DVCon 2012

Design & Verification Conference & Exhibition

February 28 – March 1, 2012

A Practical Approach to Measuring and Improving the Functional Verification of Embedded Software

Stéphane Bouvier
STMicroelectronics

Nicolas Sauzède
STMicroelectronics

Florian Letombe
SpringSoft

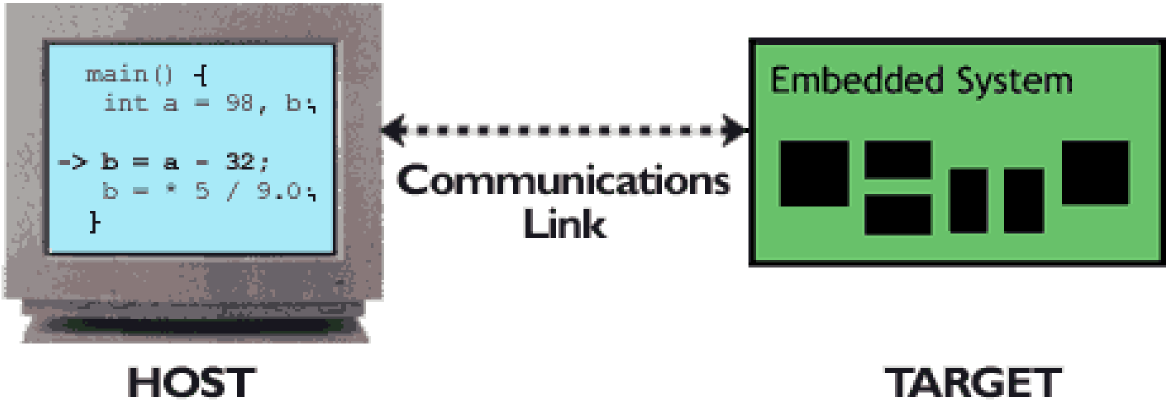
Julien Torrès
SpringSoft

Introduction

- Embedded systems development:
  - Systems tend to become more and more complex
  - Trend to have mixed hardware/software systems (RTL + firmware)
- Verification is difficult:
  - Are all scenarios covered?
  - Are all the specified functionalities checked?
  - This is even more difficult with mixed hardware/software systems
- But verification is important:
  - Bugs may be very costly

Embedded software development

- Host-target approach:
  - Develop on a host machine
  - Test on a target machine

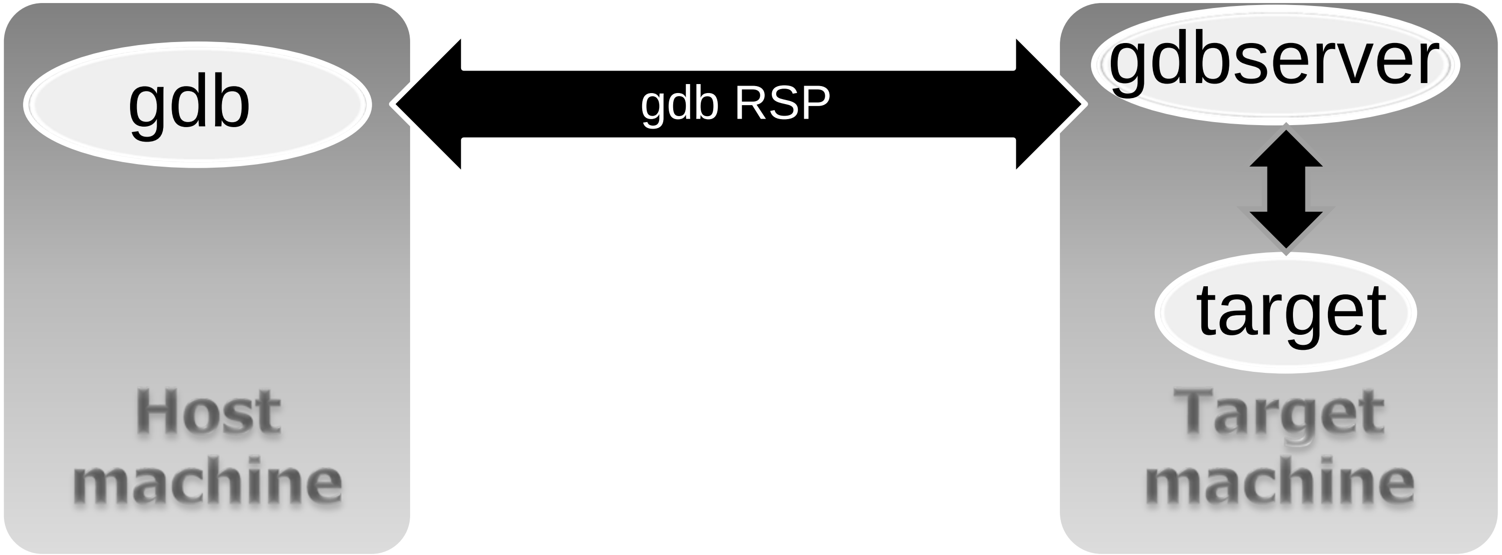


```
  main() {
   int a = 98, b;

-> b = a - 32;
   b = * 5 / 9.0;
  }
```

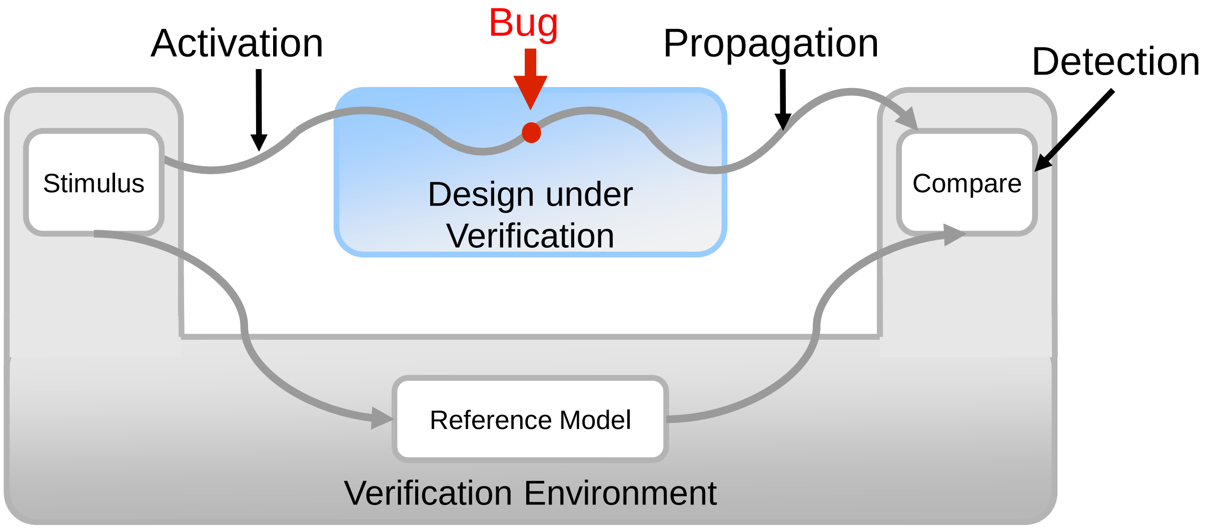
Embedded software debugging

- GNU Project debugger (gdb)
  - gdb running on the host machine
  - `gdbserver` running on the target machine
  - Communication through the gdb Remote Server Protocol (RSP)



Effective verification

It's all about activation, propagation, and detection

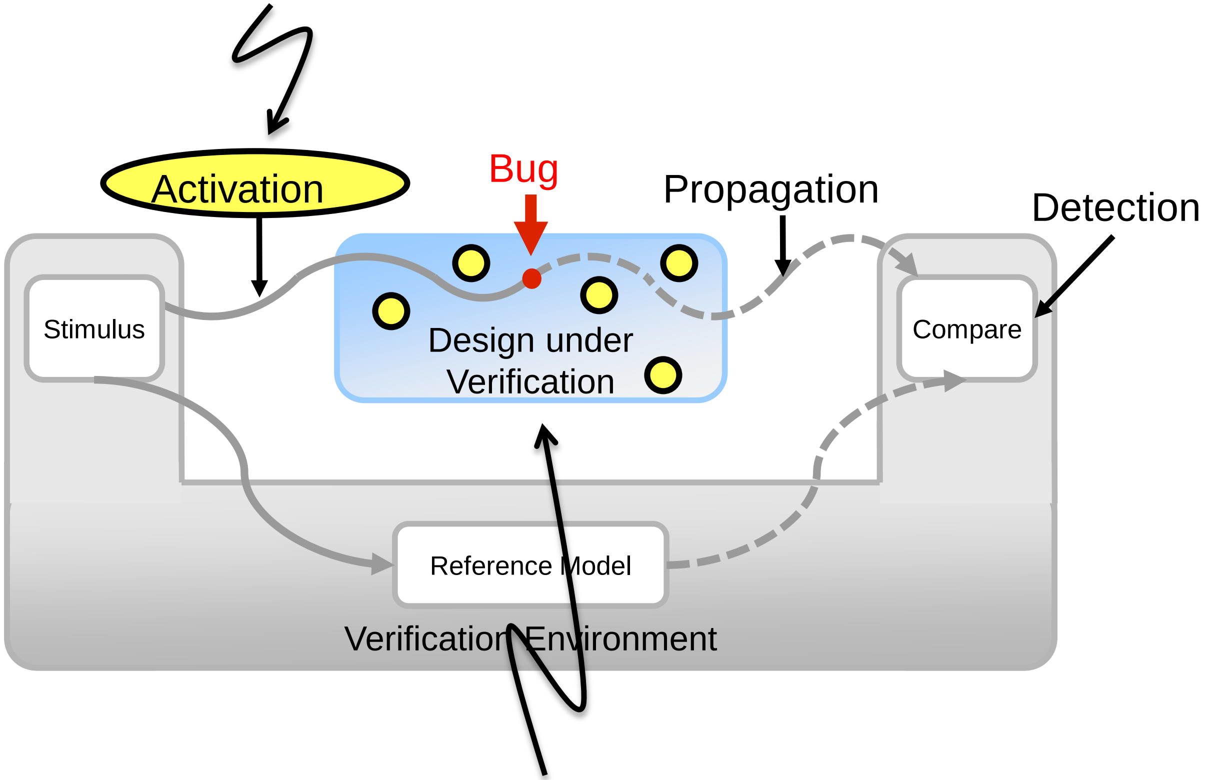


To detect a bug...

- The stimulus must **activate** the buggy logic
- An effect of the bug must **propagate** to an observation point
- The environment must **detect** the behavior difference due to the bug

Existing tools are insufficient

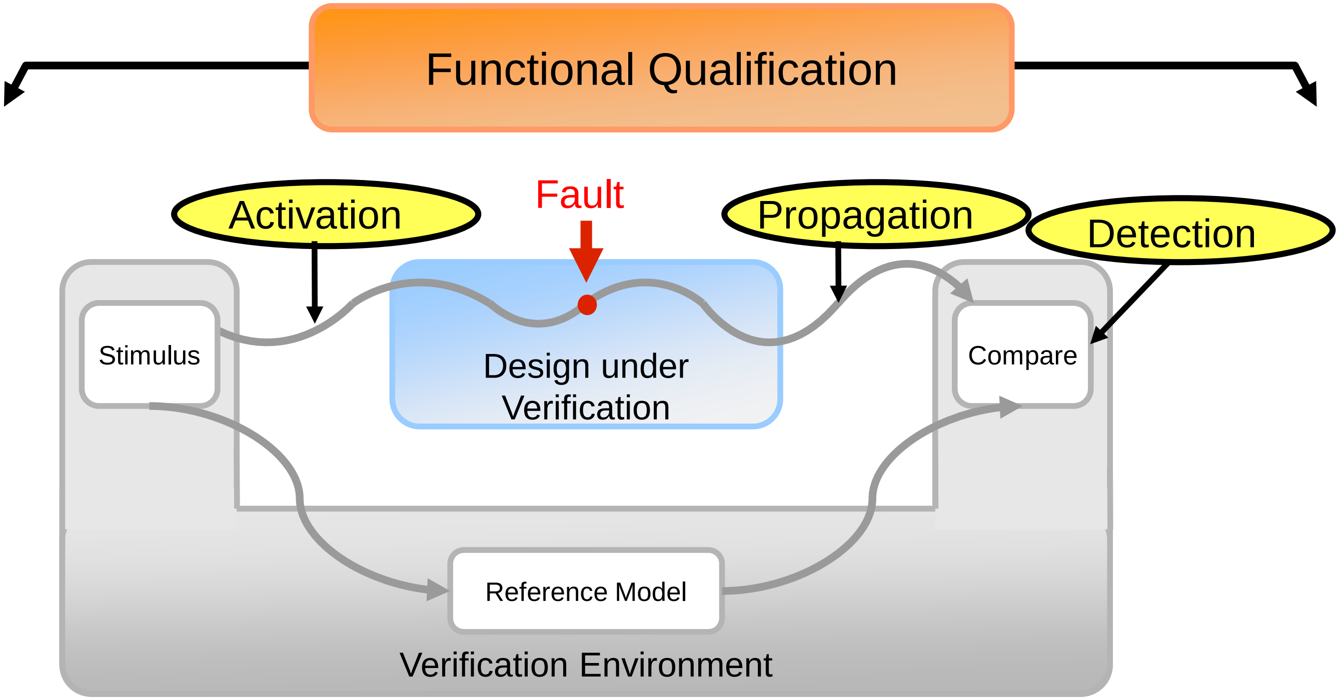
***Code coverage** measures activation, but says nothing about propagation or detection*



***Functional coverage** checks "important" functional points, but is subjective and incomplete*

Functional qualification

- Based on mutation analysis
- Inserts “artificial bugs” (mutations) called faults into the design
- Measures the ability of the verification environment to **activate**, **propagate**, and **detect** the faults
- “Qualification” of the verification environment against many inserted faults provides objective measure of overall quality and identifies holes and weaknesses



How functional qualification works

- Modifies code to insert faults

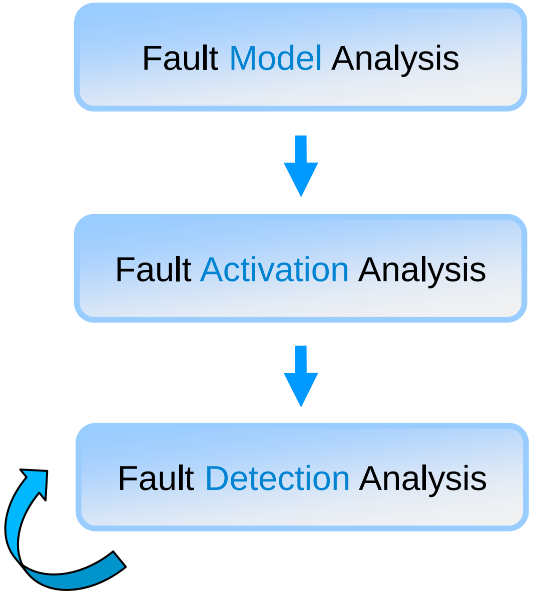
```
a = b | c   →  a = b & c   // change operator

if (a)      →  if (TRUE)   // force execution of “if” branch
  f1();           f1();
else           ~~else
  f2();           f2();~~
```

- Simulates the broken code with the test suite
  - Does at least one test fail? *Great!*
    - The environment is robust enough to detect that the code is broken
  - Do all tests pass? *Help!*
    - You now have two versions of the code, both of which are compliant with the verification environment
    - This means that the environment could miss a real bug

Certitude: a functional qualification tool

- Certitude is a functional qualification tool developed by SpringSoft
- Process and flow:



Fault Model Analysis

Analyze the design (static) to determine the possible faults

- Write instrumented code

Fault Activation Analysis

Analyze the verification environment by simulating tests once

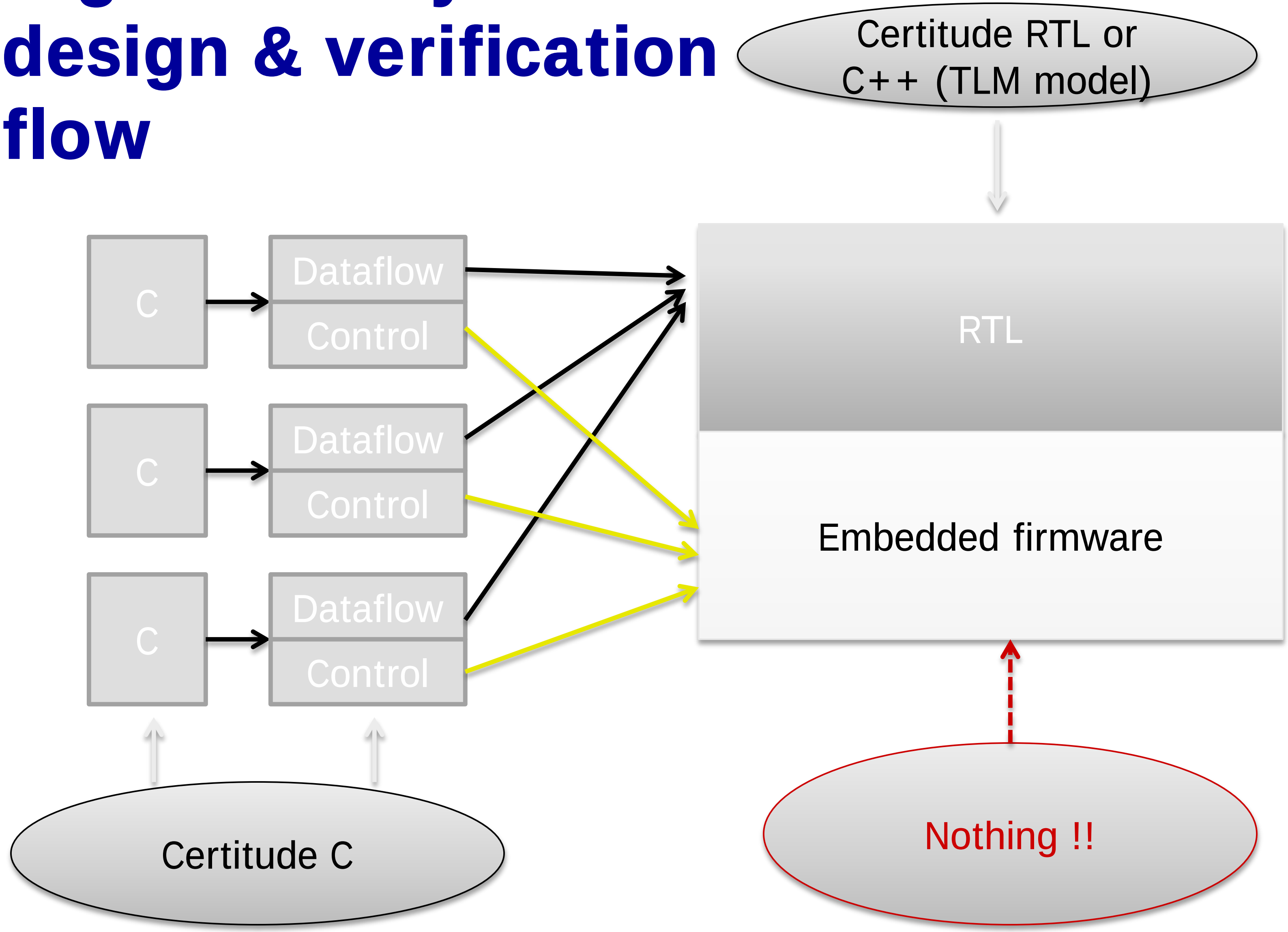
Fault Detection Analysis

Measure the ability to detect bugs by injecting faults and simulating

- ID non-detected faults
- Report details to direct fix

Fix and iterate as problems are found

High Level Synthesis design & verification flow

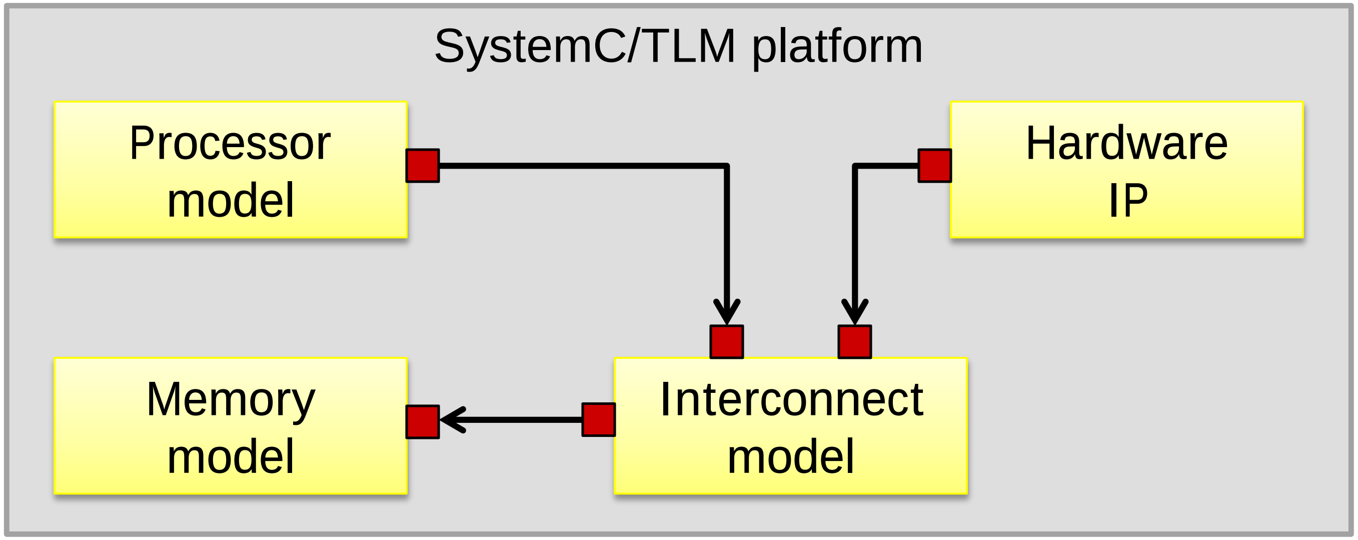


Certitude on embedded software issues

- Certitude is not usable on embedded software in its original version
- Communication issues:
  - Certitude uses control files to:
    - Inject faults
    - Monitor the simulation
    - Get the results
  - But: no file system is available on the embedded platform
- Certitude needs to be extended to support embedded software

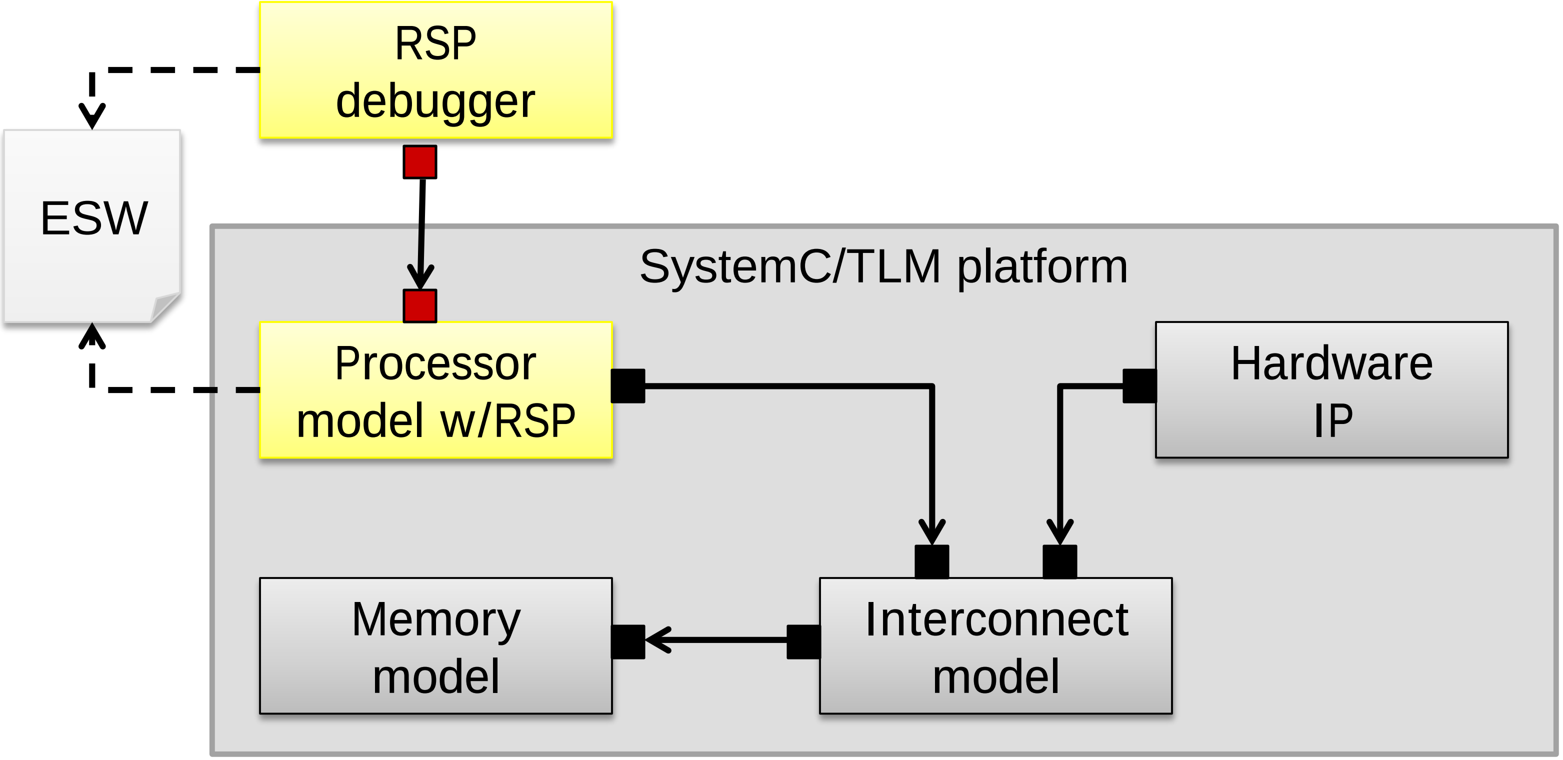
RSP enabled TLM platform (1/2)

- Transaction Level Modeling virtual platform:
  - Allows pre-silicon embedded software development
  - Accurate enough to allow software execution
  - Register-accurate, bit-accurate, loosely-timed
  - Industry standards : SystemC, TLM-2 (IEEE 1666)



RSP enabled TLM platform (2/2)

- Some TLM Processor models provide Remote Server Protocol (RSP) debug access
- Embedded software (ESW) can be debugged from outside



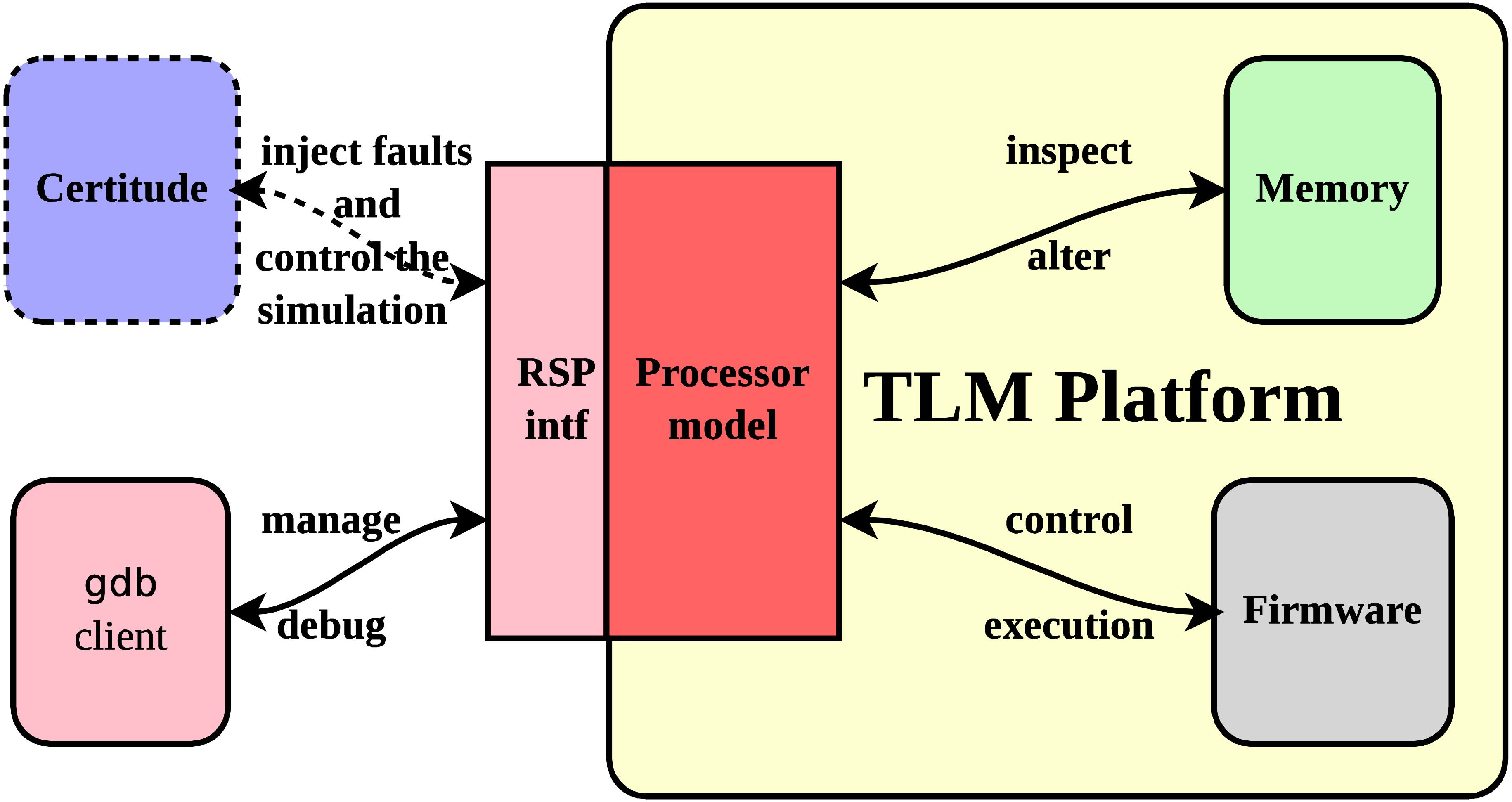
Using Certitude on embedded software (1/2)

- Certitude has been extended to solve the communication issues
- The platform supports the gdb RSP

→ The Certitude control files have been replaced by the RSP

→ Certitude behaves as a standard gdb client

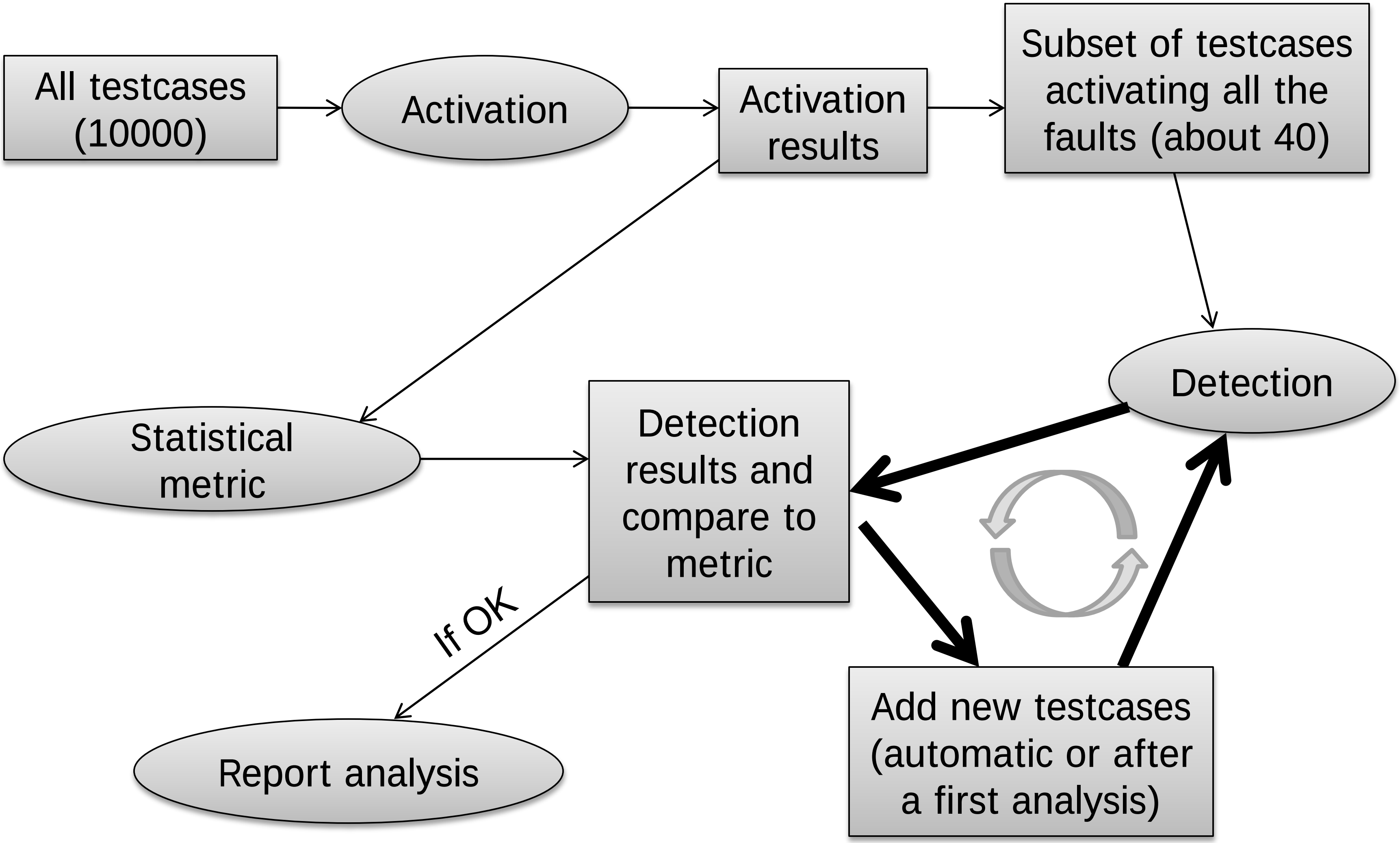
Using Certitude on embedded software (2/2)



Use case: embedded firmware qualification

- DUV is a High Quality Video Display IP
- Embedded firmware mainly implements the control part of the DUV that was formerly done in RTL
- The data flow remains in RTL
- The firmware consists of:
  - 27 files
  - 14,000 lines of C
  - 46.7 kbytes in program memory
  - 6781 faults injected by Certitude

Methodology



All testcases (10000)

Activation

Activation results

Subset of testcases activating all the faults (about 40)

Detection

Statistical metric

Detection results and compare to metric

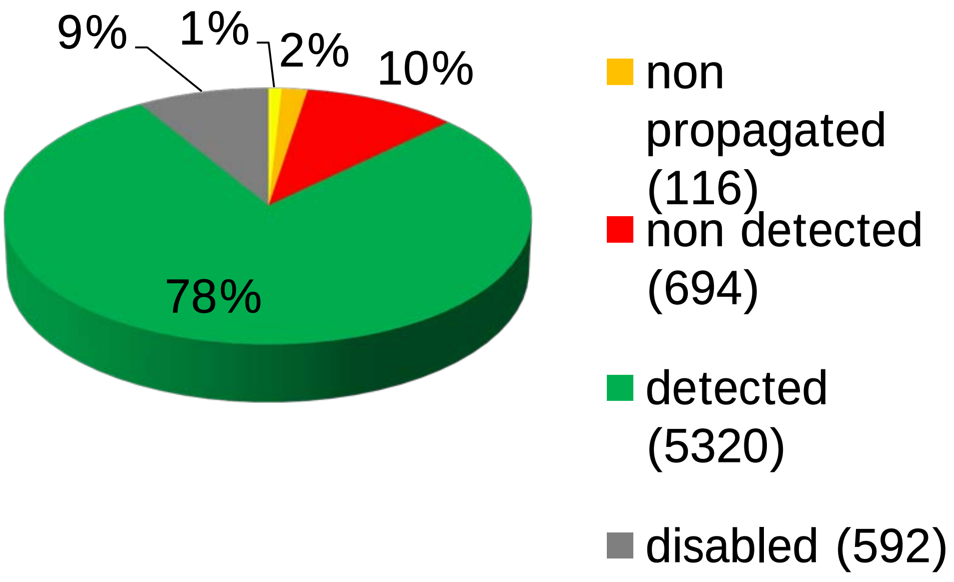
Add new testcases (automatic or after a first analysis)

If OK

Report analysis

Results

Faults status



To be analyzed

Related to timed functionality

Results analysis

- Dead code found: code that cannot be activated or is out of the specification

  -> Save 2% of room in program memory (size limited)

- Missing tests: code that can be activated and that should be activated

  -> Add new tests to cover these functions

  - 4 new tests after activation

    -> find a corner case bug

  - 6 new tests after detection

Conclusion

- Certitude already validated on RTL and C standalone
- Certitude adapted for embedded software testing environments
- Experiments on High Quality Video Display IP:
  - Removed dead code
  - Added missing tests
  - Found a design bug and avoided respin