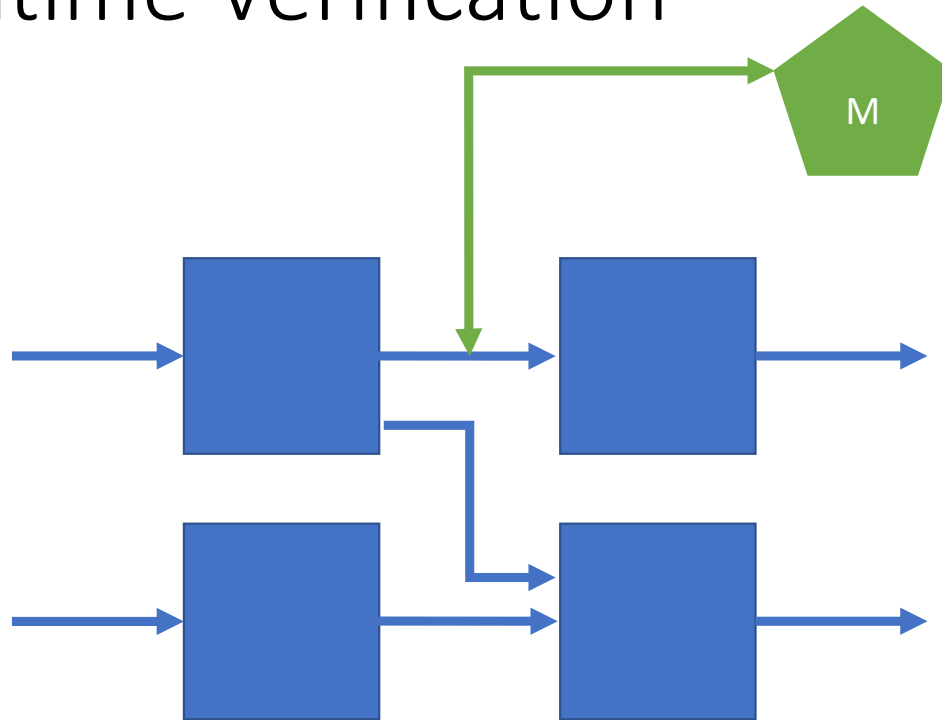


Hardware-based runtime verification with Tesla

Martin Leucker
University of Lübeck

Joint work with Normann Decker, Cesar Sanchez, Torben Scheffel, Malte Schmitz, Daniel Thoma et al.

Runtime Verification



- Monitor analyzes the execution of the system
- Synthesized from high-level specification
- Has to *see* the execution
- Used for finding bugs

Aging Bugs – Mandelbugs – Heisenbugs

- Some bugs only occur
 - after a long execution time
 - under “weird” circumstances

Observation changes the System
Heisenberg Uncertainty Principle

See what’s going on?

- Program annotation?
- Program annotation changes timing

Change of the underlying system

- Run what you test and test what you run

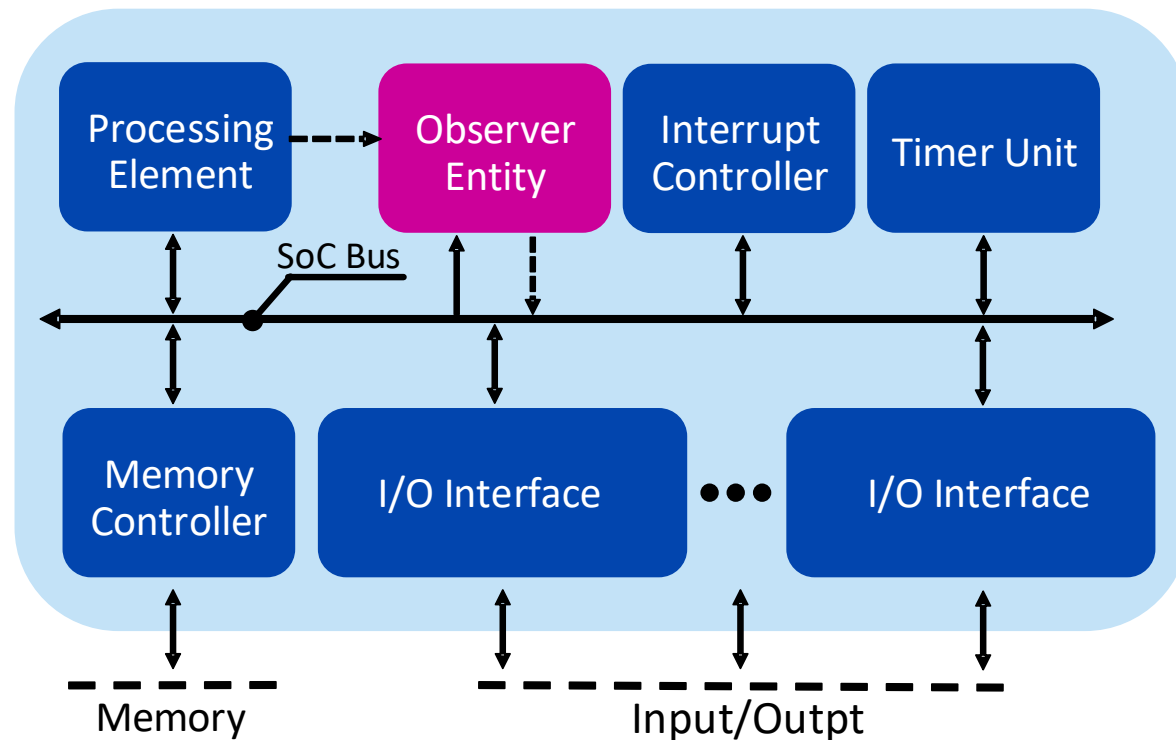
Code annotation – Program cooperates

Original source code	Instrumented source code
<pre>void foo() { bool found=false; for (int i=0; (i<100) && (!found); ++i) { if (i==50) break; if (i==20) found=true; if (i==30) found=true; } printf("foo\n"); }</pre>	<pre>char inst[15]; void foo() { bool found=false; for (int i=0; ((i<100)?inst[0]=1:inst[1]=1,0) && ((!found)?inst[2]=1:inst[3]=1,0); ++i) { if ((i==50?inst[4]=1:inst[5]=1,0)) { inst[6]=1; break; } if ((i==20?inst[7]=1:inst[8]=1,0)) { inst[9]=1; found=true; } if ((i==30?inst[10]=1:inst[11]=1,0)) { inst[12]=1; found=true; } inst[13]=1; } printf("foo\n"); inst[14]=1; }</pre>

- Slowdown typically not acceptable for production code
- Unless done in a clever way? Printf??
- Here: Use additional hardware resources instead

Hardware-based Runtime Verification

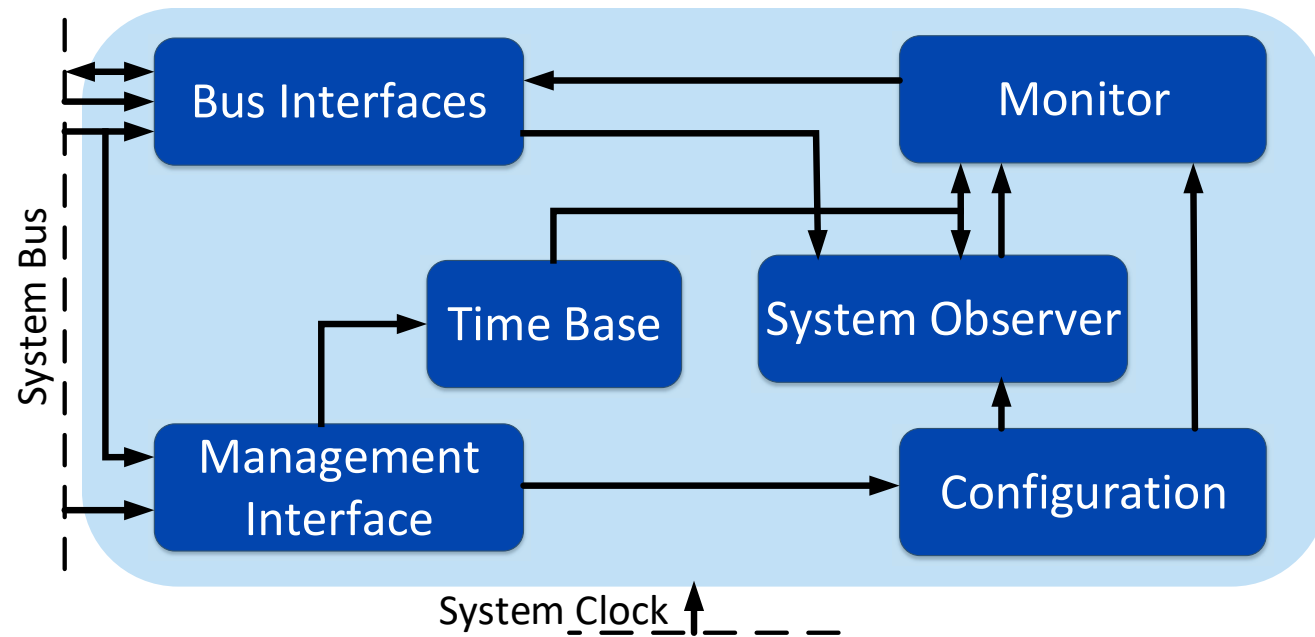
SoC approach – Architecture



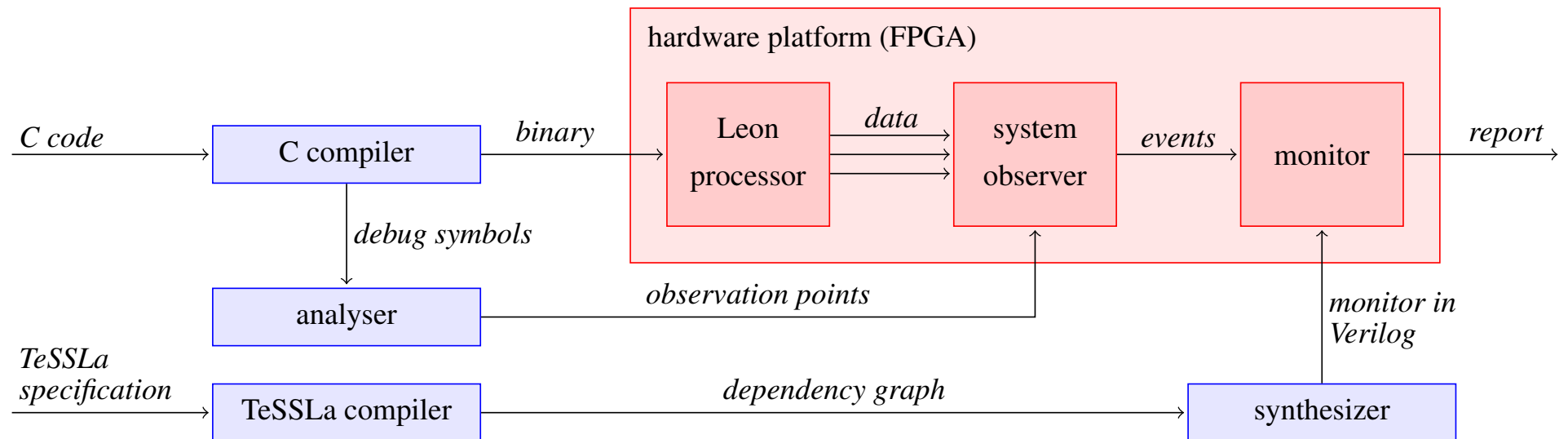
Non-intrusive Runtime Verification within a System-on-Chip, RUME 2018

José Rufino, António Casimiro, Felix Dino Lange, Martin Leucker, Torben Scheffel, Malte Schmitz, Daniel Thoma

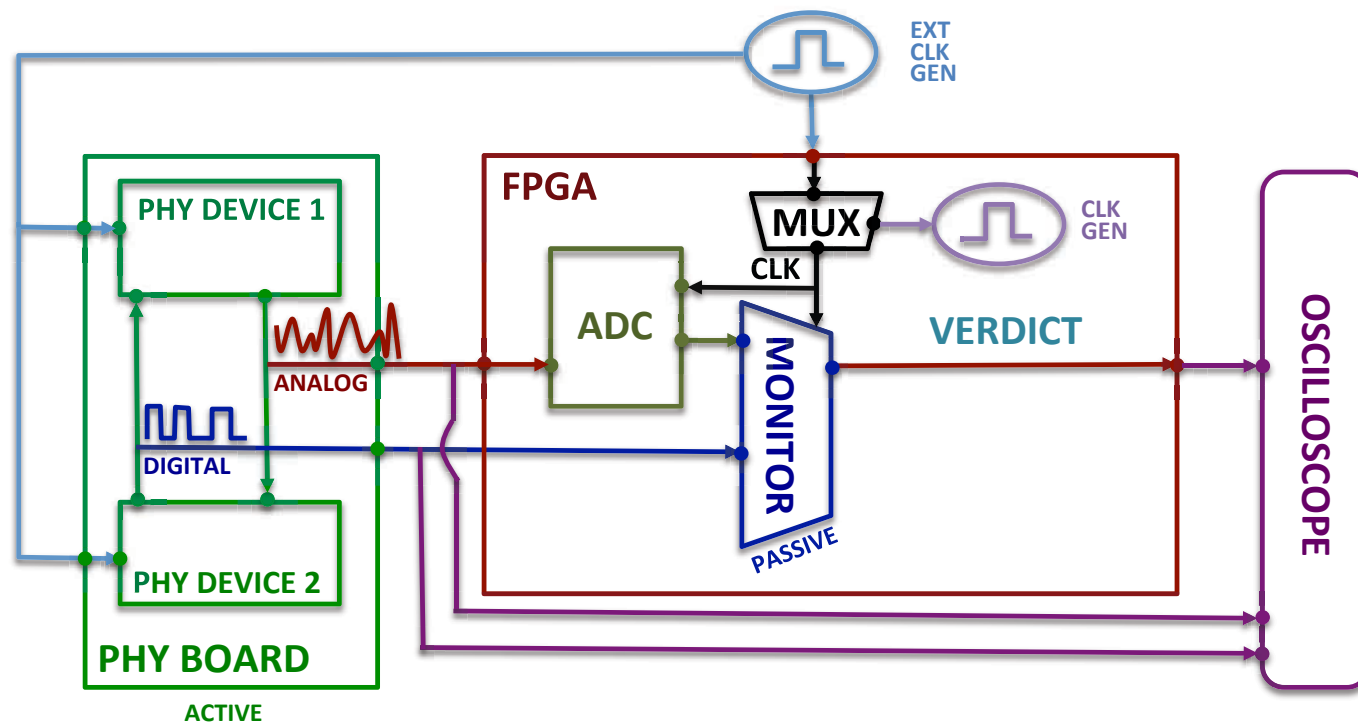
SoC approach – Observer Entity



SoC approach with TeSSLa specifications

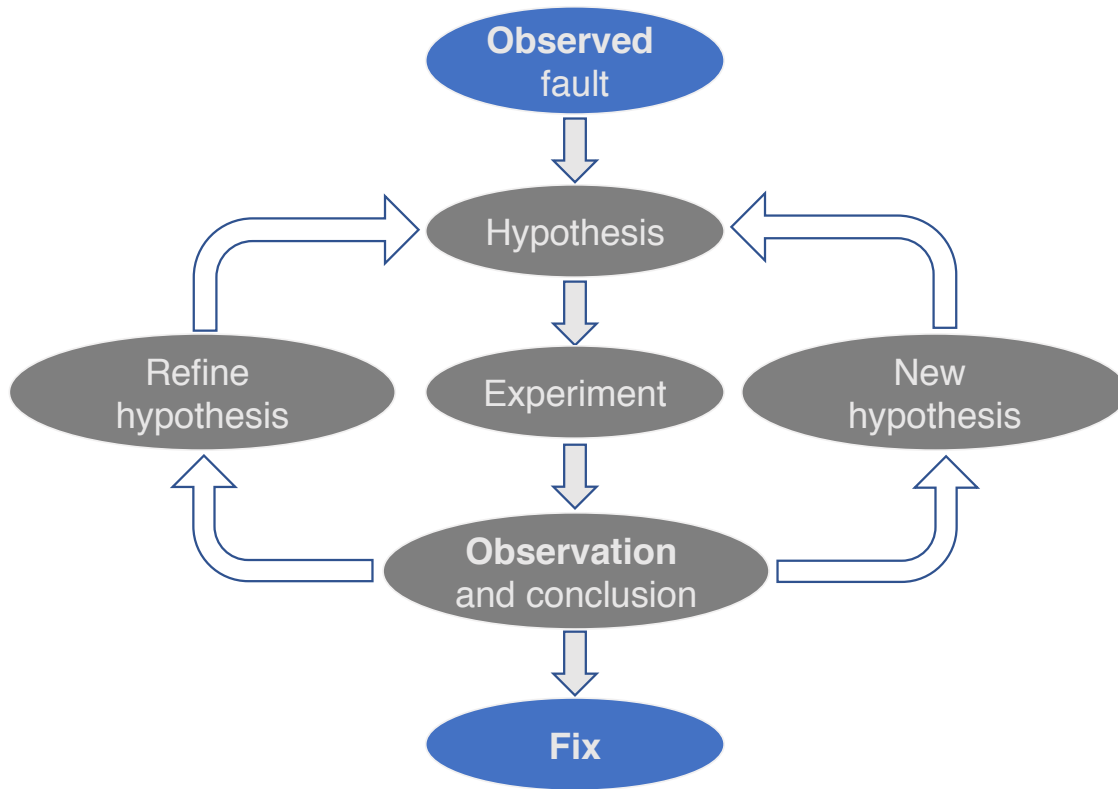


Fixed Monitors for RV



From: Ezio Bartocci, Yliès Falcone, Adrian Francalanza, Giles Reger. Introduction to Runtime Verification. Lectures on Runtime Verification. Introductory and Advanced Topics, 10457, Springer, pp.1-33, 2018, Lecture Notes in Computer Science

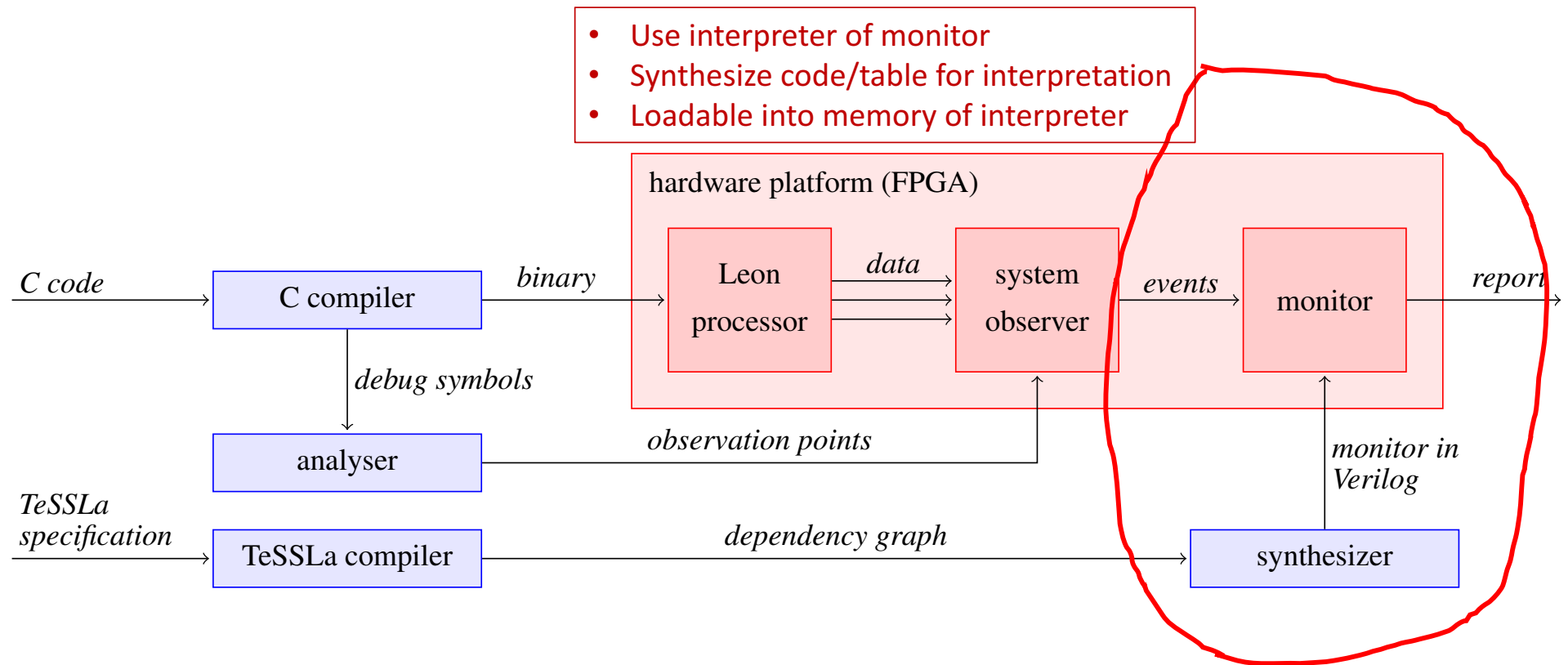
Debugging



Requirements

- Quick loop for synthesizing new properties on the test system
- Still long-term observability useful
- Change monitoring focus dependent on previous outcome

SoC approach with TeSSLa specifications



[Rico Backasch](#), [Christian Hochberger](#), [Alexander Weiss](#), Martin Leucker, [Richard Lasslop](#):

Runtime verification for multicore SoC with high-quality trace data. *ACM Trans. Design Autom. Electr. Syst.* 18(2): 18:1-18:26 (2013)

The COEMS approach

Continuous Online Observation for Embedded Multi-core Systems
EU Horizon 2020 project

The COEMS Consortium



UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

University of Lübeck



Accemic Technologies



Thales Romania

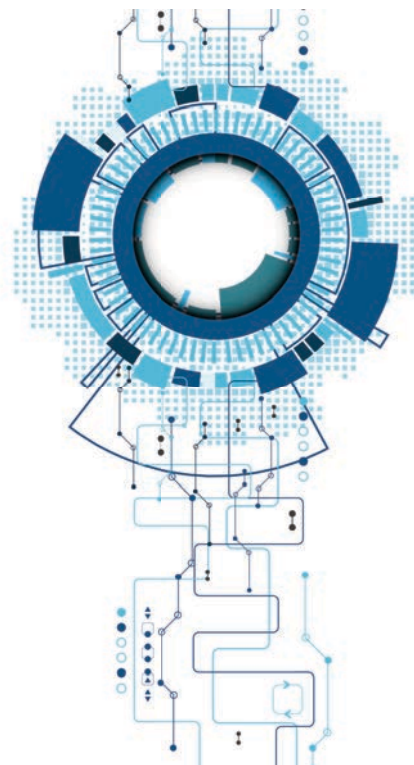
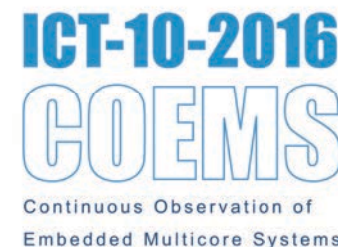
Thales Austria



Høgskulen på Vestlandet / Western Norway
University of Applied Science



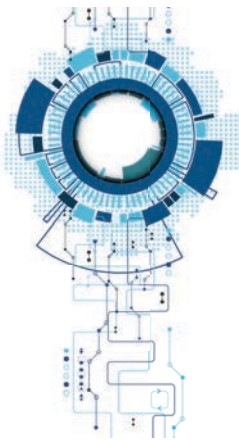
Airbus



This project has received funding from the European
Union's Horizon 2020 research and innovation programme
under grant agreement no. 732016.

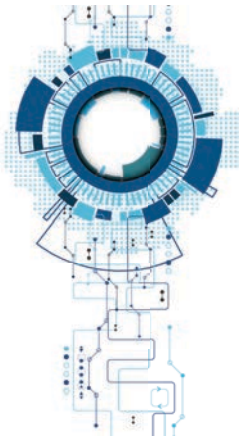
Objectives

- Increase test efficiency
 - Increase debug efficiency
 - Increase test effectivity
 - Improve embedded systems performance
-
- For embedded multicore systems (ARM etc.)
 - Running (multithreaded) C programs
 - Perhaps on Linux



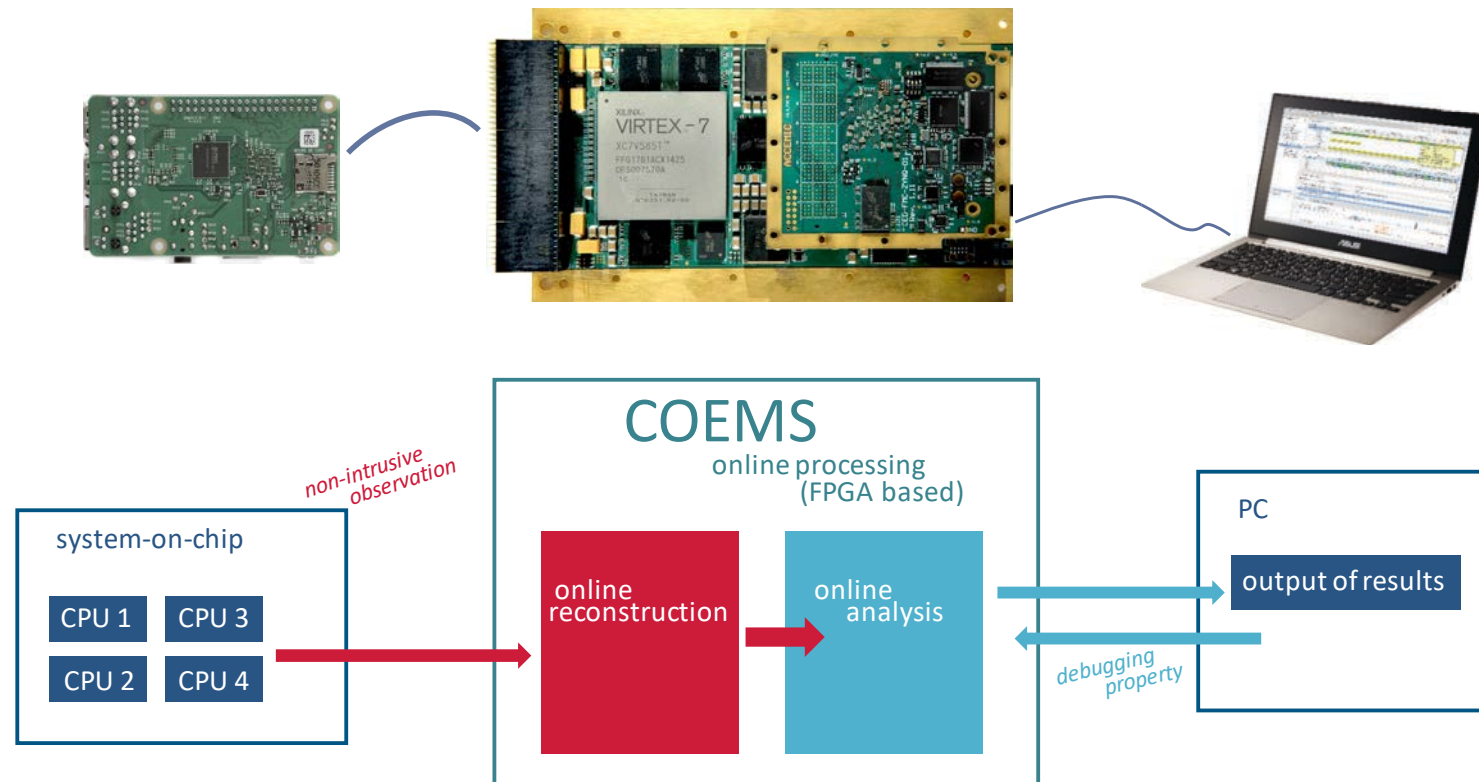
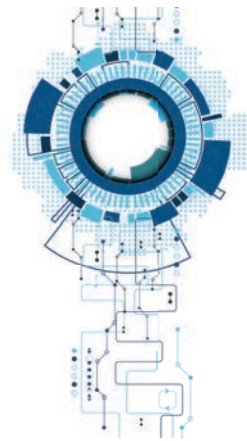
Applications – Semi-Formal Verification

- Finding Data Races
- Finding Timing Bugs
- Finding Functional Bugs
- Measuring Coverage
- Measurement of Worst-Case Execution Time
and Worst-Case Response Time

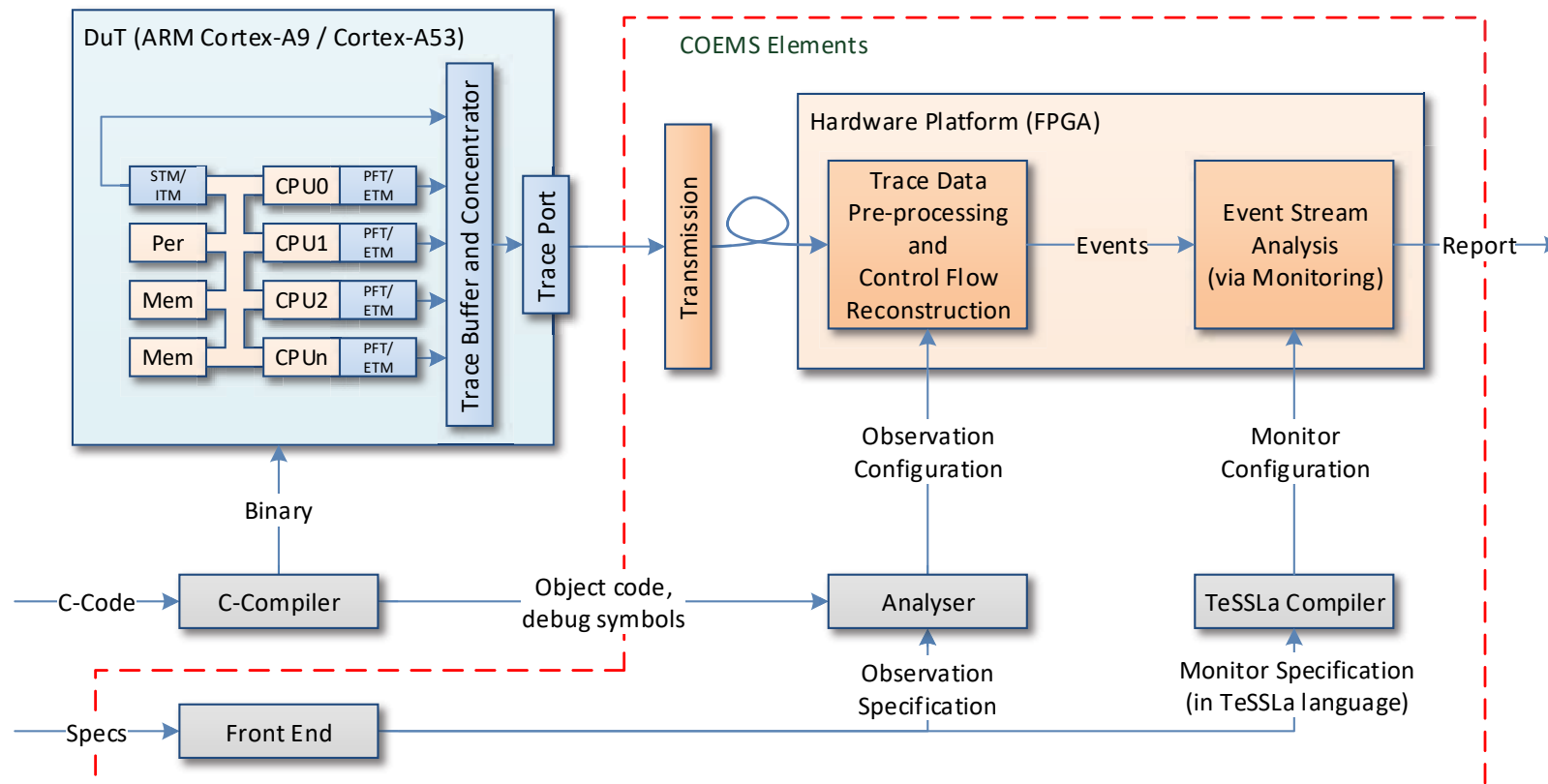
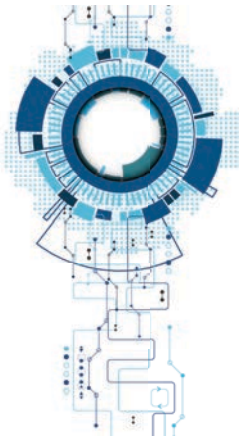


COEMS set-up

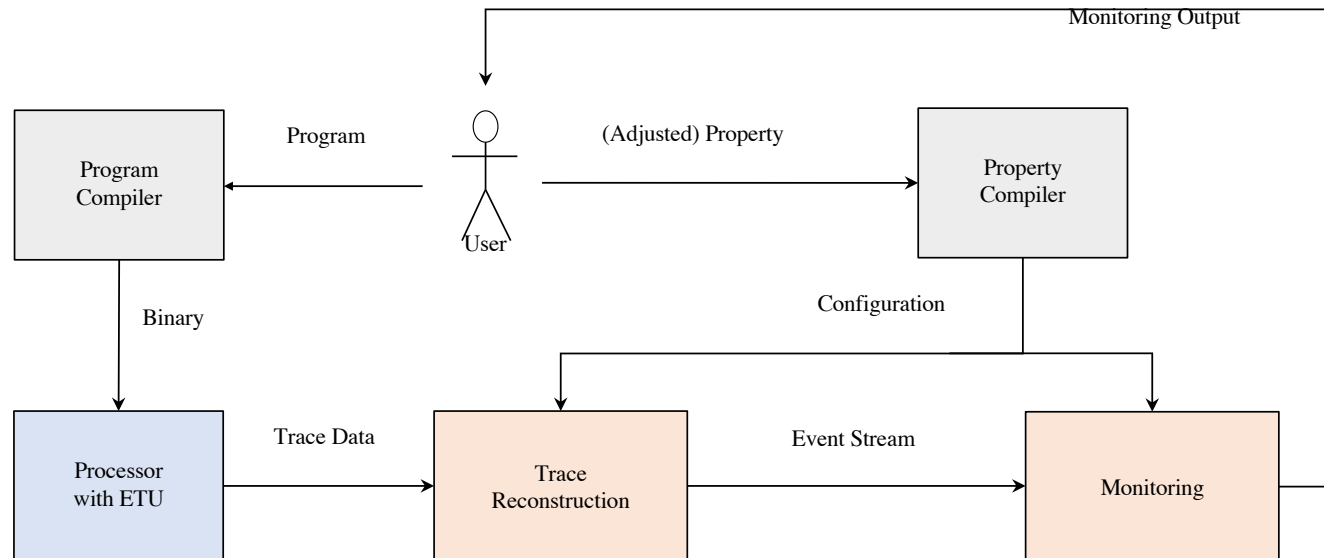
ICT-10-2016
COEMS
Continuous Observation of
Embedded Multicore Systems



System Overview



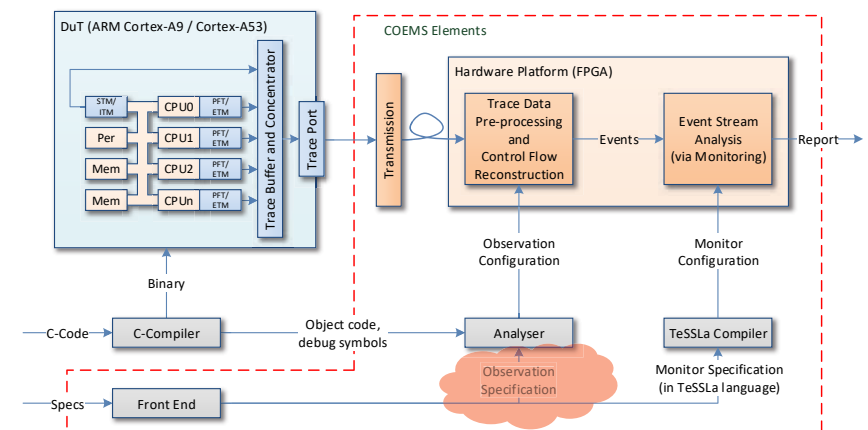
Rapidly adjustable Embedded Trace Online Monitoring – RETOM



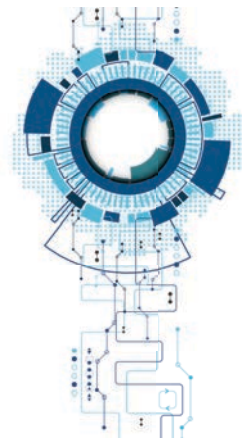
Observation Specification

Specification of atomic artefacts to be observed

- Elements of Source Code
 - Program Line
 - Entering / Leaving a Function / Exception
 - Reading / Writing variables
- Elements of the Binary
 - PC Address
 - Calls / Returns
 - Specific Operations (e.g. Floating Point Operations)
- Hardware Supported Instrumentation
 - ITM, STM



Observation Specification



C Code

```

1 int main() {
2     int sum = 0;
3     for (int i = 0; i < 5; i++) {
4         sum = add(sum, sub(i,2));
5     }
6     sum = add(sum, add(21,21));
7     for (int i = 0; i < 5; i++) {
8         sum = add(sum, sub(i,2));
9     }
10 }
11
12 int add(int x, int y) {
13     return x+y;
14 }
15
16 int sub(int x, int y) {
17     return x-y;
18 }
19

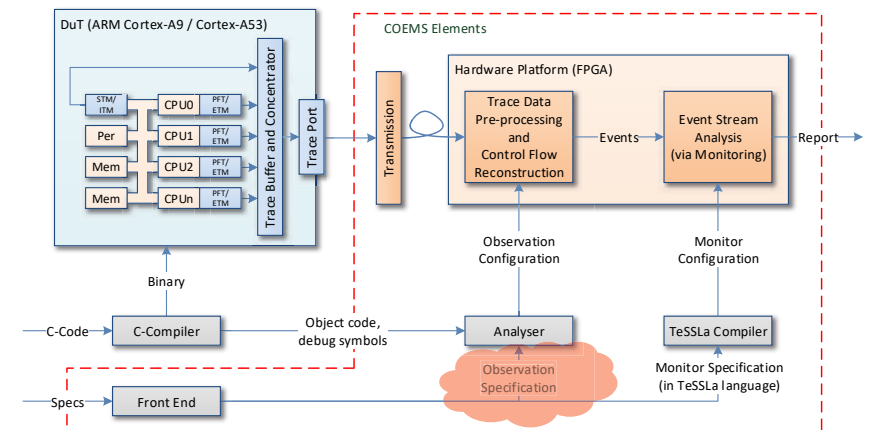
```

Specification

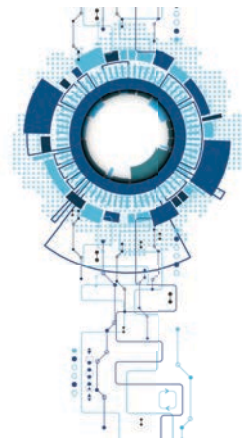
```

1 def add_event := function_call("add")
2 def add_count := eventCount(add_event)
3
4 def sub_event := function_call("sub")
5 def sub_count := eventCount(sub_event)
6
7 def diff := add_count - sub_count
8
9 def error := diff >= 2
10
11 out diff
12 out error
13

```



Monitor Specification Language (TeSSLa)



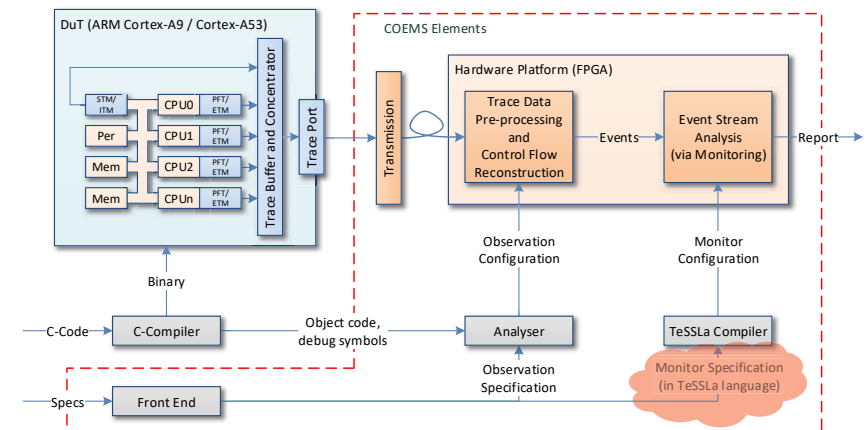
Specification

```

1 def add_event := function_call("add")
2 def add_count := eventCount(add_event)
3
4 def sub_event := function_call("sub")
5 def sub_count := eventCount(sub_event)
6
7 def diff := add_count - sub_count
8
9 def error := diff >= 2
10
11 out diff
12 out error
13

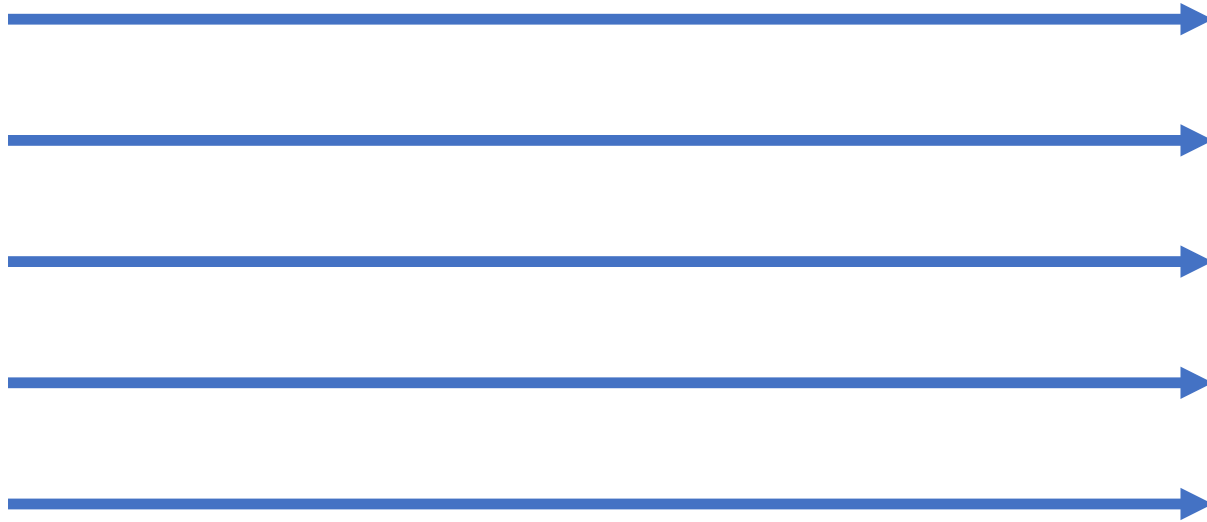
```

- **Event-Stream-Analysis**
Event ordering constraints, timing constraints and quantitative analysis
- **Declarative style**
Describe correctness criterion or analysis goal without having to think about the algorithmic check
- **Modularity**
allowing abstractions based on few primitives
- **Time**
as first-class citizen
- **Events & signals**
based on one common abstraction
- **Finite memory**
allowing execution on FPGA



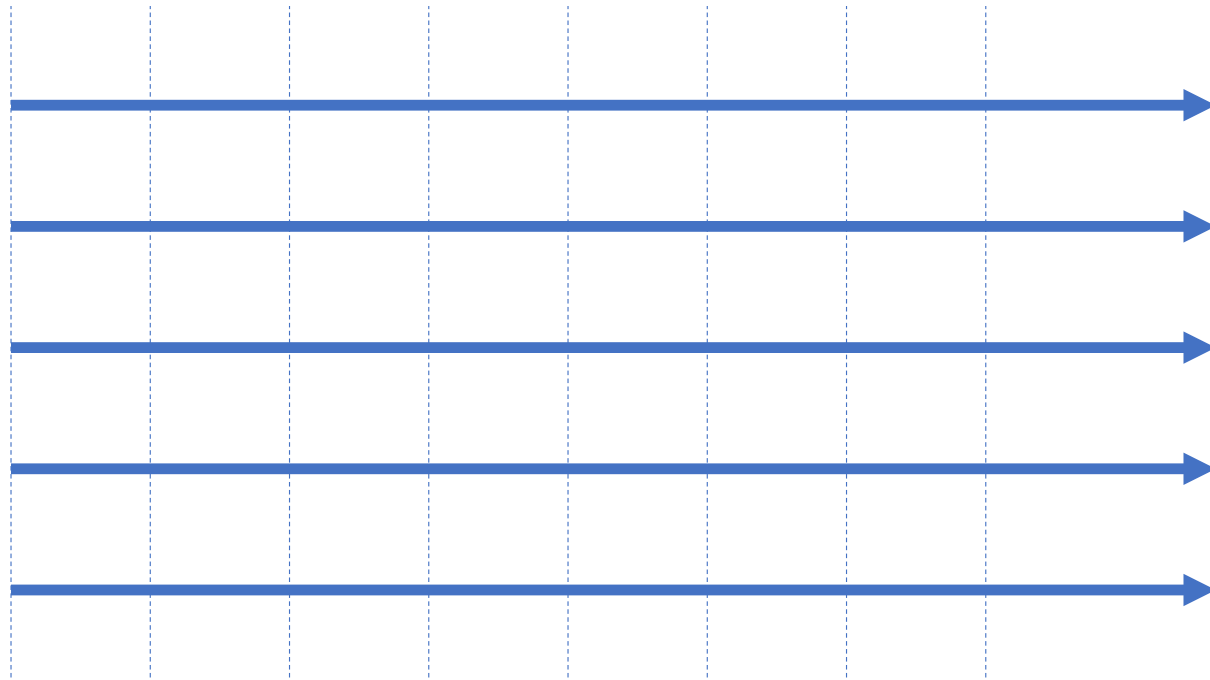
Specification Languages for RV

Streams



Concurrency/Distribution

Streams



Time? Synchrony/Ticks

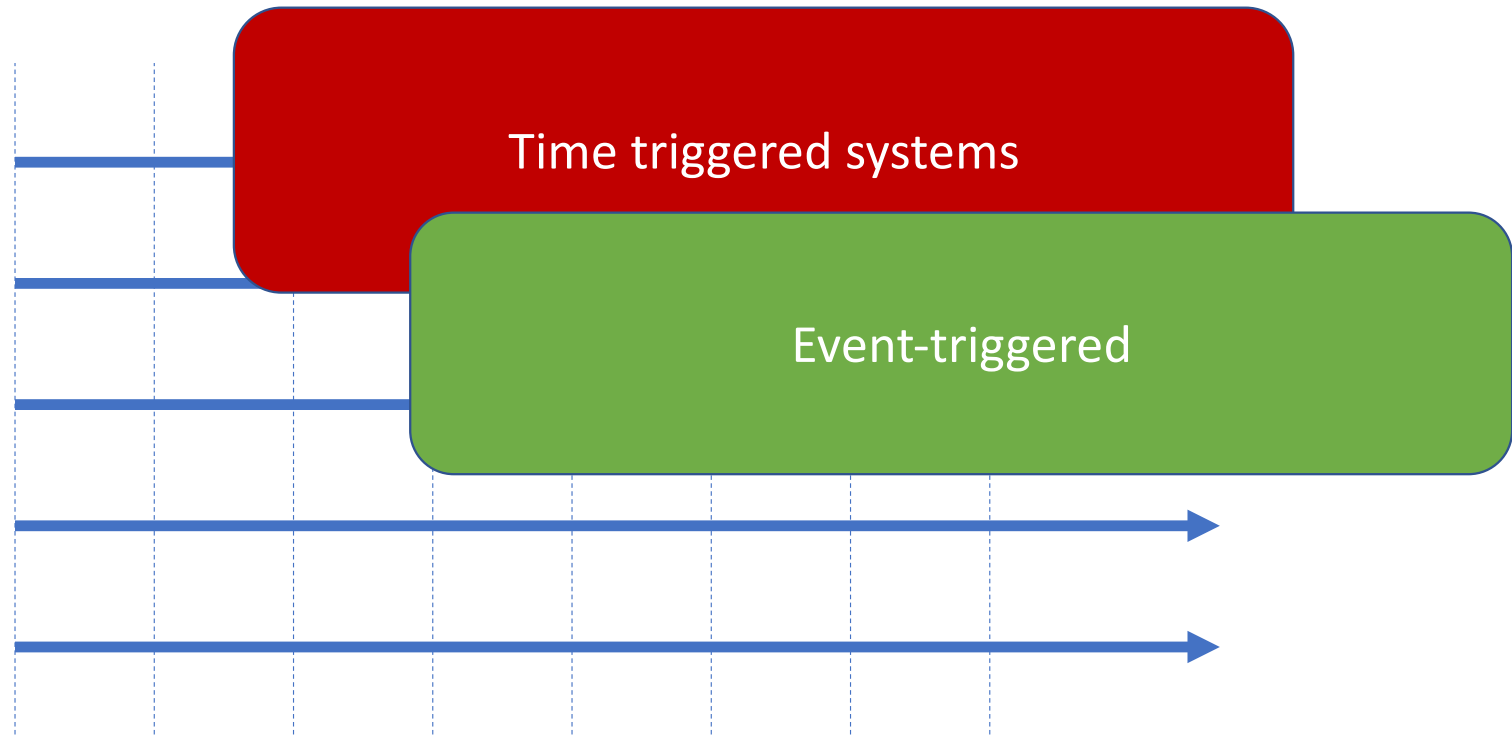
Equational specifications, data, time, concurrency

LOLA

[D'Angelo et al.]

$$\begin{aligned}s_1 &= \mathbf{true} \\s_2 &= t_3 \\s_3 &= t_1 \vee (t_3 \leq 1) \\s_4 &= ((t_3)^2 + 7) \bmod 15 \\s_5 &= \mathbf{ite}(s_3, s_4, s_4 + 1) \\s_6 &= \mathbf{ite}(t_1, t_3 \leq s_4, \neg s_3) \\s_7 &= t_1[+1, \mathbf{false}] \\s_8 &= t_1[-1, \mathbf{true}] \\s_9 &= s_9[-1, 0] + (t_3 \bmod 2) \\s_{10} &= t_2 \vee (t_1 \wedge s_{10}[1, \mathbf{true}])\end{aligned}$$

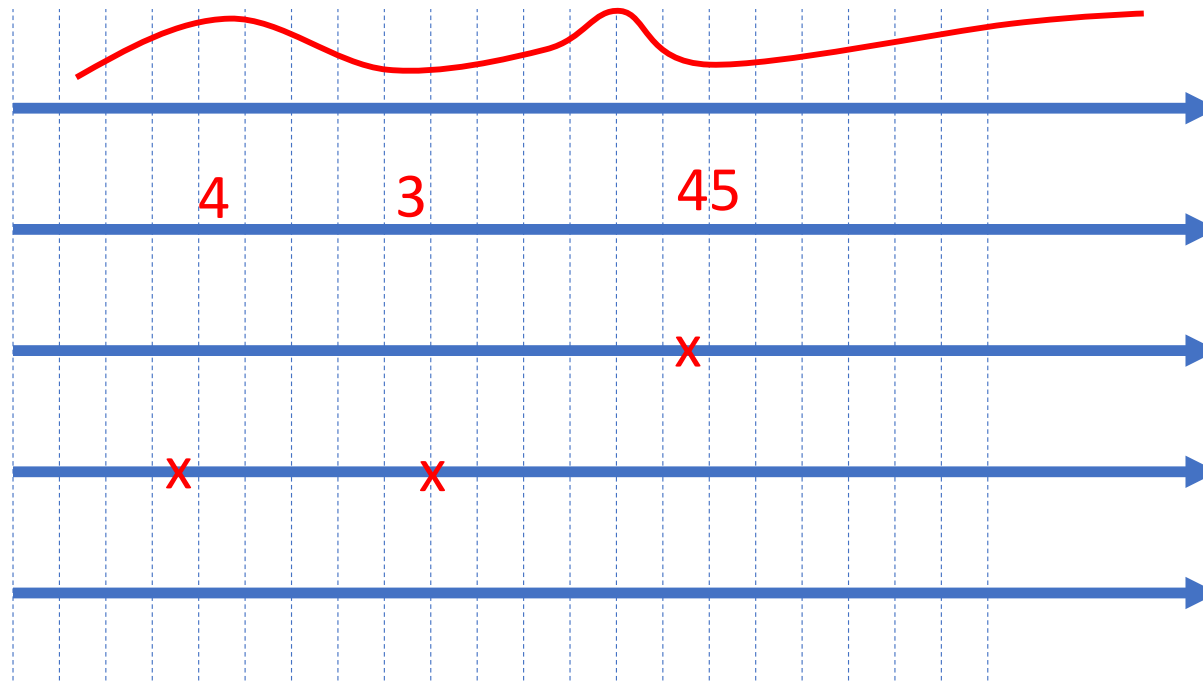
Streams



Time? Synchrony/Ticks

TeSSLa's Streams

Streams



Time? Events

Streams of Programs - After Discretization

Values

e.g., of a program
variable x



Program events

e.g., call to `my_func()`



Streams

Values

e.g., of a program
variable x

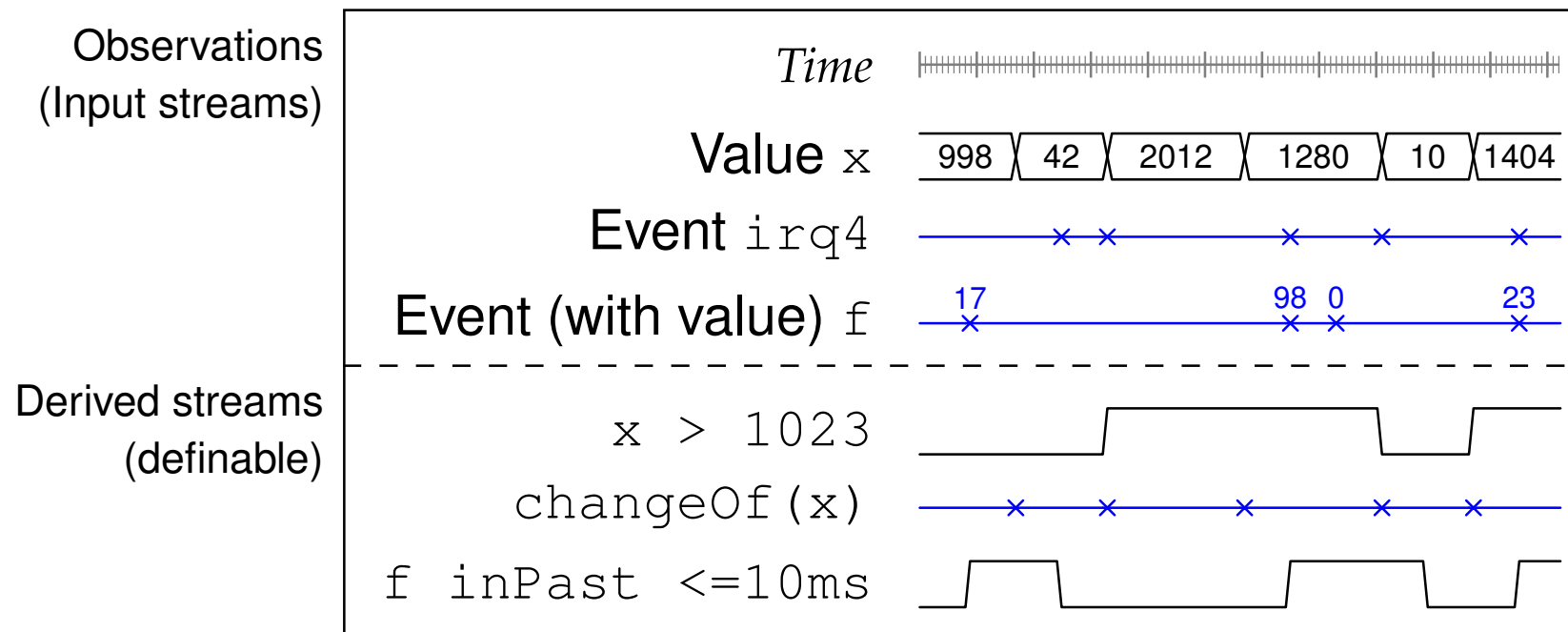


Program events

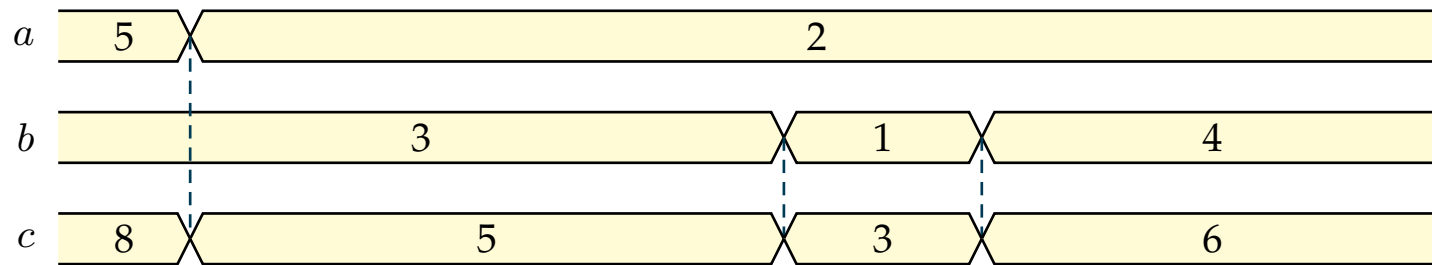
e.g., call to `my_func()`



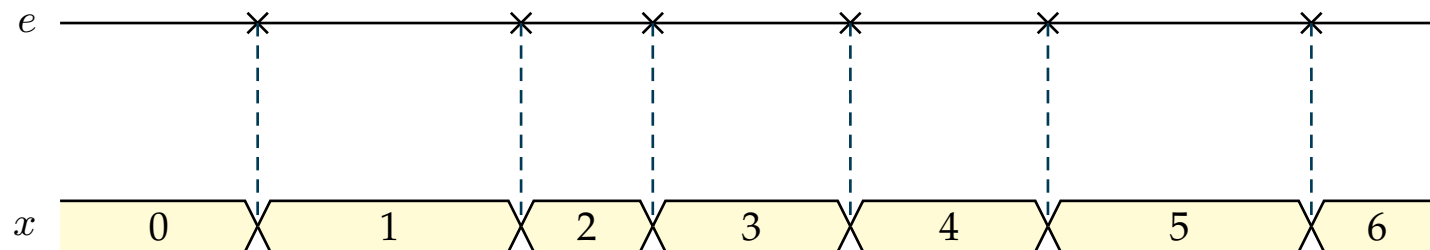
Streams here for RV



TeSSLa by Example

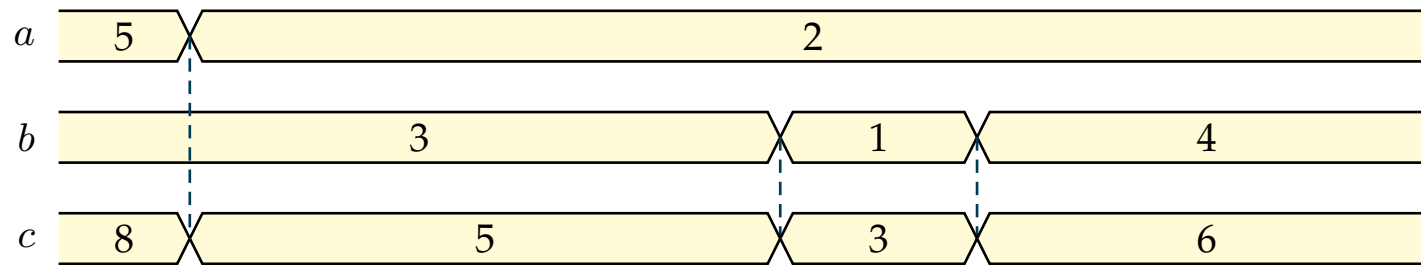


def *c* := *a* + *b*

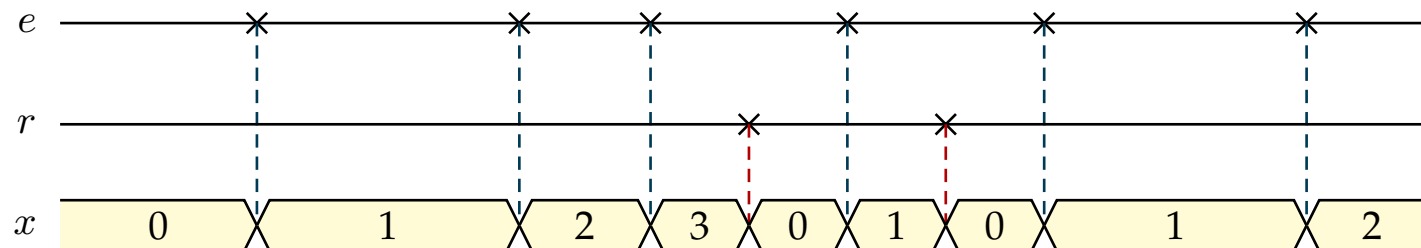


def *x* := **eventCount** (*e*)

TeSSLa by Example

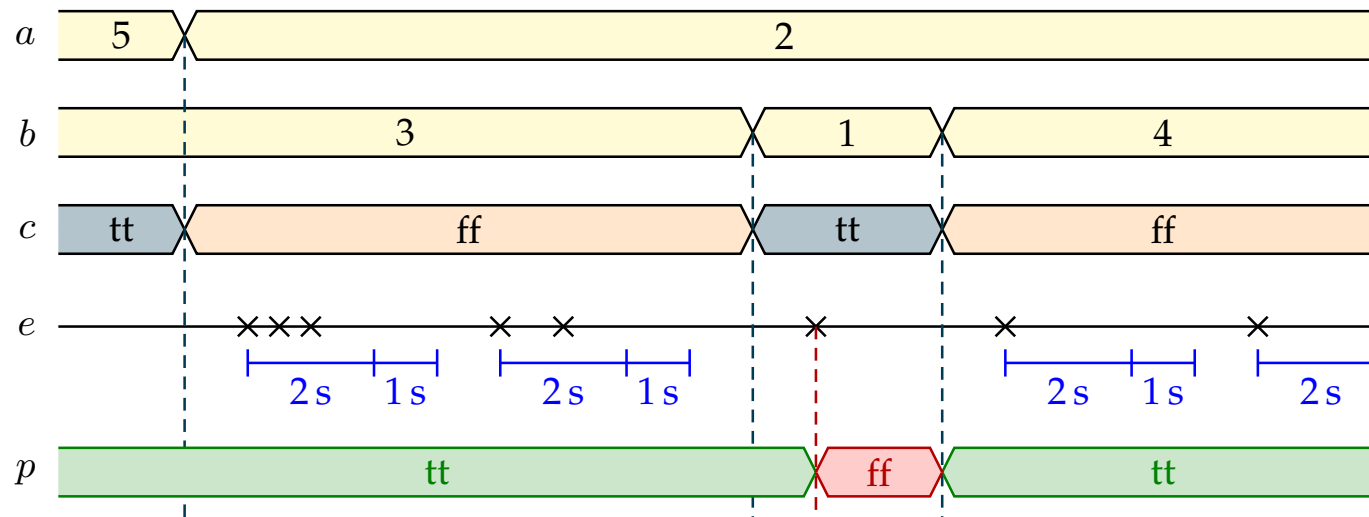


def *c* := *a* + *b*



def *x* := **eventCount** (*e*, reset = *r*)

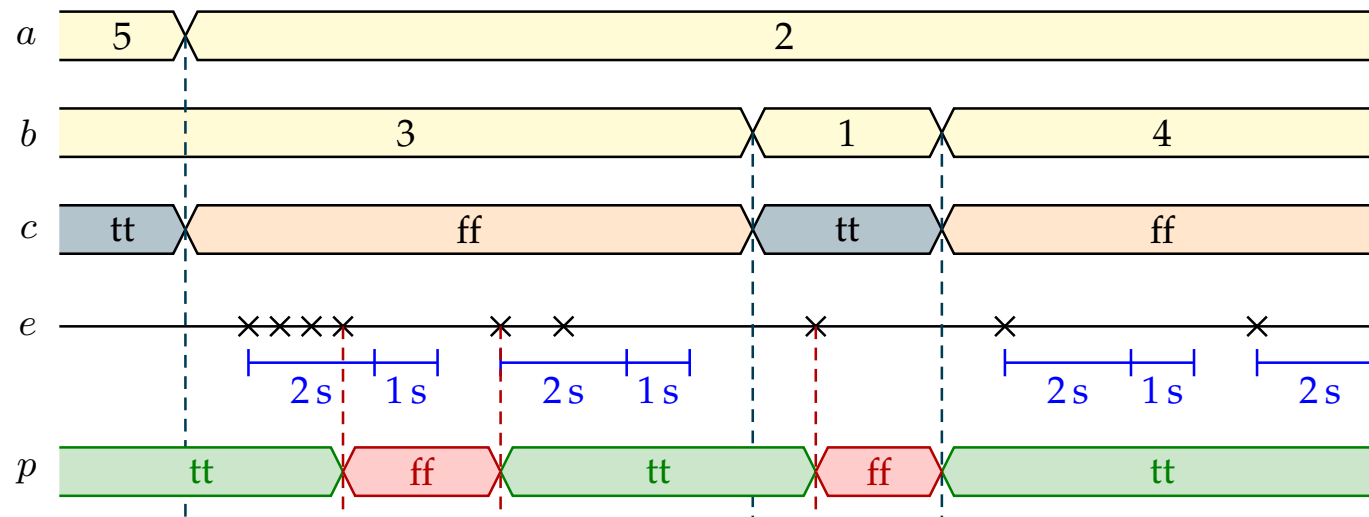
TeSSLa – Burst Pattern (Macros)



```

def c := a > b
def p := if c
    then noEvent(e, since = rising(c))
    else bursts(e, burstLength = 2s,
               waitingPeriod = 1s,
               burstAmount = 3,
               since = falling(c))
    
```

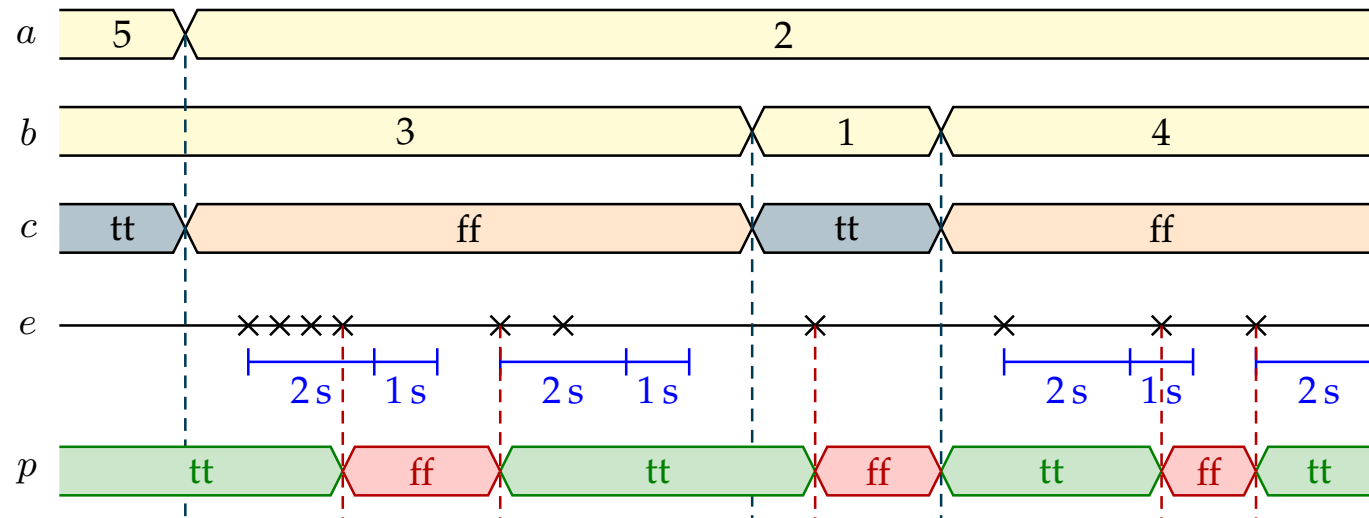
TeSSLa – Burst Pattern (Macros)



```

def c := a > b
def p := if c
    then noEvent(e, since = rising(c))
    else bursts(e, burstLength = 2s,
                waitingPeriod = 1s,
                burstAmount = 3,
                since = falling(c))
    
```

TeSSLa – Burst Pattern (Macros)



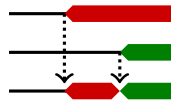
```

def c := a > b
def p := if c
  then noEvent(e, since = rising(c))
  else bursts(e, burstLength = 2s,
    waitingPeriod = 1s,
    burstAmount = 3,
    since = falling(c))
  
```

TeSSLa's operators

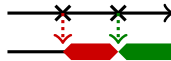
default, defaultFrom

- ▶ Initialize streams
- ▶ Start of recursion



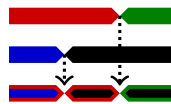
time

- ▶ Get timestamps of stream
- ▶ Replaces data values with timestamps
- ▶ Only way to read timestamps



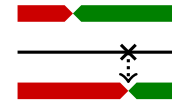
lift

- ▶ Lifts standard functions to streams
- ▶ Used to manipulate data, events, ...



last

- ▶ Refers to previous value of a stream
- ▶ Recursion

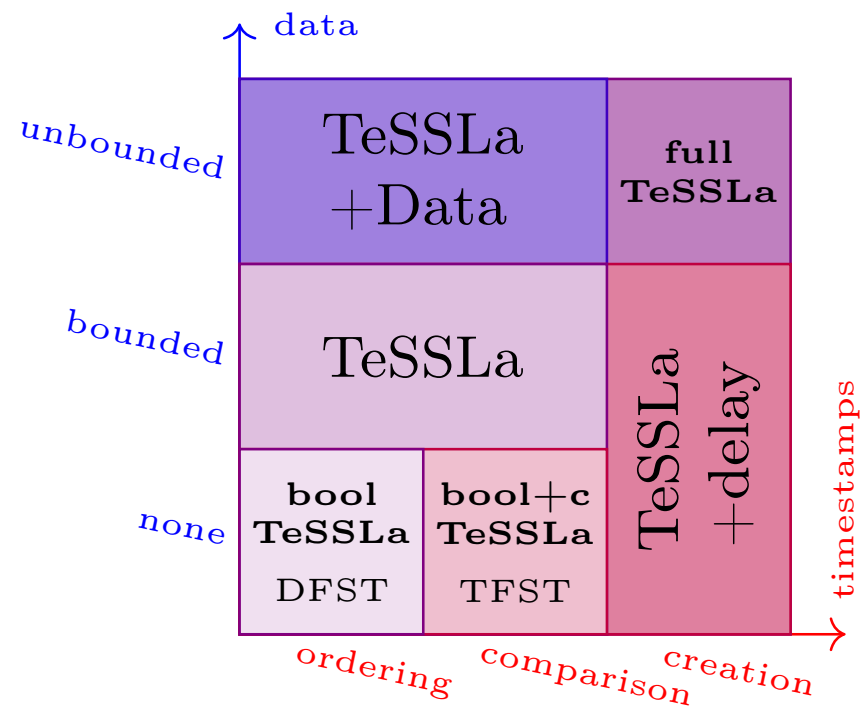


delayedLast

- ▶ Only way to create events
- ▶ Takes a stream and delays events by its current value
- ▶ Output events have the previous value of another given stream



TeSSLa's fragments



EU Horizon 2020 Project: COEMS

The COEMS Consortium



UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES



THALES



AIRBUS

University of Lübeck

Accemic Technologies

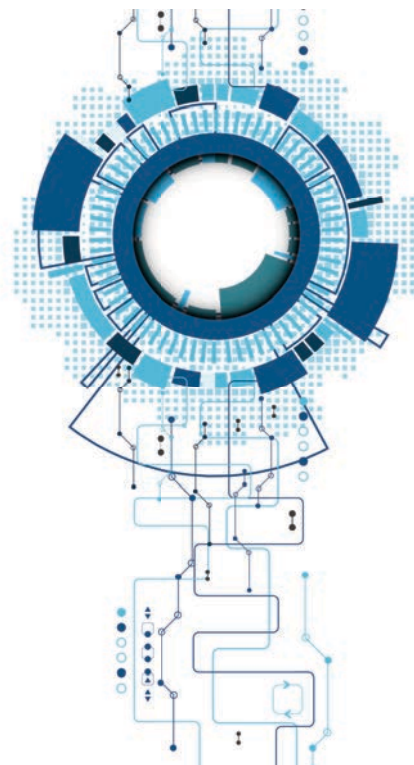
Thales Romania

Thales Austria

Høgskulen på Vestlandet / Western Norway
University of Applied Science

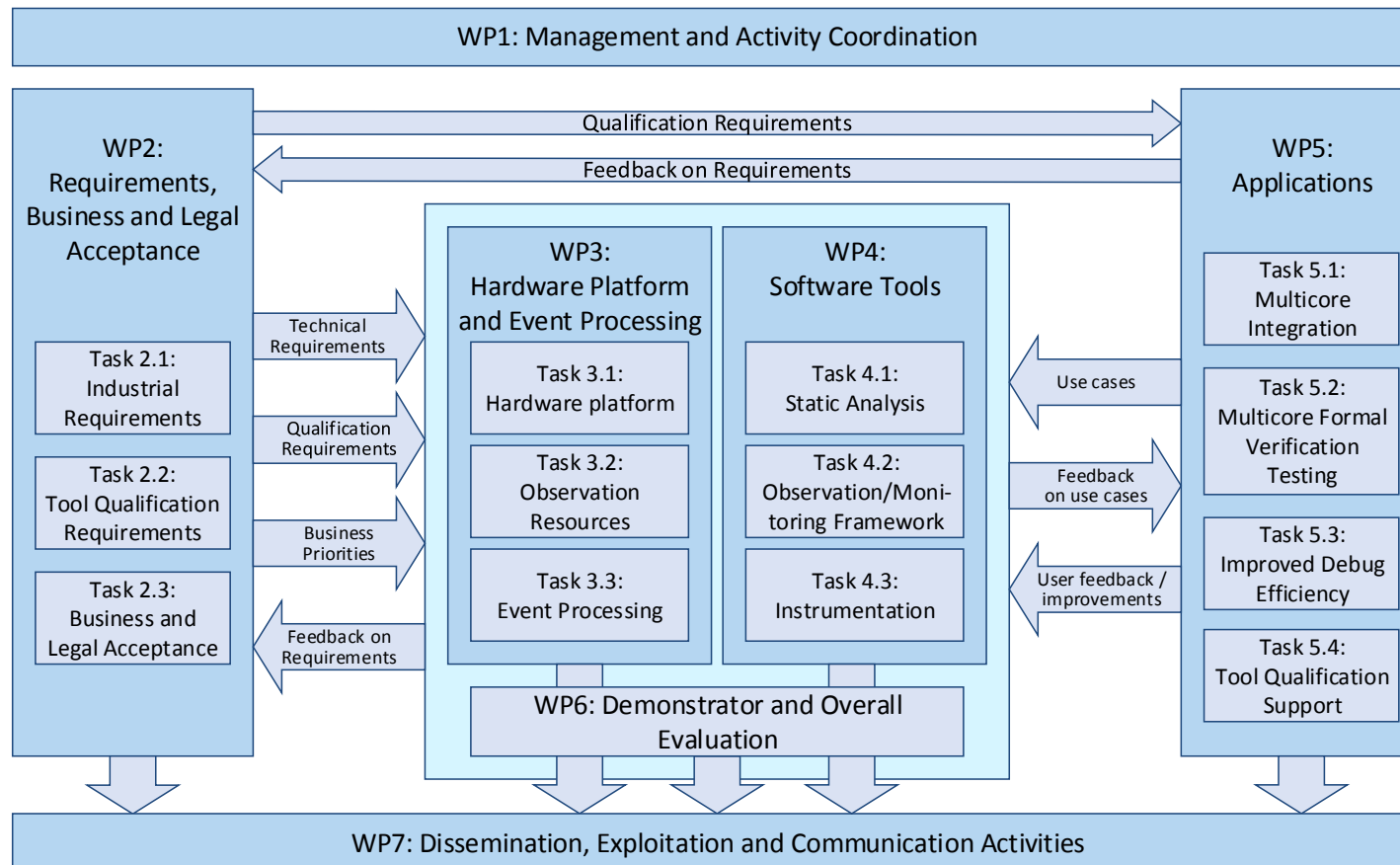
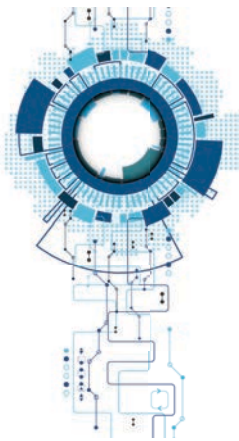
Airbus

ICT-10-2016
COEMS
Continuous Observation of
Embedded Multicore Systems

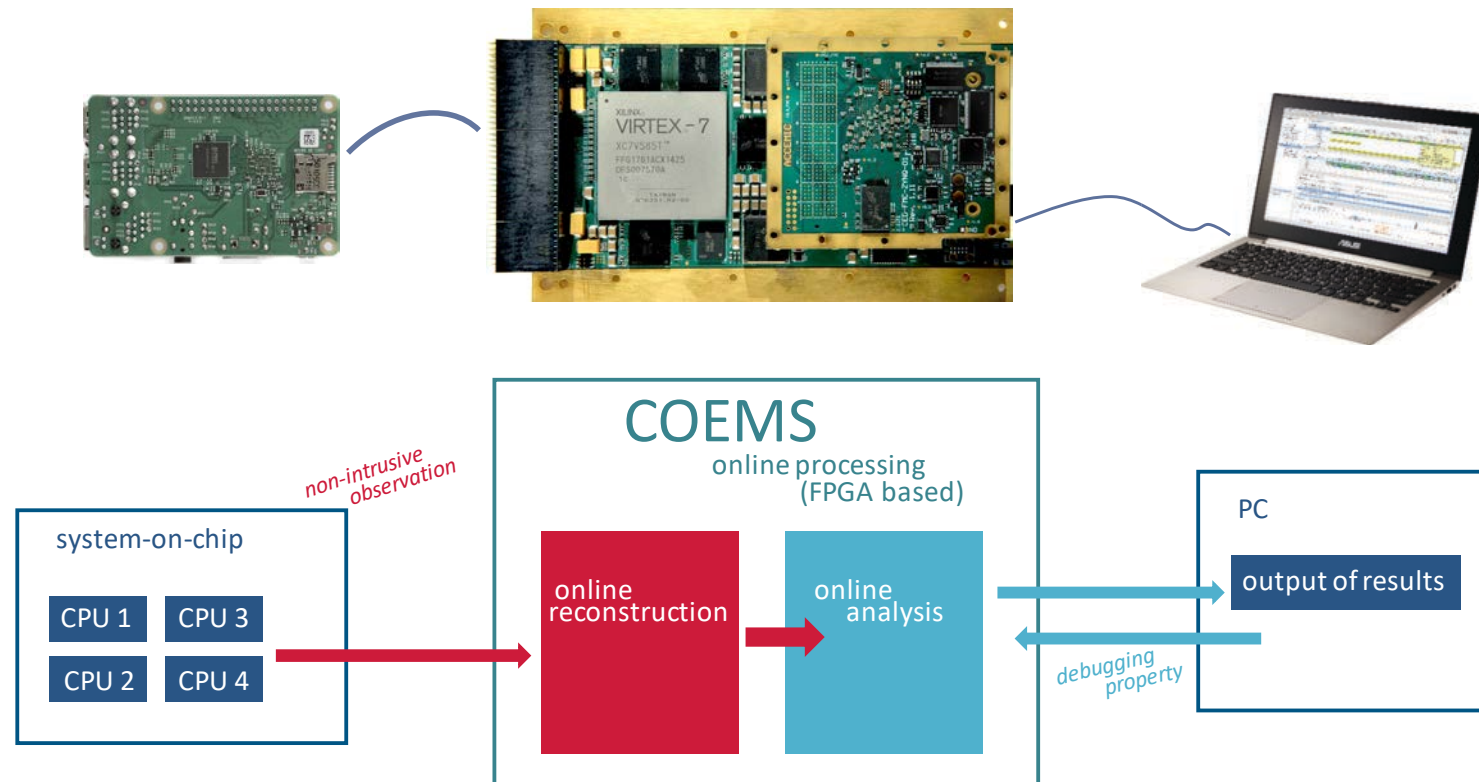


This project has received funding from the European
Union's Horizon 2020 research and innovation programme
under grant agreement no. 732016.

WP Structure



COEMS set-up



Hardware Platform

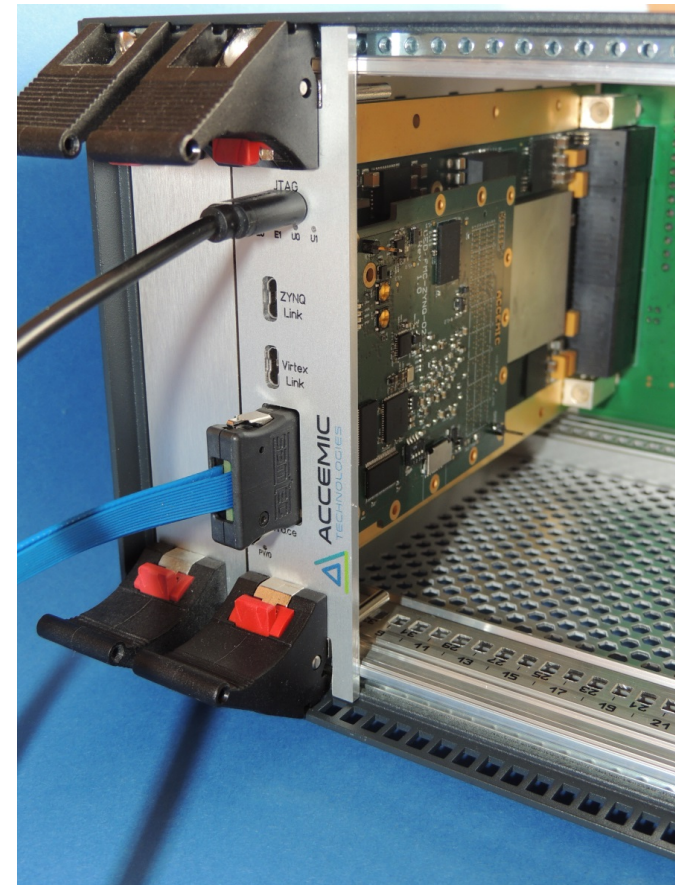
- Hardware

- Virtex-7 series FPGA (available)
- Zynq Ultrascale+ SOC (under development)
- RLDRAM3 memory for fast lookup tables
- Interface to Aurora (Nexus, HSSTP)
- VPX / FMC form factor

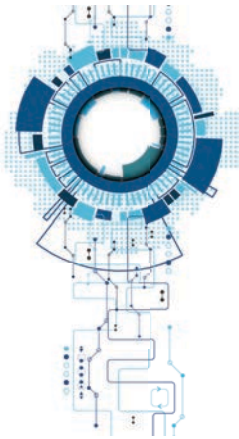
- Functionality

- Online trace data processing
(Coresight trace data -> event stream)
- Supported architectures:
ARM Cortex-A9, ARM Cortex-A53*,
QorIQ PPC*, Infineon Aurix*
- Online processing of event stream

*under development



ICT-10-2016
COEMS
Continuous Observation of
Embedded Multicore Systems



Web IDE – with trace

TeSSLa

Run

TeSSLa Examples ▾RV Examples ▾EPU Examples ▾

Trace

C Code (SW)C Code (EPUs)

```
1 0: temperature = 5
2 5: temperature = 15
3 10: temperature = 0
4 15: temperature = -8
5 16: temperature = -9
6 17: temperature = -10
7 18: temperature = -11
8 19: temperature = -12
9 20: temperature = -13
```

Specification

```
1 in temperature: Events[Int]
2
3 def low := !(temperature > -10)
4 def high := temperature > 10
5
6 def unsafeTemperature := low || high
7
8 out unsafeTemperature
9
```

Status and Compiler Output

```
1
```

TeSSLa Output

```
1 0: unsafeTemperature = false
2 5: unsafeTemperature = true
3 10: unsafeTemperature = false
4 15: unsafeTemperature = false
5 16: unsafeTemperature = false
6 17: unsafeTemperature = true
7 18: unsafeTemperature = true
8 19: unsafeTemperature = true
9 20: unsafeTemperature = true
10
```

Web IDE with C-code – software backend

The screenshot displays the TeSSLa Web IDE interface, which is divided into four main panels. At the top, there is a dark blue header bar with the 'TeSSLa' logo, a green 'Run' button, and navigation links for 'TeSSLa Examples', 'RV Examples', and 'EPU Examples'. The 'C Code (SW)' panel on the top left contains C code for a program that includes `<stdio.h>`, `<stdlib.h>`, and `<unistd.h>`. It defines a `compute()` function that sets a duration, adds a random value, and sleeps, and a `main()` function that calls `compute()` in a loop. The 'Specification' panel on the top right shows a formal specification with variables for `duration`, `error`, and `maximum`, and functions for `runtime` and `function_call`. The 'Status and Compiler Output' panel at the bottom left shows the compilation and execution status. The 'TeSSLa Output' panel at the bottom right shows the execution results, including time units and values for `maximum`, `error`, and `duration`.

TeSSLa ▶ Run TeSSLa Examples ▾ RV Examples ▾ EPU Examples ▾

Trace C Code (SW) C Code (HW)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 void compute() {
6     int duration = 40000;
7     duration += (rand() % 10) * 1000;
8     usleep(duration);
9 }
10
11 int main() {
12     for (int i = 0; i < 10; i++) {
13         compute();
14     }
15 }
```

Specification

```
1 def duration := runtime("compute")
2 out duration
3
4 def error := if duration > 45ms then ()
5 out error
6
7 def maximum := max(duration)
8 out maximum
9
10
11
12 -- Standard Library
13 def runtime(name) := {
14     def call := function_call(name)
```

Status and Compiler Output

```
1 STATUS    compiling and instrumenting main.c...
2 STATUS    starting /tmp/bin/main...
3
```

TeSSLa Output

```
1 $timeunit = "ns"
2 0: maximum = 0
3 78733488: maximum = 56131578
4 78733488: error = ()
5 78733488: duration = 56131578
6 125350346: maximum = 56131578
7 125350346: error = ()
8 125350346: duration = 46245885
9 172614641: maximum = 56131578
10 172614641: error = ()
11 172614641: duration = 47204220
12 218188106: maximum = 56131578
13 218188106: error = ()
14 218188106: duration = 45290357
```

Web IDE – with C-code – hardware backend

TeSSLa

Run

TeSSLa Examples ▾ RV Examples ▾ EPU Examples ▾

Trace C Code (SW) C Code (HW)

```
179 double Vz_control_50483_delta_e_c_delta_e_c;
180
181 int Va_filter_100449_fun(void* args)
182 {
183     double Va_f;
184     static int Va_rcell=0;
185     const struct write_proto_t Va_f_Va_control_50474_Va_f_write =
186     { NULL, 0, (int []){ true , false }, 2 };
187     static int Va_f_Va_control_50474_Va_f_wcell=0;
188     static int instance=0;
189
190     Va_f=Va_filter_100(aircraft_dynamics495_Va_Va_filter_100449_Va[Va_rcell]);
191     Va_rcell=(Va_rcell+1)%2;
192 }
```

Specification

```
1 in call: Events[Unit] # function_call 'Va_filter_100'
2 in return: Events[Unit] # function_return 'Va_filter_100'
3
4 def duration := runtime(call, return)
5 out duration
6
7 def error := if duration > 100us then true
8 out error
9
10
11
12 -- Standard Library
13 def runtime(call, return) := {
14     def diff := time(return) - time(call)
```

Status and Compiler Output

```
16 STATUS wrote Pipeline (PDF) to /wd/bin/pipeline.pdf
17 STATUS wrote Event Input Configuration to /wd/bin/ev_in.txt
18 STATUS wrote Configuration to /wd/bin/epu_cfg.txt
19 STATUS executing on CEDAR hardware...
20 EBUS Iterating ports ...
21 EBUS Device found SN:201802042416A
22 EBUS Device found SN:201802042416B
23 EBUS Opened Port
24 EBUS Wrote 21 Events
25 EBUS Read 1 Events
26 EBUS Read 1 Events
27 EBUS Read 1 Events
28 EBUS Read 1 Events
29 EBUS Read 1 Events
```

TeSSLa Output

```
1 185963: duration = 78103
2 471542: duration = 63959
3 760420: duration = 80947
4 1089009: duration = 84208
5 1367065: duration = 73269
6 1713848: duration = 64488
7 1982241: duration = 82190
8 2259470: duration = 65770
9 2507209: duration = 65311
10 2789164: duration = 58865
11
```

Conclusions

Summary

- Sometimes software annotations not acceptable
- Usage of trace functionality of modern processors feasible
- Extra hardware may be used to monitor non-intrusively
- Sophisticated ideas necessary to make overall approach feasible
- Hardware-based RV might be a game changer

Future Work

- Abstractions in TeSSLa
- Precise Relation of TeSSLa fragments to STL
- Partial-Order Semantics
- Support for ITM traces
- Increase performance of implementation
- Achieve TRL6
- Enhance training material