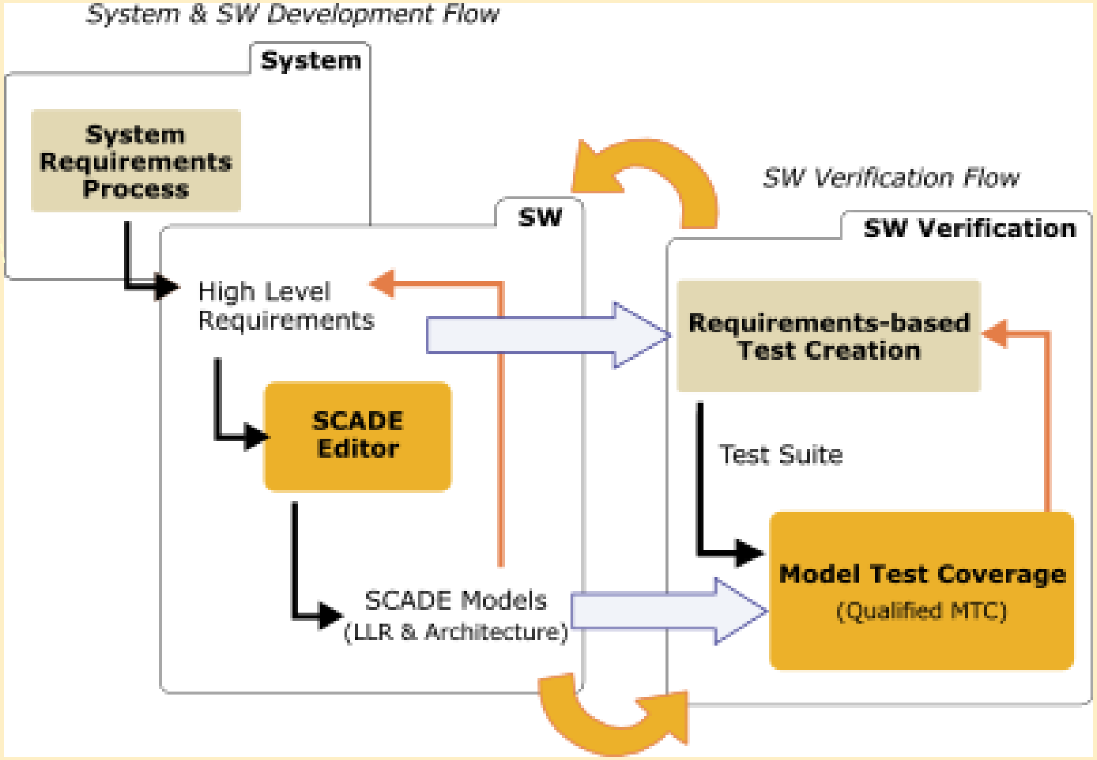
- Proof: Exhaustive search, counterexample-generation (SAT)



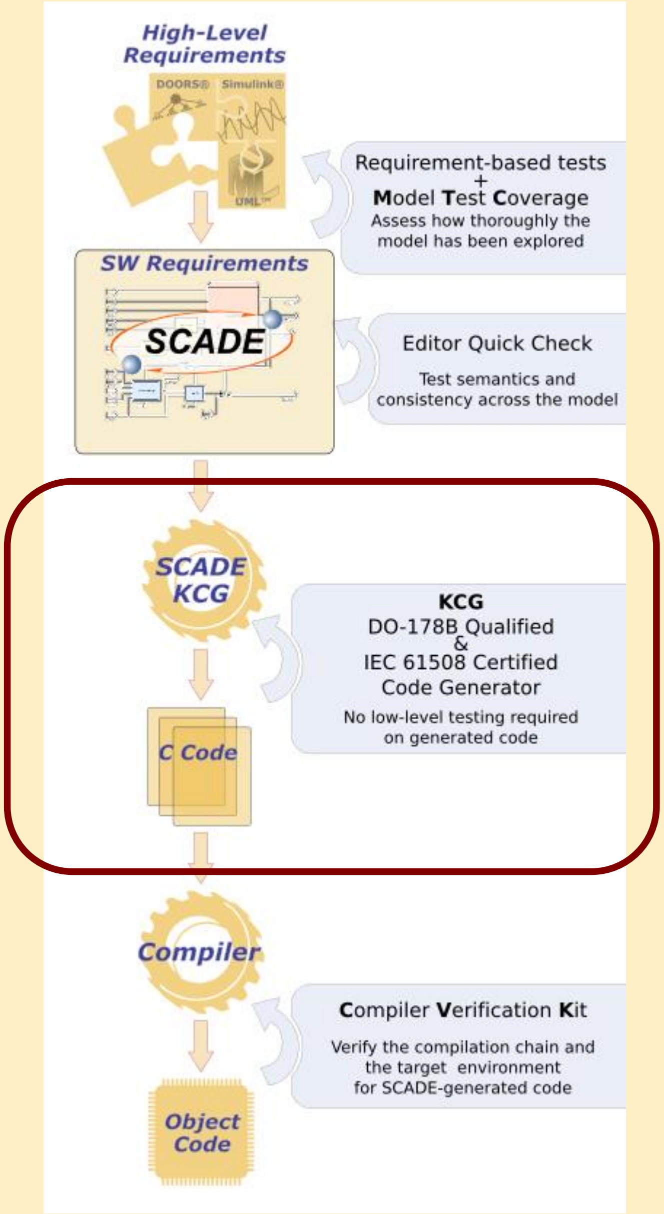
Analysis techniques: Model-based testing



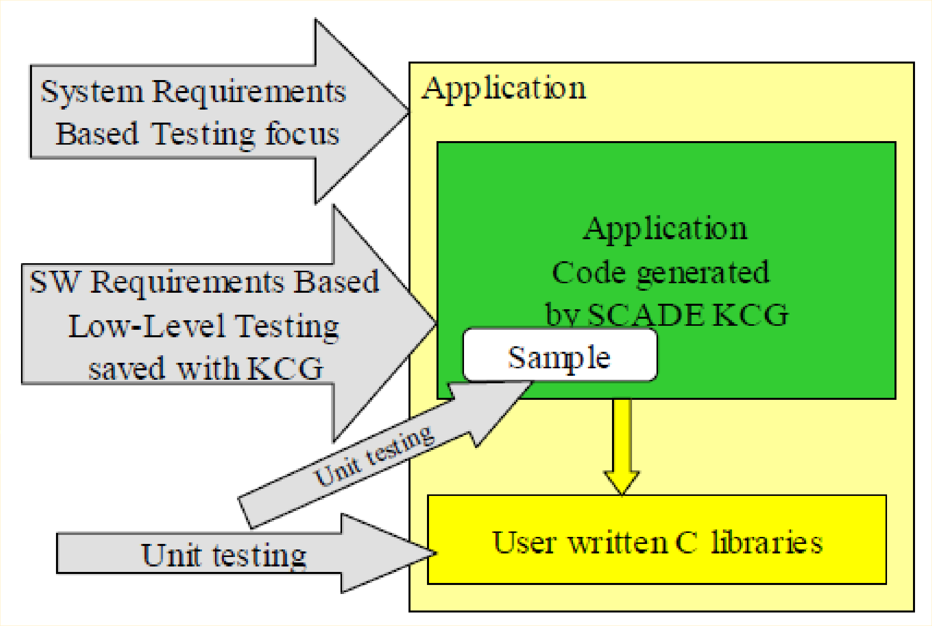
- Requirement-based test cases
- Model-based test coverage:
  - Has every element of the model been activated dynamically?
  - Untested functions can be discovered



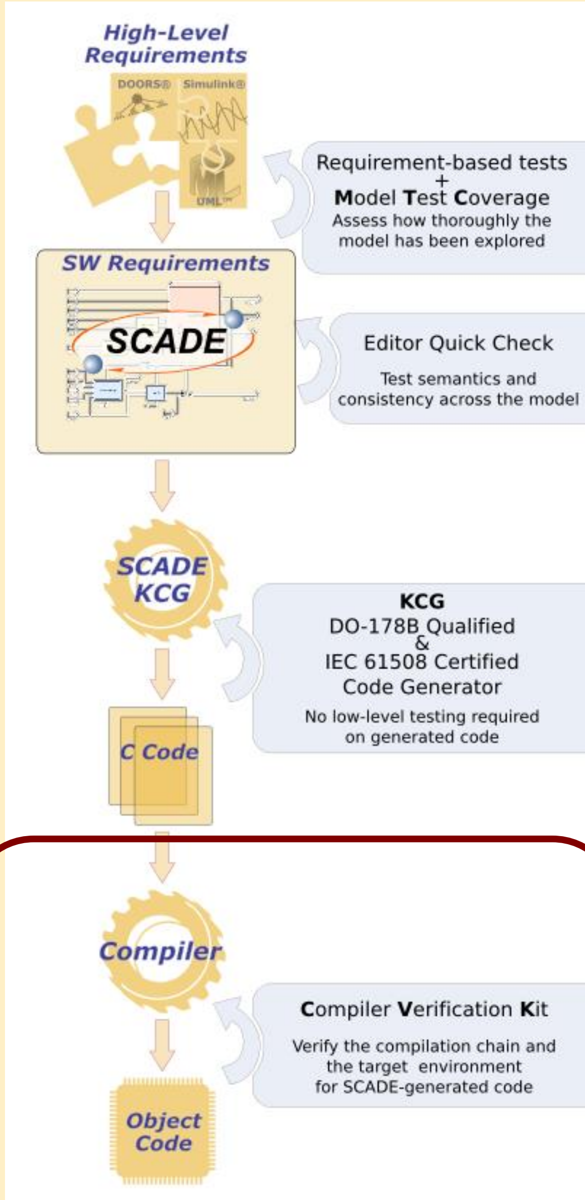
Code generation: Based on the verified model



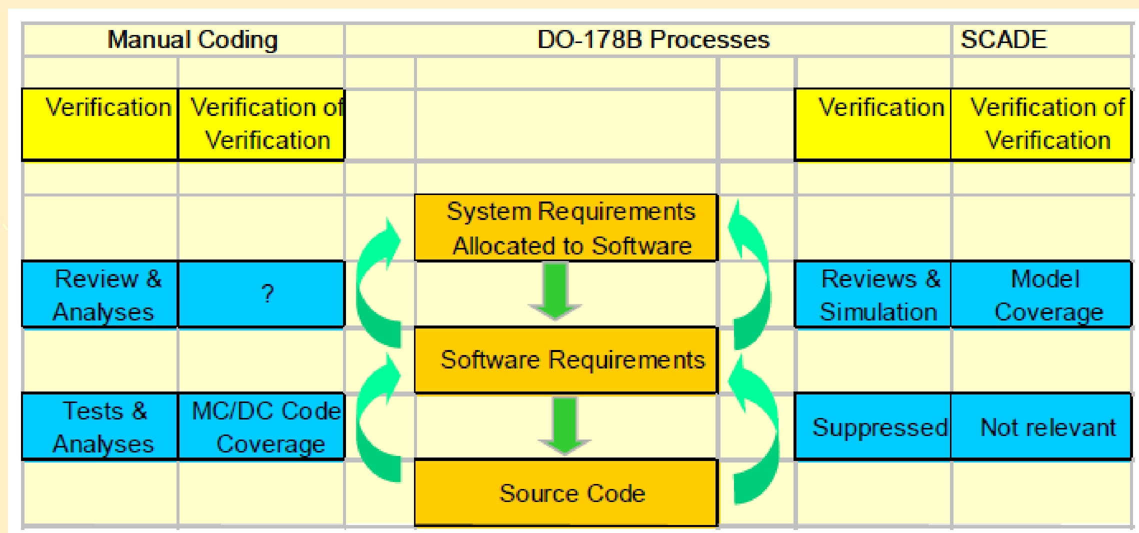
- KCG: certified code generator
  - DO-178B, IEC 61508, MISRA
  - No need for unit/model tests in case of generated code
- External or manually written code:
  - E.g. 3rd party software libraries
  - Testing is still mandatory!



Code generation: Based on the verified model



- Compiler Verification Kit:
  - Source code patterns
  - Test cases for compiled code patterns (all must pass)
- To be verified:
  - (Own) compiler
  - Execution platform



Compare manual and SCADE verification

SCADE Suite: The "certified software factory"