Hardware-Assisted Online Monitoring

Volker Stolz

Invited talk

VORTEX 2022

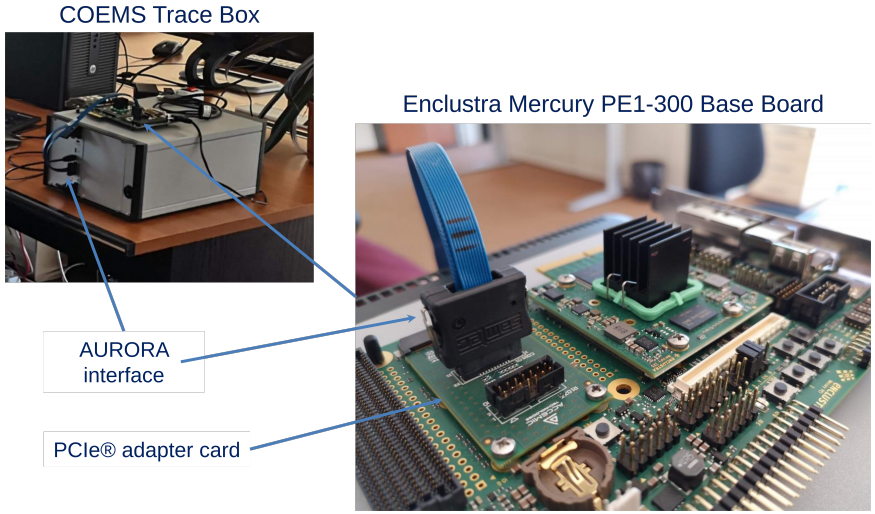
6 June 2022

Western Norway
University of
Applied Sciences

Supported by the Norwegian Research Council through grant
"COEMS Training Network" (309527)

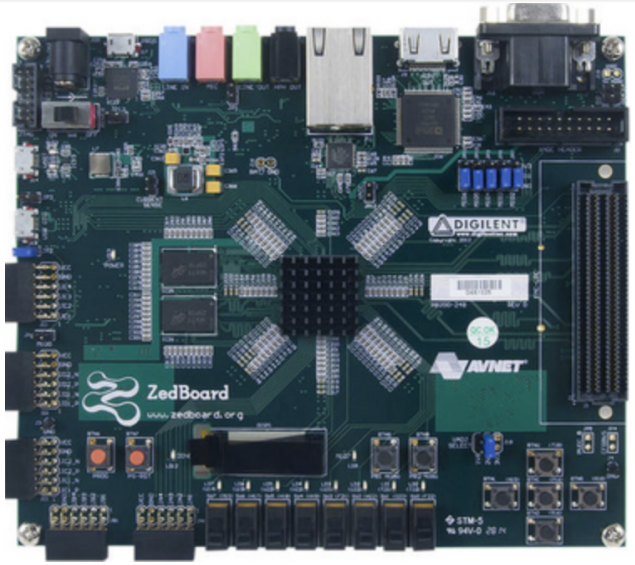
Hardware-assisted Online Monitoring

- Overview
  - Why? How?
  - Some examples from the RV community and beyond
- An example in depth: the COEMS hardware platform
  - Data race detection

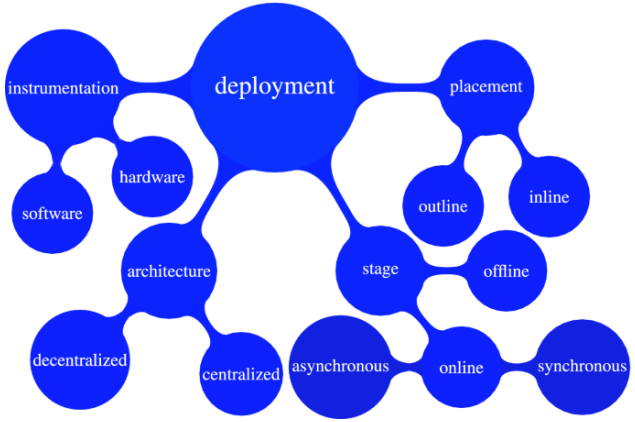


Why Hardware-assisted Online Monitoring?

- Advantage of FPGAs [Baumeister et al. 2019]:
  - faster processing speed due to the inherent parallelism
  - energy consumption
  - ease of integration within a CPS
- With respect to taxonomy of Runtime Verification:
  - may be used with or without(!) instrumentation
  - requires centralized architecture
  - easiest to use for asynchronous online monitoring

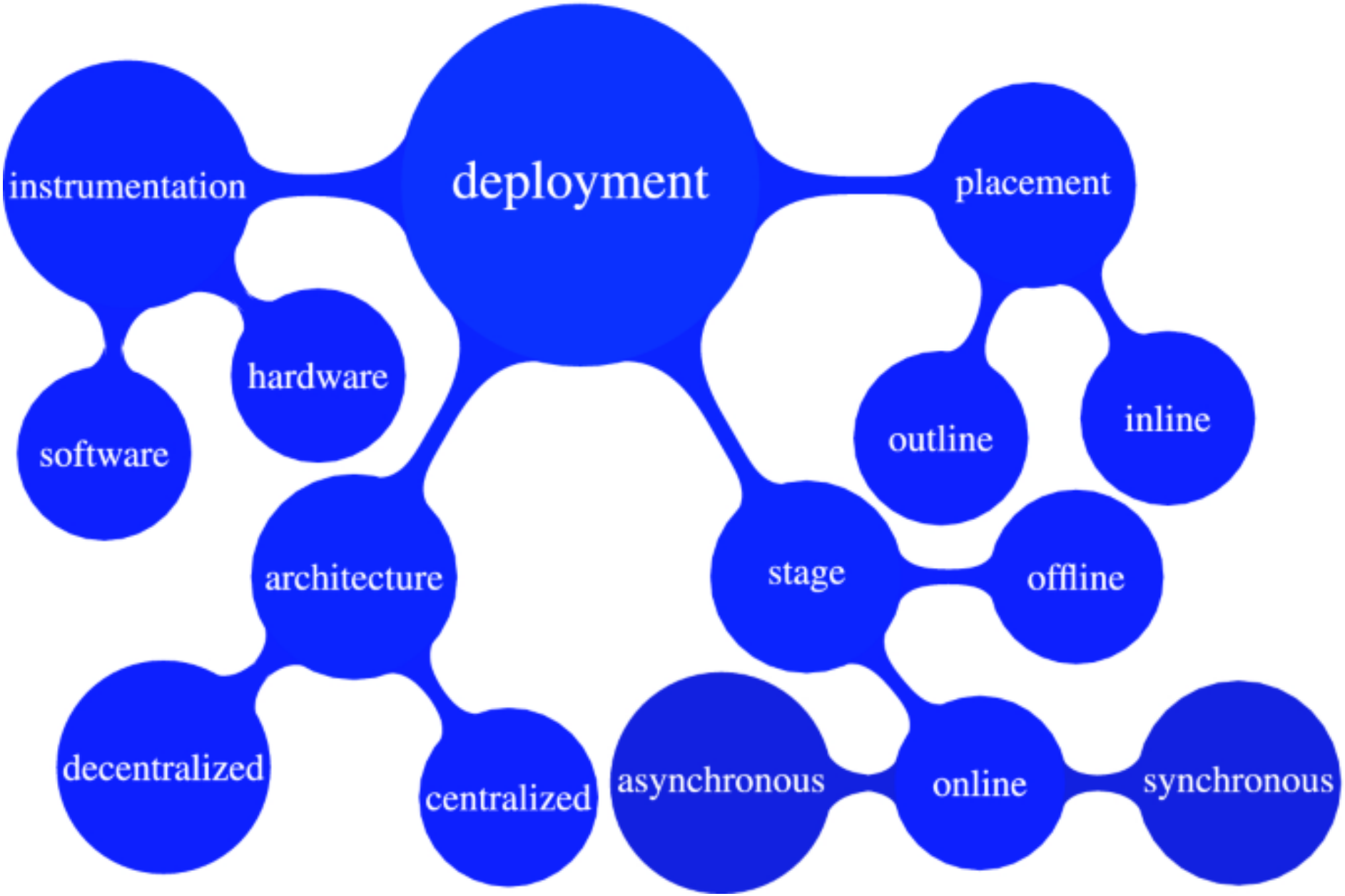


ZedBoard with Zynq-7000



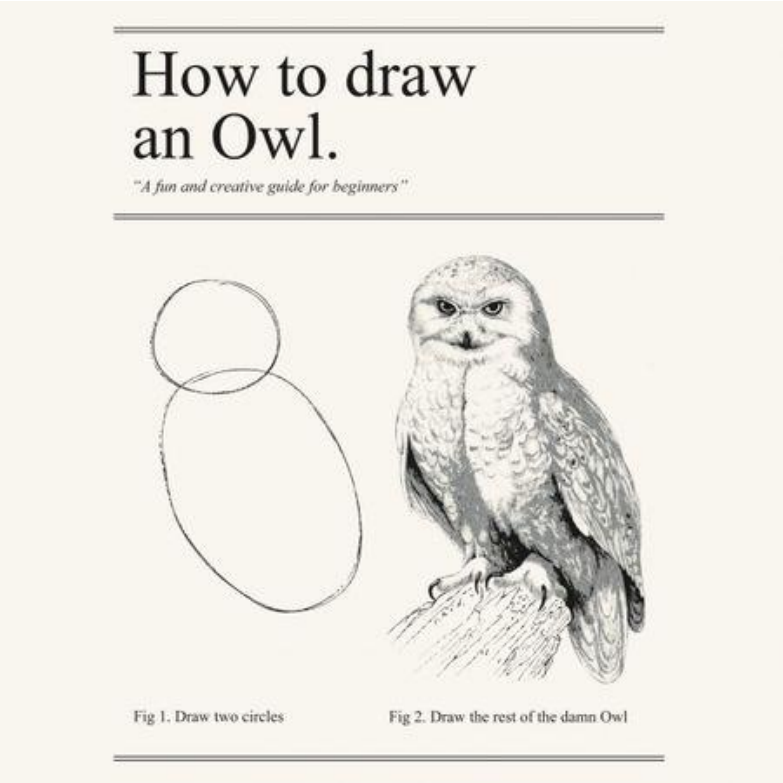
Taxonomy [Falcone et al. 2018]

Taxonomy [Falcone et al. 2018]



Some Assembly Required

- General worklow:
  - Translate your favourite specification language into VHDL or Verilog. . .
  - . . . either directly or through some middleware (RustHDL, Kôika, Cava, . . .).
  - Provide adapters for I/O
  - Synthesize & deploy!
- Challenges:
  - How to develop adapters?!
  - Synthesis sloooooow
  - Limited space on FPGA
  - How to reconfigure running system with monitor?



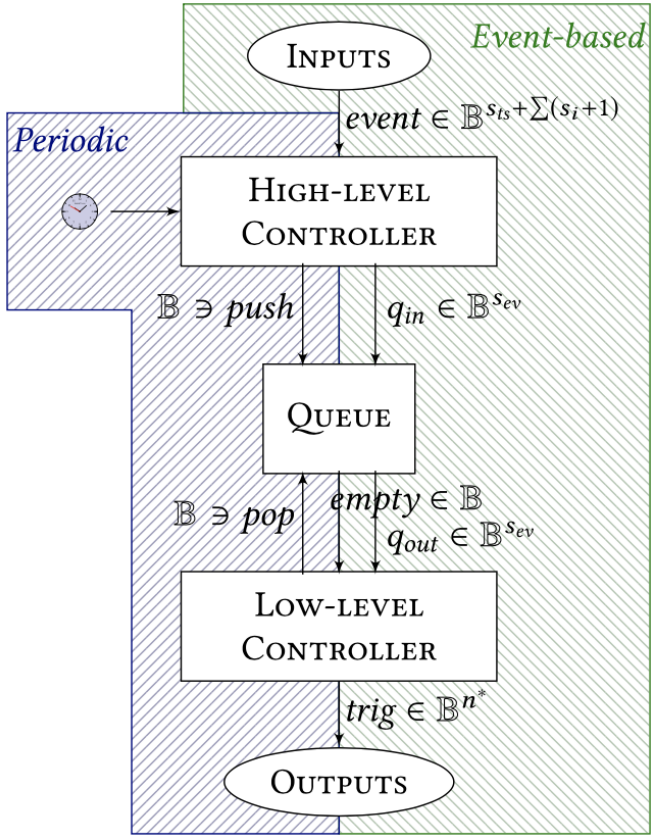
Application Area CPS

Baumeister, J., Finkbeiner, B., Schwenger, M., Torfah, H.: **FPGA stream-monitoring of real-time properties**, ("RTLola") 2019.

- Stream-based specification language
- Sensors = input, output = *periodic* or *event-based streams*

```
input gps: (Float64, Float64)
output gps_glitch: Bool@1Hz := gps.aggregate(over:2s,using:c
trigger gps_glitch "GPS sensor frequency < 5Hz"
```

- Compiled into VHDL components: High-/Low-level controller + queue



Similarly: Jaksic, S., Bartocci, E., Grosu, R., Kloibhofer, R., Nguyen, T., Nickovic, D.: **From signal temporal logic to FPGA monitors**, 2015.

Application Area Info-Sec

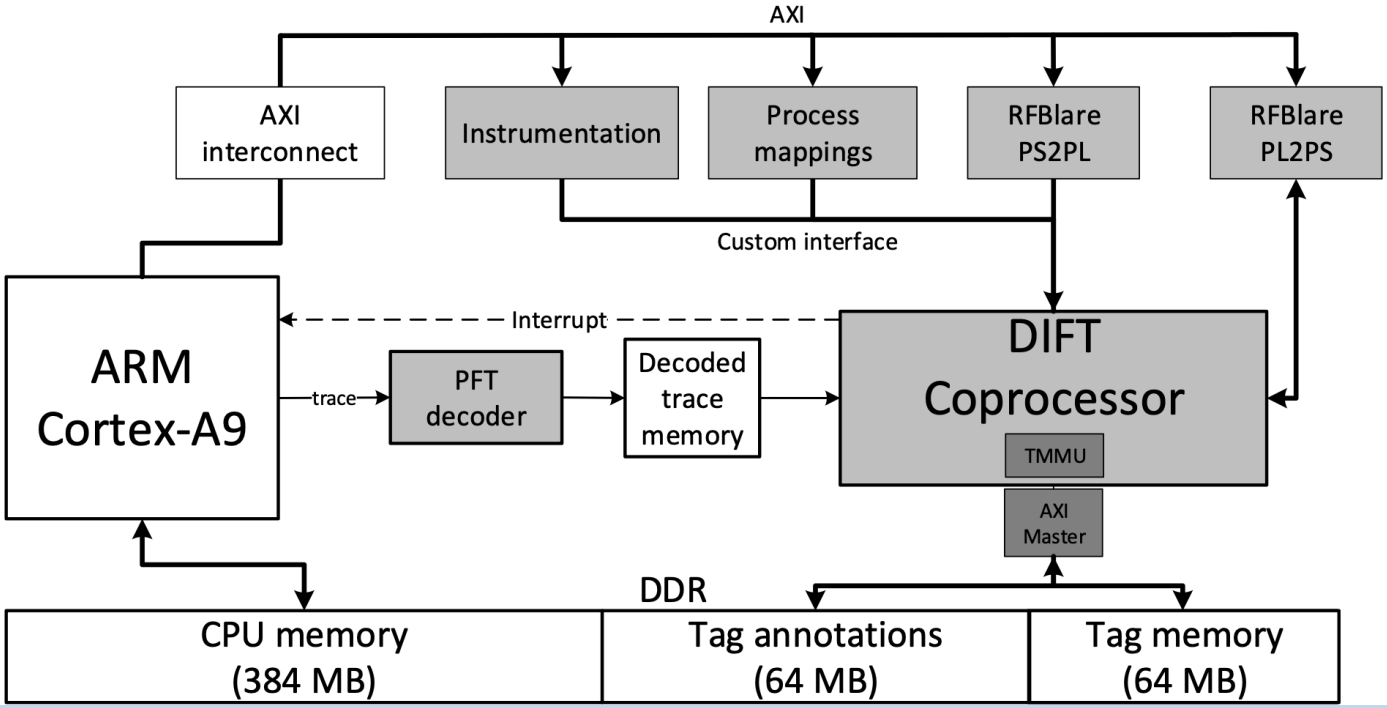
Hiet et al., INRIA Hardblare project: track information flow

Event sources:

- *Program Flow Trace* ("jumps" between basic blocks)
- Instrumentation for data accesses

Processing:

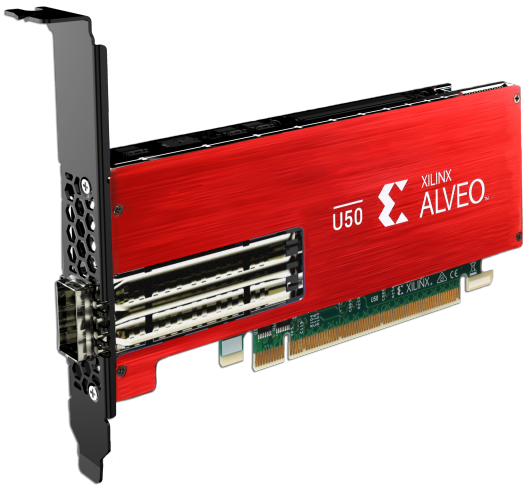
- DIFT coprocessor on FPGA
- Executes abstract program on IF data (separate RAM)
- Precomputed effect of each BB through static analysis



Design Space & Alternatives

- Tracing abstract events?
  Send data from system/ application to FPGA
- Tracing actual binaries?
  Use hw-tracing facilities!
  - reconstruct program flow
  - combine with static analysis
  - intersperse with necessary data through instrumentation

- Alternative to SoC:
  PCI Express-based cards



- "Easy" to combine with ARM CoreSight or Intel PT
- You still need the trace decoder wrt. the binary
- Issue: sourcing
  (*thank you, Bitcoin*)

Hardware-based Approach to Monitoring Races

Idea of the `https://www.coems.eu` project:

- Software-/instrumentation-based approaches have high overhead
- Offloading runtime checks to separate hardware
- Out-of-band monitoring, no runtime reflection possible
- Avionics & railway domain:
  - predictable performace independant from whether or not race checking is enabled;
  - can't swap between *normal* and *debugging* version in the field.

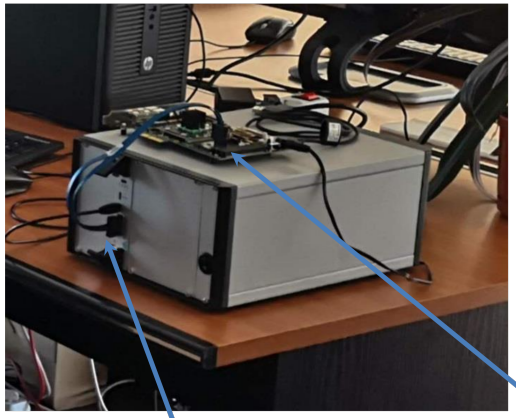
- F. Ahishakiye, J. Requeno Jarabo, V. Pun, V. Stolz: **Hardware-Assisted Online Data Race Detection**. Formal Methods in Outer Space, LNCS 13065, 2021.

Background: Lockset Algorithm

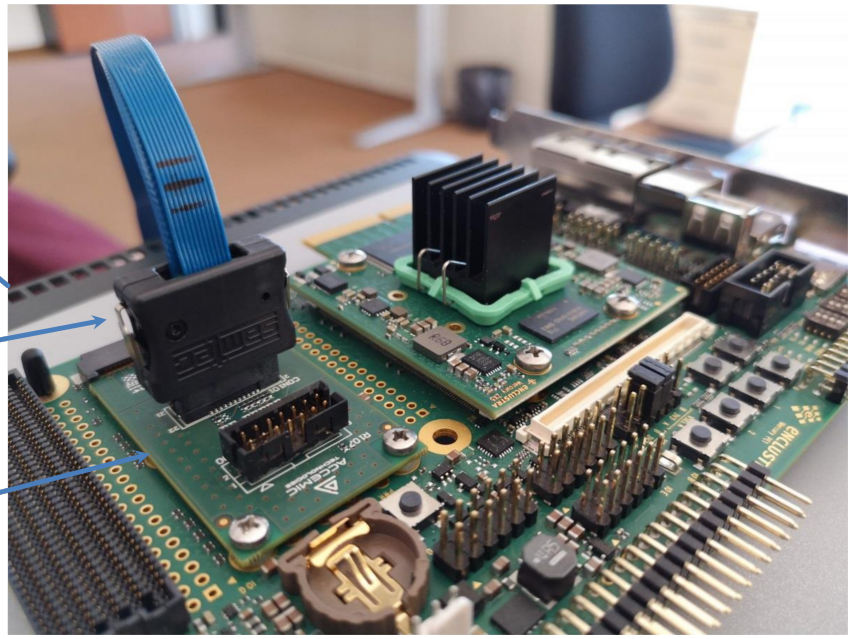
- Variable/memory accesses
- Compute sets of locks held by threads which are accessing variables/memory
- Initialize the set of locks guarding variable/memory with the set of all locks
- On each access, `guarding` := `guarding` $\cap$ `holding`
- Error when guarding == $\emptyset$

COEMS Trace Box containing FPGA (left) and SoC (right)

COEMS Trace Box



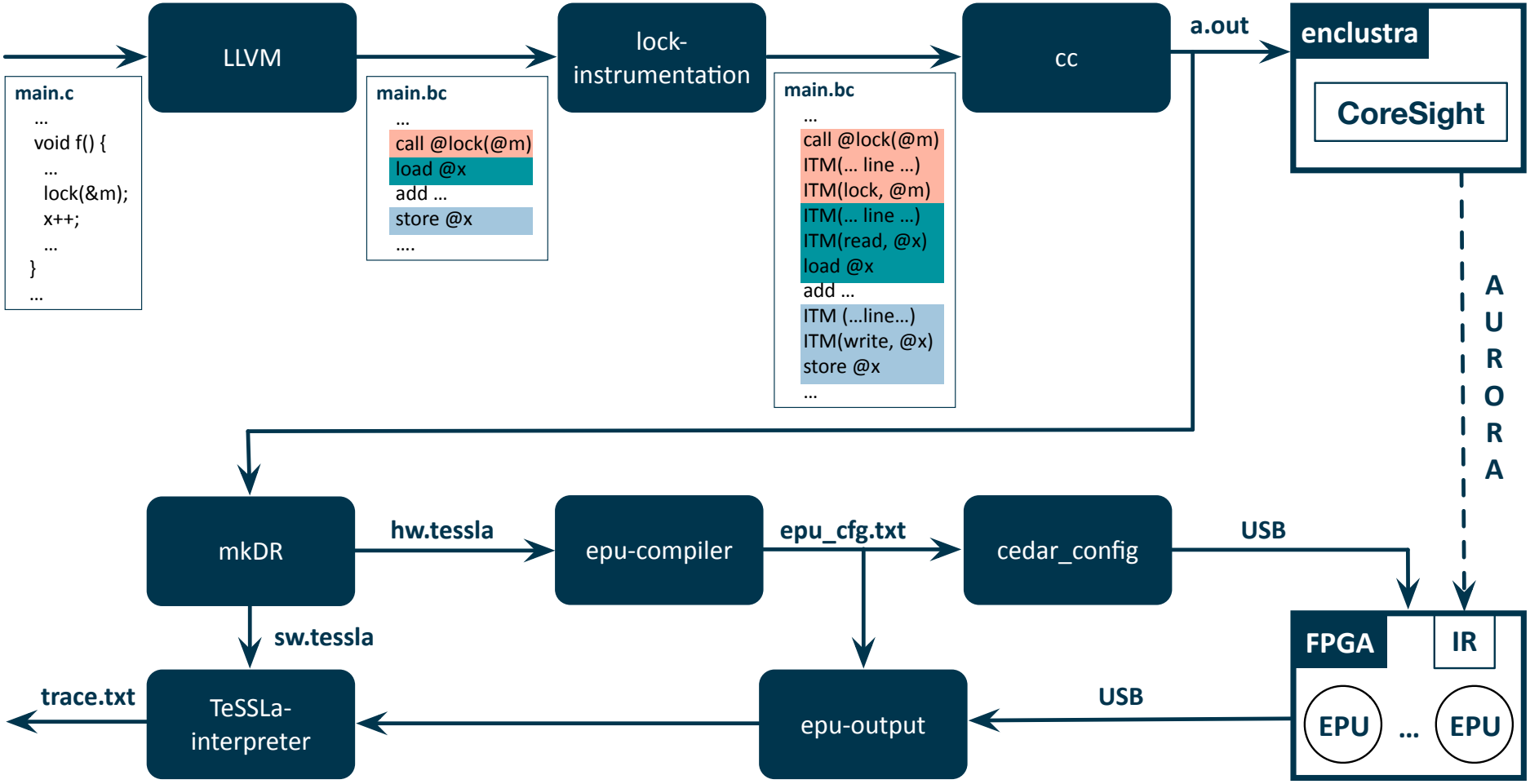
Enclustra Mercury PE1-300 Base Board



AURORA interface

PCIe® adapter card

Lock instrumentation & race monitoring with COEMS



L. Convent, S. Hungerecker, T. Scheffel, M. Schmitz, D. Thoma, A. Weiss: **Hardware-Based Runtime Verification with Embedded Tracing Units and Stream Processing**, RV'18, LNCS.

Example of incorrect locking

```
/* get an exclusive lock on both balances before
   updating (there's a problem with this, see below) */
pthread_mutex_lock(transaction_mtx);
if( !DATA_RACE || (DATA_RACE & (id != 0)) ){
    /* In case of DATA_RACE flag is 'on', the thread_id 0
    forgets to lock the accts[from].mtx mutex */
    pthread_mutex_lock(&accts[from].mtx);
                }
pthread_mutex_lock(&accts[to].mtx);
pthread_mutex_unlock(transaction_mtx);
/* Do the actual transfer. */
if (accts[from].balance > 0) {
        payment = 1 + rand_range(accts[from].balance);
        accts[from].balance -= payment;

pthread_mutex_unlock(&accts[to].mtx);
if( !DATA_RACE || (DATA_RACE & (id != 0)) ){
    /* For symmetry -- don't unlock if racy: */
    pthread_mutex_unlock(&accts[from].mtx);
}
```

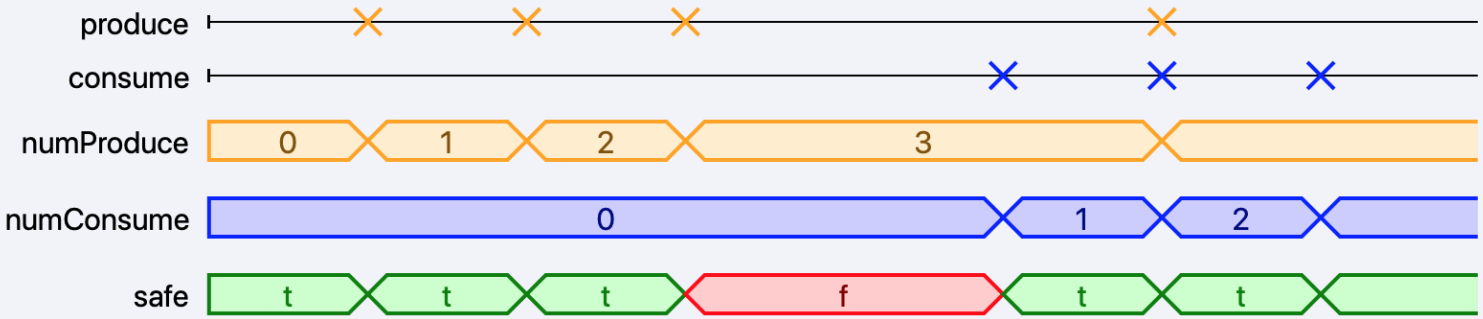
```
in threadid: Events[Int]
in readaddr: Events[Int]
in writeaddr: Events[Int]
in mutexlockaddr: Events[Int]
in mutexunlockaddr: Events[Int]
@FunctionCall("__pthread_create_2_1")
in pcreateid: Events[Unit]
in line: Events[Int]
in dyn_base: Events[Int]
in dyn_lock: Events[Int]
```

Header of the TeSSLa spec, including all the incoming events from the instrumented code

Details on Leucker's

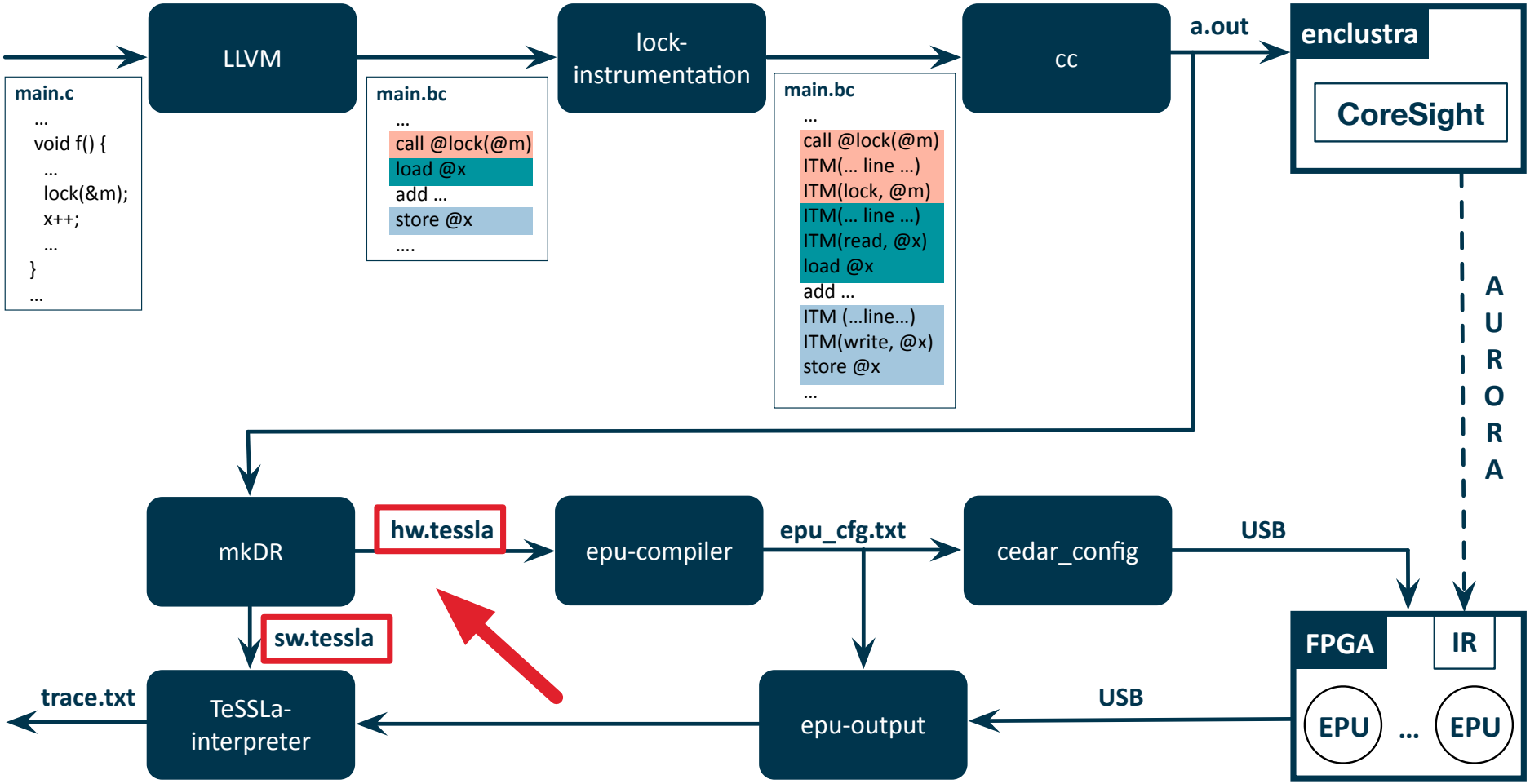
https://www.tessla.io.

```
in produce: Events[Unit]
in consume: Events[Unit]

def numProduce := count(produce)
def numConsume := count(consume)
def safe := numProduce - numConsume <= 2
```



```
1  def lock_0 := filter(mutexlockaddr ==1, mutexlockaddr ==1)
2  def unlock_0 := filter(mutexunlockaddr ==1, mutexunlockaddr ==1)
3  def lock_1 := filter(mutexlockaddr ==24808, mutexlockaddr
       ==24808)
4  def unlock_1 := filter(mutexunlockaddr ==24808, mutexunlockaddr
       ==24808)
5  def dyn_temp := 4 * (((dyn_lock >> 1) >> 1))
6  def slot := dyn_lock - dyn_temp
7  def dyn_lock_0 = filter(dyn_temp, slot == 0)
8  def lock_4 := filter(mutexlockaddr == dyn_lock_0, mutexlockaddr
       == dyn_lock_0)
9  def unlock_4 := filter(mutexunlockaddr == dyn_lock_0,
       mutexunlockaddr == dyn_lock_0)
10 def read_0 := filter(readaddr ==24532, readaddr ==24532)
11 def write_0 := filter(writeaddr ==24532, writeaddr ==24532)
12 def access_0 := merge(read_0,write_0)
13 def access_after_pc_0 := on(last(pcreateid,access_0),line)
14 def thread_accessing_0 := last(threadid*32768 + line,
       access_after_pc_0)
15 def holding_0 := default(merge(last(threadid, lock_0), last(-1,
       unlock_0)), -1)

16 def protecting_0_with_0 := detect_change(default(
       thread_accessing_0/32768 == last(holding_0,
       thread_accessing_0), true))
17 def error_0 := on(thread_accessing_0,!(protecting_0_with_0 ||
       protecting_0_with_1 || protecting_0_with_2 ||
       protecting_0_with_3 || protecting_0_with_4))
```

Lock instrumentation & race monitoring with COEMS



Recompilation of Specifications

Advantage of COEMS:

- Can reconfigure monitoring on the fly...

Recompilation of Specifications

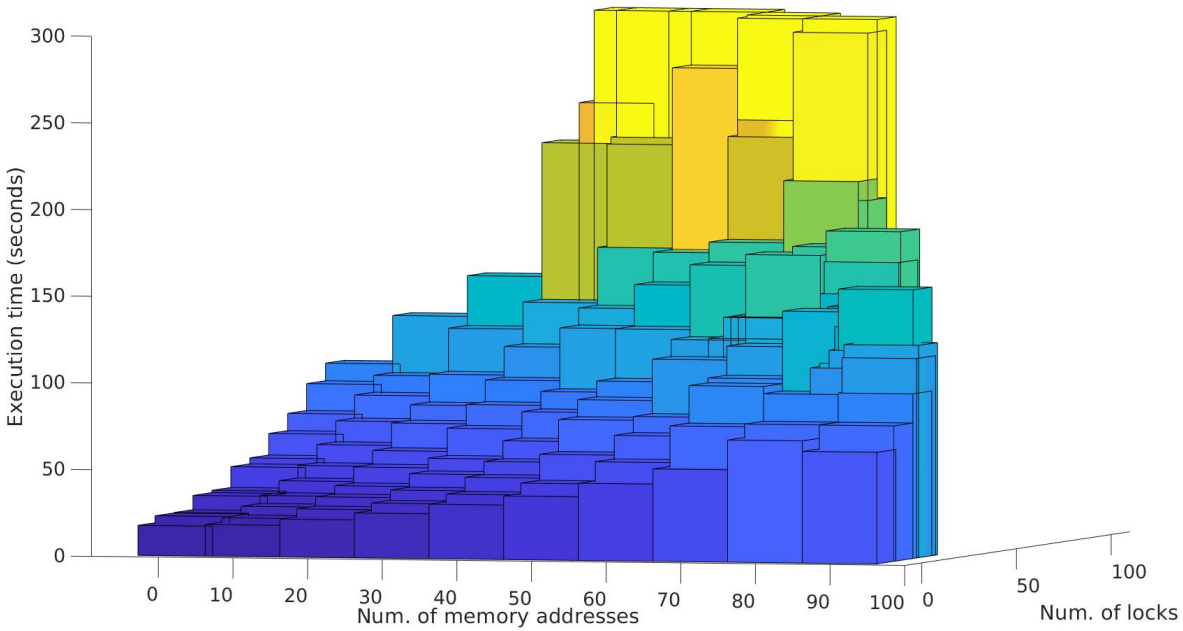
Advantage of COEMS:

- Can reconfigure monitoring on the fly...
- ...*for some value of "on the fly".*

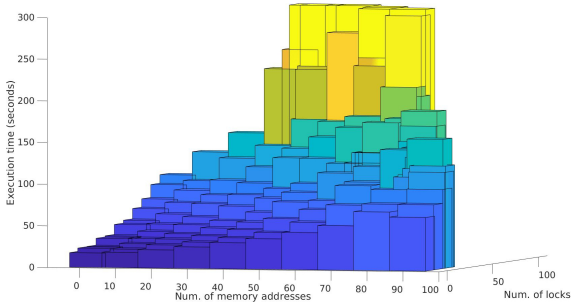
Recompilation of Specifications

Advantage of COEMS:

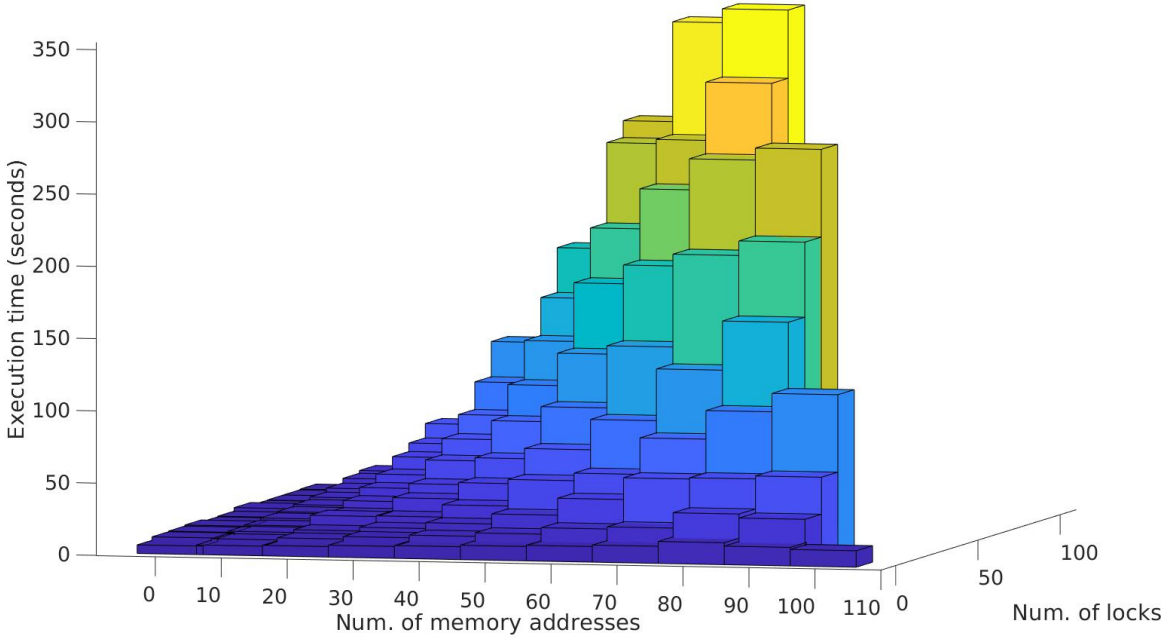
- Can reconfigure monitoring on the fly...
- ... *for some value of "on the fly".*
- Size of specification scales with number of locks & memory locations (not: threads!)



EPU compilation time for hw.tessla



EPU compilation time for hw.tessla



TeSSLa interpreter startup time for sw.tessla

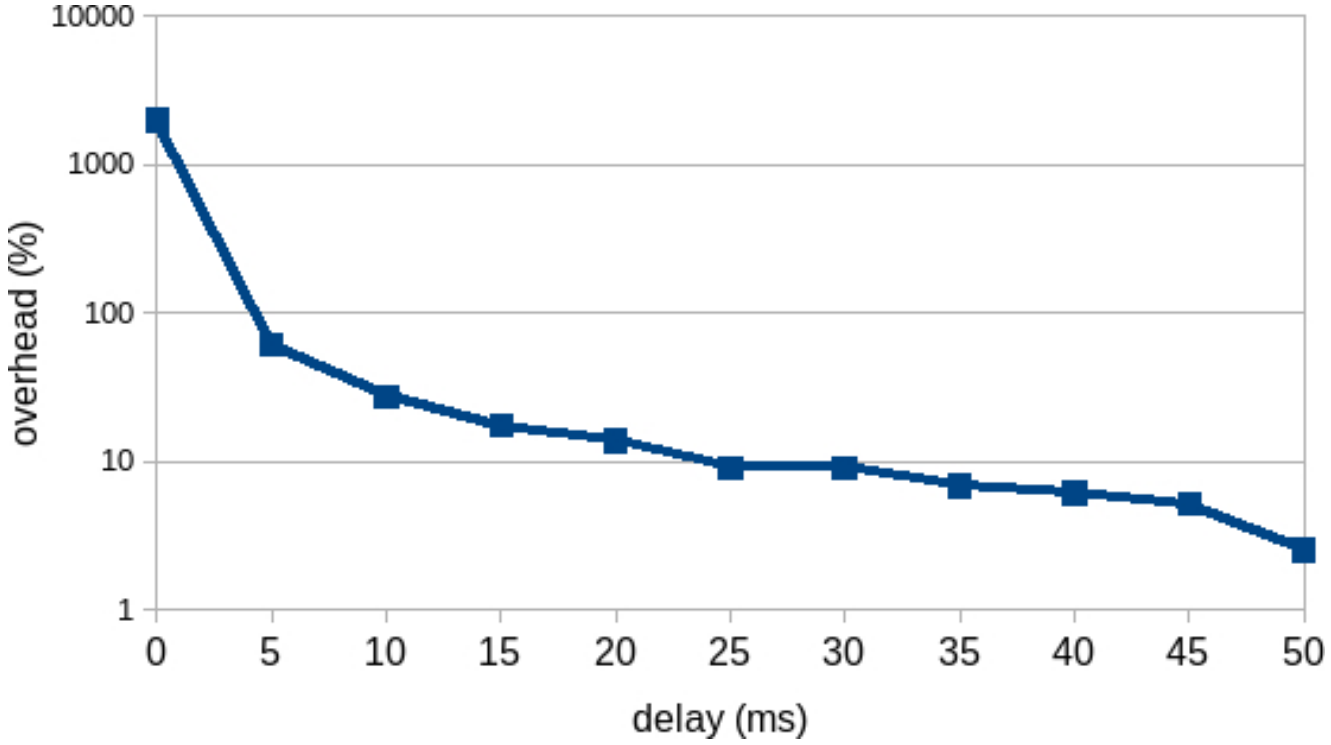
```
105923234788: holding_1 = 21653
105923846011: holding_3 = 21653
105923949854: holding_1 = -1
105924265689: holding_1 = 21525
105924621853: thread_accessing_1 = 709525559
105924716727: thread_accessing_1 = 709525559
105924764106: holding_2 = 21525
105924822656: holding_3 = -1
105924872330: holding_1 = -1
105925531285: thread_accessing_0 = 705331255
105925621081: thread_accessing_0 = 705331255
105925674037: holding_1 = 21653
105925731366: holding_2 = -1
105926266497: holding_2 = 21653
```

Events emitted by the COEMS trace box after processing the hardware half of the TeSSLa spec.

```
105847960743: error_0_in_line = 52
105847960743: error_0 = true
105848052196: error_0_in_line = 53
105848052196: error_0 = true
105848316093: error_0_in_line = 54
105848316093: error_0 = true
```

Race report obtained by processing the above events including debug information

Overhead introduced by the instrumentation



- COEMS platform in general does not need instrumentation for control-flow events...
- ...but we need instrumentation if data is involved.
- Ideally: Arm CoreSight/Intel PT can do this via simple `mov`-instruction

Summary

- We got it to work :-)
  (Hardware, specification language with compiler, toolchain)
- Acceptable overhead to industry partners
- Can't rule out some (inherited) false-positives

```
for (i = 0; i < N_THREADS; i++)
        pthread_join(ts[i], NULL);
for (total = 0, i = 0; i < N_ACCOUNTS; i++)
        total += accts[i].balance;
printf("Total money in system: %ld\n", total);
```

Summary

- We got it to work :-)
  (Hardware, specification language with compiler, toolchain)
- Acceptable overhead to industry partners
- Can't rule out some (inherited) false-positives

```
for (i = 0; i < N_THREADS; i++)
      pthread_join(ts[i], NULL);
for (total = 0, i = 0; i < N_ACCOUNTS; i++)
      total += accts[i].balance;
printf("Total money in system: %ld\n", total);
```

Visit us on

`https://www.coems.eu`

or

`https://tessla.io`