Synchronous Techniques for Software and Hardware Embedded Systems

Gérard Berry

Professor at Collège de France
Chief Scientist

www.esterel-technologies.com
Gerard.Berry@esterel-technologies.com

Global Agenda

1. Synchronous Embedded Systems
2. Scade 6 Language Design
3. Scade & Esterel Studio Verification
4. Demos

Synchronous Techniques for Software and Hardware Embedded Systems

1. Synchronous Embedded Systems

Gérard Berry

Professor at Collège de France
Chief Scientist

www.esterel-technologies.com
Gerard.Berry@esterel-technologies.com

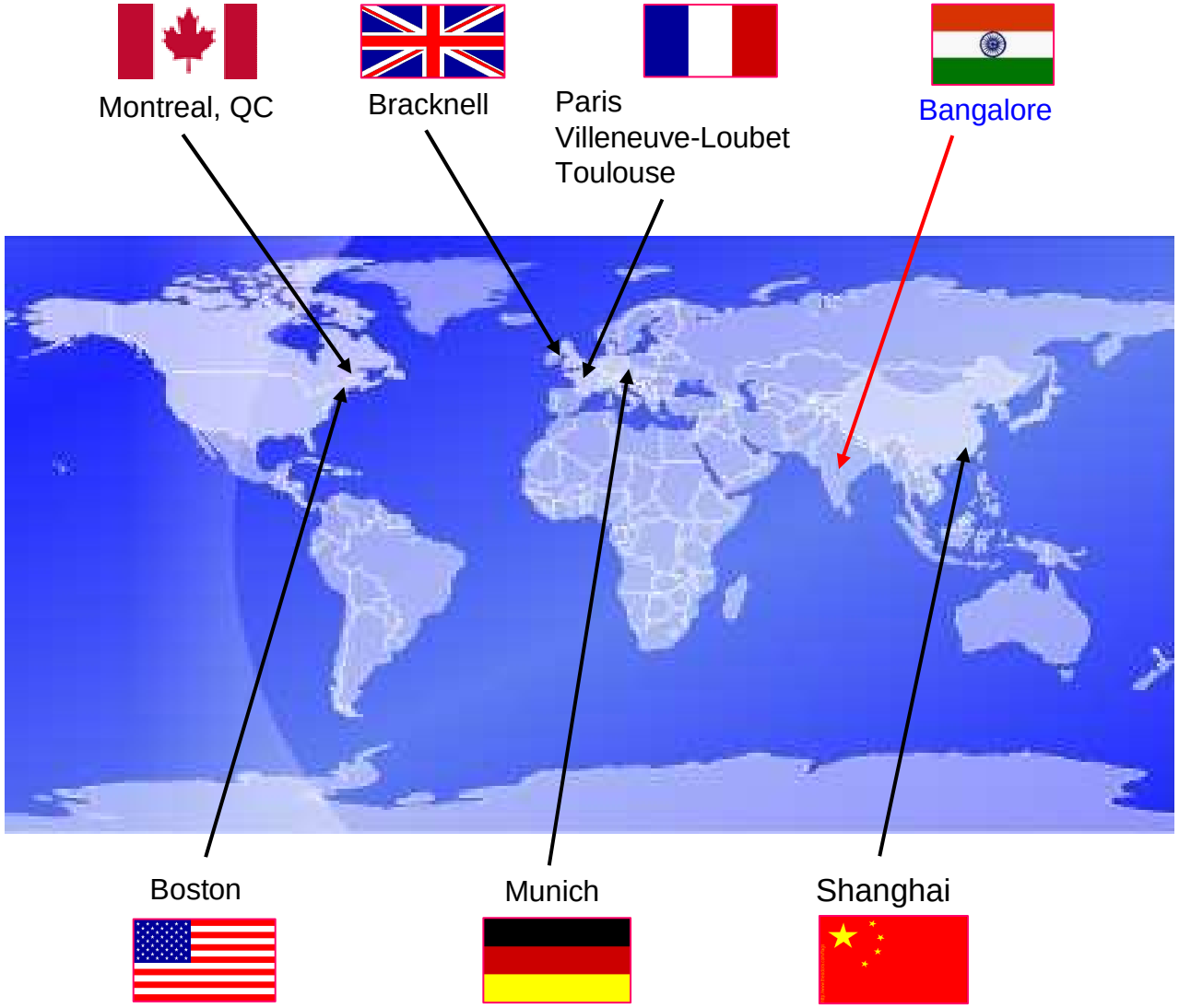
Agenda

- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

Agenda

- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

150 people, 7 countries, 150 customers



+ Partner Network:

India, Israel,

Japan, Russia…

Safety-critical certified embedded software

Avionics, railways, heavy industry, automotive

Products: SCADE Suite, Scade Display

Language: Scade 6

Circuit synthesis and verification

Consumer electronics

Product: Esterel Studio

Language : Esterel v7

SCADE Aerospace & Defense Applications

- Flight control systems
- Power management
- Reconfiguration management
- Autopilots
- Engine control systems (FADEC)
- Braking systems
- Fuel management
- Cockpit display and alarm management

EUROCOPTER – EC145

Dassault Aviation - Rafale

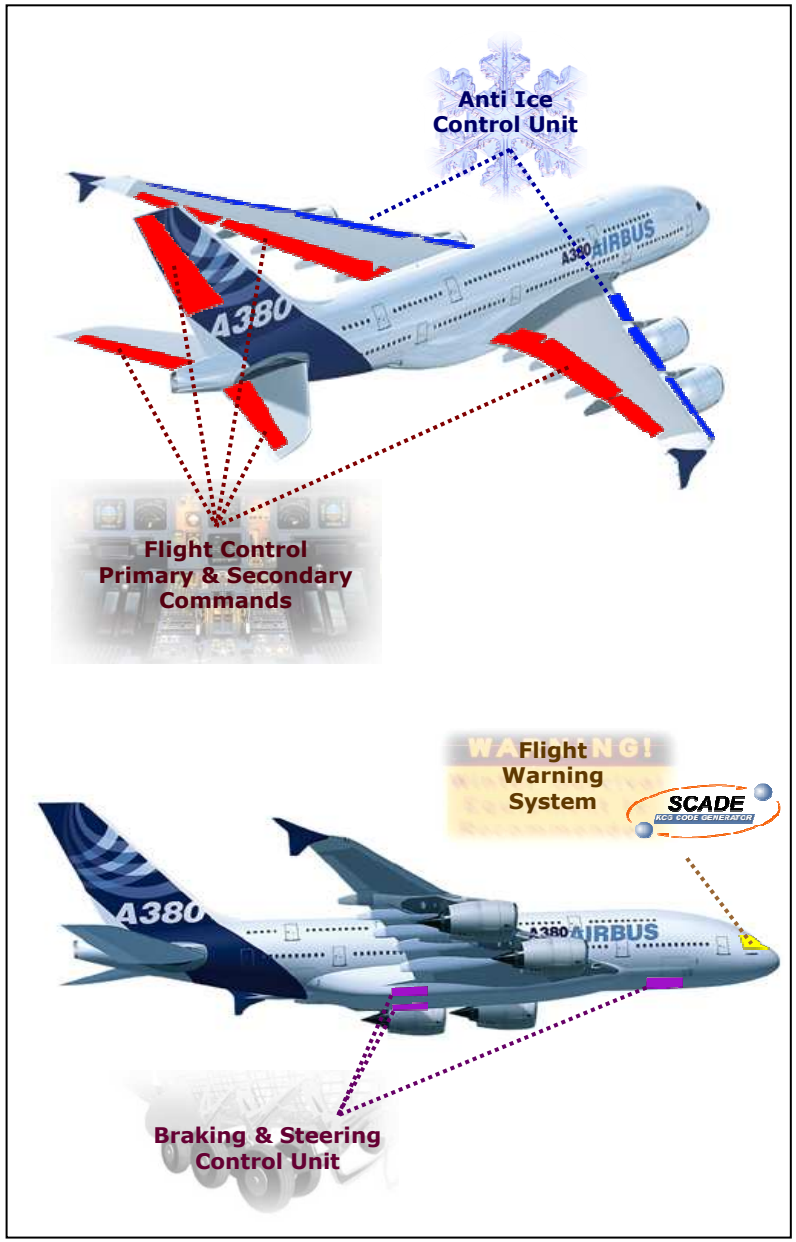
Dassault Aviation – Falcon 7X

AIRBUS – A340-600 & A380

Aeroengines by Snecma
©Snecma/Studio Pons

US Air Force - F16

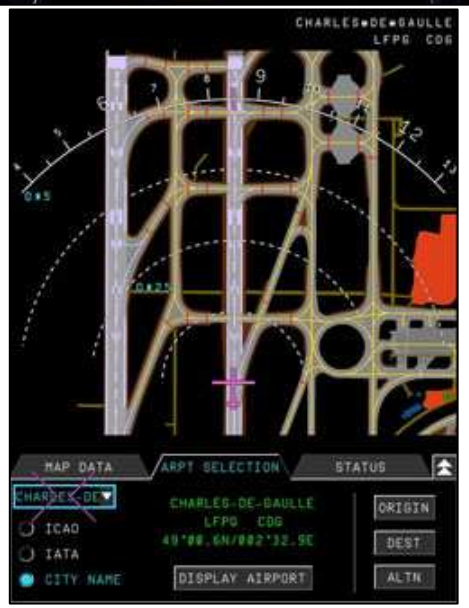
SCADE in the Airbus A380



- Flight Control system
- Flight Warning system
- Electrical Load Management system
- Anti Icing system
- Braking and Steering system
- Cockpit Display system
- Part of ATSU (Board / Ground comms)
- FADEC (Engine Control)
- EIS2 : Specification GUI Cockpit:
  - PFD : Primary Flight Display
  - ND : Navigation Display
  - EWD : Engine Warning Display
  - SD : System Display

SCADE in the A380 Cockpit

- Control and Display System (CDS)
  - Eight screens, two keyboards/cursor control devices
- Head-Up Display (HUD)
  - Incorporating LCD technology
- On-board Airport Navigation System (OANS)
  - SCADE Display & OpenGL graphics



SCADE in the Railways

- Interlocking systems control
- Signaling
- Ground stations
- Automatic Train Operations
- Train Control Systems
- Critical Graphics Displays
- Level Crossings
- Safe Platforms

(EN 50128 Certified by TÜV – up to SIL 4)

Scade in Automotive & Industrial Applications

- Automotive & 2-Wheelers:
  - Airbags
  - Braking Systems, ABS & ESP
  - Steering
  - Chassis & Suspension Systems
  - Restraining systems
  - Engine regulation
  - X-By-Wire applications

- Heavy Duty Industrial systems:
  - Cranes
  - Tractors
  - Tanks
  - Earth Moving Machines
  - Trucks
  - Construction equipment
  - Mining machines, etc…

(IEC 61508 Certified by TÜV – up to SIL3)

Scade Customers

Aerospace & Defense

Aircraft Braking Systems Corp
Airbus
AVIC1
Avionika
BAE SYSTEMS
Bundeswehr (BWB)
CASC
CETC
CS-SI
Dassault Aviation
Diehl Aerospace
EADS Military
EADS Space Transport
EADS SD&E
Edisoft
Elbit Systems
ELV
ELTA
ESA
ESG
Eurocopter
Flight Dynamics
General Electric
Goodrich
GosNIIAS
Hispano-Suiza
Honeywell CRL
Intertechnique
Liebherr-Aerospace
Lockheed Martin
MBDA
NASA
Messier-Bugatti
ONERA
Parker
Pratt & Whitney
Rockwell Collins
Rolls Royce
Rovsing
Saab Avitronics
Safran
Sagem
Snecma
Sukhoi
Turkish Aerospace (TAI)
Teuchos
Thales Airborne Systems
Thales Avionics
Turbomeca
Silver Arrow
Silver Software
Smiths Aerospace
United Arab Emirates Air Force
US Air Force
Ultra Electronics

Rail Transportation

Alstom Transportation
Ansaldo Signal
AREVA TA
Deuta Werke
RATP
SNCF
Siemens Rail Transportation
Systerel
Thales Rail Signalling Systems
Union Switch
VNIIAS

Industrial, Automotive & Energy

AREVA NP
Audi
Denso
DS&S, owned by *Rolls-Royce*
FTE
Fuji Heavy
General Motors
Hollysys
IKV
John Deere
KAERI
Korea Power
Liebherr Construction
Mitsubishi
Johnson Controls
NIAT
Nihon Seiko
Nissan
NPIC
PSA Peugeot Citroën
Qinetiq
Renault
Subaru
Toyota
Volvo Construction

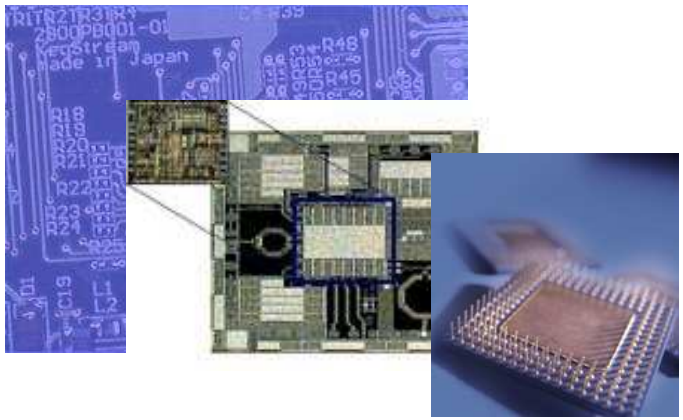
Esterel EDA Technologies & Esterel Consortium

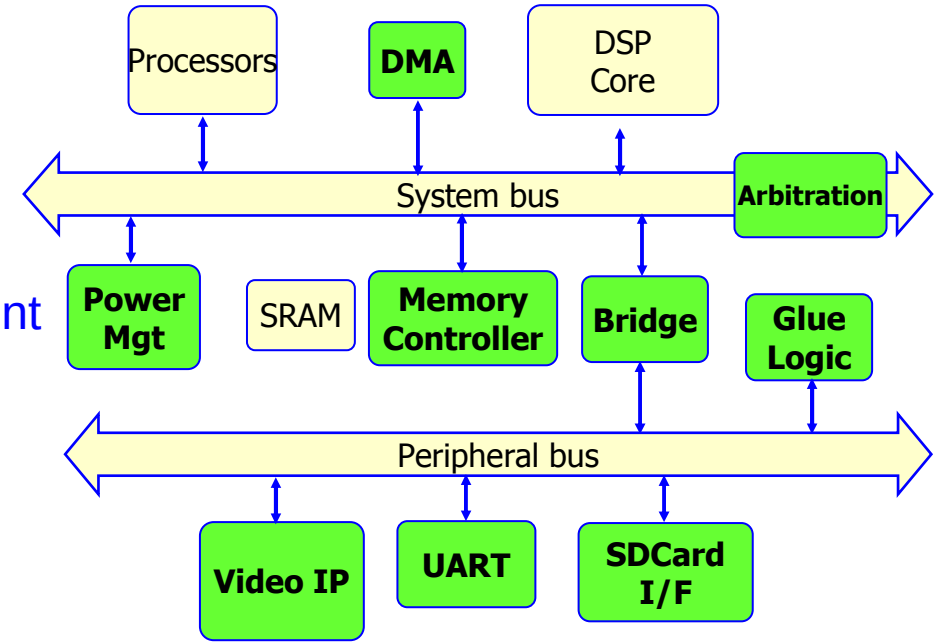
In 2001 Esterel Technologies formed a consortium of leading Semiconductor companies

- Early adopters of Esterel Studio™
- Best practice sharing about project use and design flow integration
- Collaborative specification of the main product features and roadmap
- Attended by academic partners for scientific advise
- Attended by Esterel Studio offer partners
- Support to the IEEE standardization process of the Esterel language

Esterel Application Targets

- Processor modeling and synthesis
  - Instruction Set Architecture
  - Complex instruction and data cache
  - Arbiters
  - Interrupt control
- Bus interfaces and peripheral controllers
  - Bus bridges, Networks on chips
  - Disk access, Serial ATA
  - Flash cards drivers
  - Video controllers
- Communication IPs
  - On-chip power and clock management
  - DMAs
  - Memory controllers
- Communication IPs
  - Protocols
  - Wireless links,
  - Fast serial links





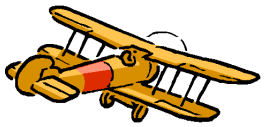
Agenda

- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

Beware of the computer!

- computers + SoCs = hardware / software mix
- complete change in device interaction
- ever-growing number of critical applications

Applications and Constraints

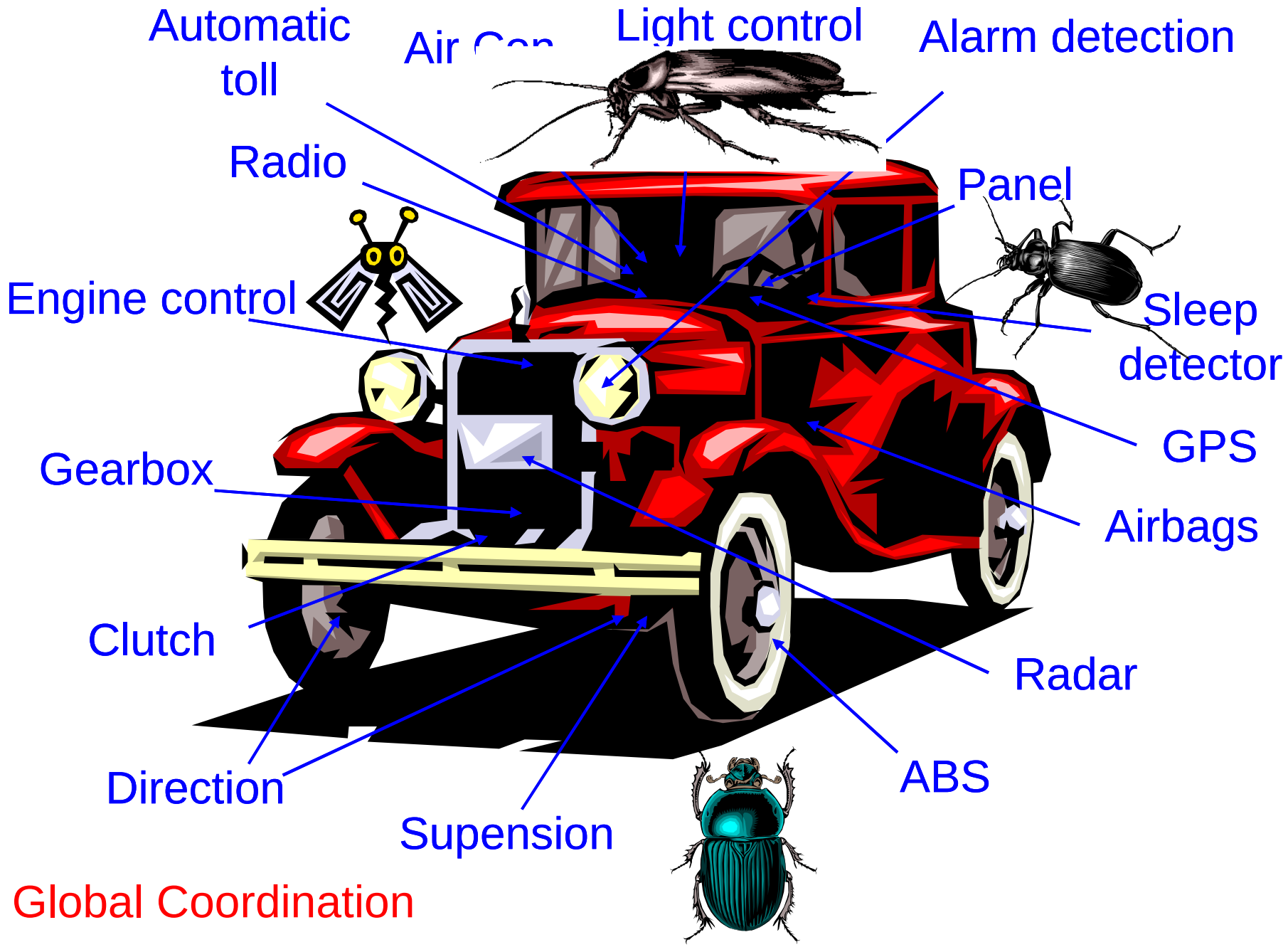


flight-control, engines, brakes, fuel, power, climate
safety-critical => certification

trajectory, attitude, image, telecom
mission-critical => very high quality

telephone, audio, TV, DVD, games
business critical => time-to market + quality

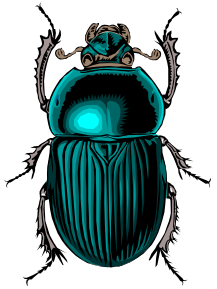
pacemakers, diabet control, robot surgeons
life-critical => TBD (I hope!)

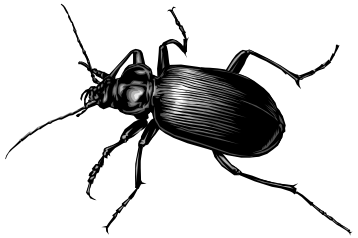


Enemy No 1 : the BUG

- Therac 25 : lethal irradiations
- Dharan's Patriot
- Ariane 501
- Mars satellites & Ro
- Automobile prob
- Intel & AMD
- Telepho amera bugs
- ...

Bug eradication campaign needed!



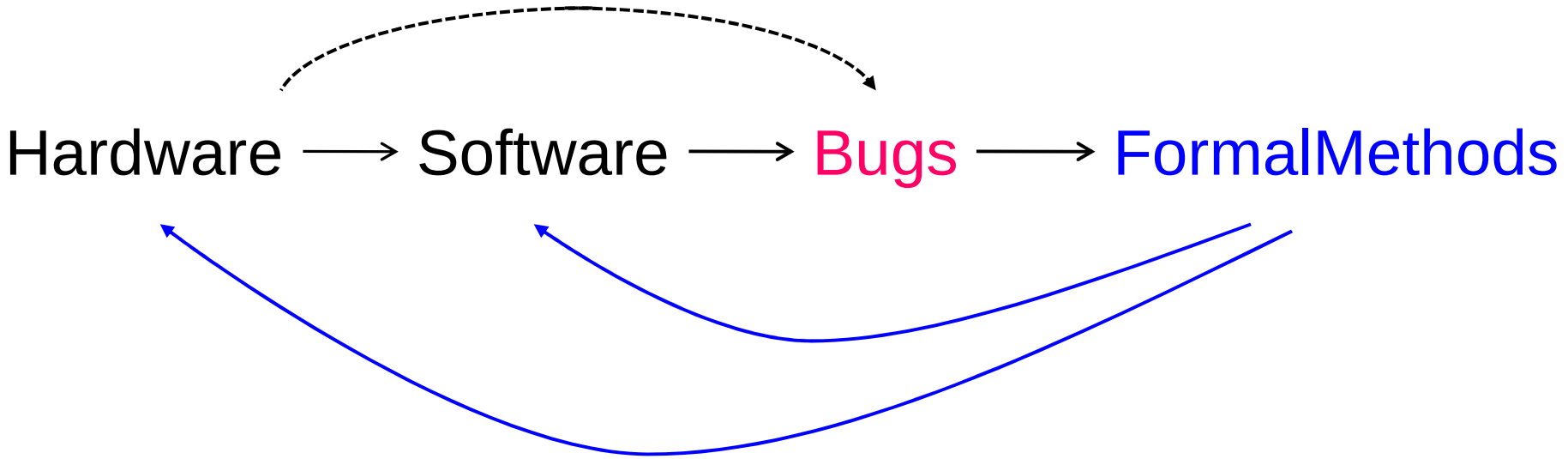


As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought.

Debugging had to be discovered.

I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.

Maurice Wilkes, 1949



How to avoid or control bugs?

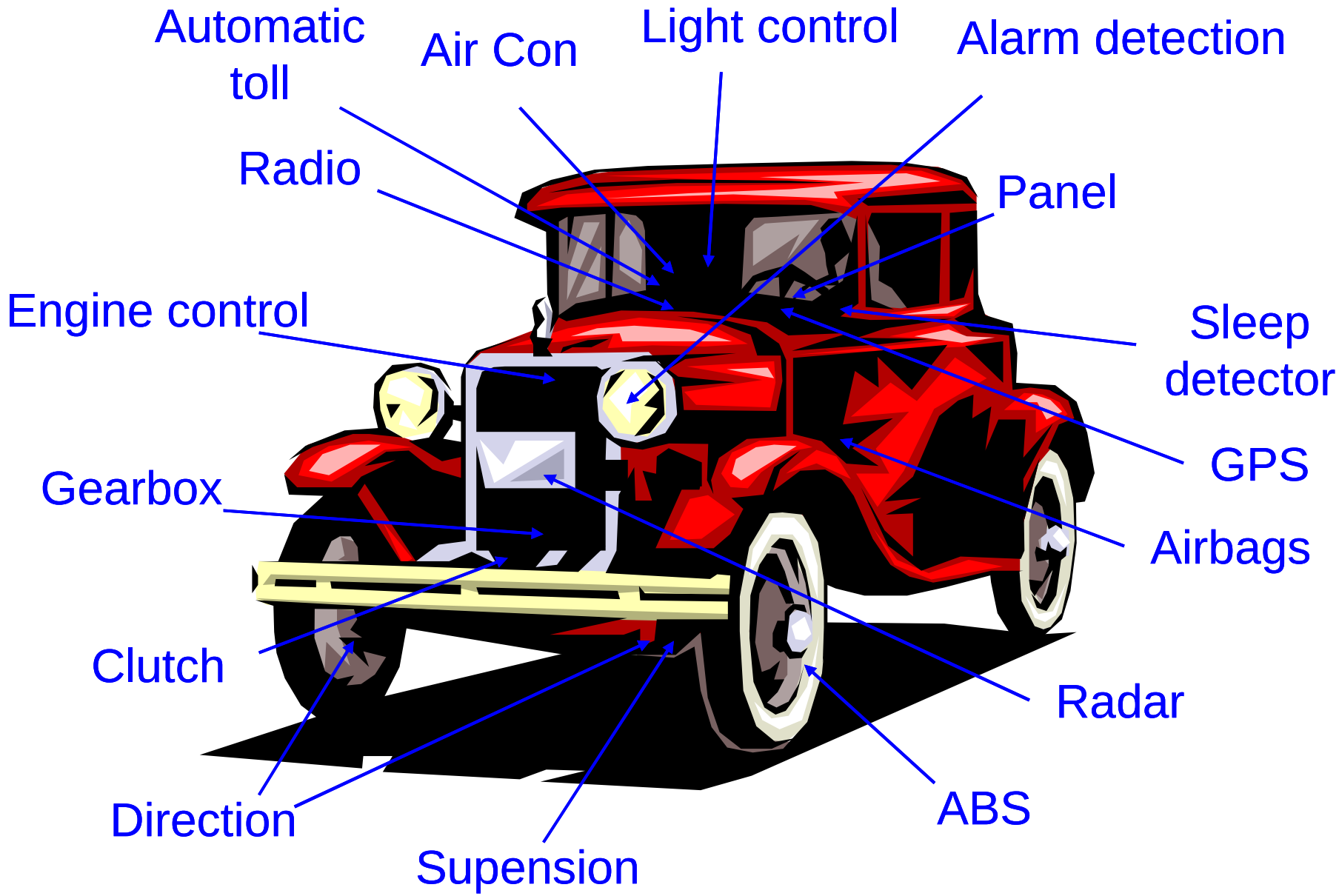
- Traditional : more verification by fancier simulation
  - but gets out of steam, more does not mean better
- Next step : better design
  - better and more reusable specifications
  - simpler computation models, formalisms, semantics
  - reduce architect / designer distance
  - reduce hardware / software distance
- Mandatory: better tooling
  - synthesis from high-level descriptions
  - formal property verification / program equivalence
  - certified libraries

Agenda

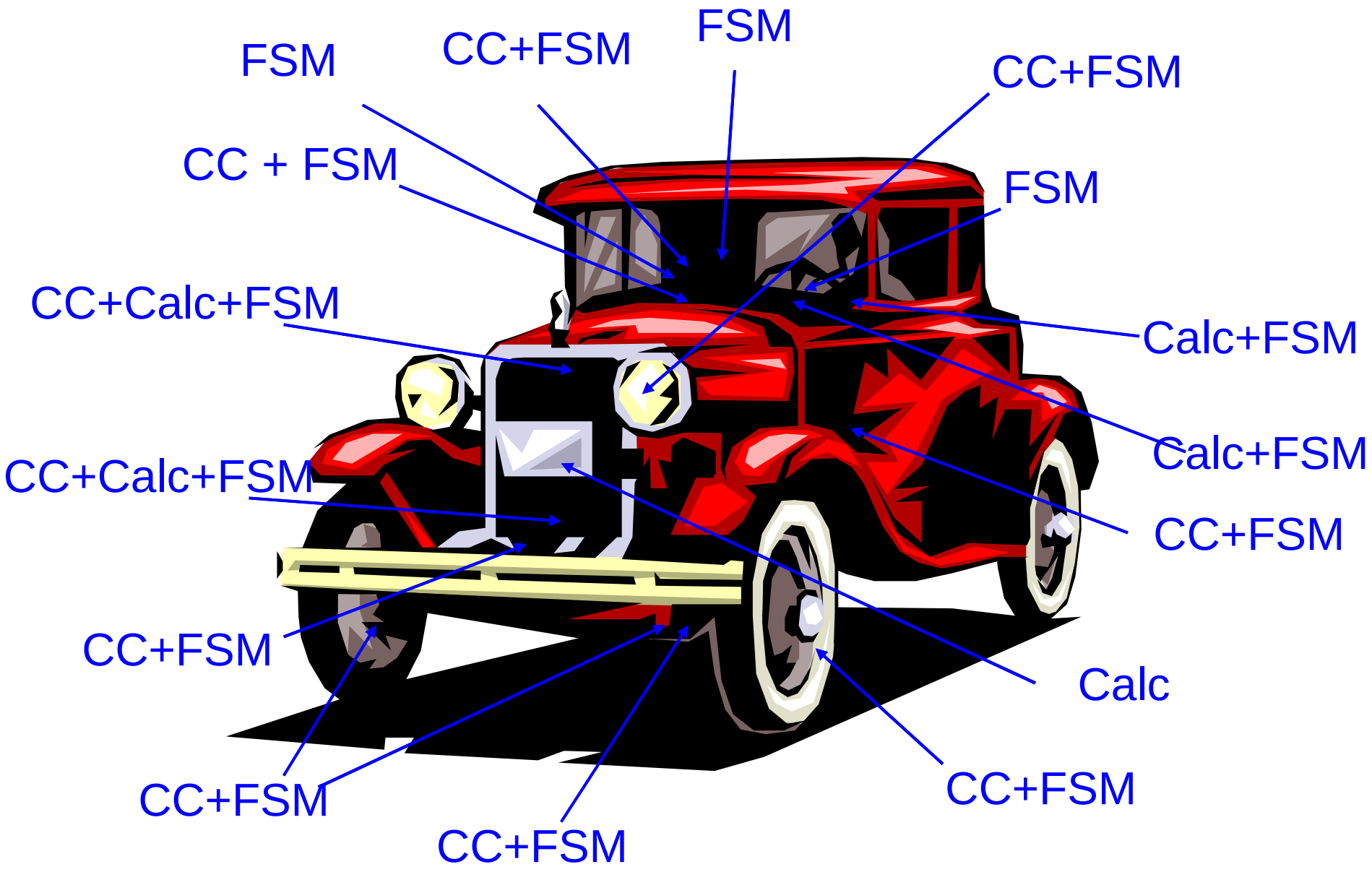
- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

Embedded Modules Anatomy

- CC : continuous control, signal processing
  - differential equations, digital filtering
  - specs and simulation with Matlab / Scilab
- FSM : finite state machines (automata)
  - discrete control, protocols, security, displays, etc.
  - flat or hierarchical FSMs
- Calc : heavy calculations
  - navigation, encryption, image processing
  - C + libraries
- Web : HMI, audio / video
  - user interaction / audio / video
  - data flow networks, Java



Global Coordination



Global Coordination : Calc+CC+FSM

Key Computation Principles

- Concurrency is fundamental
  - implicit in CC, audio / video, protocols, etc.
  - also mandatory for Web and Calc
- Determinism is fundamental
  - implicit for CC and FSM
  - who would drive a non-deterministic car?
  - can be relaxed for Web, infotainment, etc.
  - but should never be allowed to go wild !
- Physical distribution becomes fundamental
  - separation of functions, links between them
  - redundancy for fault-tolerance
  - global time needed for distributed control

Bad News or Good News?

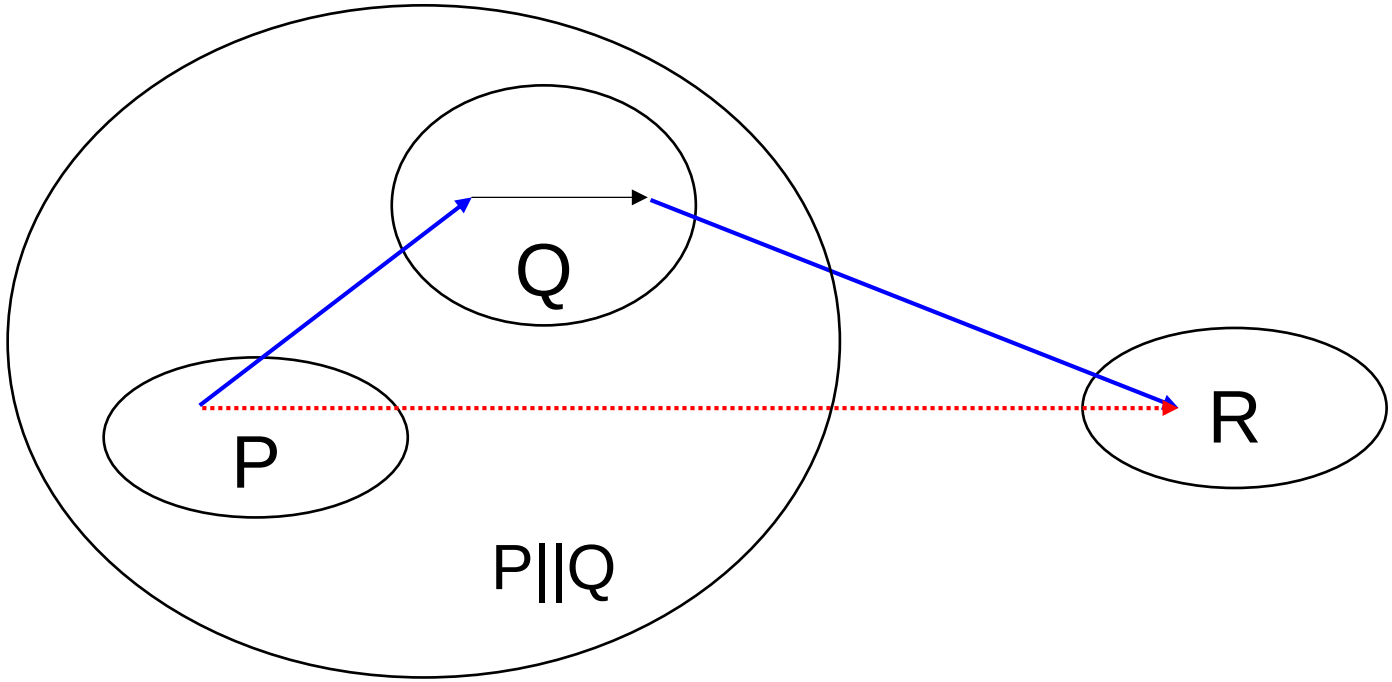
The Classical Software Development Model is Inadequate

- Turing complete => too rich, too hard to check
- OS- or thread-based concurrency => too hard too check interference, non-determinism
- CC implementation too indirect (manual action scheduling)

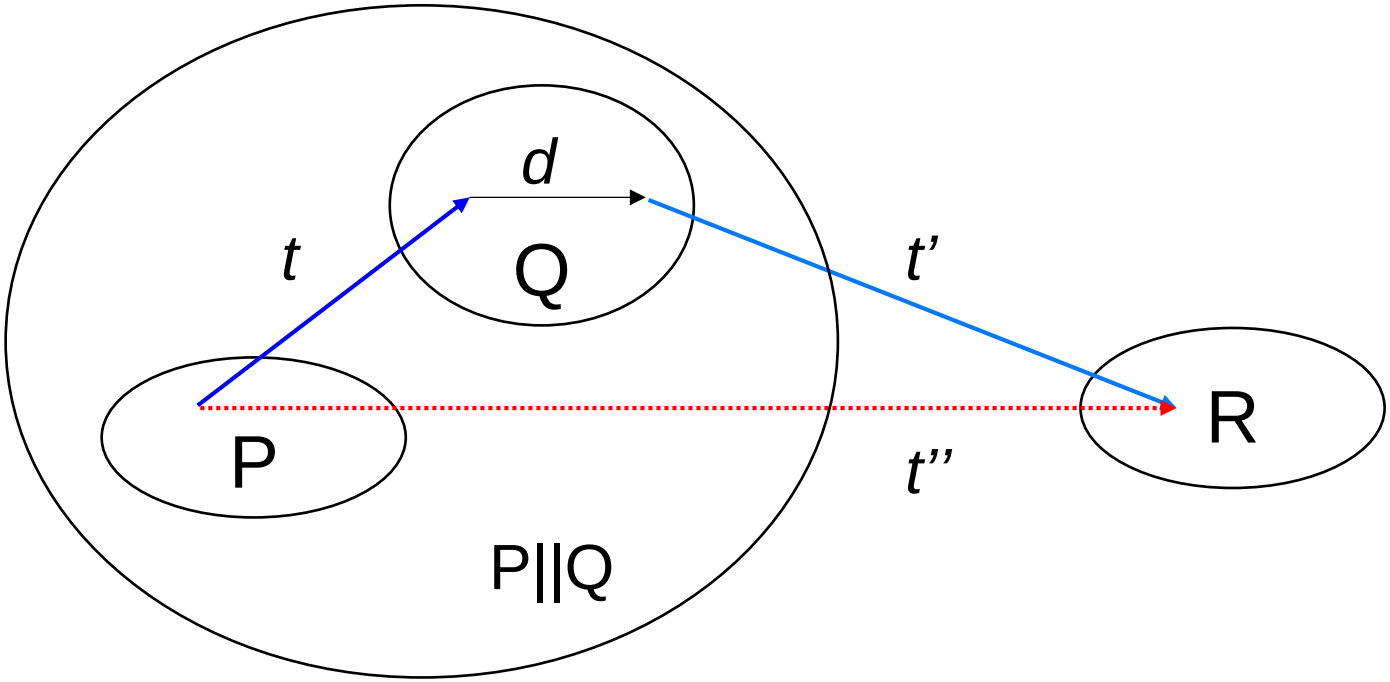
The Classical Hardware Development Model is Inadequate

- Structural RTL descriptions hide behavior dynamics
- Concurrency OK, but sequencing very indirect
- Quite old language basis, semantics too vague

Other models are needed !



Concurrency : the compositionality principle



$$t'' = t + d + t'$$

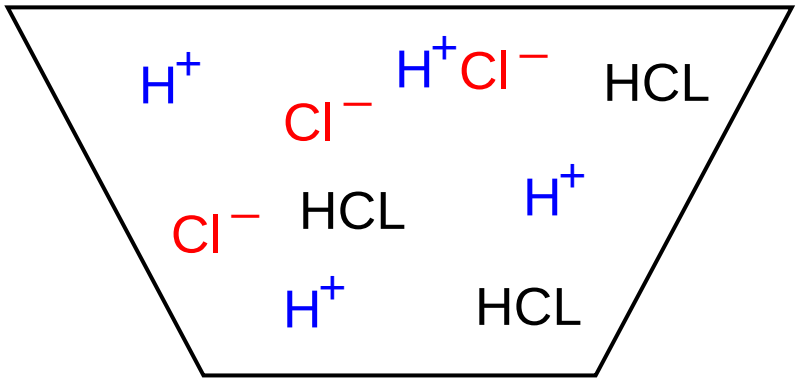
$$t'' \sim t \sim d \sim t'$$

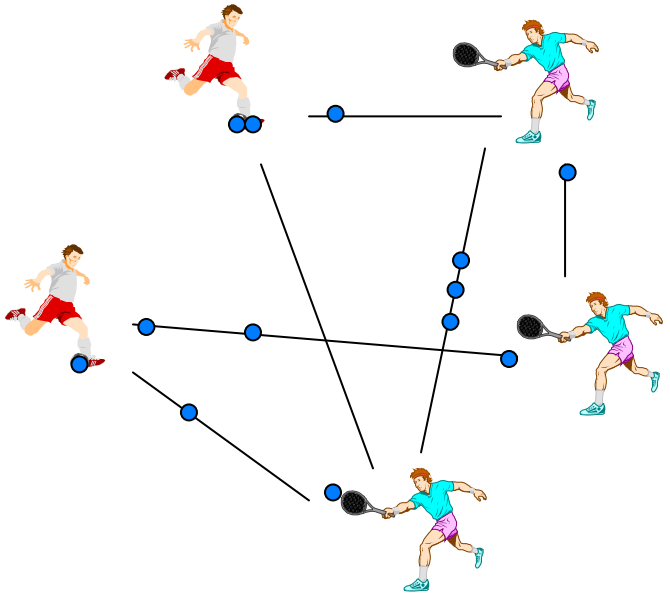
$$t \sim t + t$$

Only 3 solutions :

- *t* arbitrary asynchrony
- *t* = 0 synchrony
- t predictable vibration

Arbitrary Delay : Brownian Motion





Chemical reaction

Internet routing

$H^+ + Cl^- \longleftrightarrow HCL$

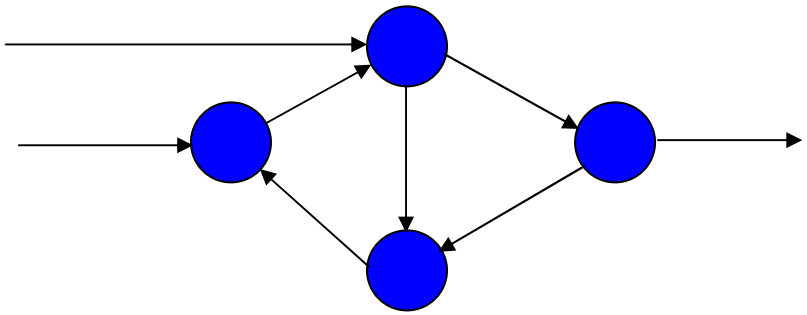
Models : Kahn networks, CSP / ADA,..., π-calculus, CHAM, Join-Calculus, Ambients,

Eratosthenes-Darwin Sieve : p, kp → p

Banâtre - Le Métayer : GAMMA

Berry - Boudol : CHAM

Kahn Networks

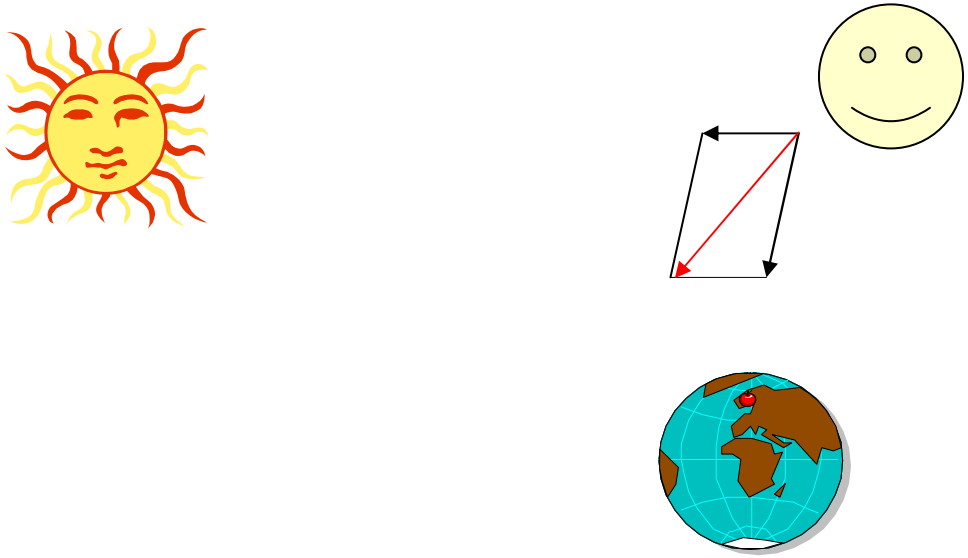


nodes = deterministic programs

arrows = infinite fifos

- result-deterministic (independent of computation order)
- easy semantics by flow equations
- heavily used in streaming applications (audio, TV)
- but semantics easily breaks down when language extended...

Zero delay: Newtonian Mechanics



Concurrency + Determinism
Calculations are feasible

The most difficult real-time maneuver ever

Refer to a fabulous drawing of Hergé's
"On a Marché sur la Lune", in English
"Explorers on the Moon". French edition, page 10,
first drawing.

Drunk Captain Haddock has become a satellite of the Adonis asteroid. To catch him, Tintin, courageously standing on the rocket's side, asked Pr. Calculus to start the rocket's atomic engine. At precisely the right time, he shouts "STOP"!

This is the trickiest real-time manoeuver ever performed by man. It required a perfect understanding of Newtonian Mechanics and absolute synchrony.

The Esterel Runner

```
trap HeartAttack in
    every Morning do
        abort
            loop
                abort  run Slowly  when  100 Meter ;
                abort
                    every Step do
                        run Jump || run Breathe || CheckHeart
                    end every
                when 15 Second ;                    exit HeartAttack
                run FullSpeed
            each  Lap
        when  4 Lap
    end every
handle HeartAttack fo
   run RushToHospital
end trap
```

Predictable time = vibration

Nothing can illustrate vibration better than Bianca Castafiore, Hergé's famous prima donna. See [1] for details. The power of her voice forcibly shakes the microphone and the ears of the poor spectators.

[1] King's Ottokar Sceptre, Hergé, page 29, last drawing.

propagation of light, sound, electrons, program counter...

Full Abstraction

Bianca Castafiore singing for the King Muskar XII in Klow, Syldavia. King's Ottokar Sceptre, page 38, first drawing.

Although the speed of sounds is finite, it is fast enough to look infinite. Full abstraction!

If room is small enough, Bianca, Walter, and listeners can neglect the speed of sound

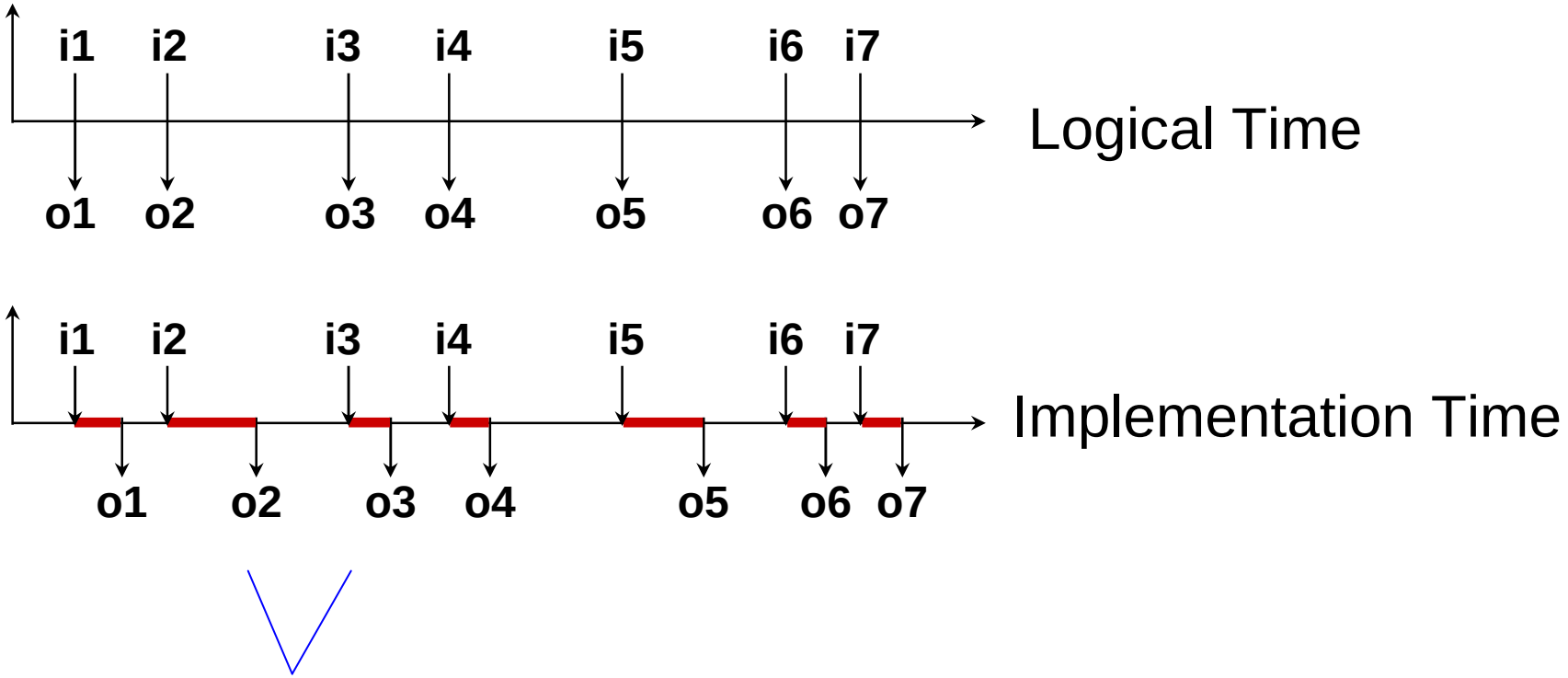
Specify with zero-delay

Implement with predictable delay

Control room size

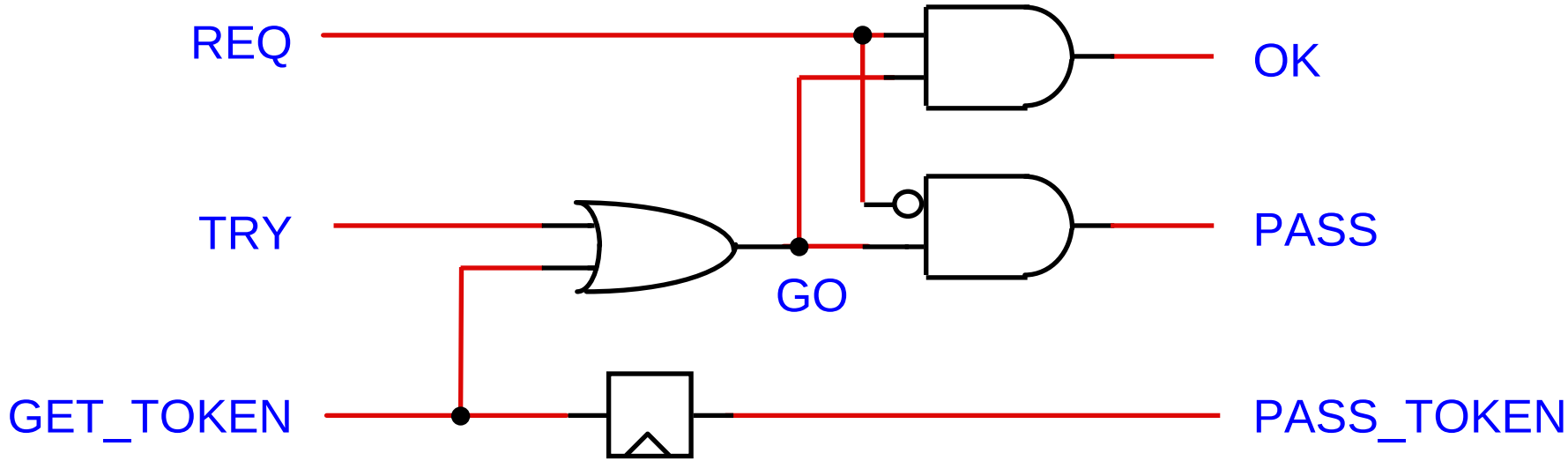
The Synchronous Models of Time

Time becomes a logical notion



WCET = guarantee of no-overlap

Hardware Synchrony: the RTL model

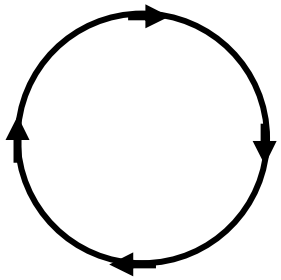


OK = REQ and GO
PASS = not REQ and GO
GO = TRY or GET_TOKEN
PASS_TOKEN = reg(GET_TOKEN)

Room size control = timing closure

Software Synchronous Systems

Cycle based



read inputs

compute reaction

produce outputs

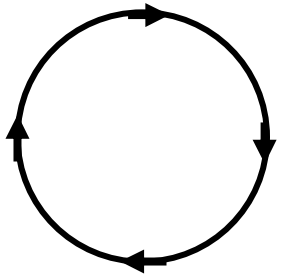
Synchronous = 0-delay = within the same cycle

propagate control

propagate signals

No interference between I/O and computation
Room size control = Worst Case Execution Time (AbsInt)

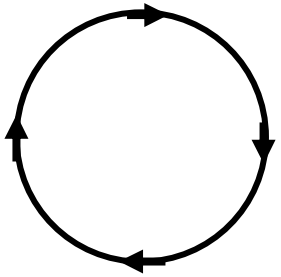
Concurrency = Cycle Fusion



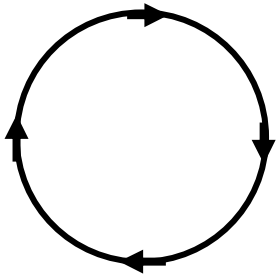
```
input X, Z;
output Y, T;
Y = X+1;
T=Z/2
```

Y

Z



```
input Y;
output Z;
Z = Y*3;
```

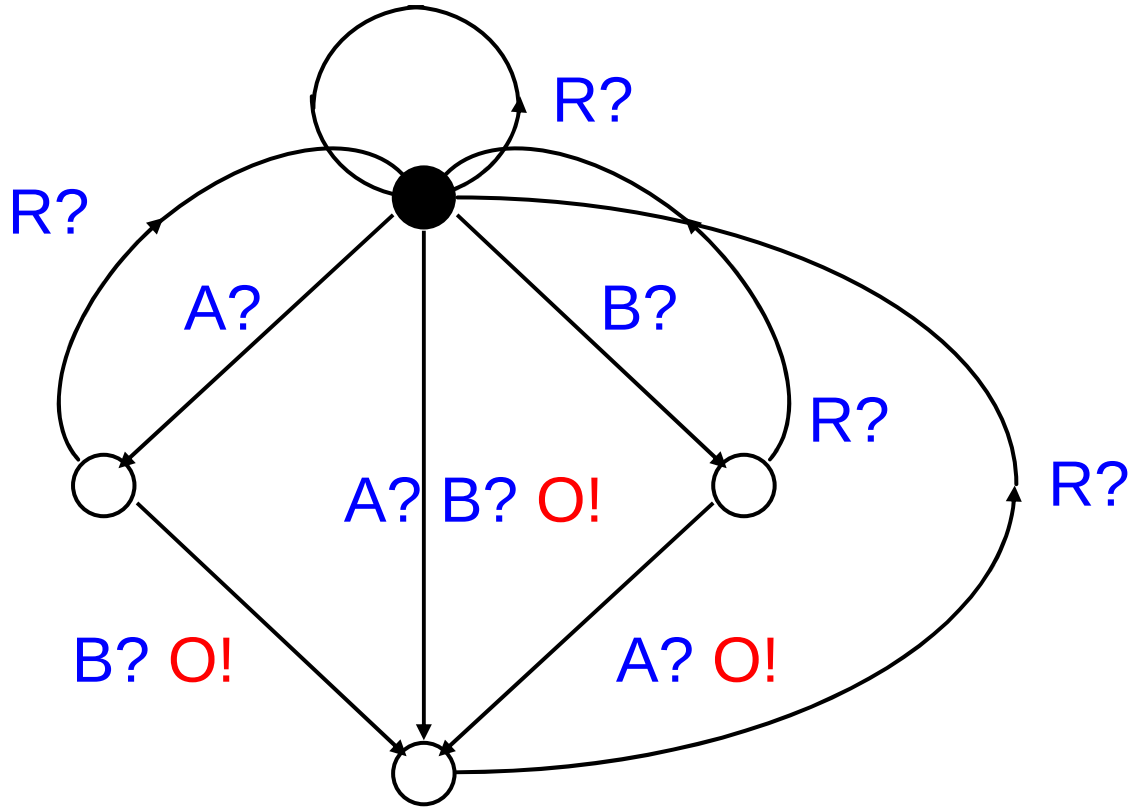


```
input X;
output T;
local Y, Z;
Y = X+1;
Z = Y*3;
T=Z/2;
```

Safe deterministic global variable sharing
No context-switching cost, makes WCET easier

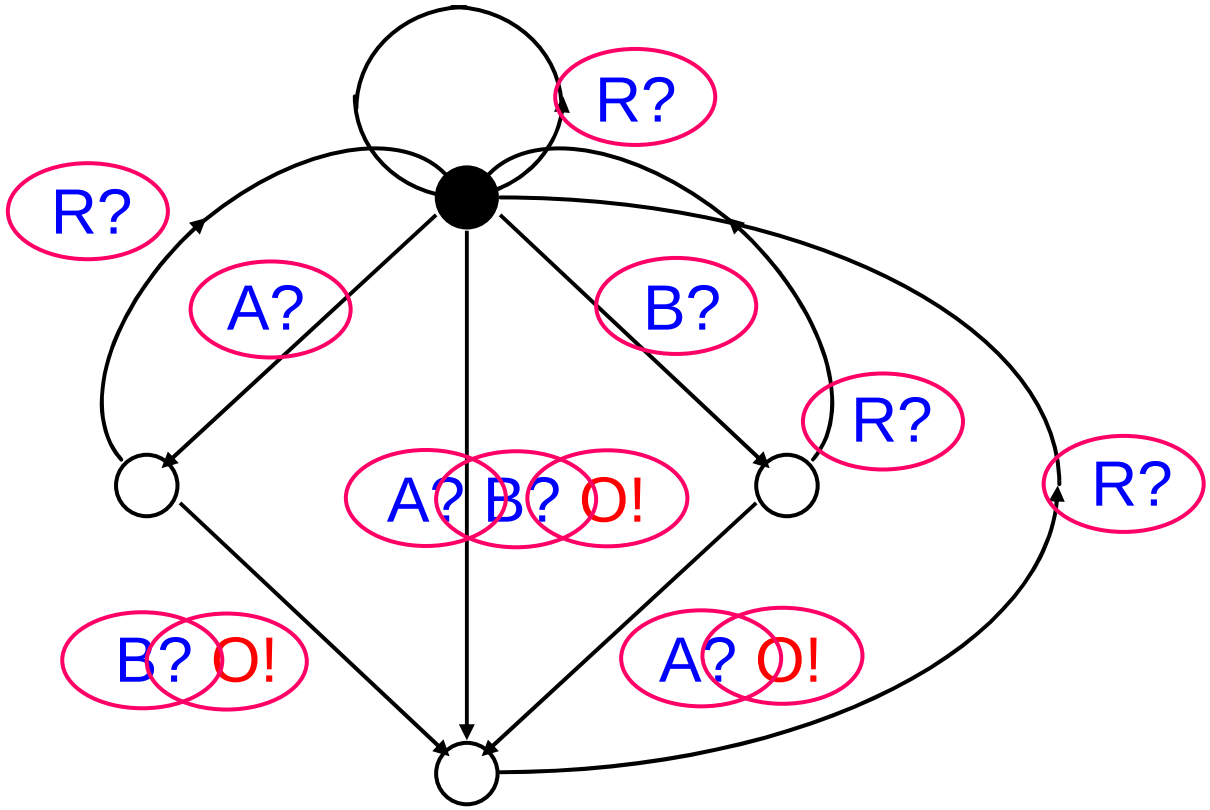
The need for behavior hierarchy

Emit O as soon has A and B have arrived.
Reset behavior each R



The need for behavior hierarchy

multiple copies => explosion

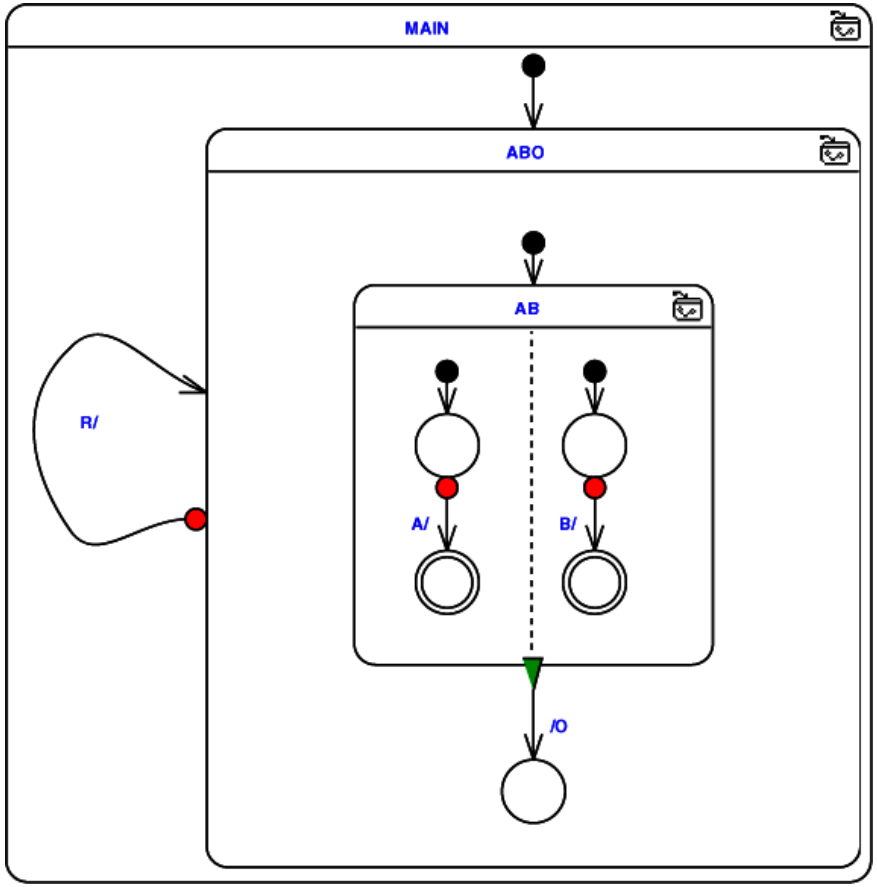


The key idea: Write Things Once

```
loop
   { await A  ||  await B } ;
   emit O
each  R
```

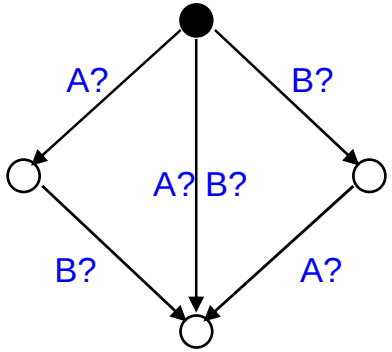
- concurrency
- sequencing
- preemption
- full orthogonality

SyncCharts (C. André)
Synchronous Hierarchical Automata

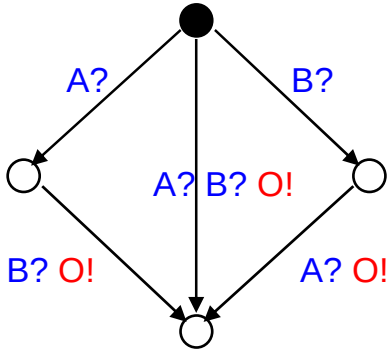


Esterel synchronous semantics

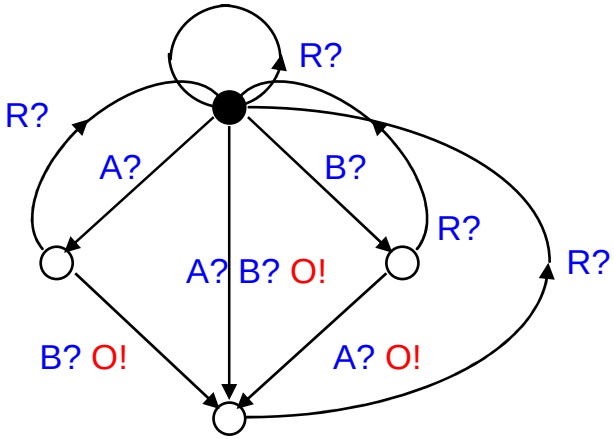
{ await A || await B }



{ await A || await B } ;
emit O



loop
{ await A || await B } ;
emit O
each R



N-way Concurrency

```
loop
   [ await A  ||  await B || await C ] ;
   emit O
each  R
```

scales linearly
vs. exponential automata blowup

A short history of synchrony

- 1982-1985 : first ideas, languages, and semantics
  - Esterel : Berry – Marmorat - Rigault, Sophia-Antipolis
  - Lustre : Caspi – Halbwachs, Grenoble
  - Signal : Benveniste – Le Guernic, Rennes

Computer Science
Control Theory

- 1985-1998 : more languages, semantics, compiling & verification
  - SyncCharts (André), Reactive C (Boussinot), TCC (Saraswat), etc.
  - causality analysis (Berry, Gonthier, Shiple)
  - links to dataflow (Ptolemy), to hardware (Vuillemin), etc.
  - formal optimization & verification techniques (Madre & Coudert, Touati)
  - Creation of SCADE (IMAG, Verilog, Airbus, Schneider)

1991: extensive BDD-based formal verif. at Dassault Aviation

- 1998 –2008 : more research, maturation

  S. Edwards, Synopsys

  R.K. Shyamasundar, TIFR, S. Ramesh, IIT Mumbai

  V. Saraswat, Xerox

  K. Schneider, Karlsruhe / KaisersLautern : Quartz project

  R. van Hanxleden, C. Traulsen, Kaiserslautern

  L. Zaffalon, EIG Geneva

- 2001-2008 : industrial expansion

  Development of Esterel v7 for hardware circuit design

  Creation of the Esterel Consortium, IEEE Standardization of Esterel v7

  Massive usage of SCADE in certified avionics embedded systems

  Growing usage of SCADE in railways and automotive industries

  Addition of SCADE Display

Esterel Studio and SCADE Suite are free for teaching activities

Agenda

- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

Design and Verification Flows

- Industrial develoment is about flows, not just tools
- Flow : full path from requirements to final object
- Methods and tools make sense only if integrated in official (non-R&D) production flows
- Verification is not a single activity but appears everywhere, and should be itself verified
- Flows cannot evolve fast

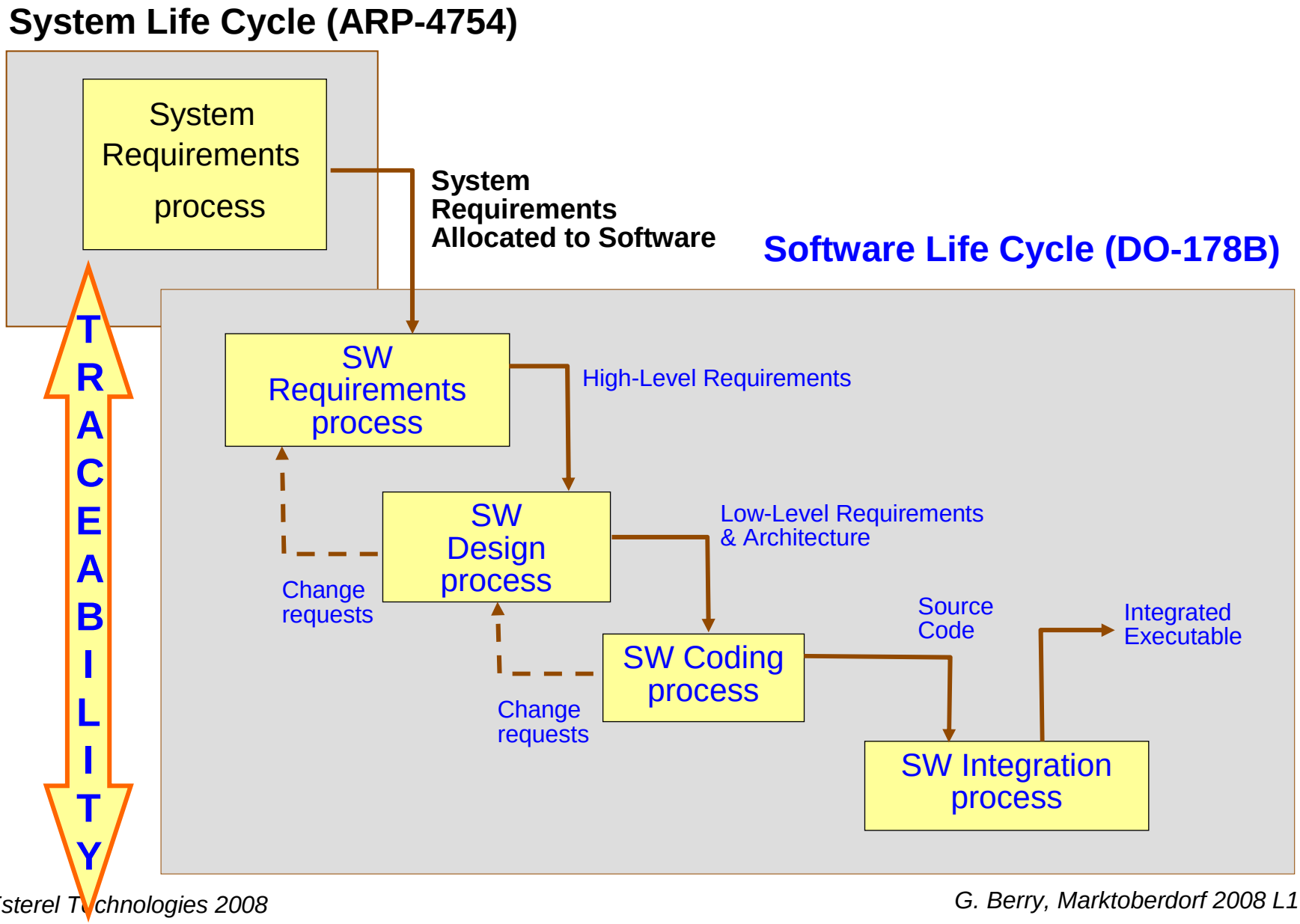
DO-178B certified avionics software flow

- Process based certification by independent authority (FAA, CEAT, JAA, etc.), use worldwide since 1992
- Goal: detect and report errors introduced during software development
- Verification objectives defined, but no specific development / verification techniques promoted
- Verification is not just testing. It contains also reviews and deep traceability-based analyses of the entire process
- Verification of verification is mandatory

Special nature of software is acknowledged

Level	Effect of anomalous behavior
A	Catastrophic failure condition (crash)
B	Hazardous/severe failure condition for the aircraft (several persons could be injured)
C	Major failure condition for the aircraft (Flight Management failure, manual management required)
D	Minor failure condition for the aircraft (Pilot/Ground communication lost)
E	No effect on aircraft operation or pilot workaround (entertainment features down)

DO-178B Development Process



Verification of Verification

- Show that the tests cover the High-level Requirements (HLR)
- Show that the tests cover the Low-Level Requirements (LLR)
- Show the source code structure that have been exercised during testing
  - Level C: 100% Statement Coverage
  - Level B: 100% Decision Coverage
  - Level A: 100% Modified Condition / Decision Coverage

Verification of Verification Objectives

	Table A-7	Objective
	1	Test procedure are correct
	2	Test result are correct and discrepancies explained
	3	Test coverage of high-level requirements is achieved
	4	Test coverage of low-level requirements is achieved
	5	Test coverage of software structure (MC/DC) is achieved
Level A	6	**Test coverage of software structure (decision coverage) is achieved**
Level B	7	**Test coverage of software structure (statement coverage) is achieved**
Level C	8	Test coverage of software structure (data coupling and control coupling) is achieved

Towards DO-178C

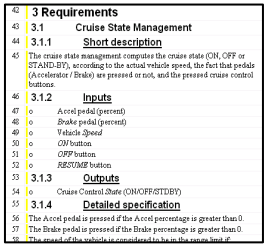
- 2005-2009 : Working Group, 120 people, 1000 on Web site
- From process-based to product-based
  - tool qualification
  - model-based development
  - OO design
  - automatic code generation
  - formal verification
- Full consensus needed to publish the document

Other Standards

- DO-254: avionics hardware development
- IEC 61508: function safety of systems made with Electrical, Electronic, Programmable electronic components
- EN 50128: Adaptation of IEC 61508 to Railways
- MIL-STD-498: Military standard for SW development
- DEF-STD-055/056: Safety management for Defense Systems
- Chinese Standards

The SCADE™ Certified Software Factory

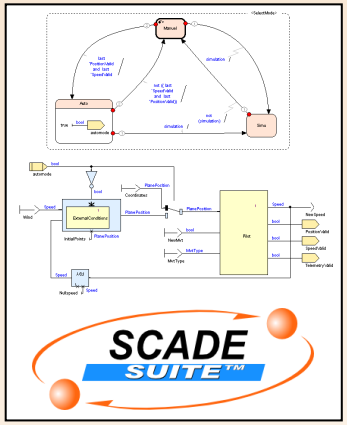
SYSTEM SPEC

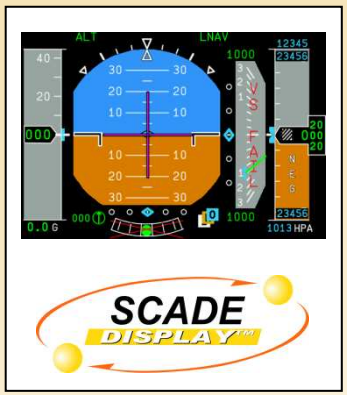


Algorithm Design Capture

Architecture Design Capture

DESIGN





VERIFY

Debugging & Simulation

Formal Verification

Model Coverage Analysis

Object Code Verification

SCADE Suite/SCADE Display Integration

Graphical Animation

Ergonomics Checking

GENERATE

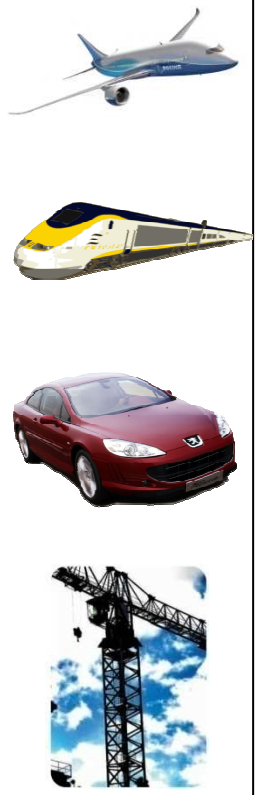
SCADE Suite KCG

RTOS Wrappers

SCADE Display KCG

OpenGL SC Compliant

SYSTEM TEST



MANAGE & TRACE

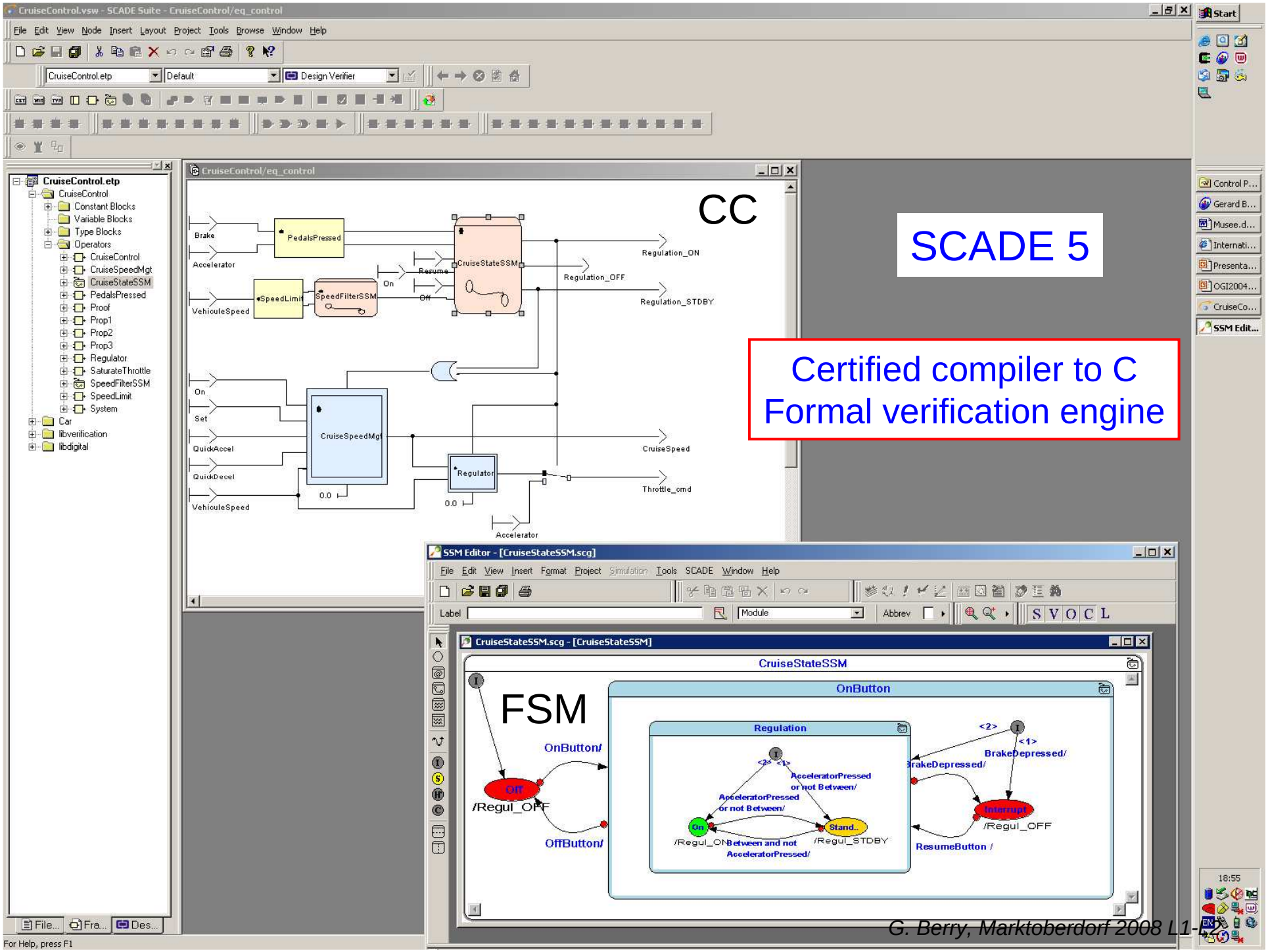
Requirements Management Gateway

Integrated Configuration Management

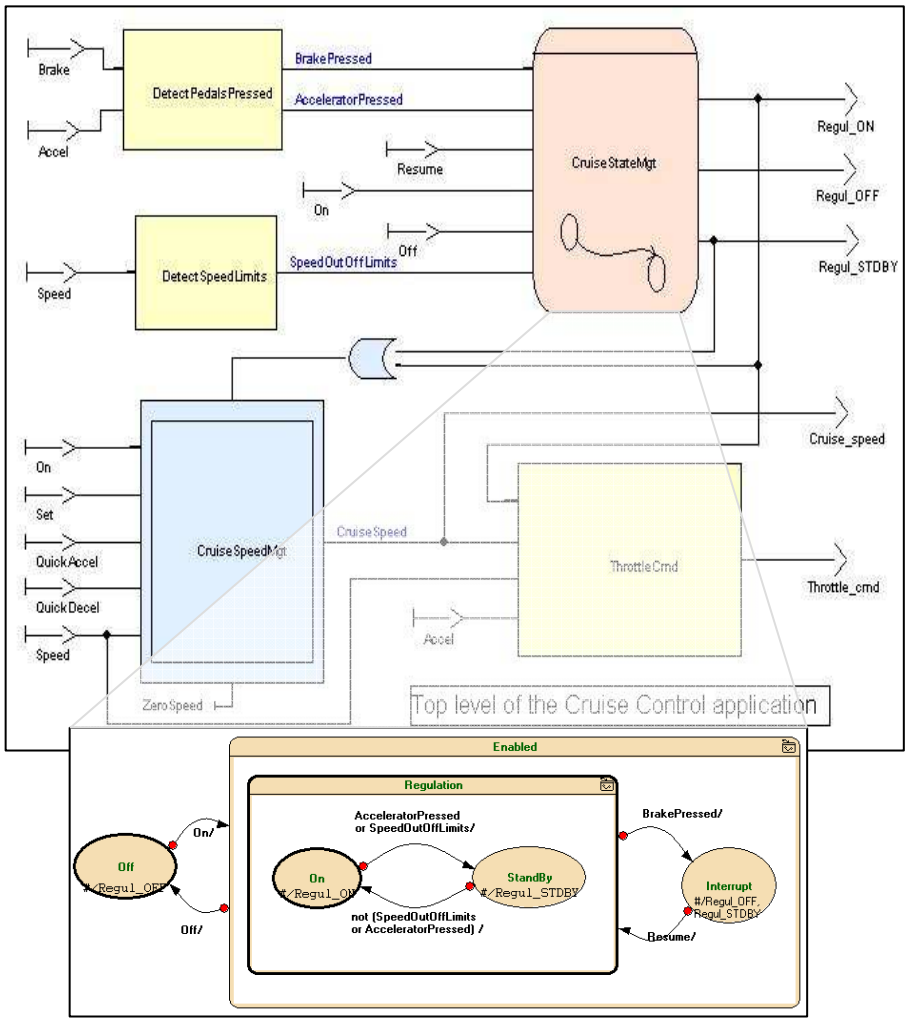
Automatic Design Documentation

DO-178B
IEC 61508
EN 50128

Qualification Kits, Certificates & Handbooks

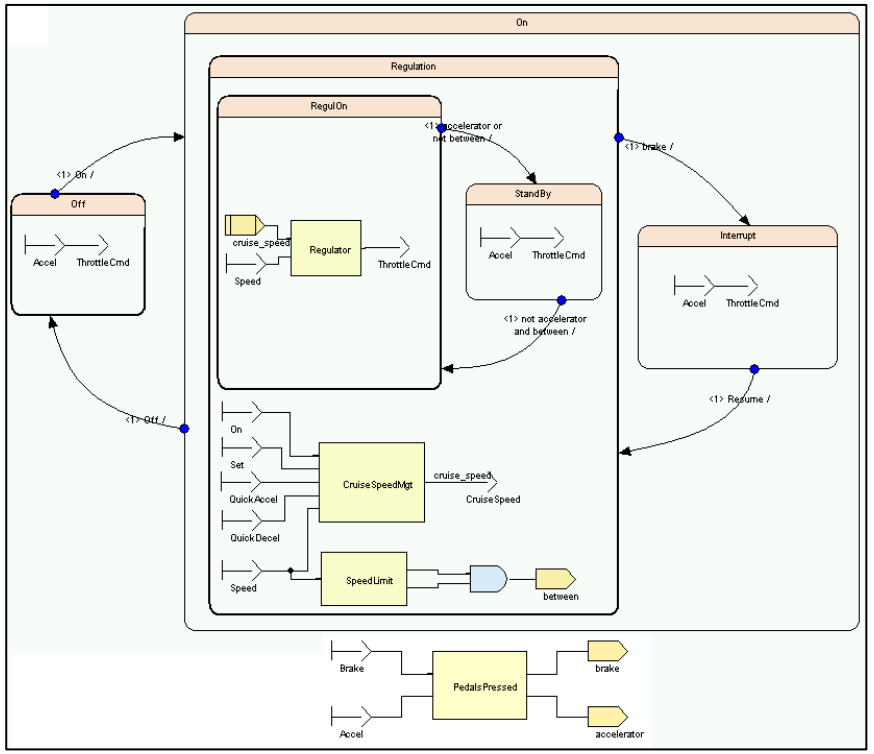

CruiseControl.vsw - SCADE Suite - CruiseControl/eq_control
CruiseControl/eq_control
CC
SCADE 5
Certified compiler to C
Formal verification engine
SSM Editor - [CruiseStateSSM.scg]
CruiseStateSSM.scg - [CruiseStateSSM]
CruiseStateSSM
FSM
OnButton
Regulation
Off
/Regul_OFF
OnButton/
OffButton/
On
/Regul_ON
Stand..
/Regul_STDBY
Interrupt
/Regul_OFF
BrakeDepressed/
ResumeButton /
AcceleratorPressed
or not Between/
Between and not
AcceleratorPressed/
G. Berry, Marktoberdorf 2008 L1-22

Scade 6 : Data - Control Flow Unification



SCADE 5

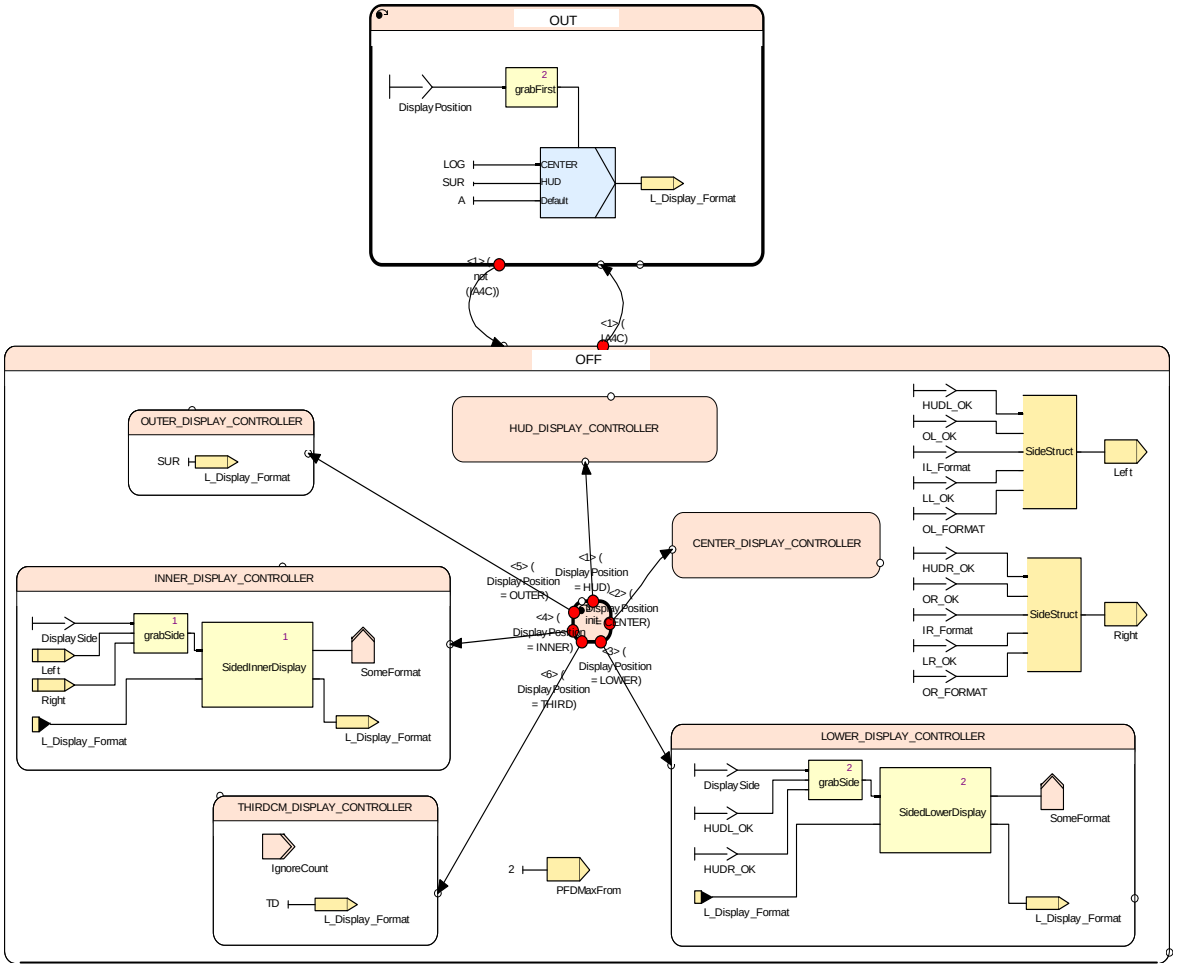
Pure Control Flow into
Pure Data Flow



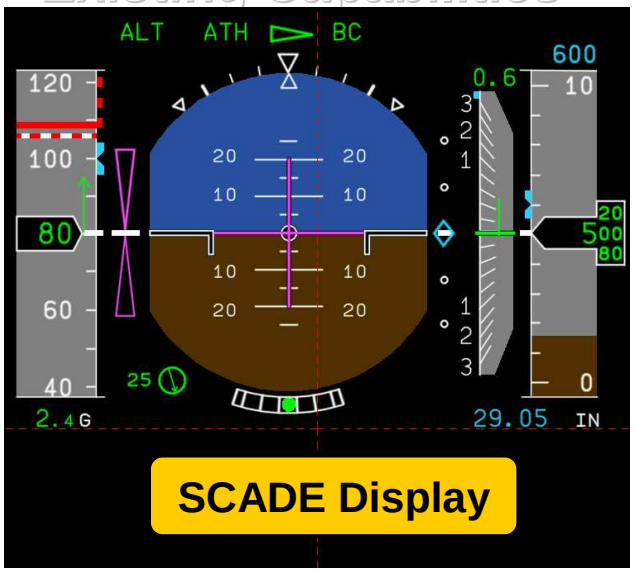
SCADE 6

Unified Data Flow & Control Flow
Freely mixable in hierarchy

A Typical Application: Cockpit Display



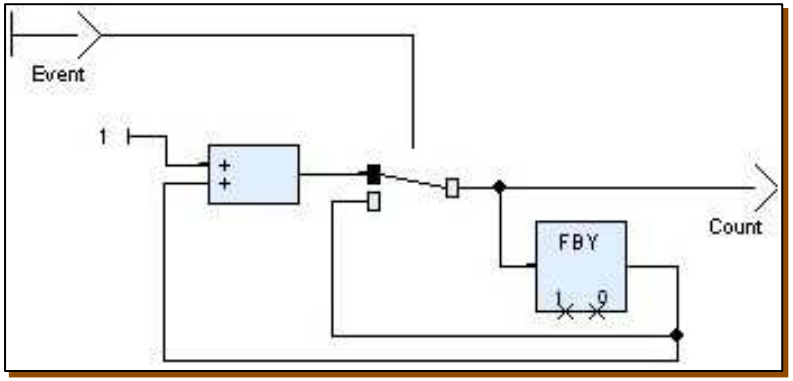
Existing Capabilities



Lustre = Synchronous Kahn Networks

A simple counter

$$\begin{cases} Count(0) = 0 \\ \forall t > 0,\ Count(t) = \begin{cases} Count(t-1)+1, & if\ \ Event(t) = true \\ Count(t-1), & otherwise \end{cases} \end{cases}$$




```
Count = 0 ->
    (if Event
     then pre(Count)+1
     else pre(Count))
```

The **`Count`** flow is the solution of the equation

Code Generation with KCG

- KCG is the qualifiable C code generator
  - developed with a DO-178B Level A process
  - certification authorities certified that “KCG can fly” and qualified it as a development tool

  => no need to unit-test the generated C code
- Evidences provided to users
  - qualification kit
  - verifiable, traceable, and safe code
  - C compiler verification kit

Synchronous Semantics

- Ensures every data is produced exactly once
- Additional static checks
  - no access to undefined data
  - no race condition (combinational cycle)

    => deterministic scheduling-independent result
  - no recursion in node calls

    => static memory allocation, bounded stack

Checked by the qualified code generator as a prerequisite to code generation

Generated Code Properties

- Small C subset
- Portable (compiler, target and OS independent)
- Structured (by function or by blocks)
- Readable, traceable (names/annotations propagation)
- Safe static memory allocation
- No pointer arithmetic
- No recursion, no loop
- Bounded execution time
- Size and / or speed optimizations

Eases verification and static analysis (Astrée, AbsInt)

Agenda

- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

System-on-Chips Flows

- Entirely in-house
- Long chain of individually hard flow components
  - informal documentation (English)
  - manual coding at RTL level (VHDL, Verilog)
  - semi-automatic design for testability (DFT) additions
  - automatic logic synthesis
  - automatic place and route
  - mask fabrication
  - final chip on-line testing
- A hard milestone : RTL sign-off
  - after that, mask patches needed, 100,000$ +

Architecture — components, dimensioning, communication — Word, Excel, Visio, System C

↓

Micro-Architecture — concurrency, pipeline, resource sharing — Word, Visio, C

↓

RTL design — gates, clocks, registers, RAMs, critical path — VHDL, Verilog

↓

circuits — cells, clock trees, area, speed — netlists

↓

DFT (test) — testability, scan insertion — netlists

↓

Place&Route — physical & electrical constraints — P&R netlists

↓

$ 1,000,000 Masks — printing — pseudo-rectangles

↓

Chips — fabrication — silicon dies

Architecture

functionality OK?
throughput OK?
marketing OK?

Experience
Reviews

↓

Micro-Architecture

breakdown OK?
performance OK?

C-based modeling

↓

RTL design

functionality OK?
area/speed OK?
power OK?

Random-directed testing,
Formal verification

↓

circuits

equivalent to RTL?

formal equivalence checking

↓

DFT (test)

test coverage
~100% ?

ATPG

↓

Place&Route

connections?
electrical constraints?
timing?

Design Rules Checking (DRC)

↓

$ 1,000,000 Masks

No fab fault?

Scan test run

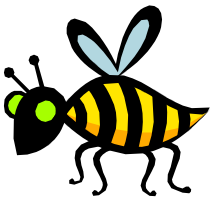
↓

Chips

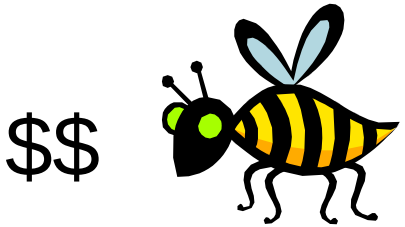
Architecture

Micro-Architecture

RTL design



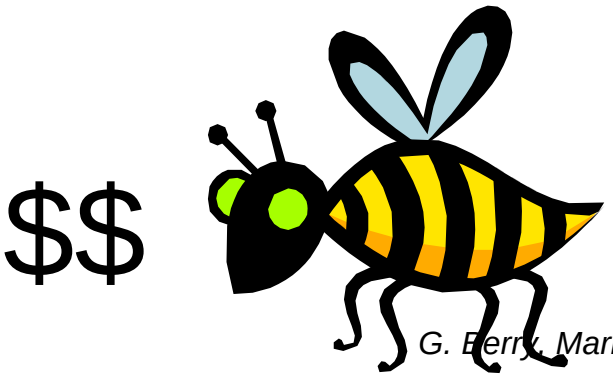
circuits



DFT (test)

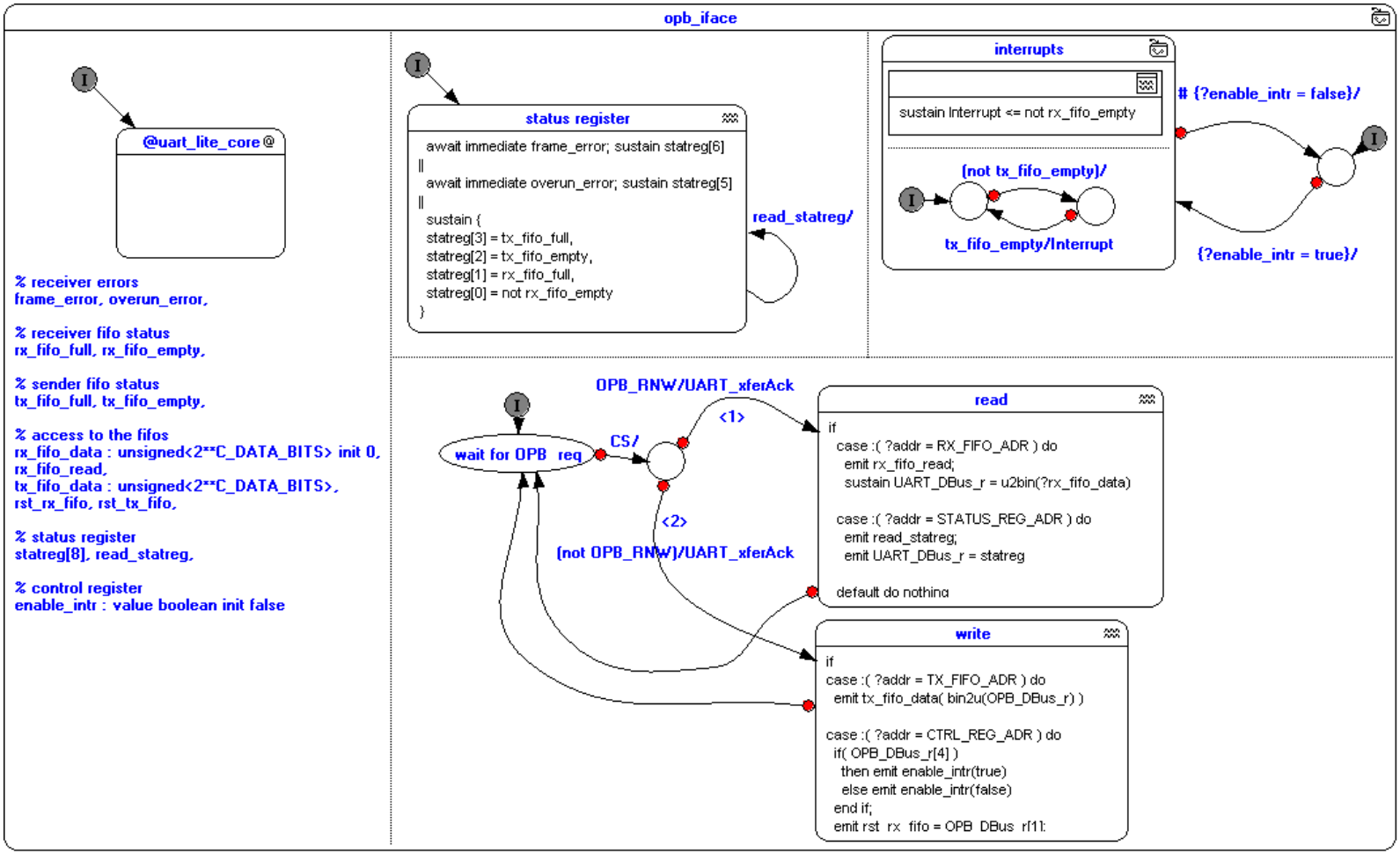
Place&Route

$ 1,000,000 Masks



Chips

Esterel v7 (Berry – Kishinevsky)



text + graphics, concurrency + sequencing
clear semantics

Esterel

Architecture

Micro-Architecture

RTL design

circuits

DFT (test)

Place&Route

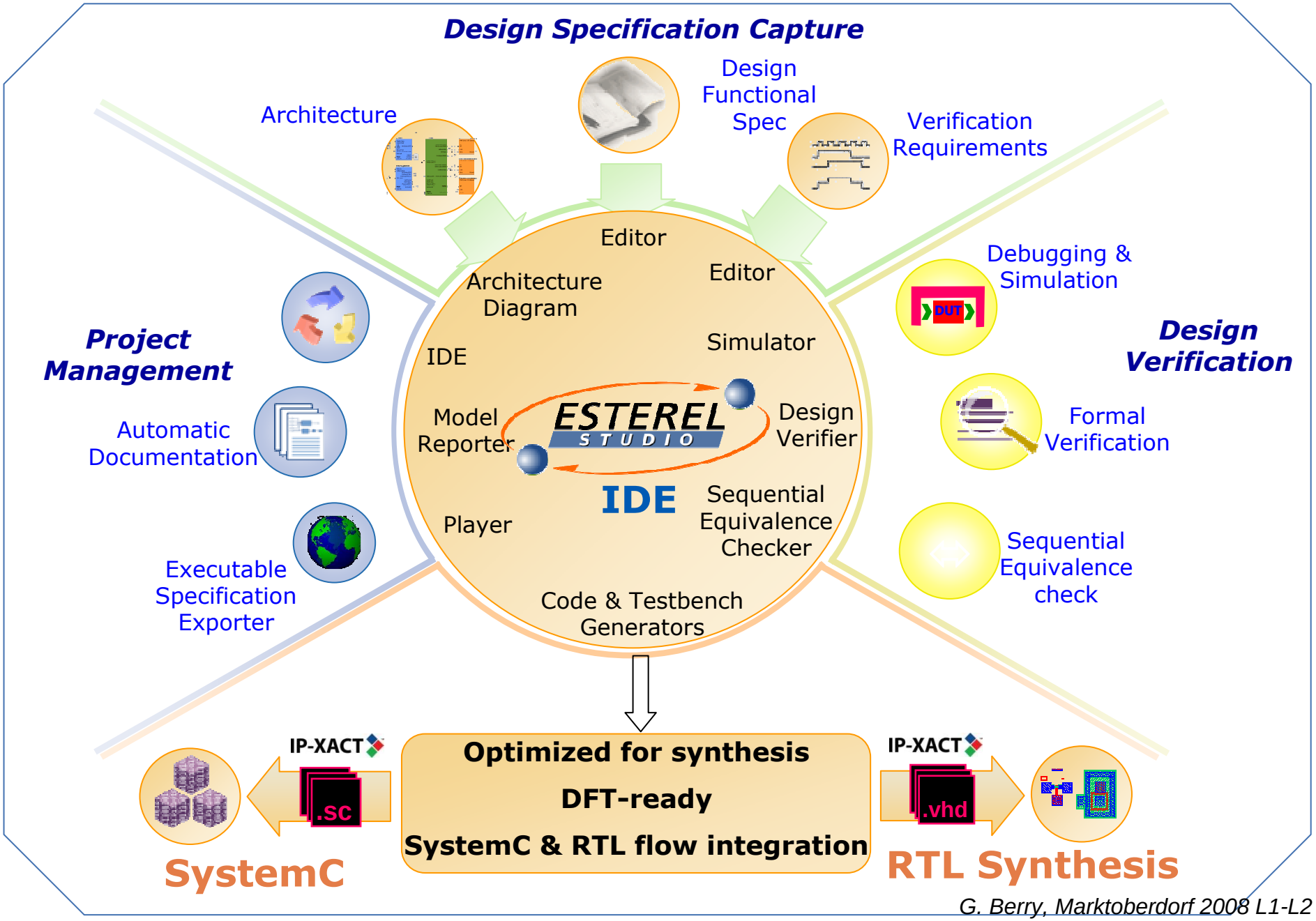
$ 1,000,000 Masks

Chips

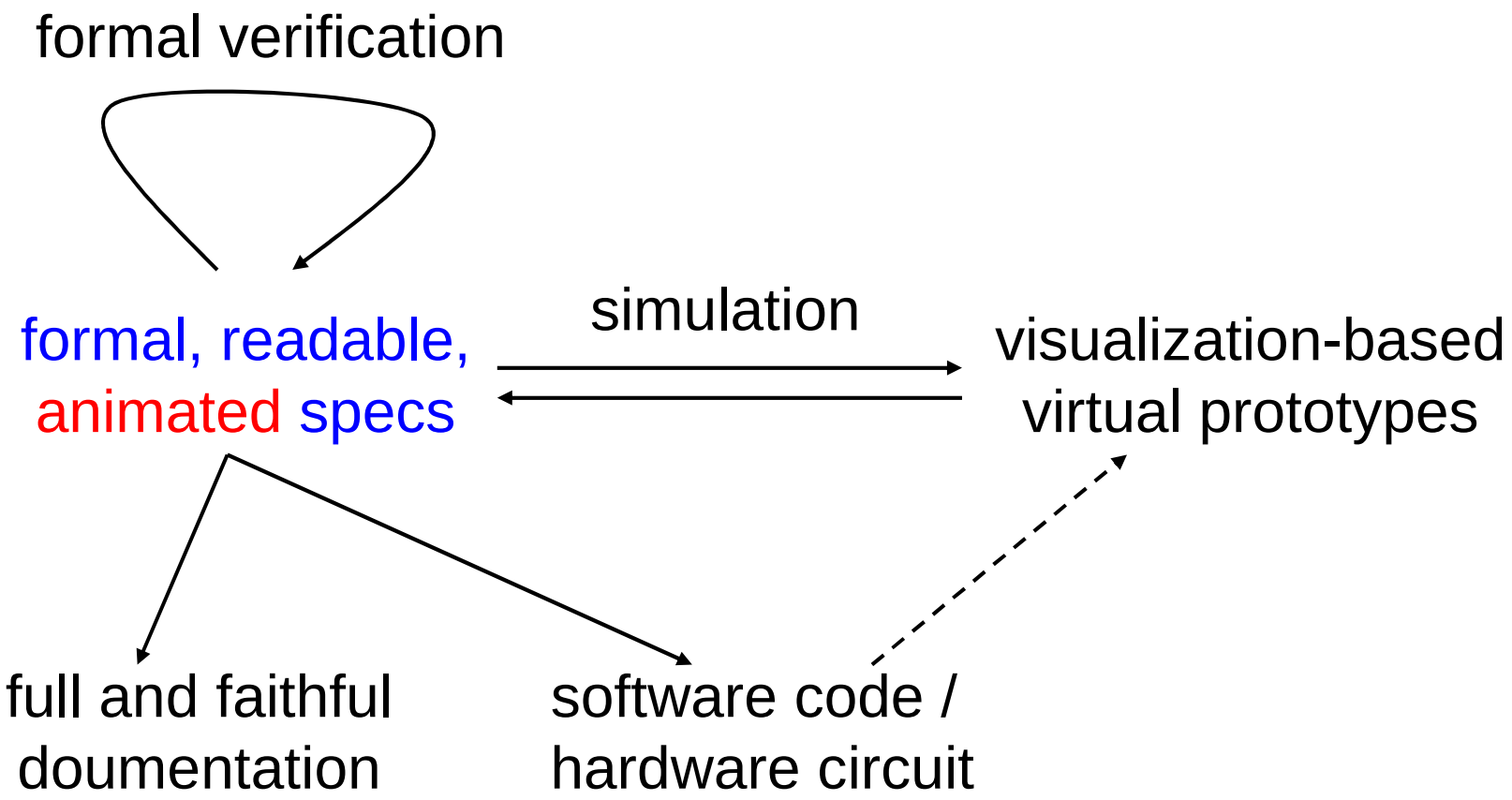
Key Messages to Users

1. Specification of dynamics cannot be accurate when written on static paper
2. Animated executable specifications key to reuse, inter-teams communication, what-if studies, etc
3. Once such specs are available, why recoding?
   - single model for HW synthesis and SW modeling (SystemC)
4. Spec-to-implementation path : formal methods and tools
   - hierarchical behavior description
   - languages with formal semantics
   - formal compiling algorithms
   - formal verification techniques
5. Formal verification= design tool usable at all design steps

Esterel Studio at a glance



The Esterel Studio Usage Model



Agenda

- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

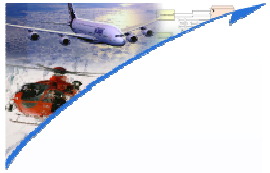
Agenda

- About Esterel Technologies
- Beware of the Computer!
- Design and Verification Flows for Embedded SW
- Design and Verification Flows for SoCs
- The Synchronous Approach to D&V
- Overview of SCADE
- Overview of Esterel Studio

Conclusion

- Synchronous formal methods are heavily used in industry
  - formal languages
  - formal compilation schemes
  - formal verification
- They make verification much simpler
  - Source language matters, hierarchy is key
- Current research & development
  - Improving the languages (SCADE 6, IEEE-standard Esterel)
  - Improving the compilers (faster and more modular output)
  - Scaling up formal verification : how big can you verify?

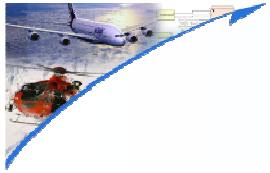
Get Esterel Studio and SCADE free
for teaching and academic usage



The SCADE 6 Language and the KCG certifiable code generator

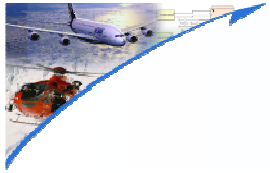
Gérard Berry

Chief Scientist
Esterel Technologies



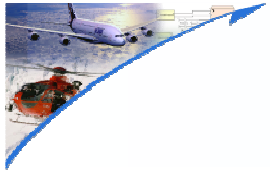
Agenda

- Scade 6 Design Goals
- The design of SCADE 6
- The KCG certifiable code generator
- Conclusion



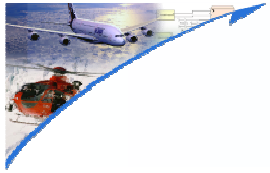
Agenda

- Scade 6 Design Goals
- The design of SCADE 6
- The KCG certifiable code generator
- Conclusion



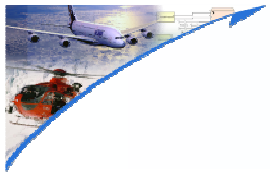
SCADE 6 Design Goals

- Unify control and data
  - full SSM / data flow mix at any hierarchical level
  - in mode automata style: different data behaviors in different states
  - but with controlled expressive power for maximal safety
- Provide functional arrays
  - simple and safe semantics, efficient implementation
- Allow polymorphic programming (generic nodes)
  - when behavior is independent of actual data contents
- Benefit from research on Esterel v7 and Synchronous Lucid
- And keep all good properties of Lustre / SCADE 5
  - all constructs should compile well
  - clock-calculus based semantics, substitution principle
  - KCG certifiable code generator

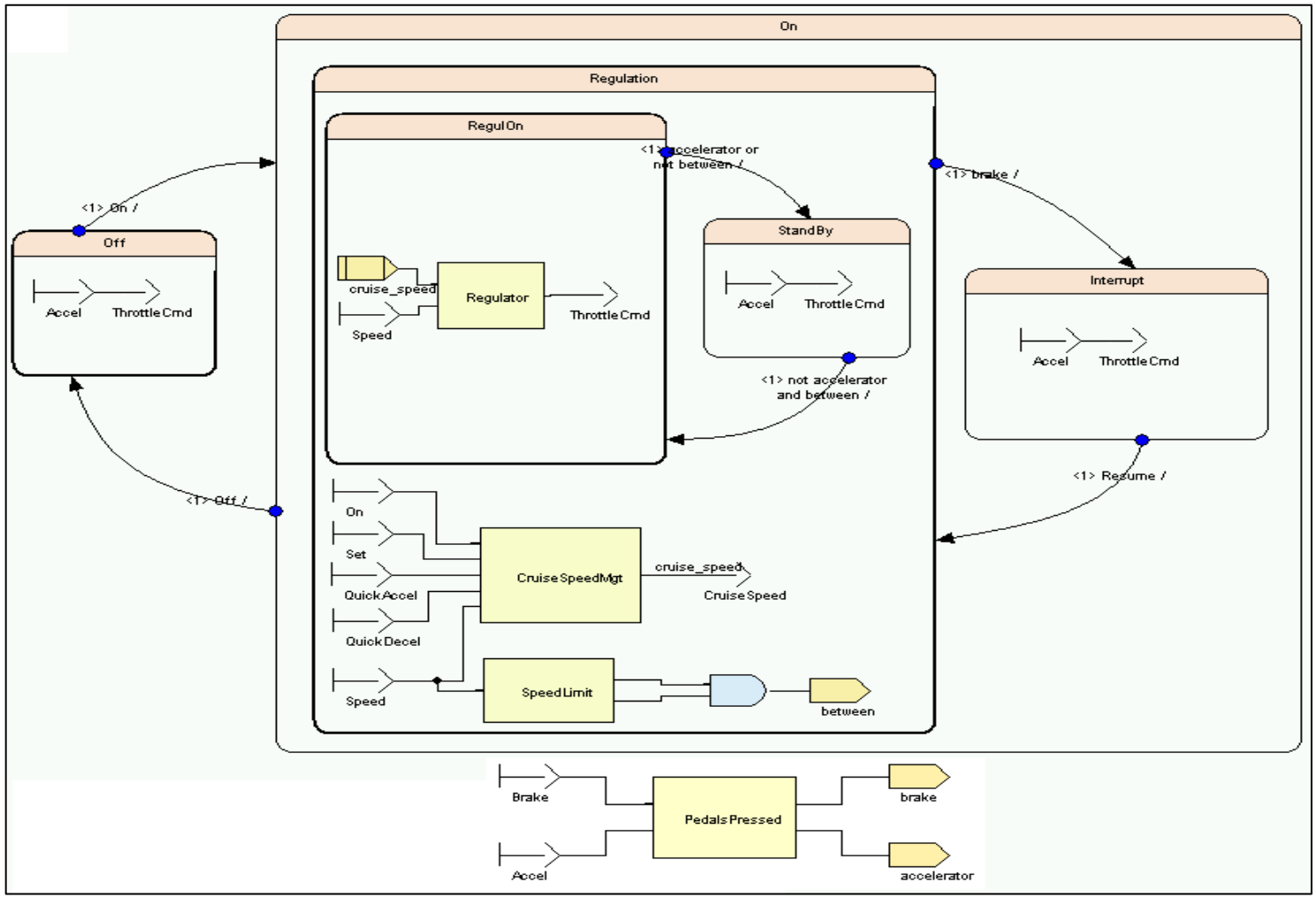


Agenda

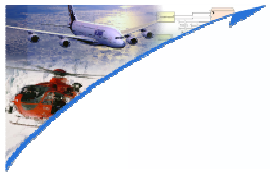
- Scade 6 Design Goals
- The design of SCADE 6
- The KCG certifiable code generator
- Conclusion



SCADE 6 Look&Feel



semantics defined by Scade 6 textual language



Scade 6 Types

- Predefined types:

```
bool, int, real, char
```

- User-defined types
  - Enumerations

```
type COLORS = enum {RED, GREEN, BLUE};
```

  - Structures

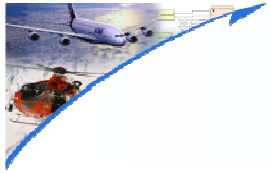
```
type Sensor = {valid: bool; value: real;};
```

  - Arrays

```
type RealArray = real^5;
type ElevatorButtons = int^(FLOORS);
```

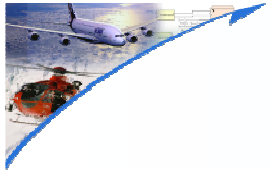
- Host language imported types

```
type imported CanMessage;
```



A flow is an infinite sequence of values

Cycle	1	2	3	4	5
Cond	false	true	true	false	true
not(Cond)	true	false	false	true	false
3.14	3.14	3.14	3.14	3.14	3.14
0 -> 5	0	5	5	5	5
I	14	13	11	12	16
pre(I)	*nil*	14	13	11	12
fby(I; 1; 0)	0	14	13	11	12

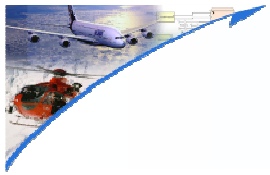


Scade 6 Equations

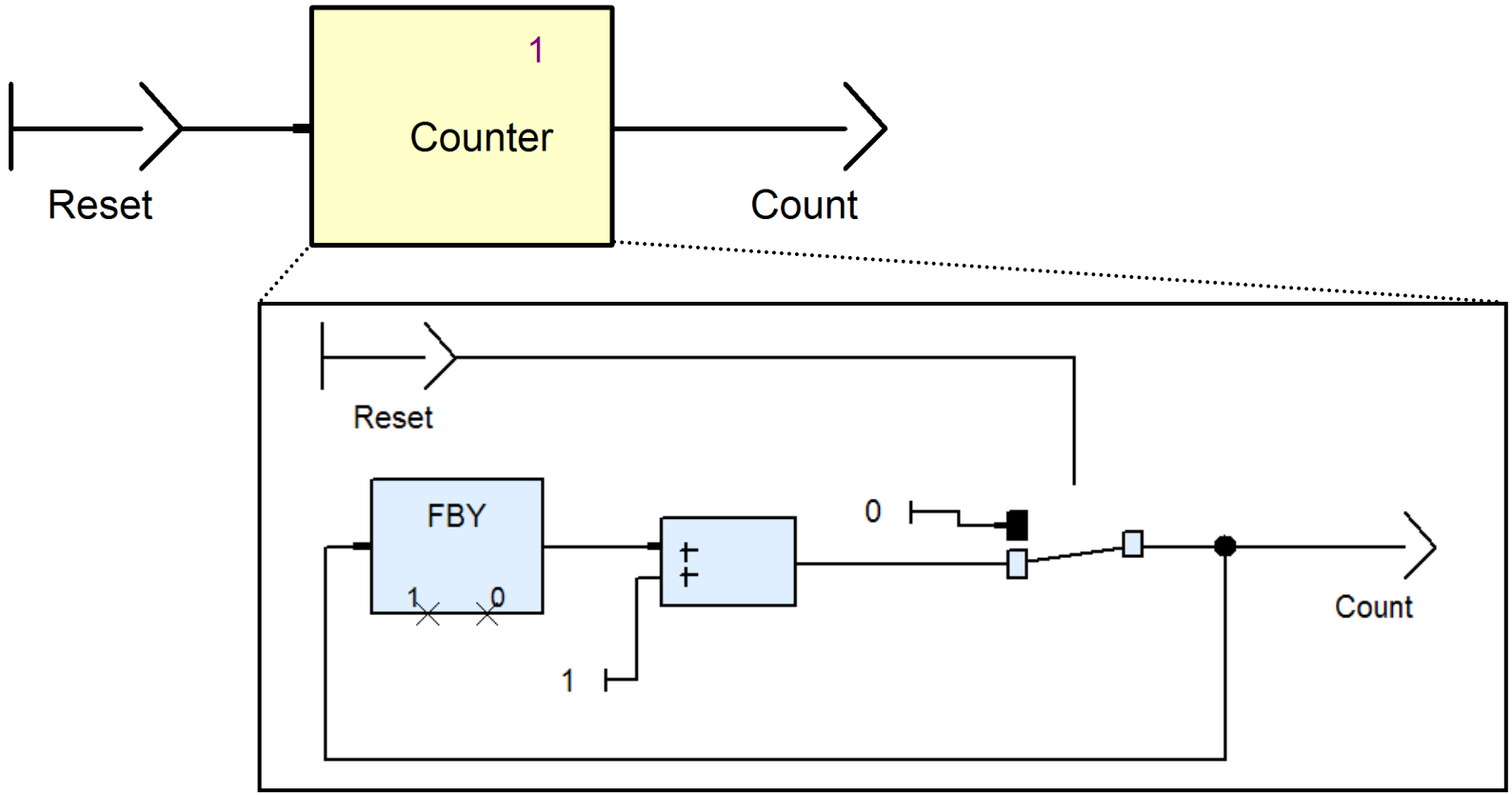
Equations are equalities that define flows as their solutions

```
nat: int;
nat = 1 -> (pre(nat) + 1);
```

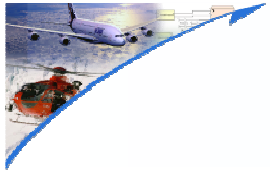
Cycle	1	2	3	4	5
1	1	1	1	1	1
pre(nat)	*nil*	1	2	3	4
1 + pre(nat)	*nil*	2	3	4	5
1 -> (pre(nat) + 1);	1	2	3	4	5



A Simple Counter

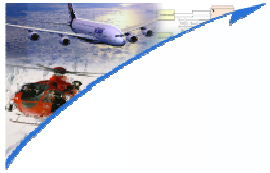



```
node Counter (Reset: bool) returns (Count: int)
Count = 0 -> if Reset then 0 else 1 + fby(Count,1,0)
```



SCADE Clocks

- SCADE defines clocks over flows and operators
- Clocks allow flows to be sampled and sub-systems to run at different rates
  - Clocks define sampling rates
  - Each flow has a clock, which defines when new values are available
  - An operator is executed when all its inputs are available; this defines the clock of its result
- The language primitives operators **`when`** and **`merge`** introduce clocks
- Clocks are mostly used for semantics. In practice, activation structures are more convenient



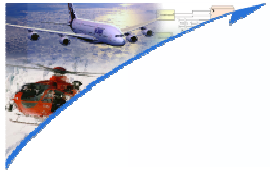
The when Operator

- Sampling of a flow

C	true	false	true	false	false	true
X	x1	x2	x3	x4	x5	x6
X when C	x1		x3			x6

- Combining sampled flows

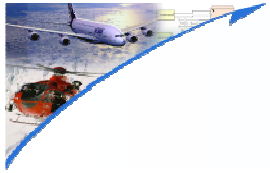
C	true	false	true	false	false	true
X	x1	x2	x3	x4	x5	y6
Y	y1	y2	y3	y4	y5	y6
Z=X when C	x1		x3			x6
T=Y when C	y1		y3			y6
Z + T	x1+y1		x3+y3			x6+y6
pre(Z+T)	*nil*		x1+y1			x3+y3



The current Operator

- Extend sampled flows

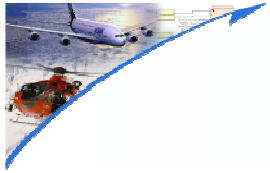
C1	true	false	true	false	false	true
C2	false	false	true	false	true	true
X	x1	x2	x3	x4	x5	y6
Z=X when C1	x1		x3			x6
current(Z)	x1	x1	x3	x3	x3	x6
T=X when C2			x3		x5	x6
current(T)			x3	x3	x5	x6



The merge Operator

- Merge flows on complementary clocks

c	true	false	true	false	false	true
X	x1	x2	x3	x4	x5	y6
Y	y1	y2	y3	y4	y5	y6
Z=X when C	x1		x3			x6
T=Y when not C		y2		y4	y5	
merge(c; Z; T)	x1	y2	x3	y4	y5	x6

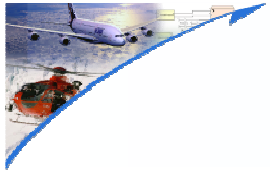


Activation Structures

- Define when an operator or an equation gets executed
  - Specify activation domains

    combinational = if-then-else, switch-case

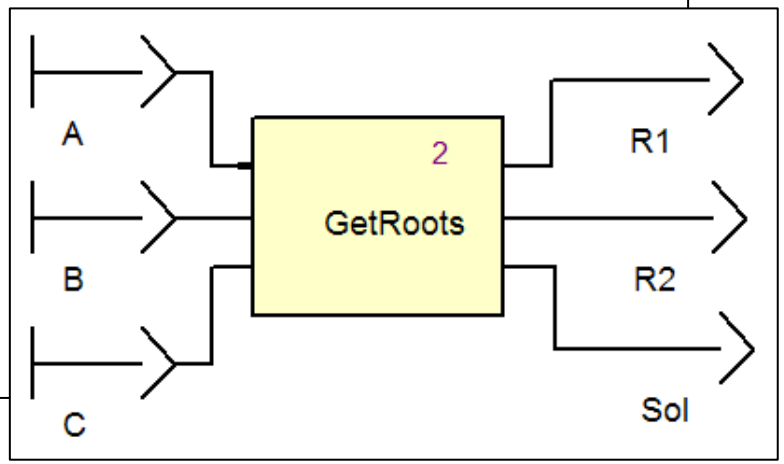
    SSMs = hierarchical synchronous state machines
- Ensure that each flow has a unique definition at each cycle
  - Note : Esterel v7 does not impose this
  - But we consider it fundamental for SCADE 6 program simplicity and clarity



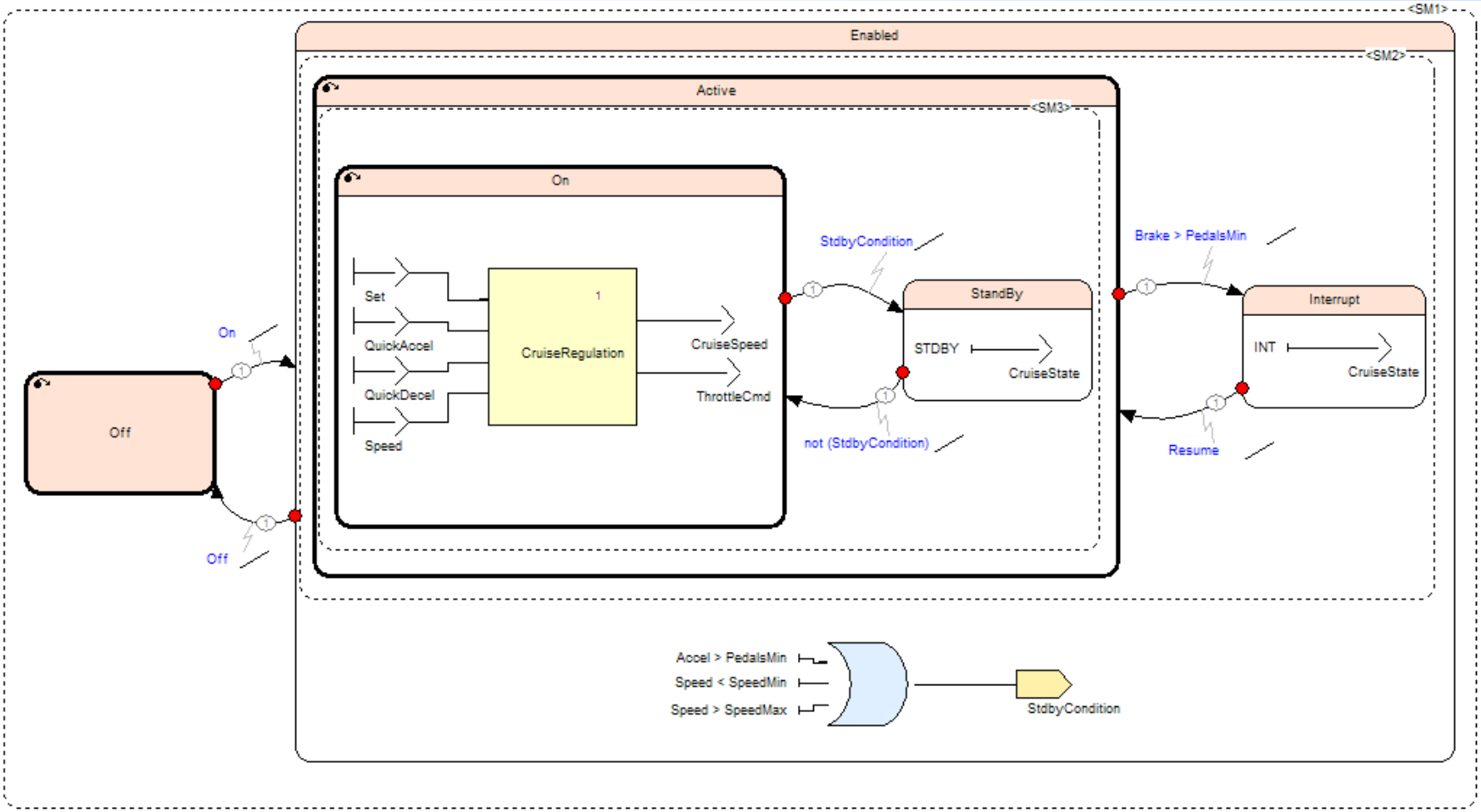
Combinational Activation

```
node GetRoots (A: real; B: real; C: real)
returns (r1: real; r2: real; sol: bool default true)
  delta = B*B – 4*A*C;
  activate if (delta > 0.0) then
    r1, r2 = GetDeltaPosRoots(A, B, delta);
  else if (delta = 0.0) then
    var r: real;
    let
      r = GetDeltaNulRoot(A, B, delta);
      r1 = r; r2 = r;
    tel
  else
    r1 = 0.0; r2 = 0.0;
    sol = false;
  returns r1, r2, sol;
```

default definition if no equation



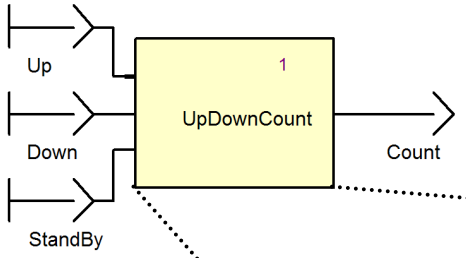
Cruise Control Combinational Defaults



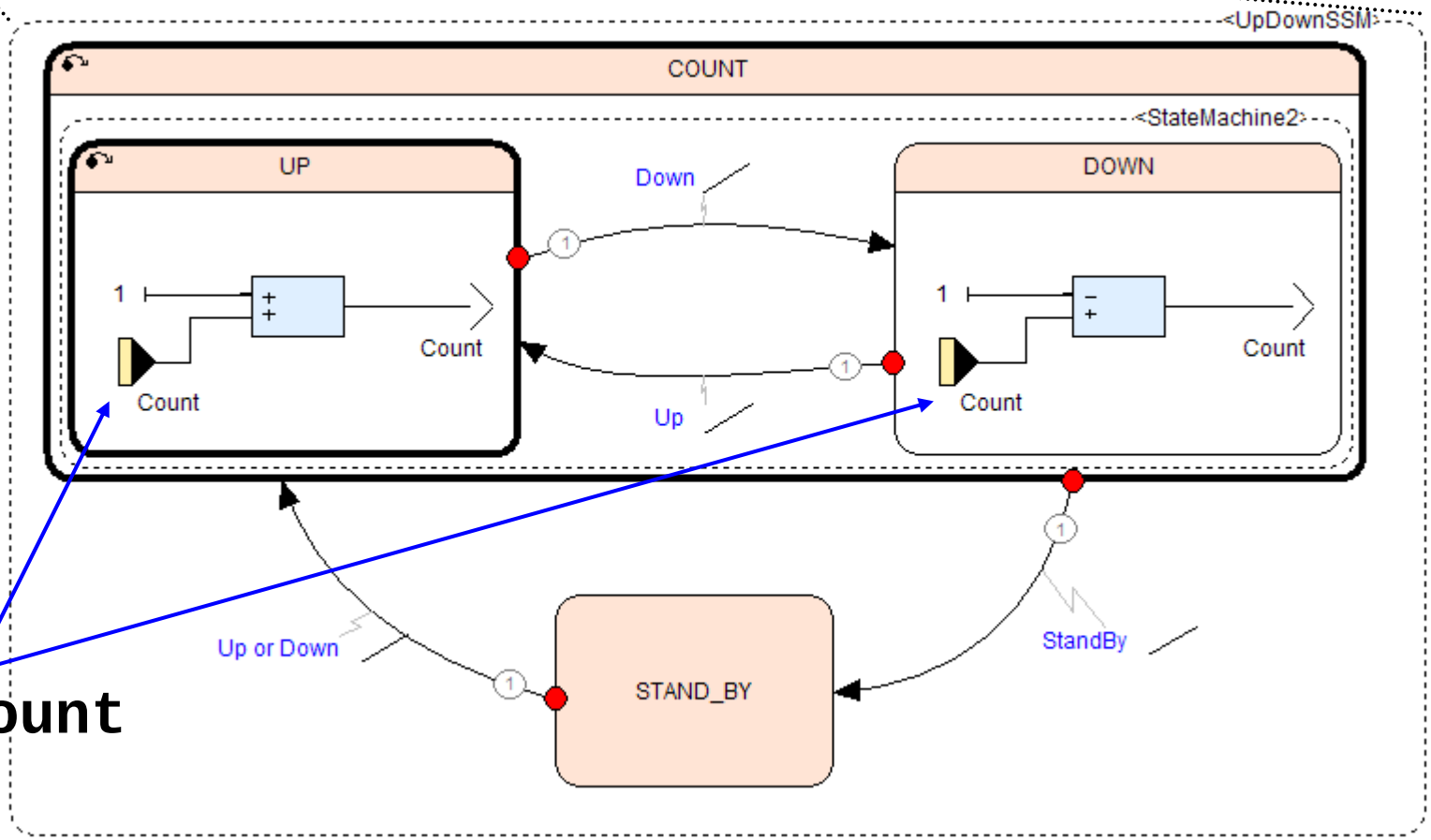
Default values for CruiseRegulation outputs :

```
ThrottleCmd: real default Accel; -- Accel: real (input)
CruiseState: enum {OFF, ON, STDBY, INT} default OFF;
```

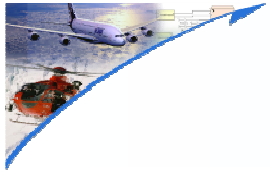
State Machine Activation




```
Count: int last = 0;
```



`last Count`



The last Operator

- When **Up** is active

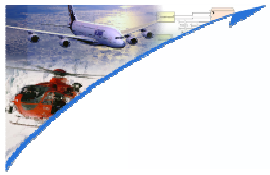
```
Count = last 'Count + 1;
```

- When **Down** is active

```
Count = last 'Count – 1;
```

- **last 'Count** gets the last value global to all the activation domains in the scope of **Count**
- When **STAND_BY** is active, **Count** keeps its previous value because of its declaration

```
Count: int last = 0;
```



Default vs. Last Values

- Combinational activation:

```
sol: bool default true
```

default value used at each cycle where **`sol`** has no equation

- SSM activation

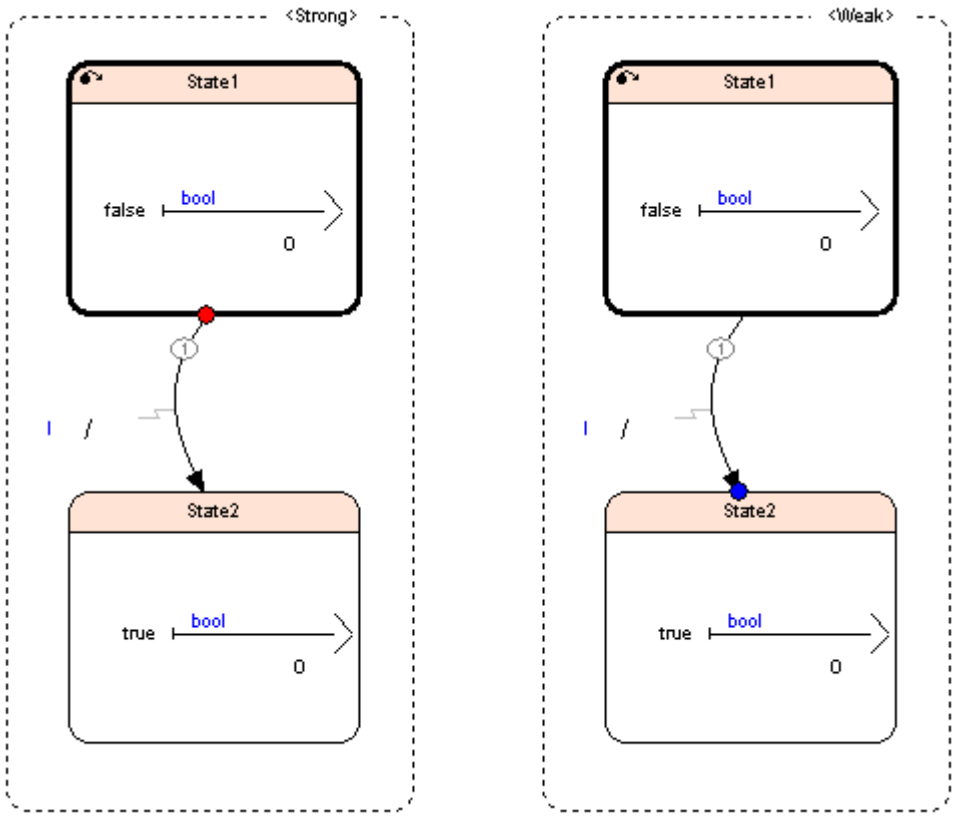
```
Count: int last = 0
```

`Count` initialized to 0

last value persists when **`Count`** has no equation

- Similar notions in Esterel v7: temporary vs. memorized

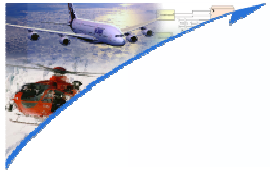
Strong vs. Weak-Delayed Preemption



In both cases, only one state active at a time

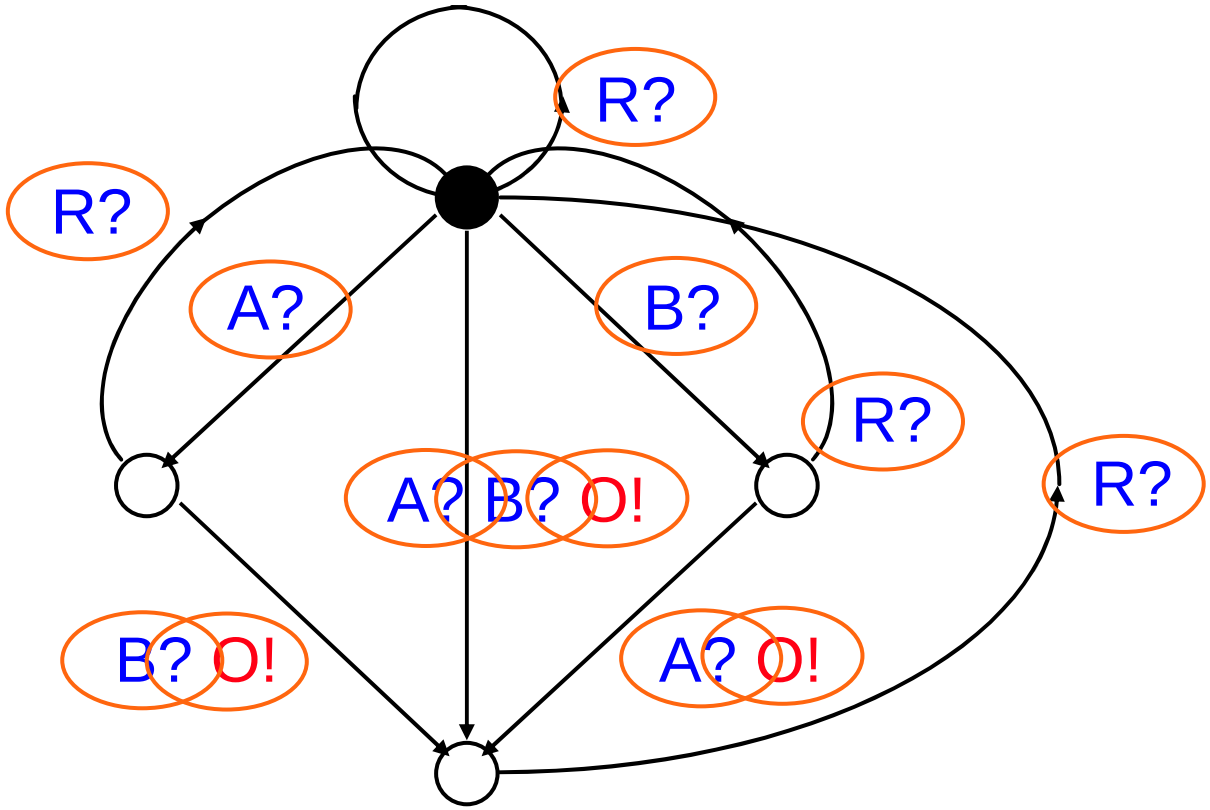
When both exist, strong has priority over weak

I	false	true	-
Strong	O = false	O = true	O = true
Weak delayed	O = false	O = false	O = true



ABRO - a very common control pattern

- Wait for A and B; when both there, output O
- Reset behavior each R



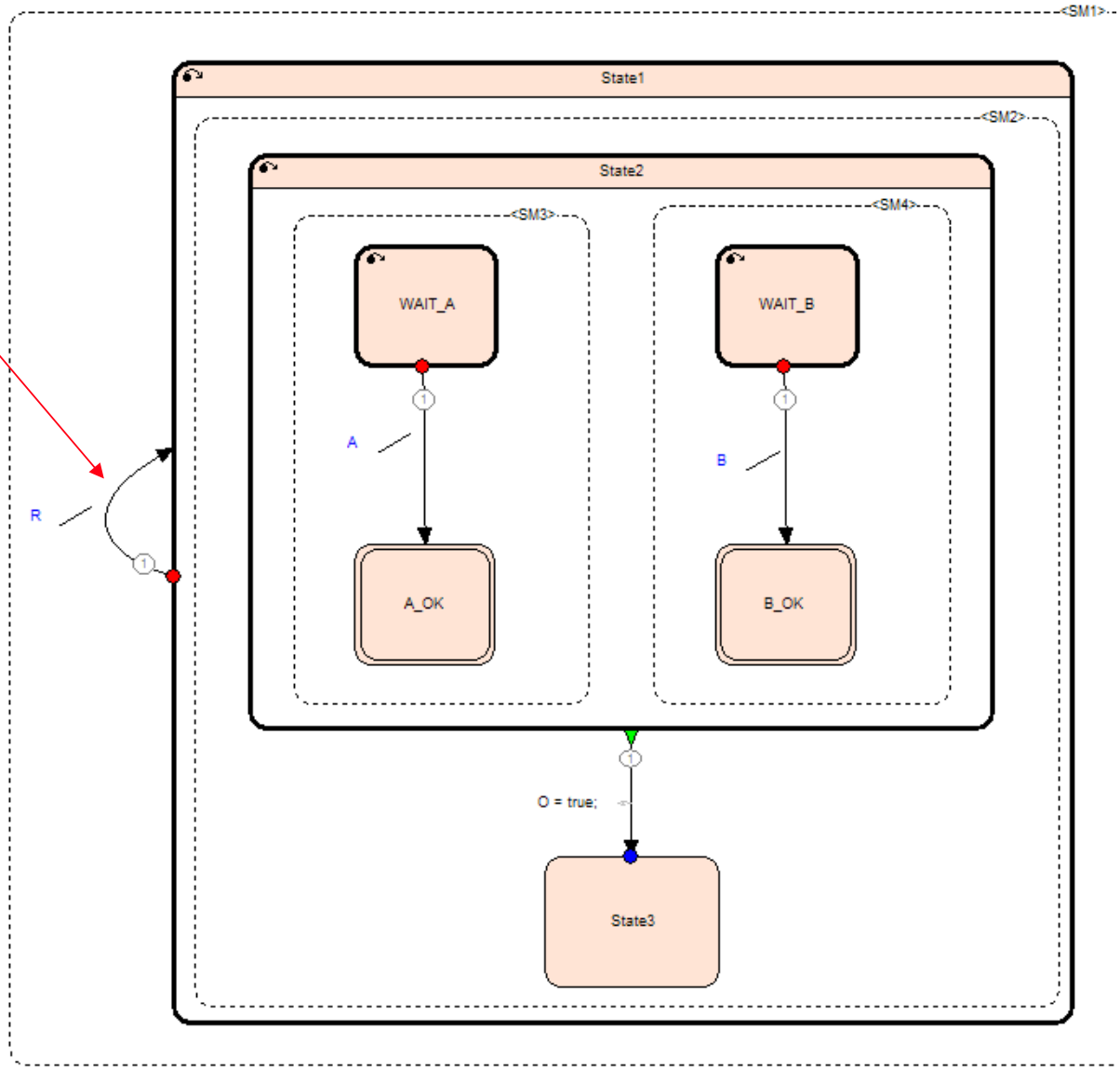
Strong ABRO

- Strong preemption

  O not set true

  if A, B, R

  simultaneous



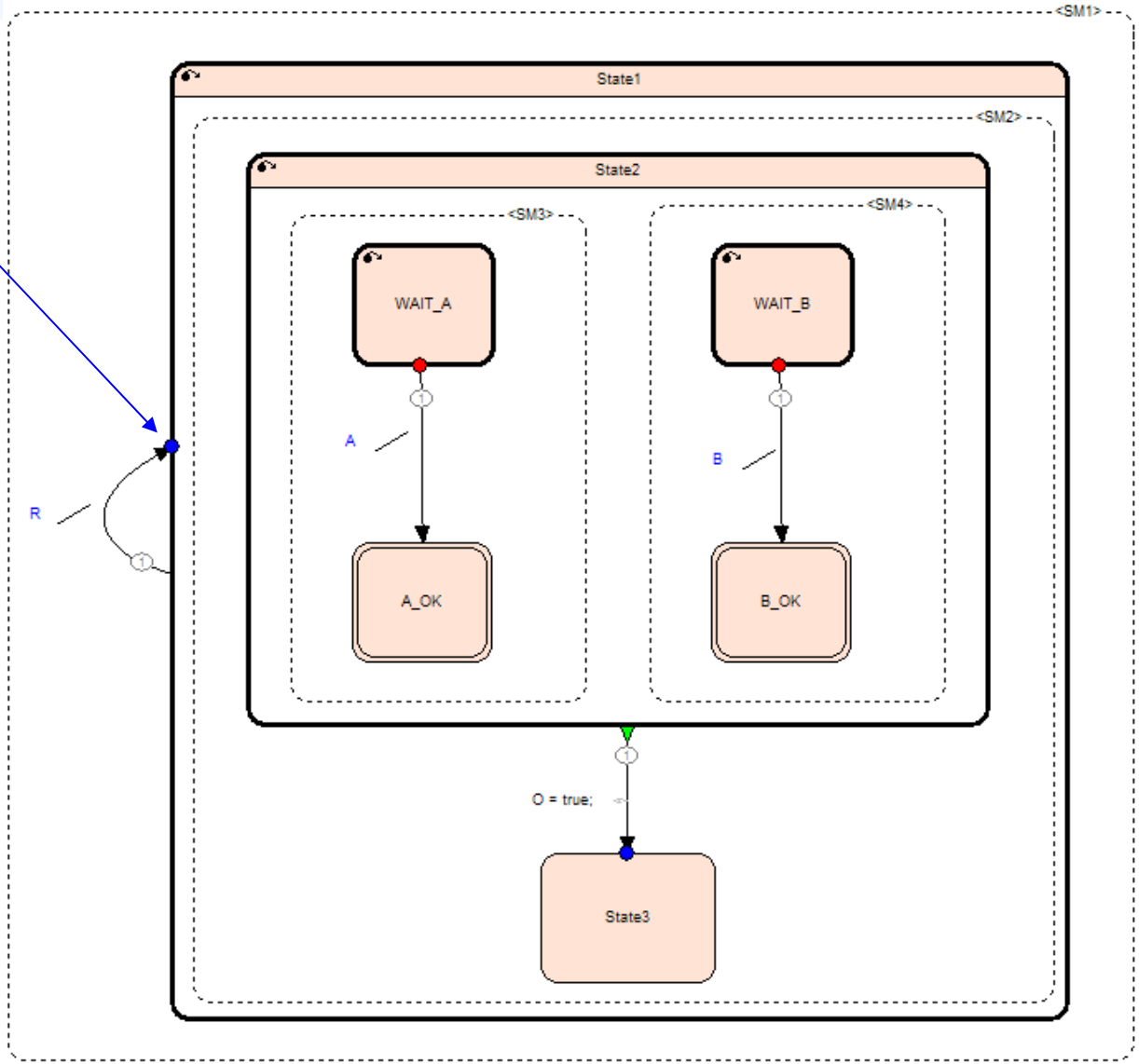
Weak ABRO

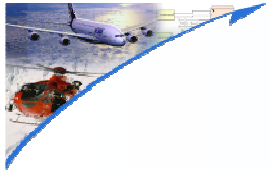
- Weak preemption

  O is set true

  if A, B, R

  simultaneous





Parametric Polymorphism

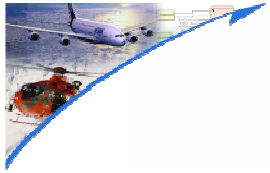
$c = a_0, b_1, a_2, b_3, a_4, b_5, ...$

```
node toggle_sample (a, b: int) returns (c: int)
var
   flag: bool
let
   flag = true -> not pre(flag);
   c = if flag then a else b
tel
```

monomorphic

```
node toggle_sample (a, b: 'T) returns (c: 'T)
var
   flag: bool
let
   flag = true -> not pre(flag);
   c = if flag then a else b
tel
```

polymorphic



Arrays

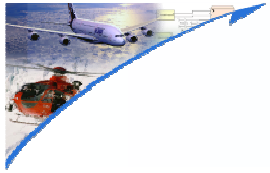
- Construction
  - Definition by extension

```
const array0: int^5 = [0,1,2,3,4];
const array1: int^2^3 = [[5,17],[0,-5],[1,1]];
```

  - Definition by replication
    - `array2 = 1^3; -- [1,1,1]`
  - Concatenation

```
array3: int^7;
array3 = array0 | [5,6]; -- [0,1,2,3,4,5,6]
```

- Access
  - Static indexing: `x = array1[3];`
  - Safe dynamic indexing: `x = array1.[i] default -1;`



Array operations

- Slicing

```
array4: int^3;
array4 = array0[1..2]; -- [1,2]
```

- Reversing

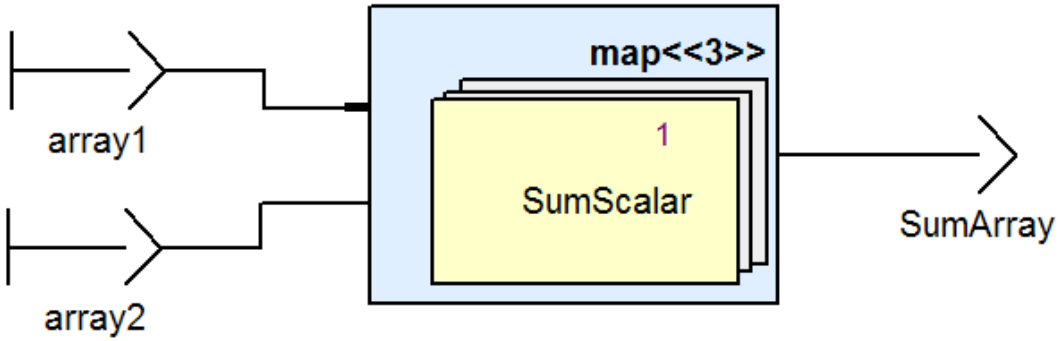
```
array5: int^5;
array5 = reverse array0; -- [4,3,2,1,0]
```

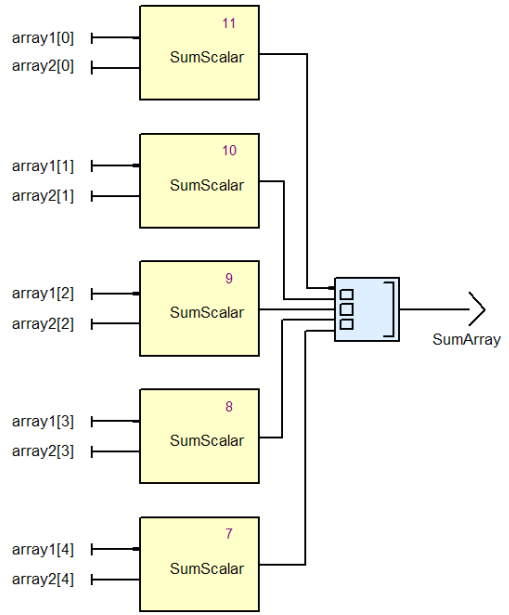
- Copy with change

```
array6 = (array0 with [2] = -1);
                -- [0,1,-1,3,4]
```

Array map Iterator

- **Point-wise sum of arrays:**

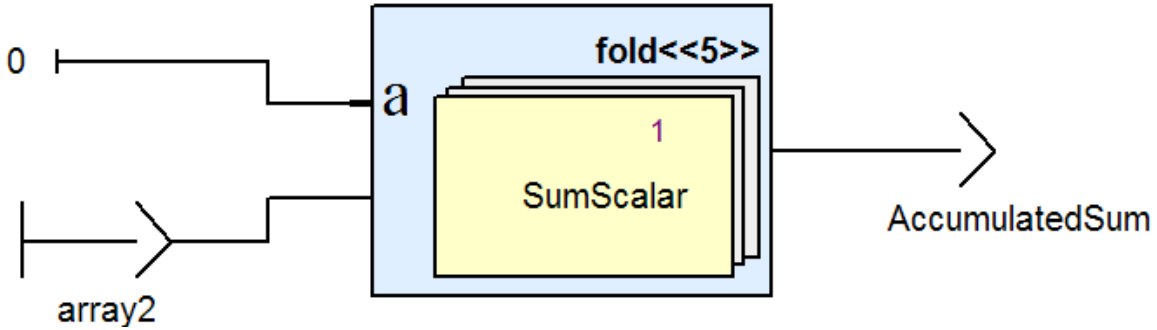


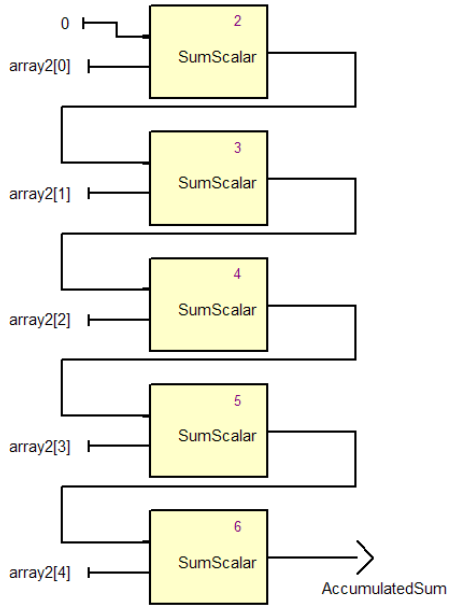



```
node SumScalar (a,b: int) returns (c: int)
let
  c = a + b;
tel
node SumArray(t,u: int^3) returns (v: int^3)
let
  v = (map SumScalar <<3>>) (t,u);
tel
SumArray([1,2,3],[2,4,0]) → [3,6,3]);
```

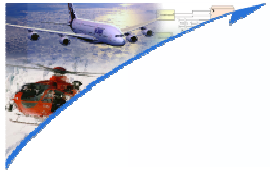
Array fold iterator

- **Accumulated sum of array elements:**



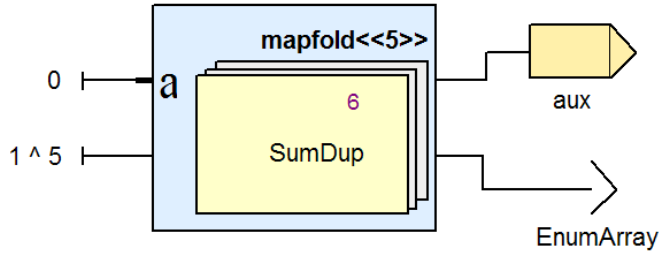


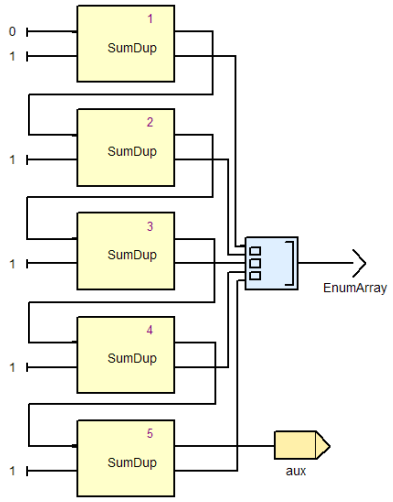

```
node AccumulatedSum(t: int^5) returns (v: int)
let
  v = (fold SumScalar <<5>>) (t);
tel
AccumulatedSum([1,2,3,4,5]) → 15;
```



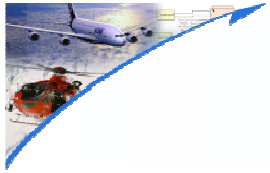
Array mapfold Iterator

- Combination of `map` and `fold`:





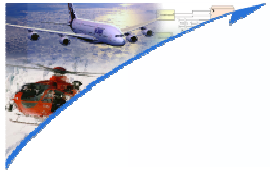

```
node SumDup (a,b: int) returns (s1,s2: int)
let
  s1 = a + b; s2 = s1;
tel
node EnumInt() returns (aux: int, EnumArray: int^5)
var aux: int;
let
  (aux, EnumArray) = (mapfold SumDup <<5>>) (0,1^5);
tel
EnumInt(0, 1^10) → 5, [1,2,3,4,5];
```



The Scade 6 Reference Manual

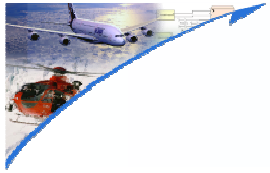
- Starts with a user-friendly Primer presenting all constructs
- Continues with a Reference Manual in English
  - syntax formally given in extended Backus-Naur form
- Then with a fully formal definition chapter
  - semantics given in Plotkin's SOS logical rules form
  - causality and initialization analysis fully formal
- Ends with the mapping of graphical constructs to textual ones

Formal definitions serves as the reference for KCG

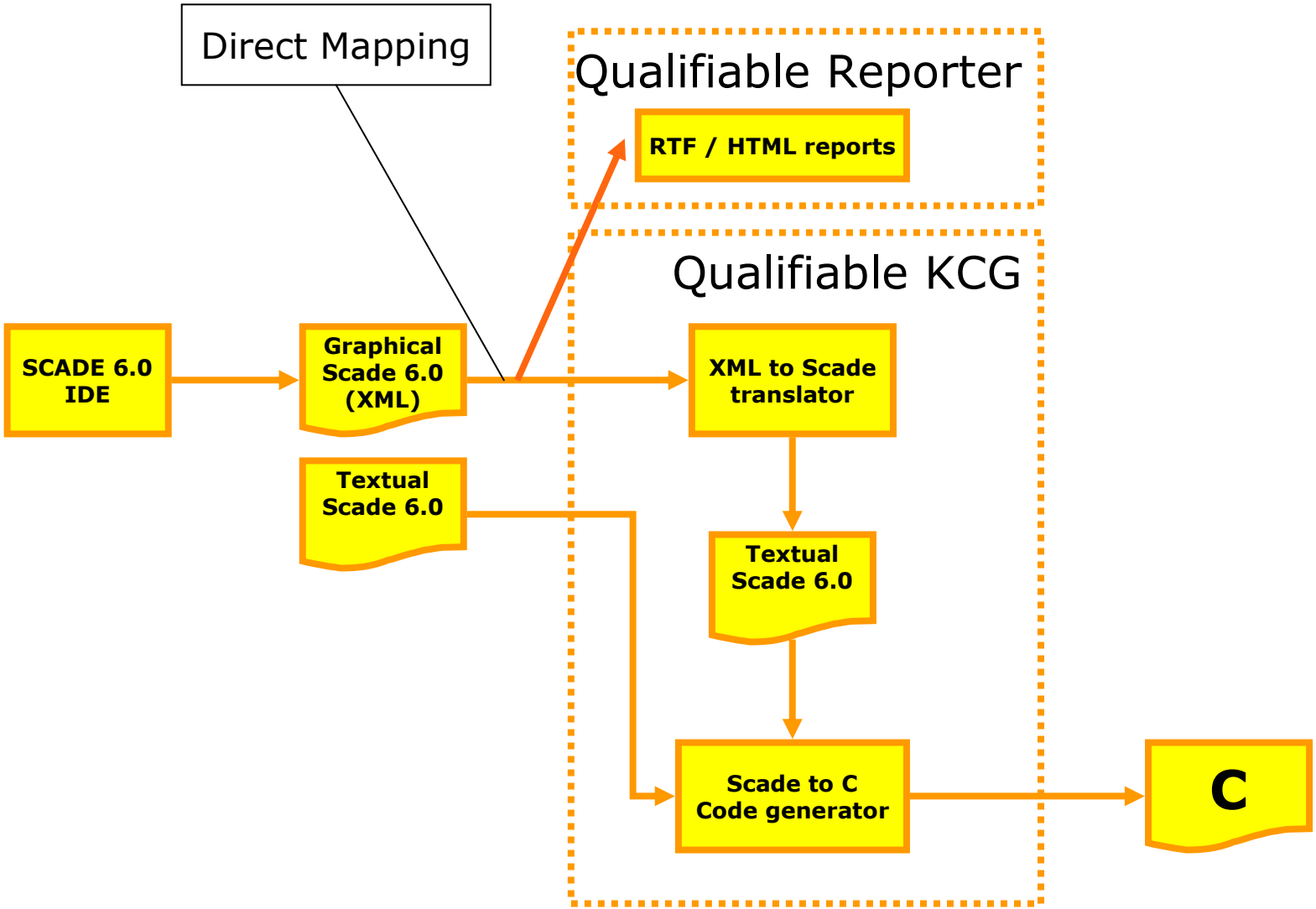


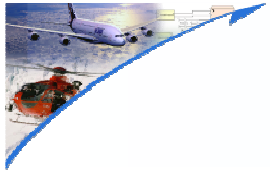
Agenda

- Scade 6 Design Goals
- The design of SCADE 6
- The KCG certifiable code generator
- Conclusion

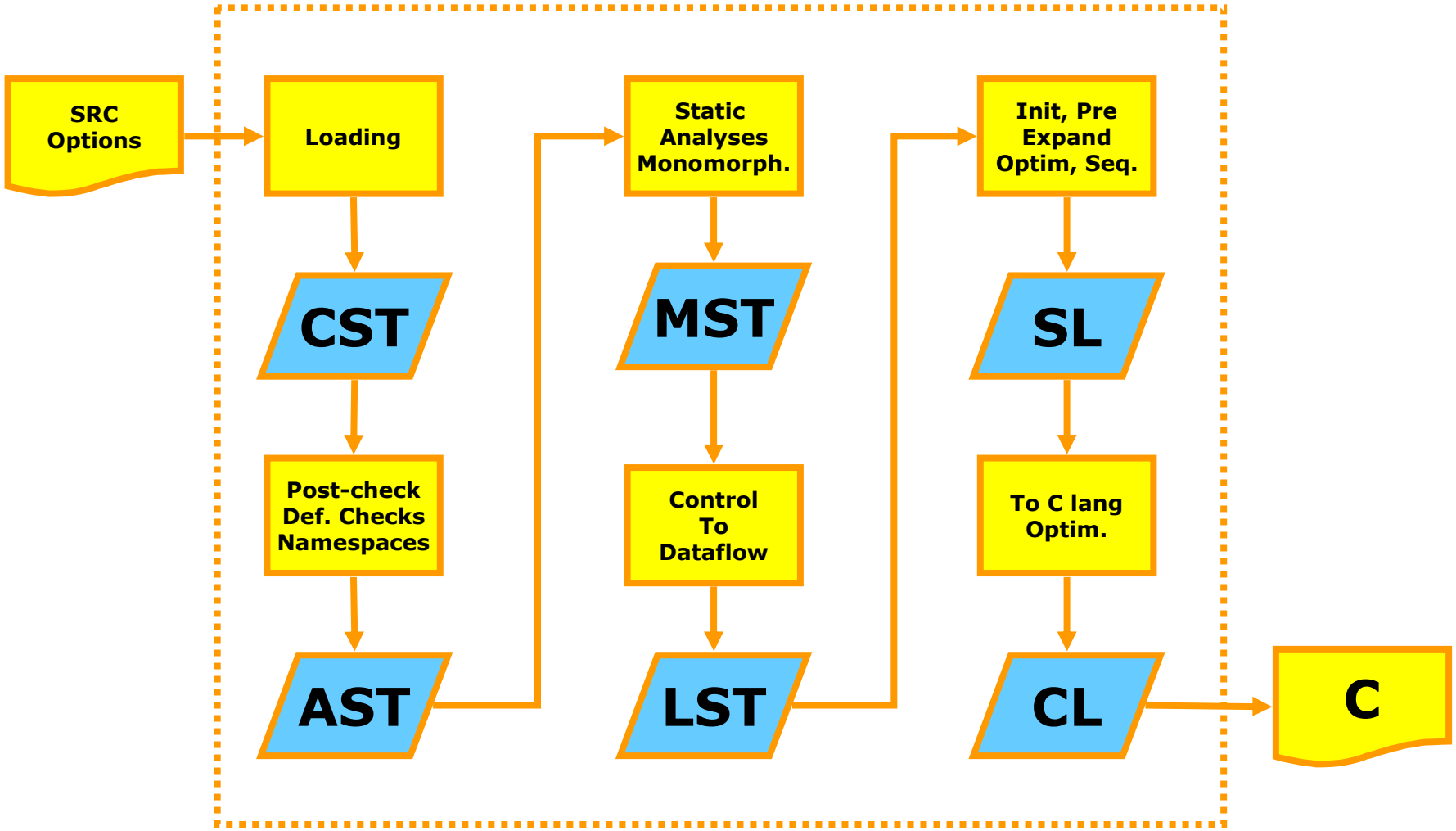


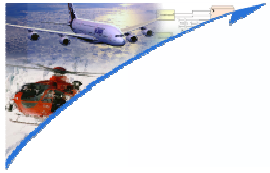
KCG Global Structure



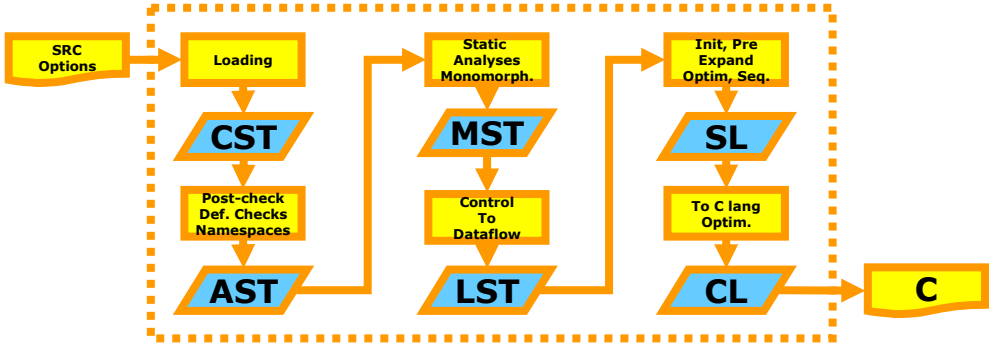


Structure of KCG Code Generator

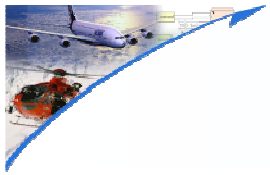




KCG Internal Representations

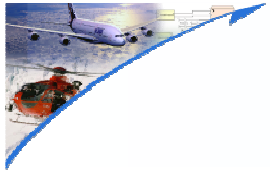


- **CST : Concrete Syntax Tree**
  - Internal form right after source loading
- **AST : Abstract Syntax Tree**
  - Naming resolution done, all variables have a definition, and all definitions correspond to declared variables
- **MST: Monomorphised Syntax Tree**
  - Polymorphism resolved, possibly by creation of corresponding instances of nodes.
- **LST : Lustre Syntax Tree**
  - Data-flow only syntax tree. SSM have been compiled into control blocks, and control blocks into equations
- **SL : Sequential Language**
  - Internal representation of a sequential (imperative) language after data-flow scheduling
- **CSL: C Syntax Language tree**
  - Internal representation of the used C subset.
- **IEC 1131 Language tree ?**



Static Checks

- **Classical**
  - Type-check
  - Connection check
  - Activation check
  - Check that all flows are used (DO-178B constraints)
- **Flow definition checks**
  - Clock compatibility checks
  - Check that all flows are initialized (static analysis)
- **Abstract interpretation based checks**
  - Absence of arithmetic exceptions (Astrée)
  - Worst-case execution time (WCET, AbsInt)
  - (Much more difficult!)



Conclusion

- SCADE 6 : major evolution of SCADE
  - fuses data flow and state machines into a single hierarchy
  - provides the user with safe functional arrays
  - and with node polymorphism to build generic libraries
  - preserves the well-founded SCADE formal semantics
- SCADE 6 is built upon
  - previous experience in SCADE / Synchronous Lucid compiling
  - previous design and implementation of Esterel v7
  - simplifications to make source code semantics straightforward and generated code efficient and traceable
  - A new KCG (written in CAML)

Semantics and Circuit Translation for Esterel v7

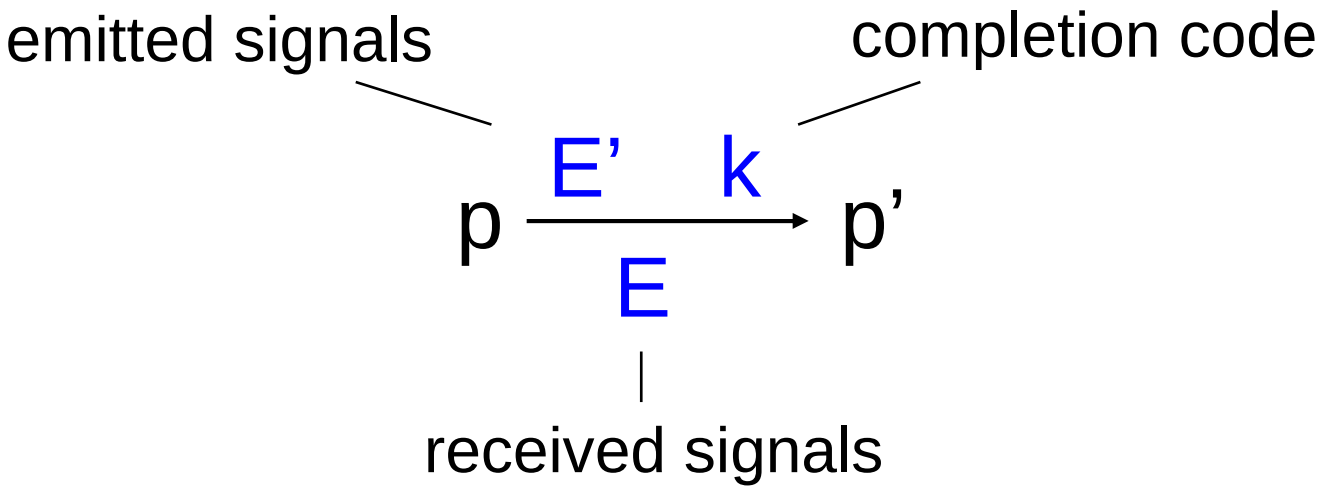
G. Berry

Marktoberdorf, 2008

The Pure Esterel Kernel

nothing	0
pause	1
emit S	! s
if S then p else q end	s ? p, q
suspend p when S	s ⊃ p
p; q	p; q
loop p end	p*
p || q	p | q
trap T in p end	{p} ↑ p
exit T	k, k > 1
signal S in p end	p \ s

The Behavioral Semantics



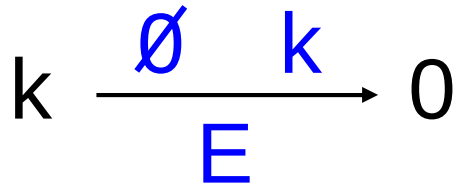
Broadcasting : E’ ⊂ E

k
- 0 : termination
- 1 : waiting
- 2 : exiting one trap level
- 3 : exiting two trap levels

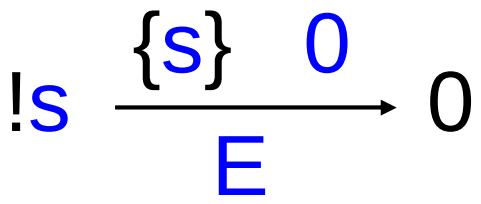
Nested traps numbering

```
trap T in
   trap U in
      ... exit U2
      ... exit T3
   end trap
||
   trap V in
      ... exit V2
      ... exit T3
   end trap
||
   ...exit T2
end trap
```

When two traps are exited concurrently, only the outermost one matters
  => compute max(k1, k2)



(for k=0, k=1, k>1)



$$\frac{s \ni E \qquad p \xrightarrow[E]{E' \quad k} p'}{s ? p, q \xrightarrow[E]{E' \quad k} p'}$$

$$\frac{s \not\ni E \qquad q \xrightarrow[E]{F' \quad l} q'}{s ? p, q \xrightarrow[E]{F' \quad l} q'}$$

$$\frac{p \xrightarrow[E]{E' \quad 0} p'}{s \supset p \xrightarrow[E]{E' \quad 0} 0}$$

$$\frac{p \xrightarrow[E]{E' \quad k} p' \qquad k \neq 0}{s \supset p \xrightarrow[E]{E' \quad k} s \supset\!\bullet\, p'}$$

[7] with $s \supset\!\bullet\, p' = \{(s\ ?\ 1\ ,\ 2)*\}\ ;\ s \supset p'$

$$\frac{p \xrightarrow[E]{E' \;\; k} p' \qquad k \neq 0}{p \; ; \; q \xrightarrow[E]{E' \;\; k} p' \; ; \; q}$$

$$\frac{p \xrightarrow[E]{E' \;\; 0} p' \qquad q \xrightarrow[E]{F' \;\; l} q'}{p \; ; \; q \xrightarrow[E]{E' \cup F' \;\; l} q'}$$

$$\frac{p \xrightarrow[E]{E' \quad k} p' \qquad k \neq 0}{p* \xrightarrow[E]{E' \quad k} p' \,;\, p*}$$

$$\frac{p \xrightarrow[E]{E' \quad k} p' \qquad q \xrightarrow[E]{F' \quad l} q'}{p \mid q \xrightarrow[E]{E' \cup F' \quad \max(k,l)} p' \mid q'}$$

$$\frac{p \xrightarrow[E]{E'\ k} p' \qquad k = 0 \text{ or } k = 2}{\{p\} \xrightarrow[E]{E'\ 0} 0}$$

$$\frac{p \xrightarrow[E]{E'\ k} p' \qquad k = 1 \text{ or } k > 2}{\{p\} \xrightarrow[E]{E'\ \downarrow k} \{p'\}}$$

$\downarrow k = 1$ if $k=1$,
$k-1$ if $k>2$

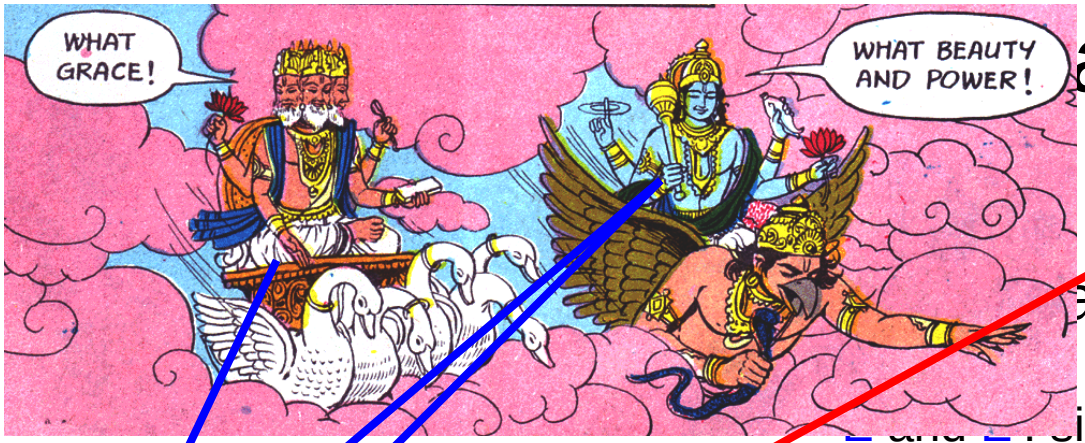
$$\frac{p \xrightarrow[E \cup \{s\}]{E' \cup \{s\} \quad k} p'}{p \setminus s \xrightarrow[E]{E' \quad k} p' \setminus s}$$

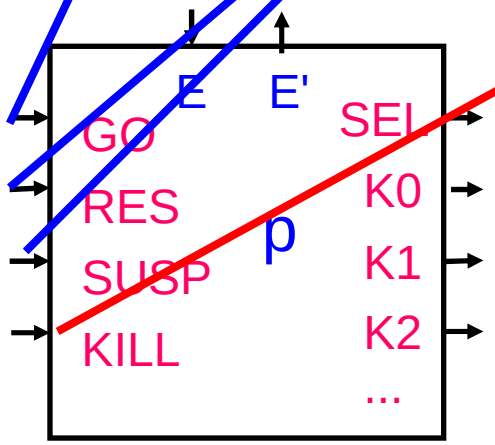
$$\frac{s \notin E \quad s \notin E' \quad p \xrightarrow[E]{E' \quad k} p'}{p \setminus s \xrightarrow[E]{E' \quad k} p' \setminus s}$$

Unique solution => determinism

But no solution or several solutions possible!

...ai...eme



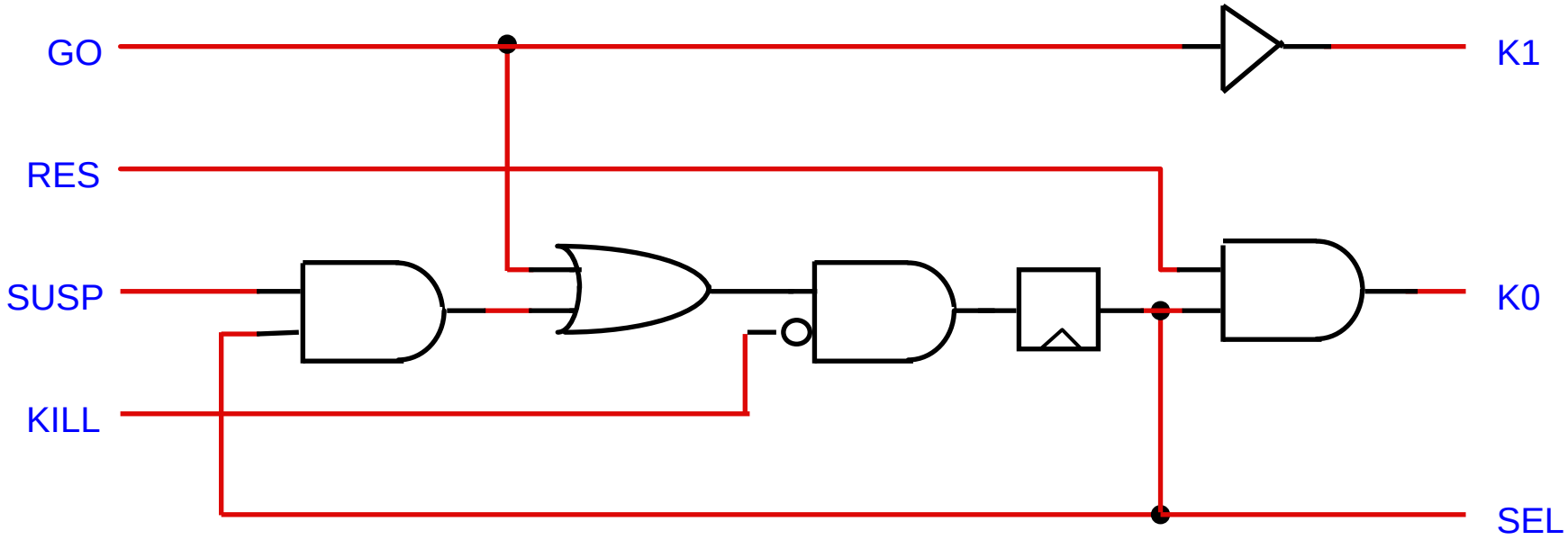


...a b...

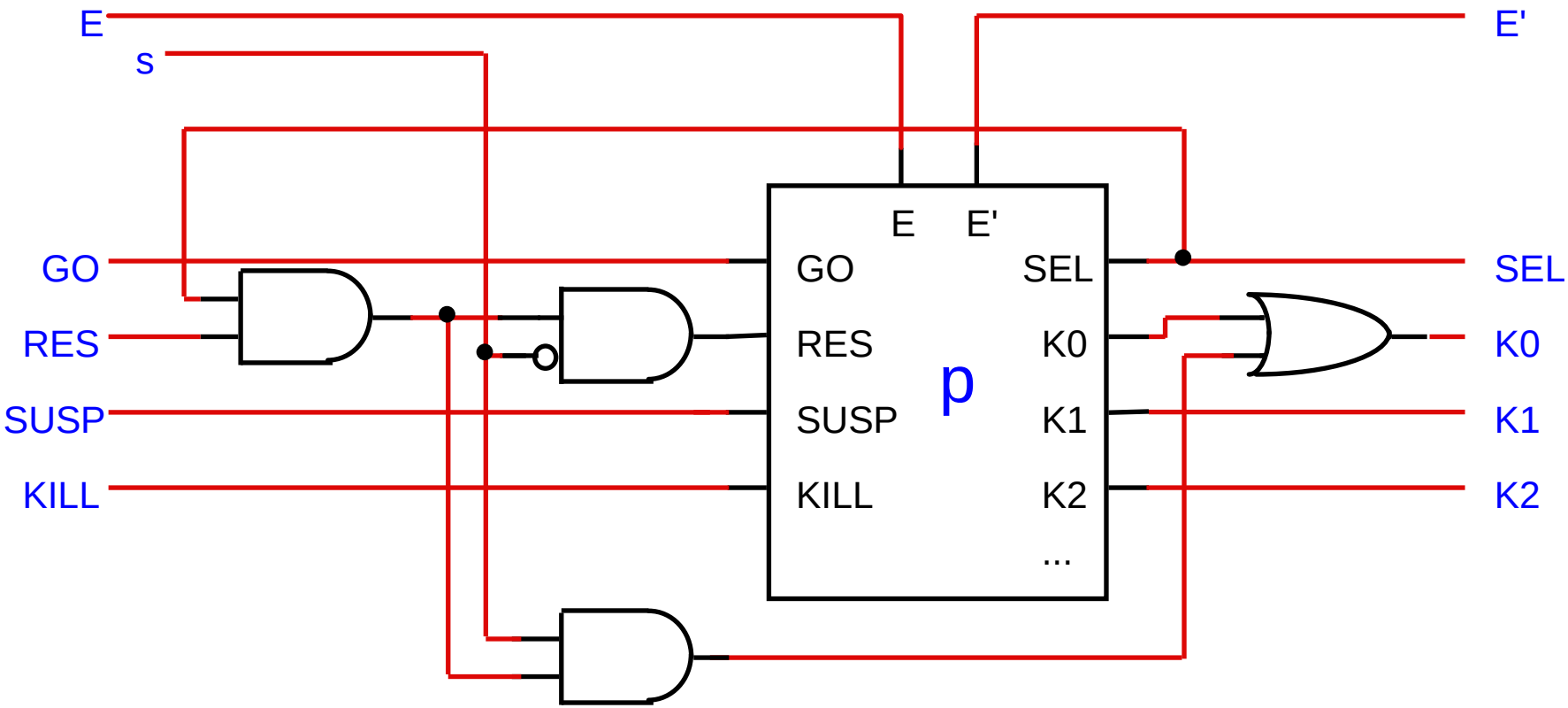
...E and E'...signals received and... emitted

- GO: start p (first cycle)
- RES: continue from the previous state
- SUSP: freeze for a cycle (keep registers)
- KILL : reset registers
- SEL: at least one register set = statement *alive*
- Ki : 1-hot encoded *completion code*
  - K0: terminate
  - K1: pause for a cycle
  - K2,K3,… - exit enclosing traps

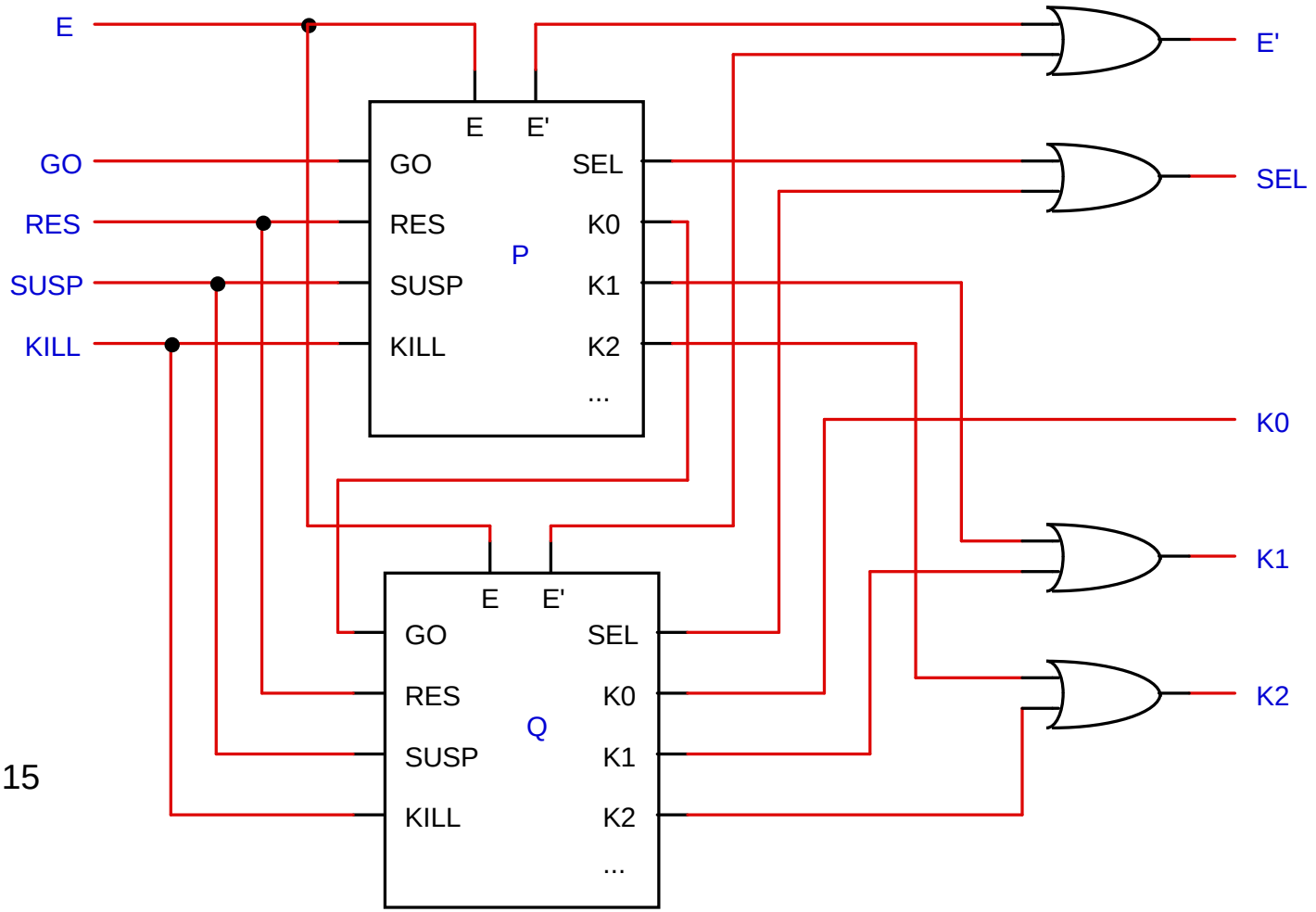
Circuit for 1 (pause)



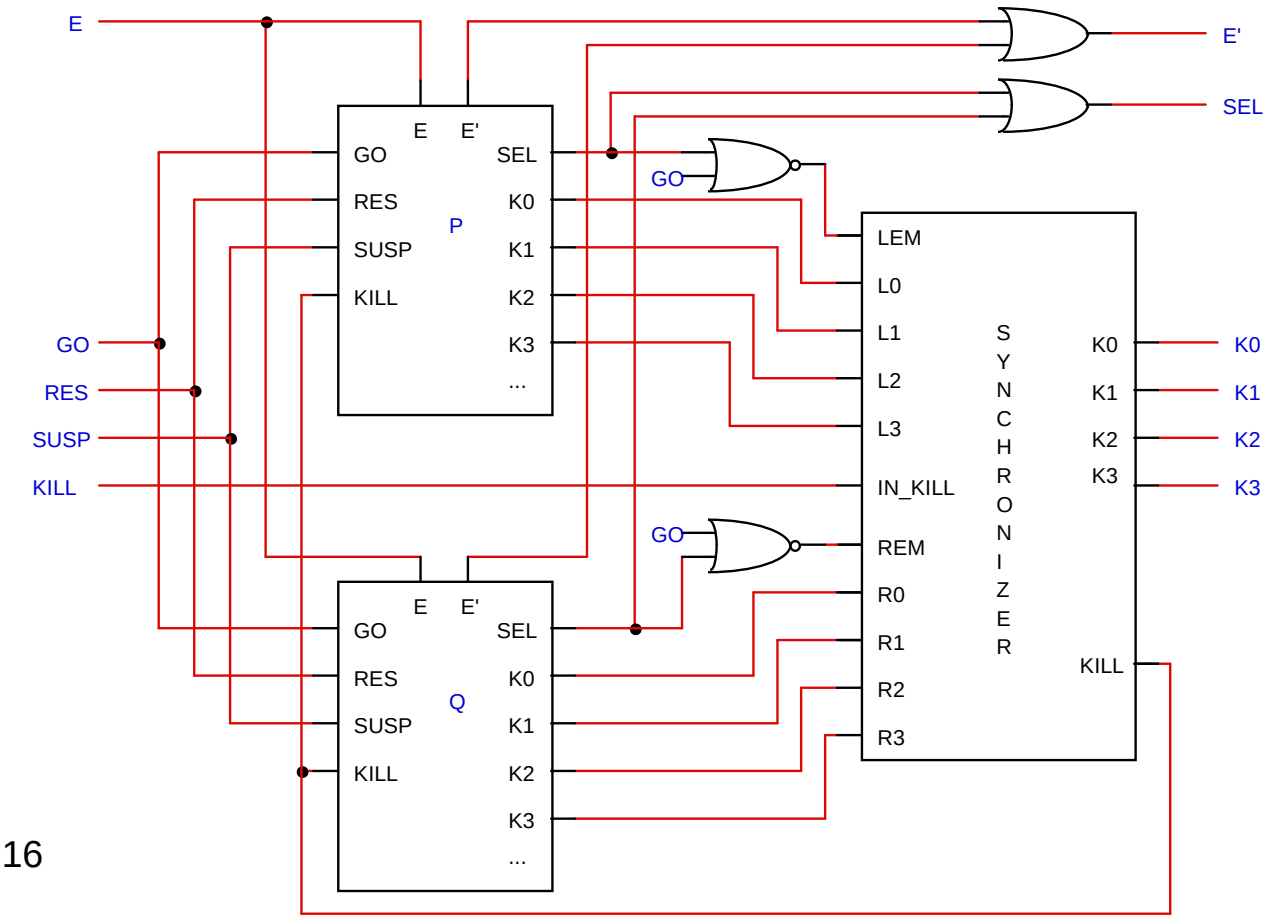
Circuit for abort p when s



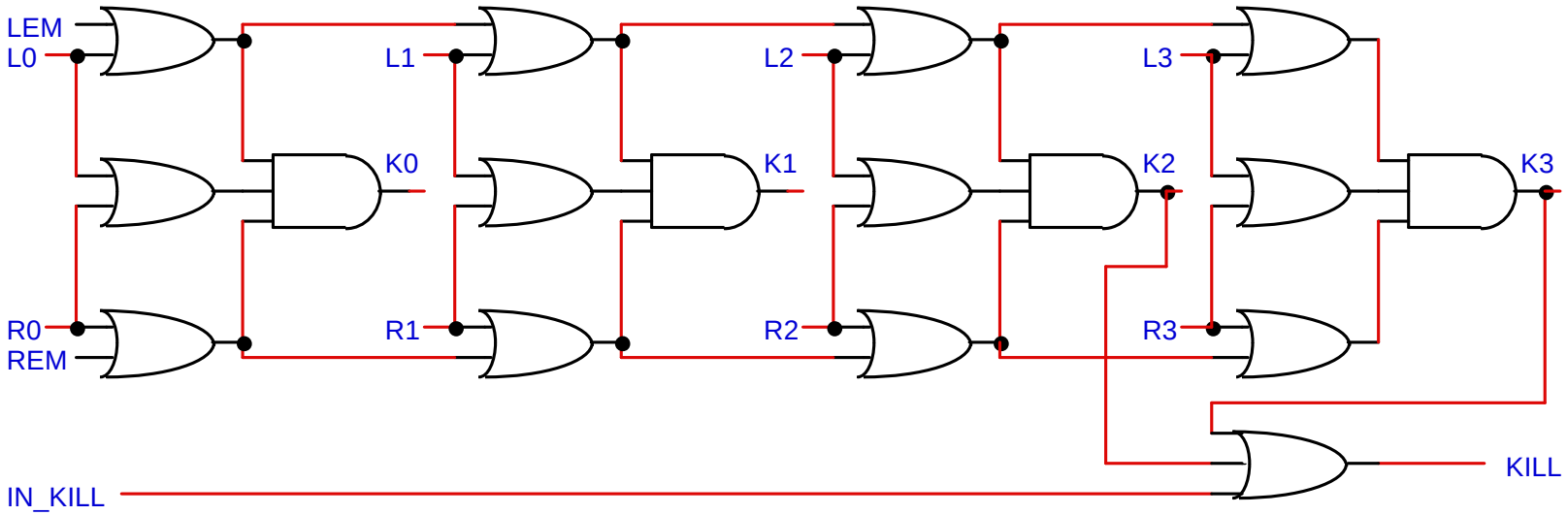
Circuit for sequencing P; Q



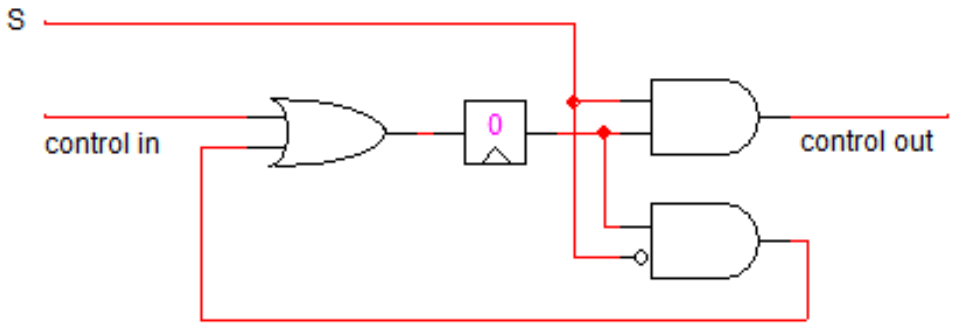
Circuit for parallel P||Q



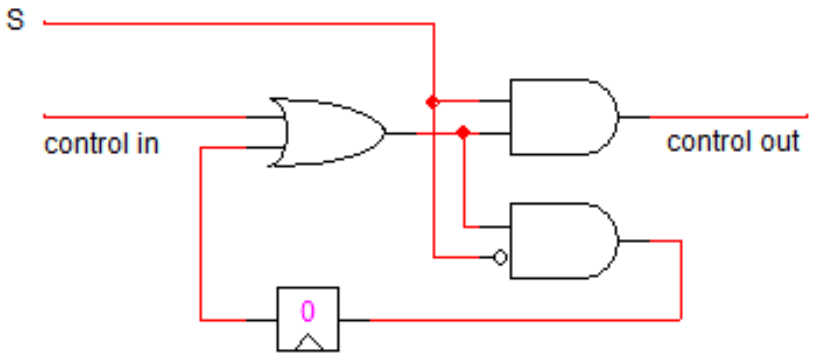
The parallel synchronizer



Examples : await

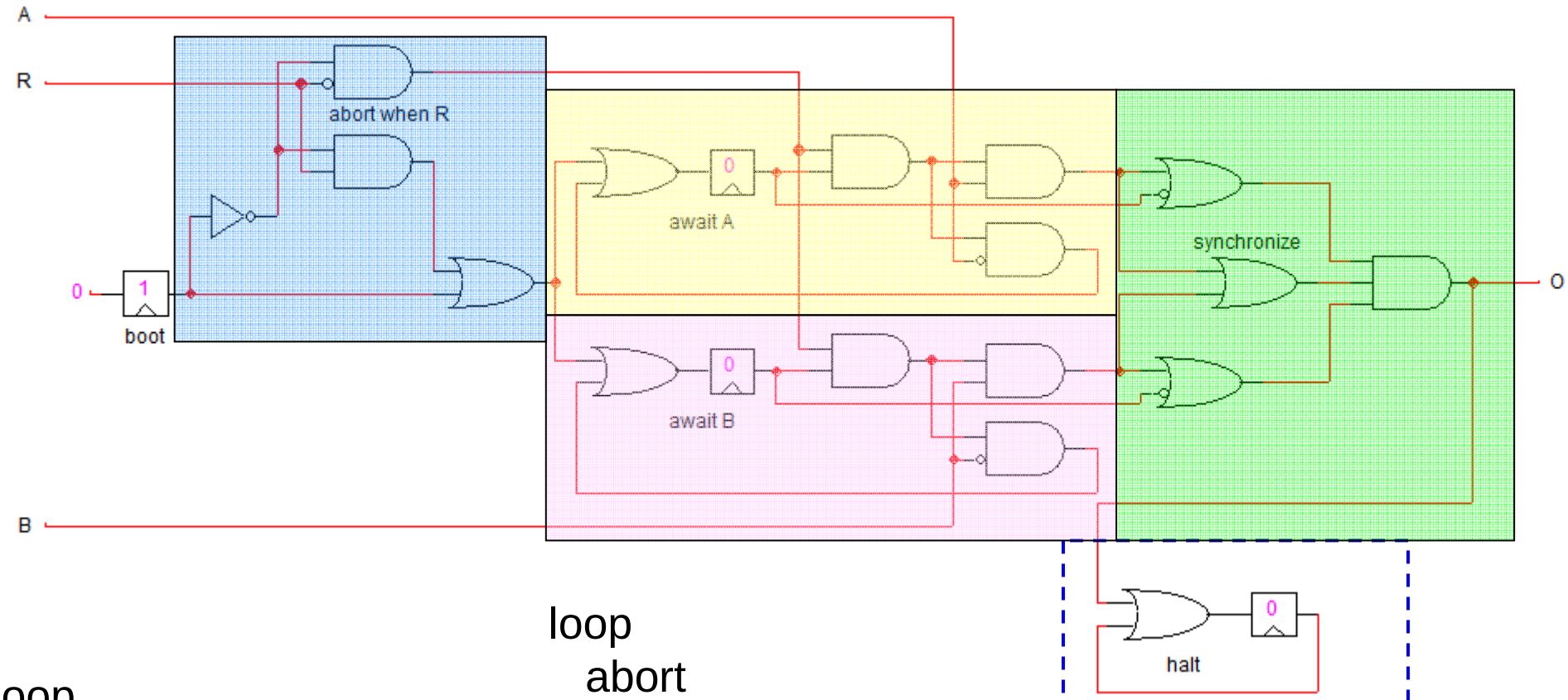


await S



await immediate S

Hierarchical ABRO circuit



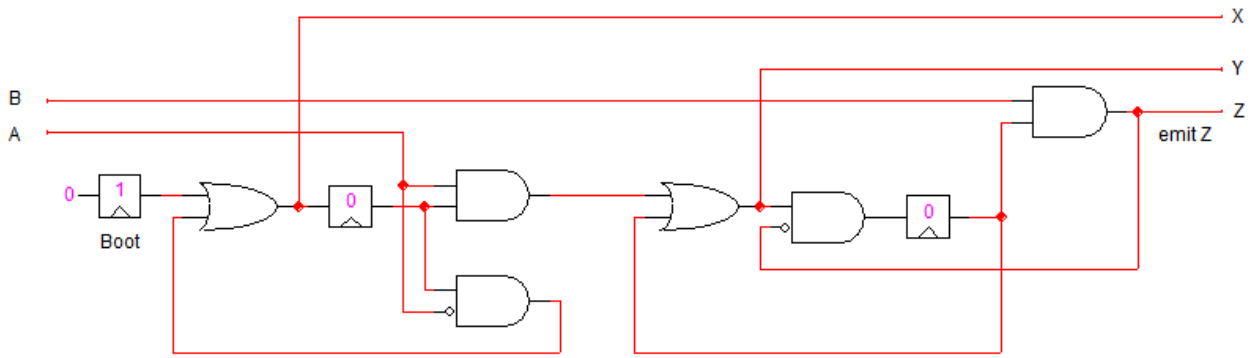

```
loop
   { await A || await B };
   emit O
each R
```

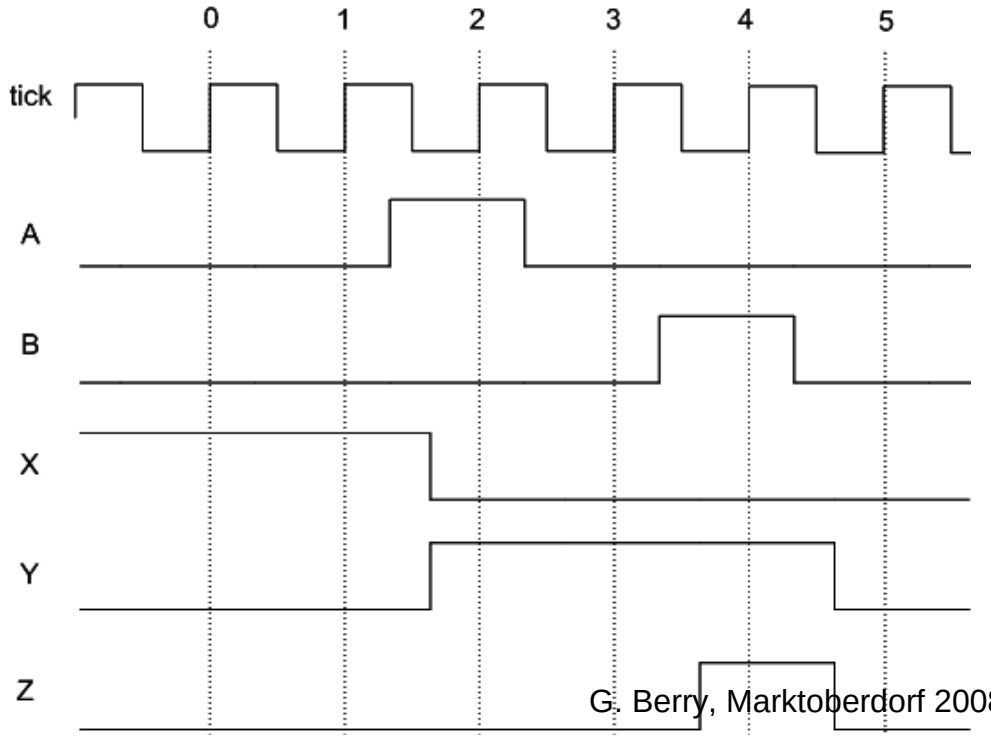
```
loop
   abort
      { await A || await B };
      emit O
   when R;
   halt
end loop
```

useless!

Example 9 : abort and weak abort

```
abort
   sustain X
when A;
weak abort
   sustain Y
when B;
emit Z
```

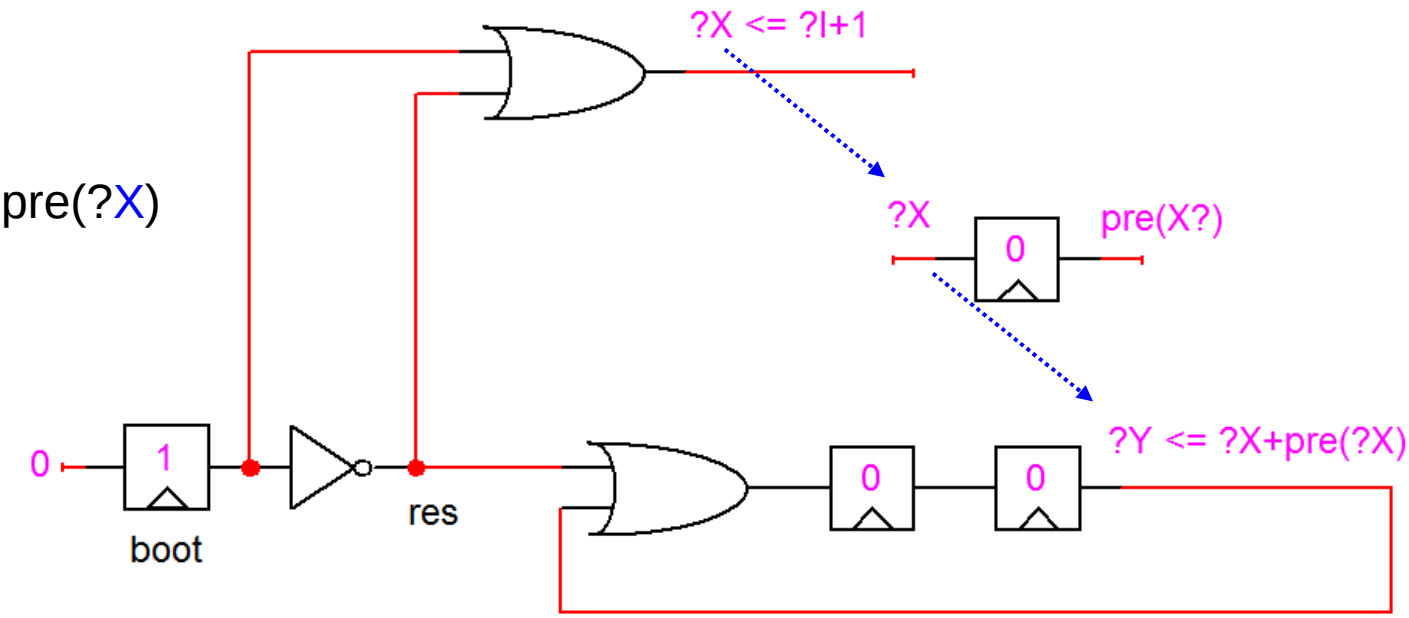




Data path handling

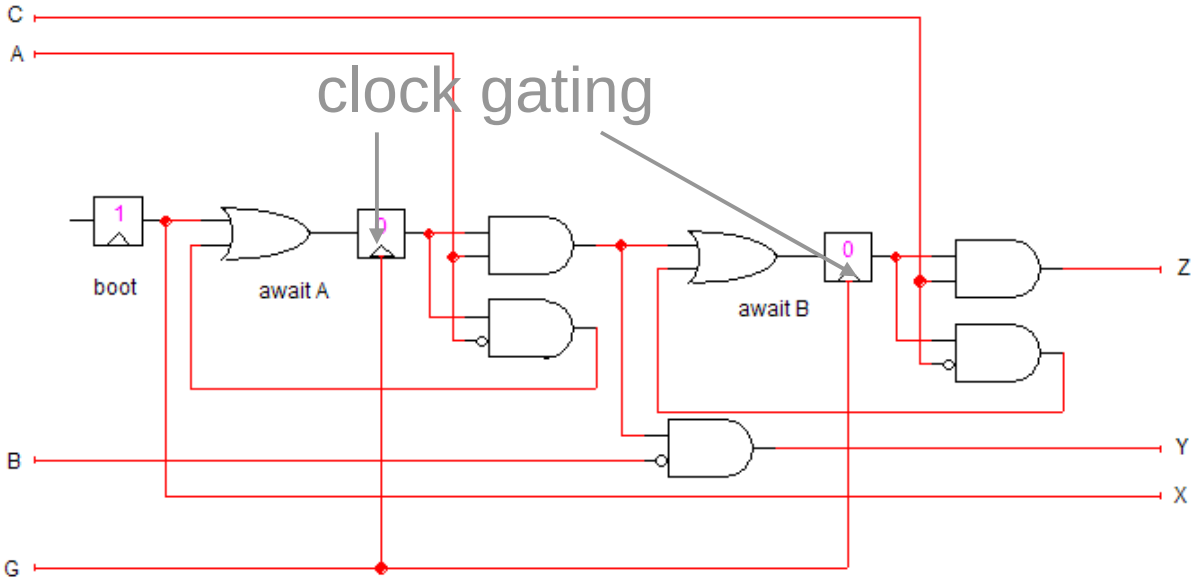
```
signal X : unsigned in
   sustain ?X <= ?I+1
||
   loop
      pause;
      pause;
      emit ?Y <= ?X+pre(?X)
   end loop
end signal
```

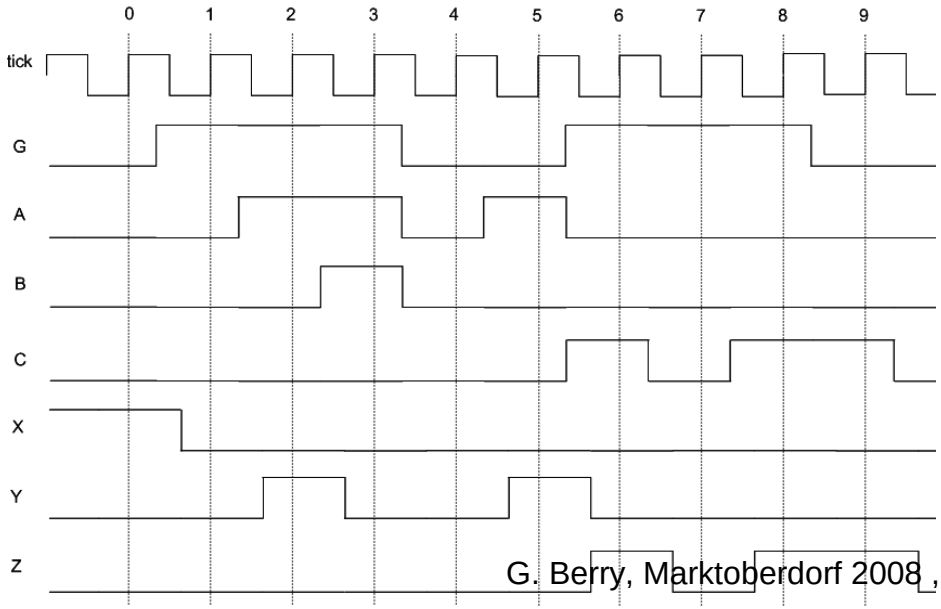
data dependencies
(for sorting)



Clock gating

```
weak suspend
   emit X;
   await A;
   emit Y <= not C
   await B;
   emit Z;
when not G
```



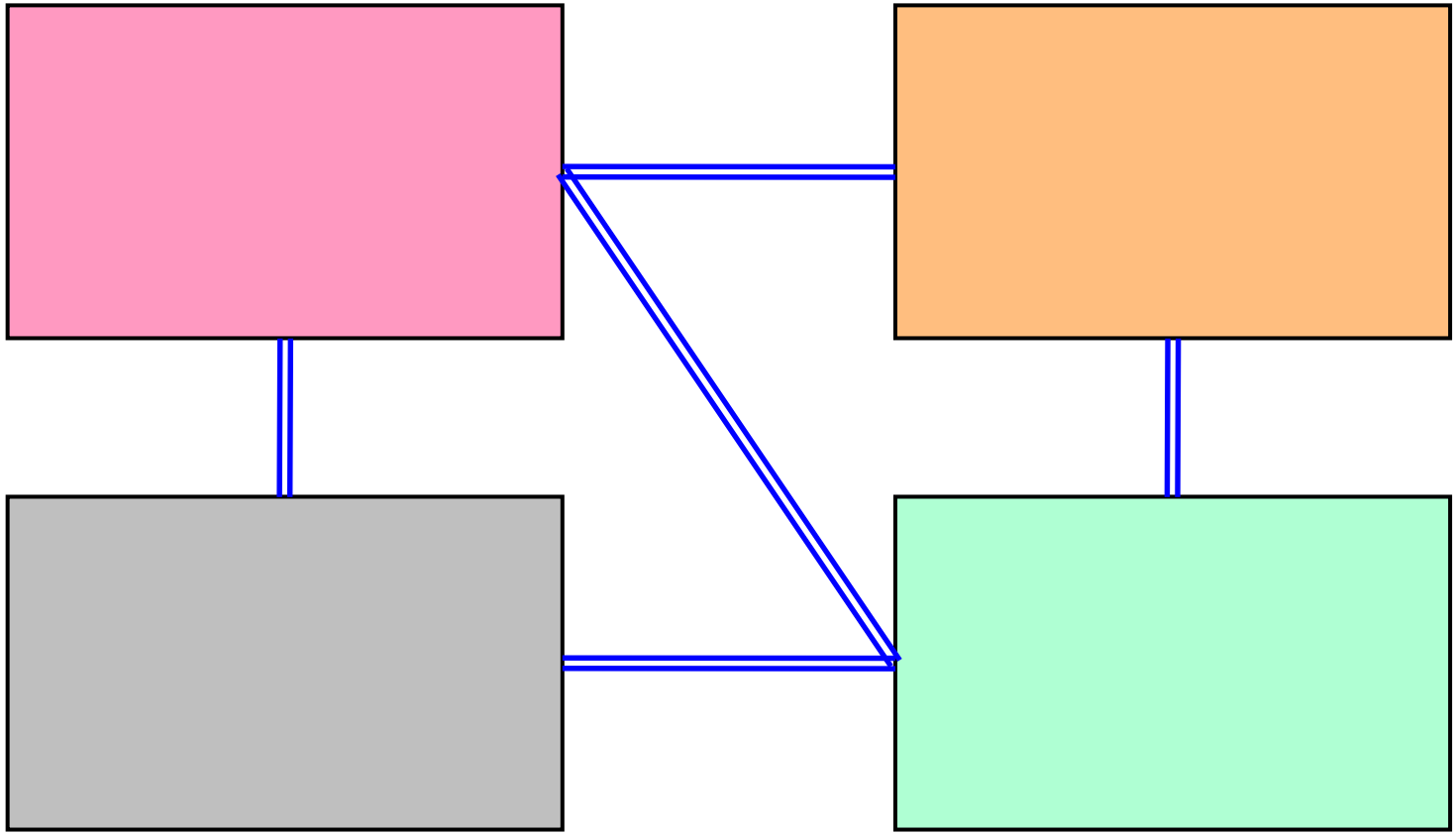


From Rooms to Castles: Synchrony + Asynchrony

Gérard Berry

Marktoberdorf 2008

Castles =
Rooms connected by corridors



Preserving synchrony : Time-Triggered Local Area Networks

- TTP (Kopetz), FlexRay, Deterministic Ethernet
  - clock synchronization
  - guaranteed transmission time (optional)
  - reconfiguration on failure
- Make it possible to be (quasi) castle-synchronous
  - adding known latency to communication
  - very strongly causal!

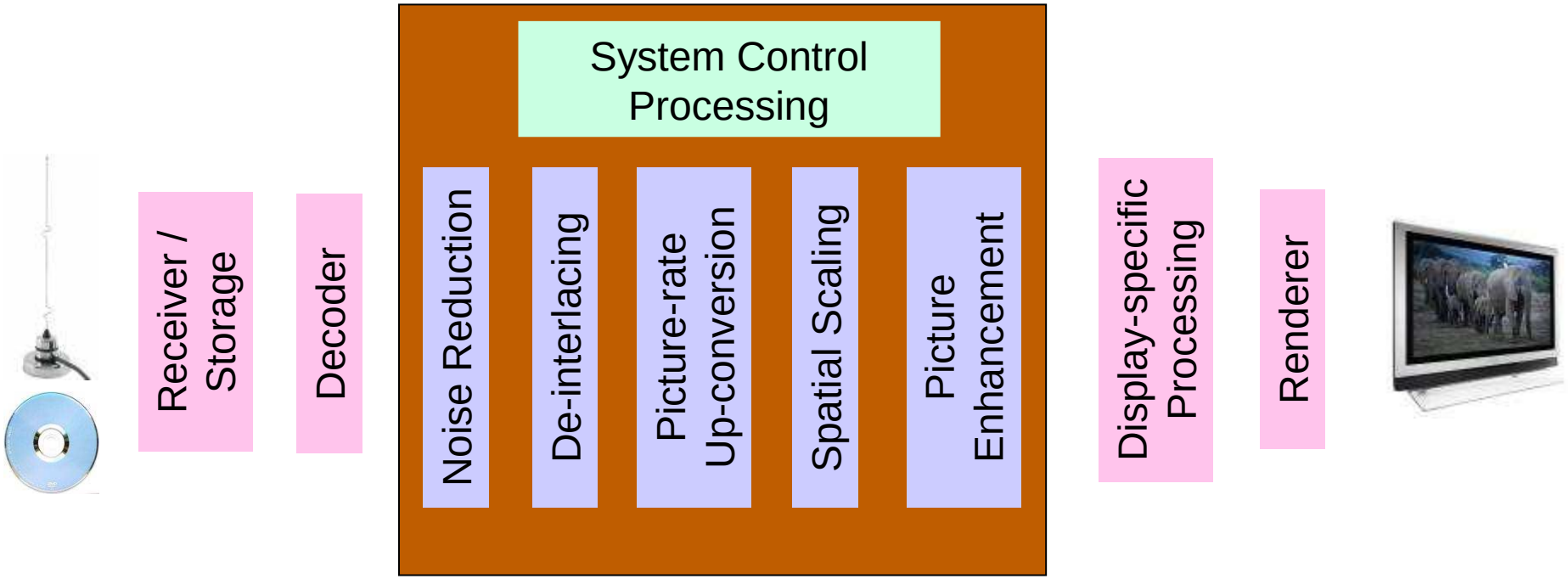
Never leave synchronization and failure recovery to application engineers, put them in the infrastructure

(H. Kopetz)

Asynchronous corridors

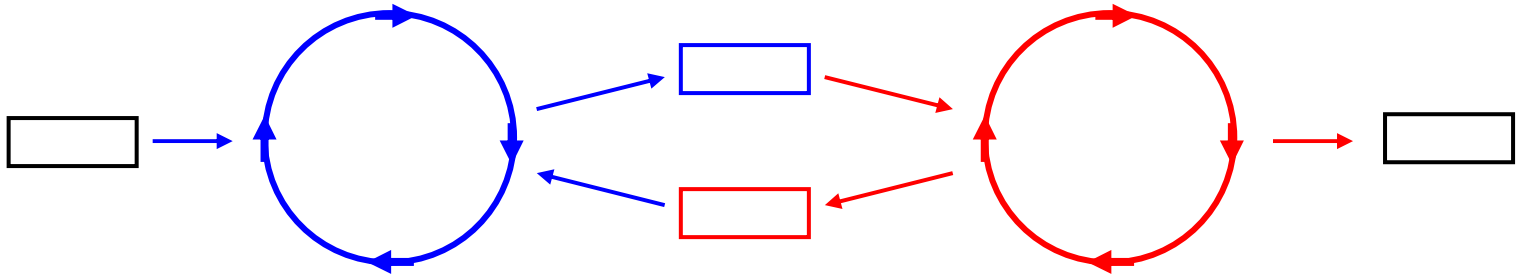
- Unbounded fifos : Kahn networks
  result deterministic, no time guarantee
- Bounded fifos, flow control
  risk of deadlocks, ok for feed-forward designs
- Bounded fifos, test for emptiness
  looses result determinism
- Rendezvous : CRP = Communicating Reactive Processes
  (Berry, Ramesh, Shyamasundar)
- One-place buffer
  Polis (Sangiovanni, Lavagno, et. al.)
  Disributed control (Caspi-Beveniste)
- Multiclock hardware
  now routine!

HDTV pipeline (TeraOps)



Courtesy Marc Duranton, NXP

Correct-by-control-theory asynchrony



- Distribution by mutual sampling

  computers sample each other at regular rates

- Works because of control theory stability results

  valid with easy-to-implement clock synchronization

  (Caspi & Benveniste)

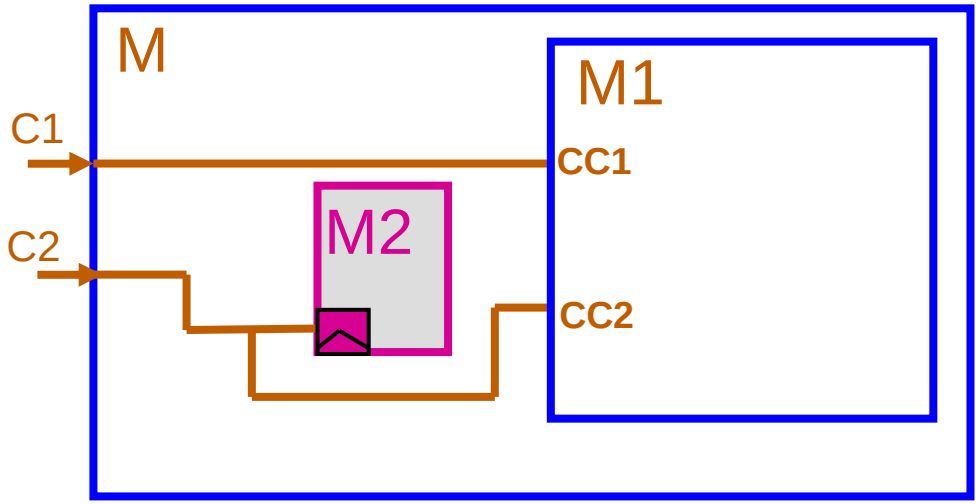
Clocks and Multiclock Units

Clocks

- special signal declared clock
- can clock the states of conventional single-clock modules
- can be downsampled or muxed
- no other combinational or sequential calculation allowed

Multiclock units

- module interface + clock interface
- can only do the following:
  - perform combinational (unclocked) calculations
  - run clocked modules
  - run multiclock units (hierarchy)
  - define new clocks



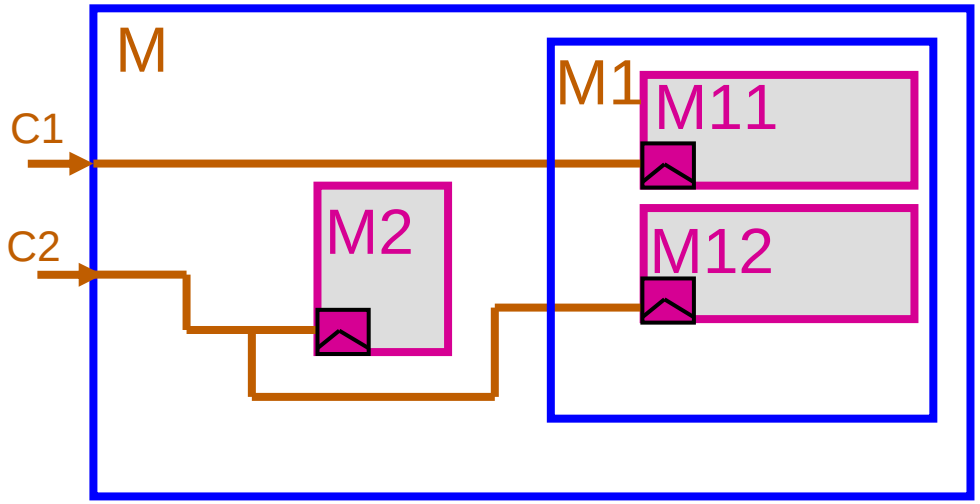
clock as a primitive special signal

```
multiclock M:
input C1,C2: clock;
...
 run M1[clock C1/CC1, C2/CC2]
||
 run M2[clock C2]
end module
```

```
multiclock M1:
input CC1,CC2:clock;
   ...


end module
```

```
module M2:
...
end module
```



hierarchical multiclock design

```
multiclock M:
input C1,C2: clock;
...
 run M1[C1/CC1,C2/CC2]
||
 run M2[clock C2]
end module
```

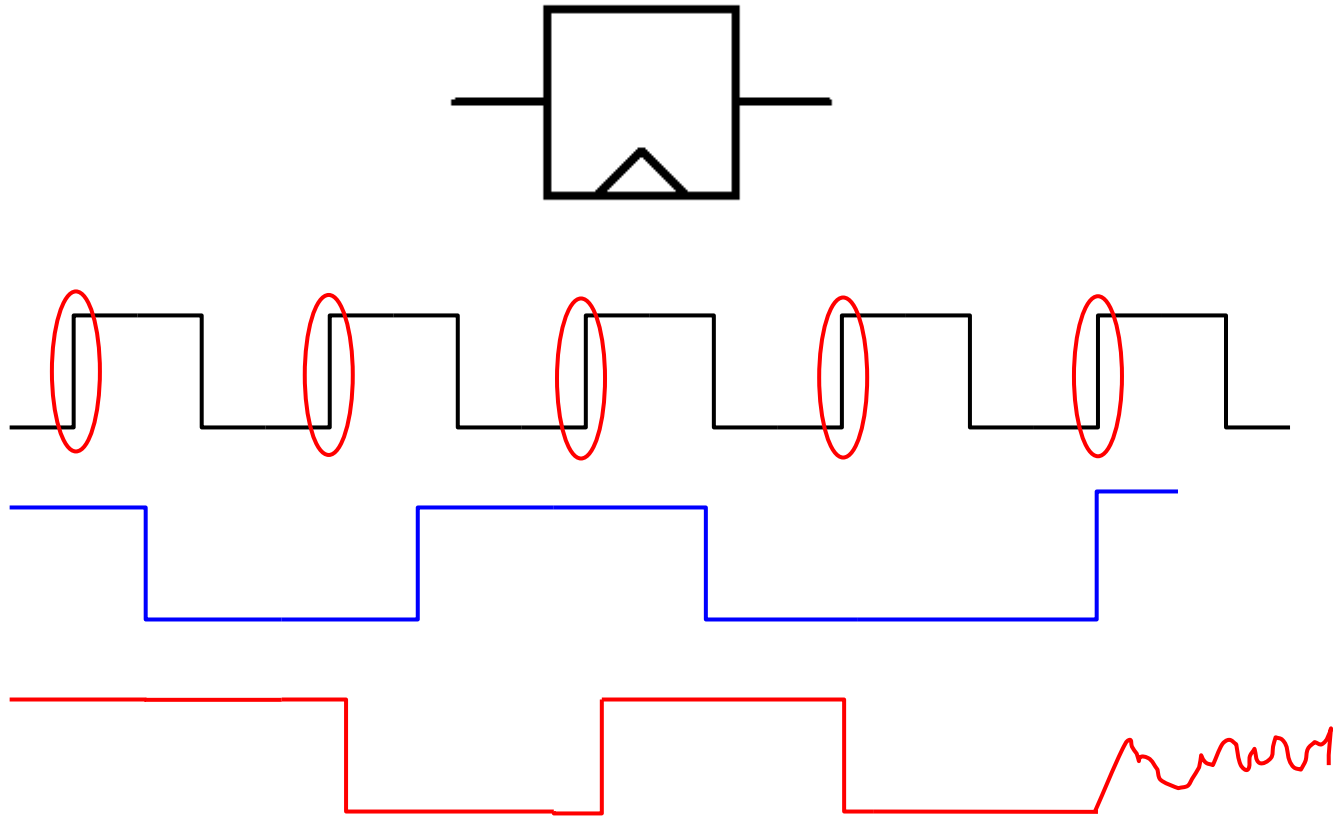
```
multiclock M1:
  input CC1,CC2:clock;
  ...
   run M11[clock CC1]
  ||
   run M12[clock CC2]
end module
```

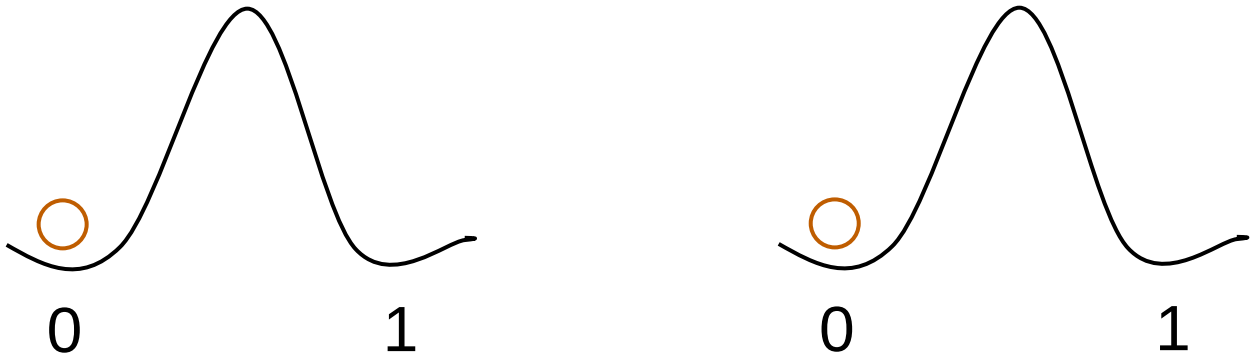
```
module M11:
...
end module
```

```
module M12:
...
end module
```

```
module M2:
...
end module
```

Metastability





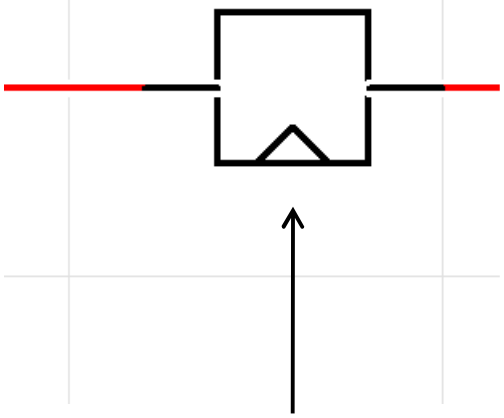
- Wind will make the ball fall some side
- In circuits, noise will do the same
- Theoretically, unbounded time
- Practically, less than one cycle (?)

How to synchronize?

- Multiclock fifo's (writer clock, reader clock)
  key components, delicate to code and verify
  fifo full and fifo empty cannot be exact
  $\Rightarrow$must be conservative

- Synchronizers
  push, pull, 2-phase, 4-phase, etc.
  key issue : handle metastability

Safe non-deterministic edge detector



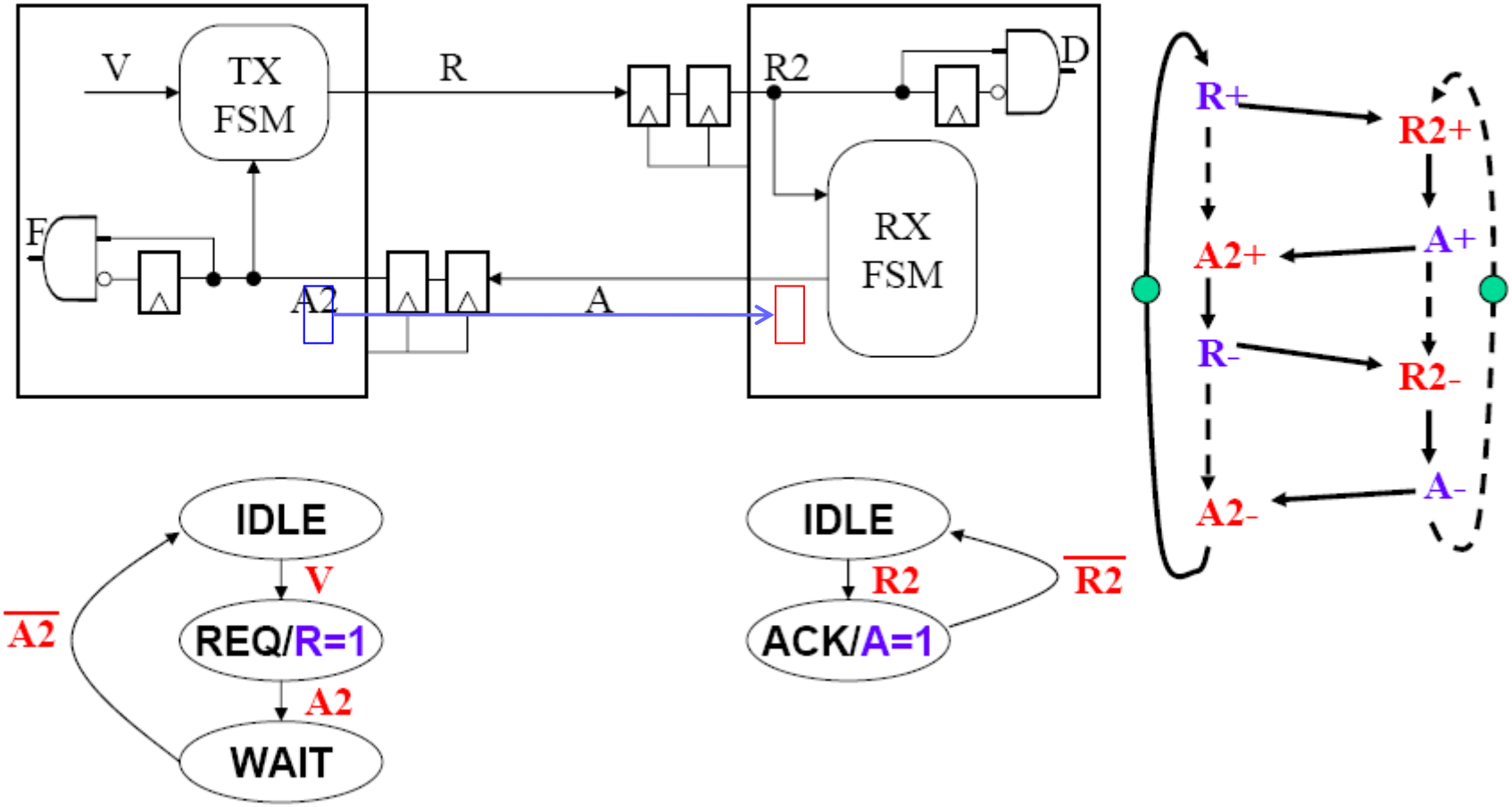
can be metastable

not metastable

change detector

Change is detected after 2 or 3 local clock cycles

Four-phase Push Synchronizer (Ran Ginosar, Technion)



Clock gating

- Usually considered as a way to save power
- Partly automated by pattern-matching

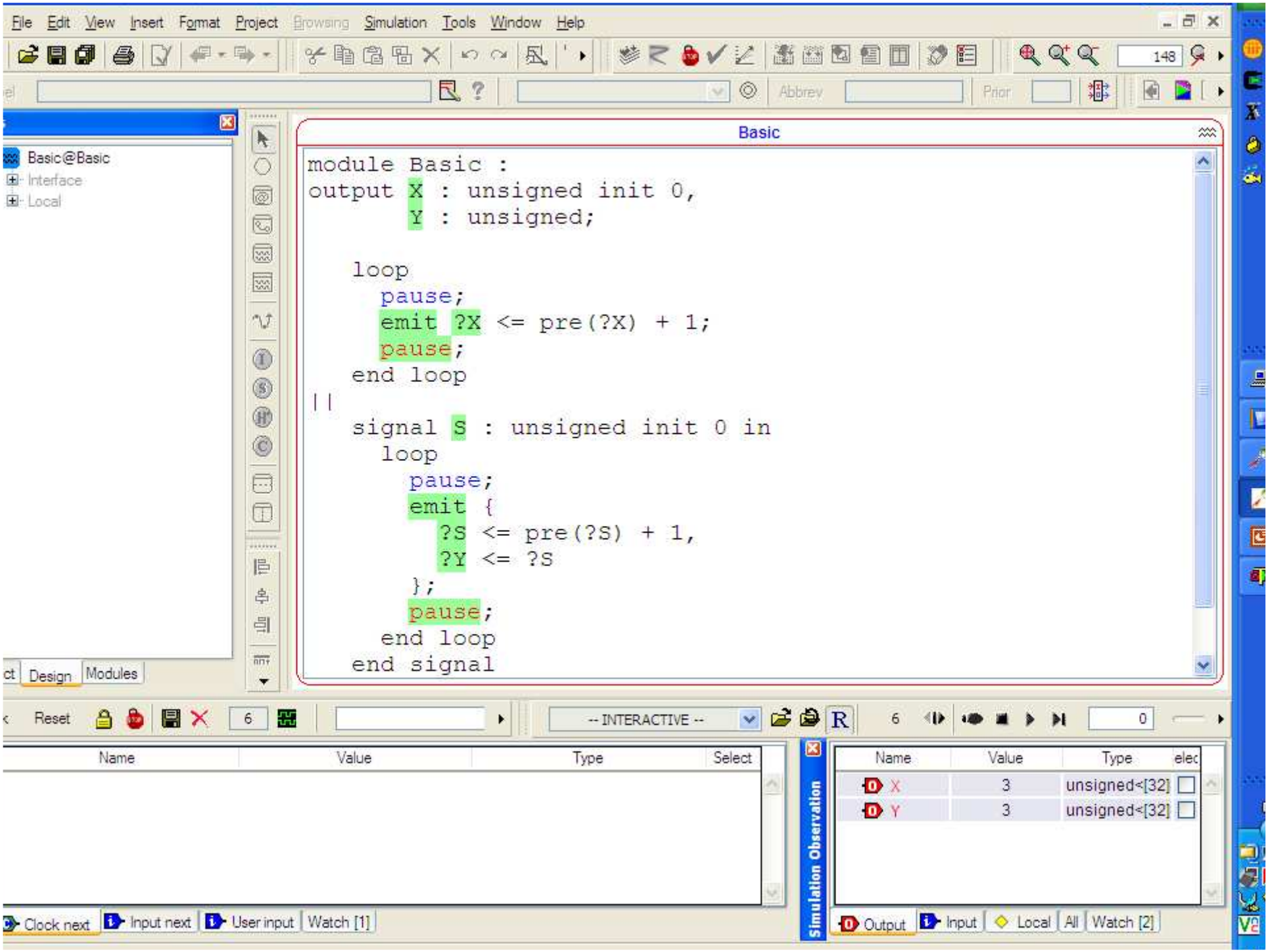
  Synopsys power compiler
- Not really in the RTL model

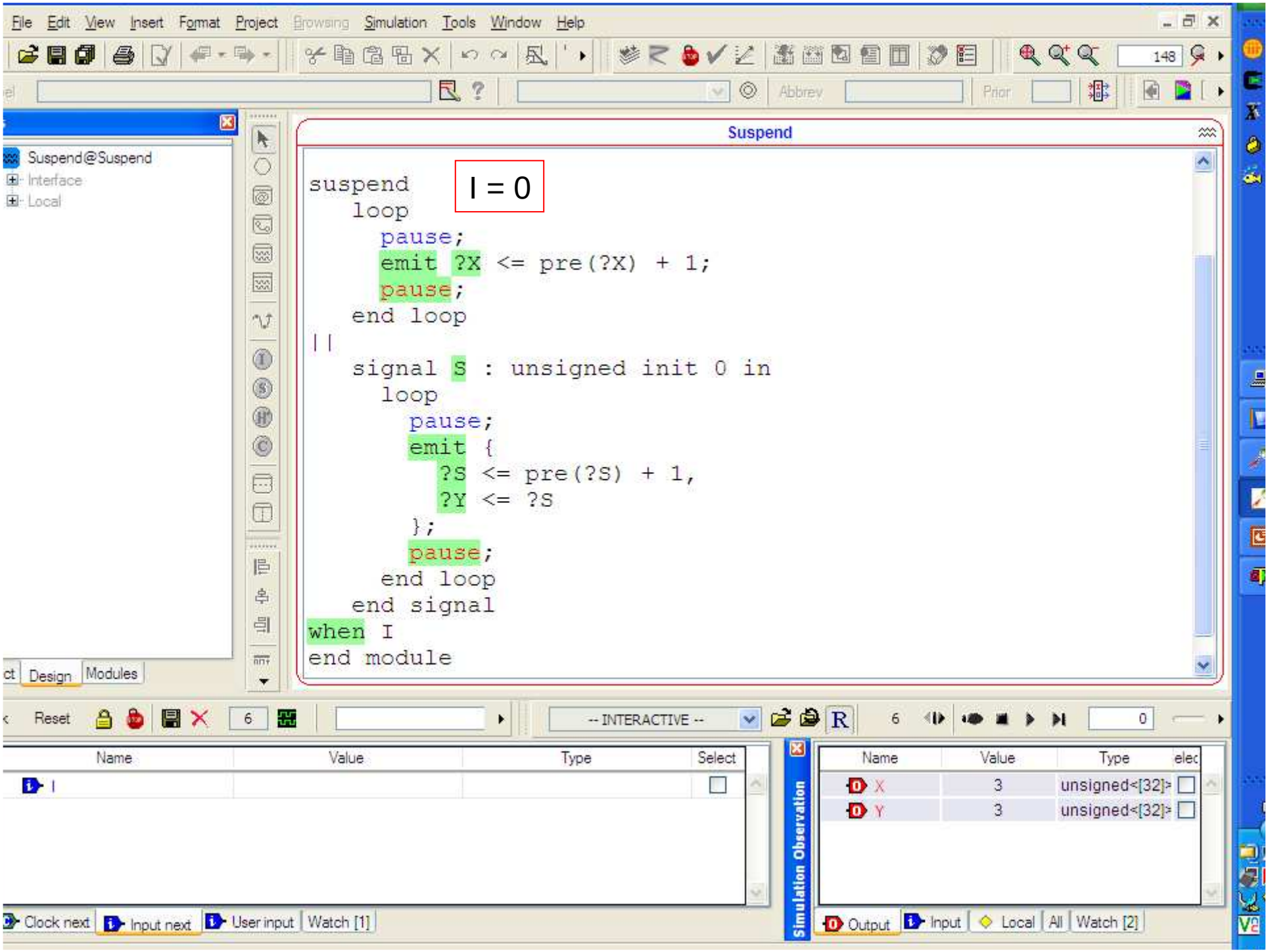
Esterel view

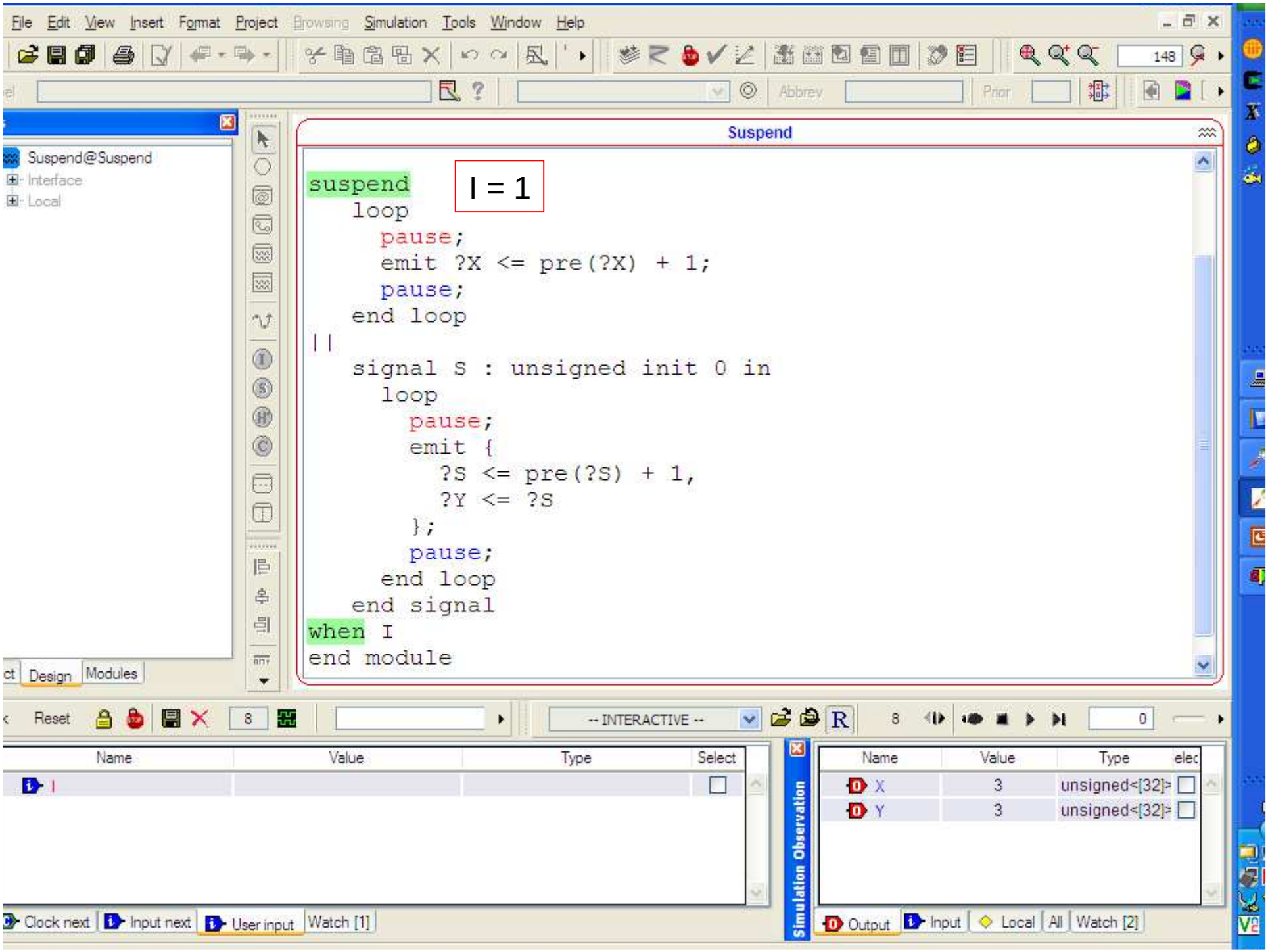
- Results from new weak suspend statement (K. Schneider)

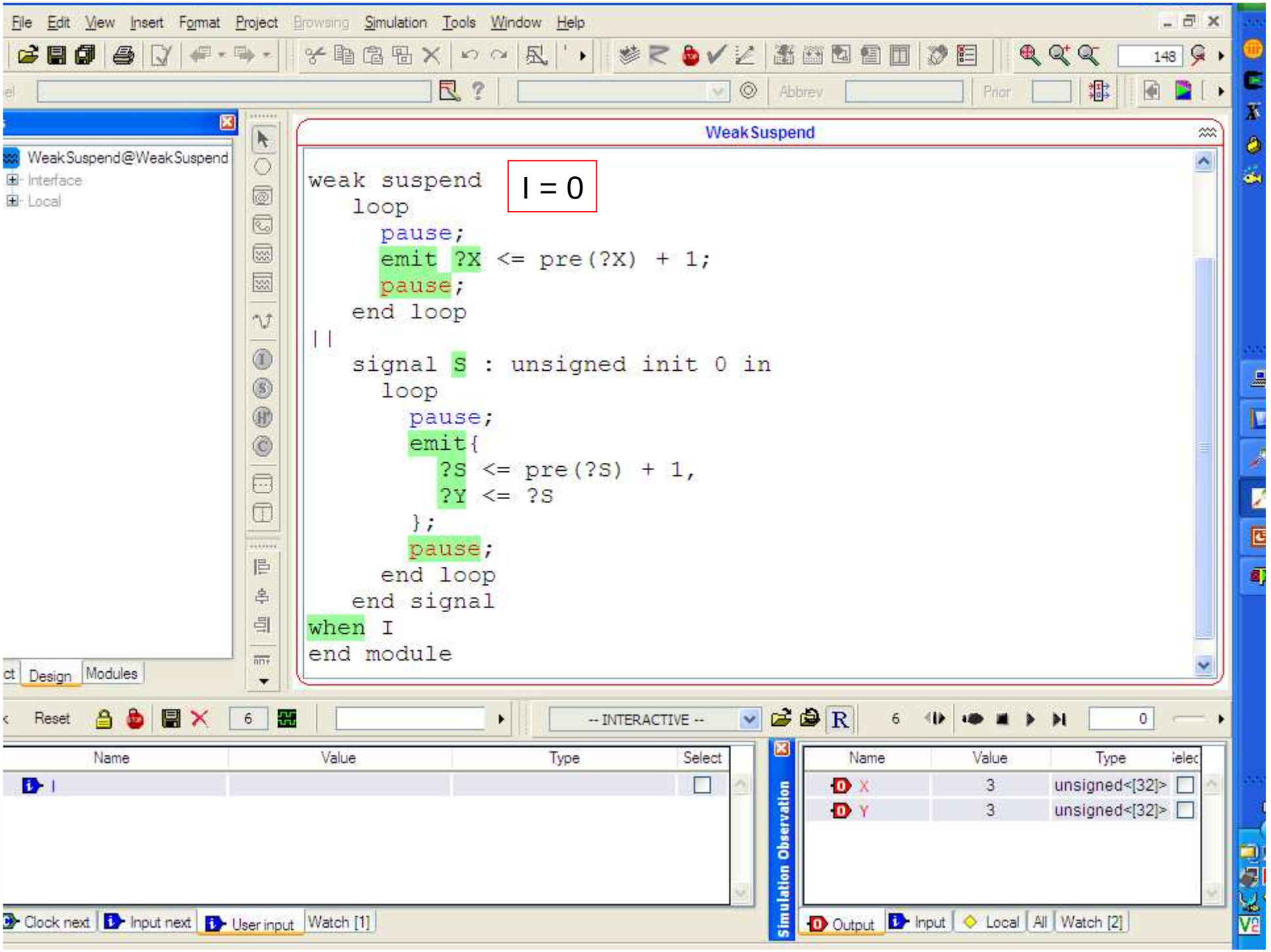
  combinational transition performed, but no state change
- Semantics ok, fits well with scoping

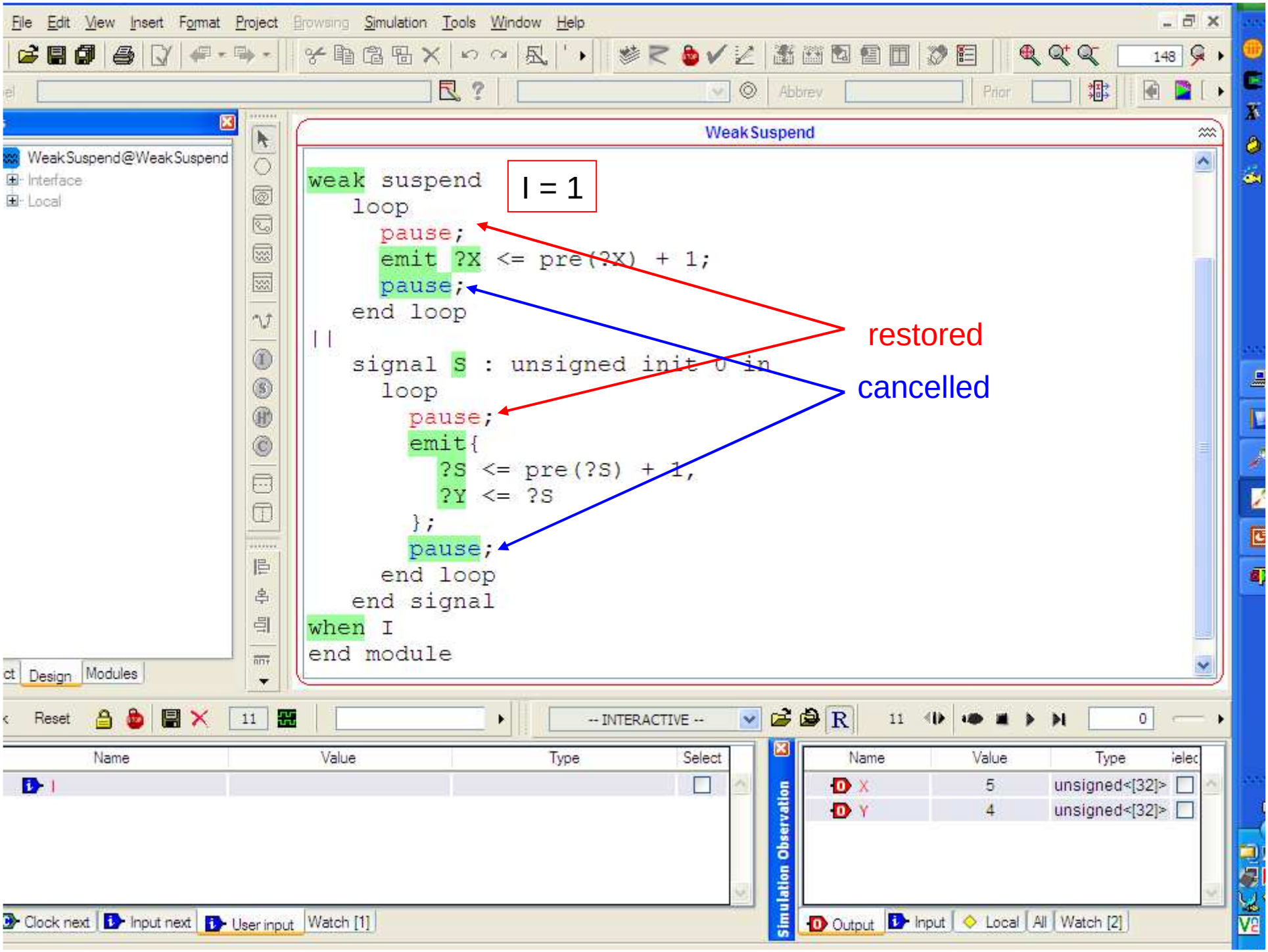
  applies to states of all control and objects declared inside
- Synthesis to actual clock gating or to enabling logic


File Edit View Insert Format Project Browsing Simulation Tools Window Help
Basic@Basic
Interface
Local
Design
Modules
Basic
module Basic :
output X : unsigned init 0,
Y : unsigned;
loop
pause;
emit ?X <= pre(?X) + 1;
pause;
end loop
||
signal S : unsigned init 0 in
loop
pause;
emit {
?S <= pre(?S) + 1,
?Y <= ?S
};
pause;
end loop
end signal
Reset
-- INTERACTIVE --
Name
Value
Type
Select
Clock next
Input next
User input
Watch [1]
Simulation Observation
Name
Value
Type
X
3
unsigned<[32]
Y
3
unsigned<[32]
Output
Input
Local
All
Watch [2]

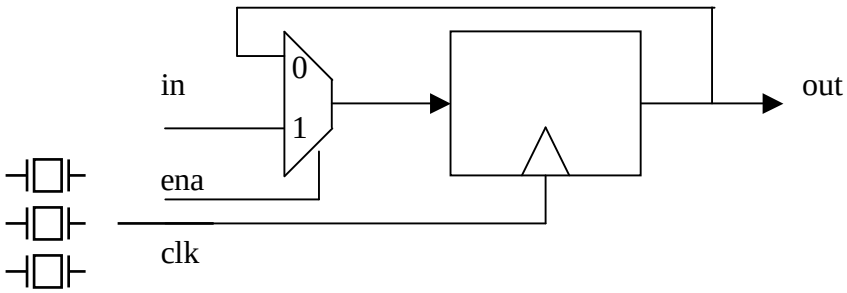

File Edit View Insert Format Project Browsing Simulation Tools Window Help
Suspend@Suspend
Interface
Local
Design
Modules
Suspend
suspend
I = 0
loop
pause;
emit ?X <= pre(?X) + 1;
pause;
end loop
||
signal S : unsigned init 0 in
loop
pause;
emit {
?S <= pre(?S) + 1,
?Y <= ?S
};
pause;
end loop
end signal
when I
end module
Reset
-- INTERACTIVE --
Name
Value
Type
Select
I
Clock next
Input next
User input
Watch [1]
Simulation Observation
Name
Value
Type
X
3
unsigned<[32]>
Y
3
unsigned<[32]>
Output
Input
Local
All
Watch [2]

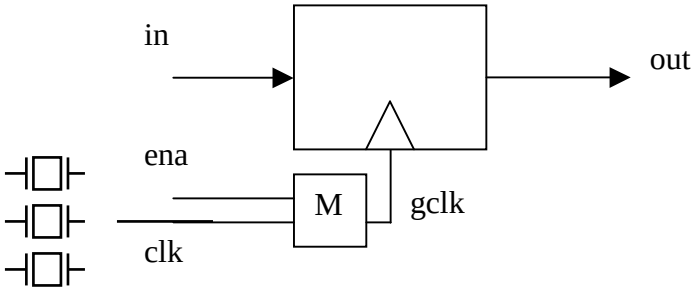

File Edit View Insert Format Project Browsing Simulation Tools Window Help
Suspend@Suspend
Interface
Local
Design
Modules
Suspend
I = 1
suspend
loop
pause;
emit ?X <= pre(?X) + 1;
pause;
end loop
||
signal S : unsigned init 0 in
loop
pause;
emit {
?S <= pre(?S) + 1,
?Y <= ?S
};
pause;
end loop
end signal
when I
end module
Reset
-- INTERACTIVE --
Name
Value
Type
Select
I
Clock next
Input next
User input
Watch [1]
Simulation Observation
Name
Value
Type
X
3
unsigned<[32]>
Y
3
unsigned<[32]>
Output
Input
Local
All
Watch [2]


File Edit View Insert Format Project Browsing Simulation Tools Window Help
WeakSuspend@WeakSuspend
Interface
Local
Design
Modules
Weak Suspend
I = 0
weak suspend
loop
pause;
emit ?X <= pre(?X) + 1;
pause;
end loop
||
signal S : unsigned init 0 in
loop
pause;
emit{
?S <= pre(?S) + 1,
?Y <= ?S
};
pause;
end loop
end signal
when I
end module
Reset
-- INTERACTIVE --
Name
Value
Type
Select
I
Clock next
Input next
User input
Watch [1]
Simulation Observation
Name
Value
Type
X
3
unsigned<[32]>
Y
3
unsigned<[32]>
Output
Input
Local
All
Watch [2]


File Edit View Insert Format Project Browsing Simulation Tools Window Help
Weak Suspend
weak suspend
loop
pause;
emit ?X <= pre(?X) + 1;
pause;
end loop
||
signal S : unsigned init 0 in
loop
pause;
emit{
?S <= pre(?S) + 1,
?Y <= ?S
};
pause;
end loop
end signal
when I
end module
I = 1
restored
cancelled
WeakSuspend@WeakSuspend
Interface
Local
Design
Modules
Reset
-- INTERACTIVE --
Name
Value
Type
Select
I
X
5
unsigned<[32]>
Y
4
unsigned<[32]>
Simulation Observation
Clock next
Input next
User input
Watch [1]
Output
Input
Local
All
Watch [2]

Weak Suspend Implementation





enabling logic

FGGA, software, formal verification

clock gating

ASIC

esterelv7 compiler option

Multiclock design

```
multiclock Multi :
   input clock C1, C2;
   run M1 [clock C1]
||
   run M2 [ clock C2 ]
end multiclock;
```

Monoclok semantics (clocks as signals)

```
module Multi :
   input C1, C2;
   weak suspend
      run M1
   when immediate (not C1)
||
   weak suspend
      run M2
   when immediate (not C2)
end multiclock;
```