PRODUCT BRIEF

CEDARTOOLS

THE STORY

With the ever-increasing complexity of embedded systems, the number of software defects is also growing[1]. Disproportionately. Even if your engineers are geniuses.
Freedom from defects is an illusion. Nonetheless, closing in on this ideal should be the ultimate goal of any serious project management. In addition to a well-thought-out system architecture and a thorough implementation, (a) tests as complete as possible and (b) precautions for dealing with errors in the field are important prerequisites for the ability to develop and market products on time and in the best possible quality.

McKinsey&Company: "Snowballing complexity is causing significant software-related quality issues, as evidenced by millions of recent vehicle recalls."[2]

Capers Jones: "Because software defect removal efficiency averages only about 85% and essentially never equals 100%, there will always be latent defects present in software at the time of delivery."[1]

THE PRODUCT

The key factor for proper testing and efficient debugging is observability – ideally without affecting the system.
This is exactly what we deliver: CEDARtools® leverages complex electronics and clever software, which create life digital twin representations of monitored processors to perform live coverage measurements or runtime checks defined in a high-level language.
CEDARtools® is the first non-intrusive, continuous live observation tool for embedded systems. Running on release code without influencing it. Measuring code coverage without instrumentation even on high functional test levels. Measuring complex timing behavior without influencing the system.

SPECIFICATIONS

Structural coverage (branch and statement coverage, performance measurement) can be measured on all test levels on the executing object code to be back-annotated to the source code.
Multiple timing constraints specified in a high-level language[3] can be monitored concurrently over an arbitrary time frame.
CEDARtools® is easily digesting the multi-gigabit trace data stream produced by most industry-standard processors:

- Arm® Cortex®-A (via HSSTP, PCIe and parallel[4])
- Infineon Aurix™ TC2xx, TC3xx (via AGBT)
- QorIQ® P- and T-series (via Aurora and PCIe)
- Intel® Atom® E39x0 (via PCIe and USB[4])

Trace data storage and time-consuming offline processing are nuisances of the past.

KEY FEATURES

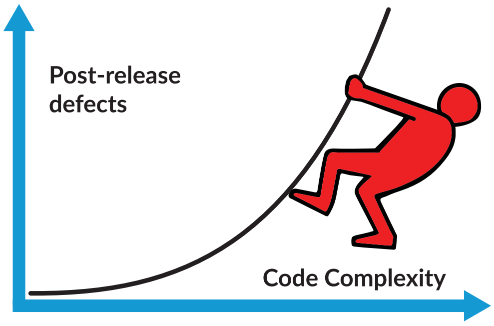
- Live, non-intrusive, and continuous observation of embedded processors
- Triggering on complex conditions which may have a high temporal spread
- Multicore support
- Multiple observation focuses
- Autonomous operation

KEY VALUES

- Long desired item in the arsenal of test tools
- Powerful tool for hunting elusive failures

APPLICATIONS

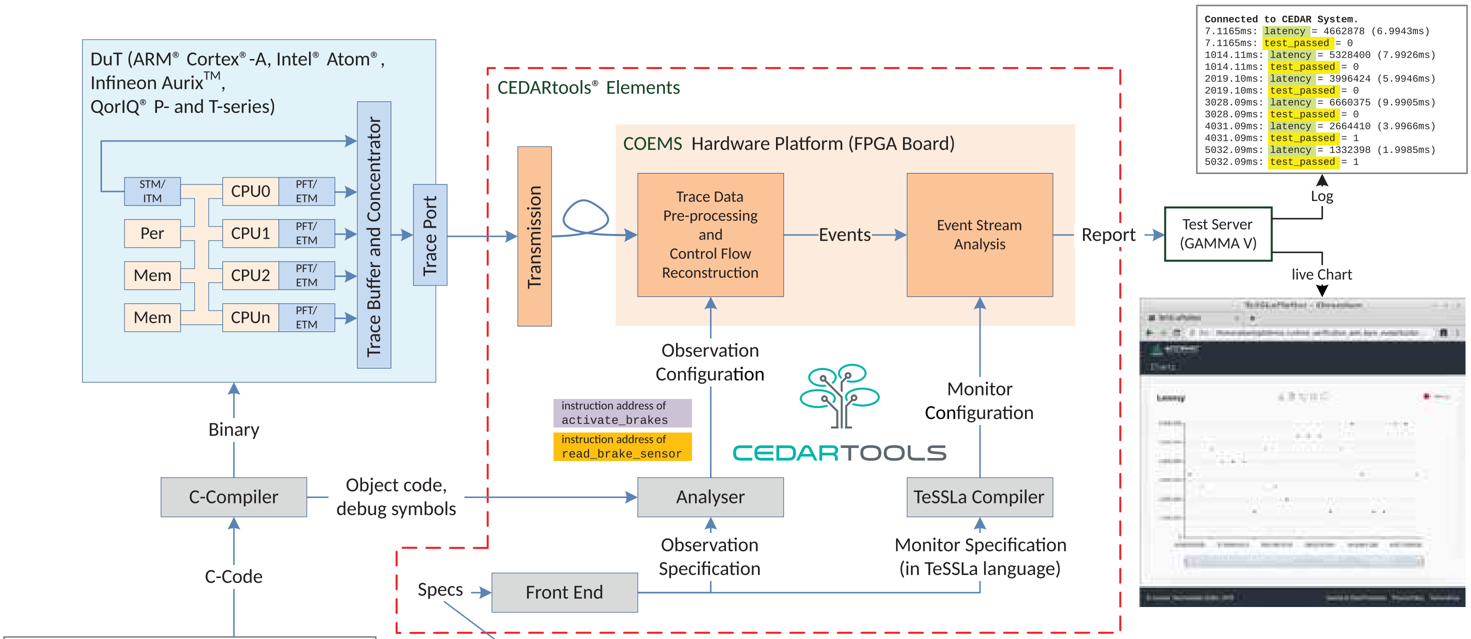
- Structural tests on object-code level without software instrumentation - just test your release code
- Performance measurement on release code
- Complex timing verification, configurable by a high-level language
- Effect chain measurement



Technology for Dependable Systems

[1] About 2500 defects (after proper testing in accordance with the state of the art) will still be in the release code of an embedded project with 10kFP complexity. (C. Jones and O. Bonsignour, The Economics of Software Quality. Addison-Wesley, 2011)
[2] O. Burkacky, J. Deichmann, G. Doll, and C. Knochenhauer, 'Rethinking car software and electronics architecture'. McKinsey, Feb-2018.
[3] see www.tessla.io to get a feeling for the awesome power of this ability
[4] still under development, but available for you soon.

VERIFICATION FLOW




```c
void run_task()
{
  float brake_angle = read_brake_sensor();

  int strength;
  strength = calculate_brake_strengh_for_angle(brake_angle);

  int motor_control;
  motor_control = calculate_motor_control_value(strength);

  if (motor_control == 1) {
    activate_brakes();
  }
  else if (motor_control == -1) {
    release_brakes();
  }
}
```

Source code

```
in read_brake_sensor : Events[Unit]
in activate_brakes   : Events[Unit]

def latency : Events[Int] := calc_latency(stimulus = read_brake_sensor,
                                           response = activate_brakes)

# Check if event chain took less than 5ms (3.333.333 CPU cycles)
def test_passed : Events[Bool] := if (latency < 3333333) then true else false

out latency
out test_passed

# domain specific library
def calc_latency[A,B](stimulus: Events[A], response: Events[B])
    := { return time(response) - last(time(stimulus), response)}
```

TeSSLa specification

Hundreds of high-level specifications can be monitored in parallel

COVERAGE ANNOTATION

```
17                          // Unfold Collatz sequence and return its length.
18                          // - n <= 1 will not execute the while loop at all.
19                          // - n = 2^k will never trigger the 3*n+1 path.
20                1970      unsigned collatz_depth(unsigned n) {
21                1970      unsigned depth = 0;
22   [+,+]        13987     while(n > 1) {
                ↪ 1970        40089B: eb 24 jmp 4008c1
                  13987       4008C1: 83 7d ec 01 cmpl $0x1,-0x14(%rbp)
     [+,+]      1970 ↪
                  12017       4008C5: 77 d6 ja 40089d
23   [+,+]        12017     n = (n&1)? 3*n+1 : n/2;
                  12017       40089D: 8b 45 ec mov -0x14(%rbp),%eax
                  12017       4008A0: 83 e0 01 and $0x1,%eax
                  12017       4008A3: 85 c0 test %eax,%eax
     [+,+]      3152 ↪ 8865   4008A5: 74 0e je 4008b5
                  3152        4008A7: 8b 55 ec mov -0x14(%rbp),%edx
                  3152        4008AA: 89 d0 mov %edx,%eax
                  3152        4008AC: 01 c0 add %eax,%eax
                  3152        4008AE: 01 d0 add %edx,%eax
                  3152        4008B0: 83 c0 01 add $0x1,%eax
                ↪ 3152        4008B3: eb 05 jmp 4008ba
                  8865        4008B5: 8b 45 ec mov -0x14(%rbp),%eax
                  8865        4008B8: d1 e8 shr %eax
                  12017       4008BA: 89 45 ec mov %eax,-0x14(%rbp)
24                12017     depth++;
25                          }
```

ABOUT ACCEMIC TECHNOLOGIES

Accemic Technologies, with offices in Kiefersfelden (Munich area) and Dresden, has developed breakthrough technology for the dynamic analysis of dependable embedded systems.
We make software test effective and efficient. We simplify the debugging process enormously and give you effective leverage even over sporadic and non-deterministic failures.

CONTACT PERSON:

Dr. Alexander Weiss
cedartools@accemic.com
+49 8033 6039795

ACCEMIC TECHNOLOGIES

Accemic Technologies GmbH
Franz-Huber-Str. 39,
83088 Kiefersfelden, Germany
www.accemic.com