Infineon

San Jose - February 26, 2013

Quantification of Formal Properties for Productive Automotive Microcontroller Verification

Holger Busch

Infineon Technologies

DVCon Design&Verification Conference&Exhibition

Infineon

Never stop thinking

Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature
- Certitude
  - – General set-up
  - – Coupling with Onespin
- Results
- Conclusions

Goals

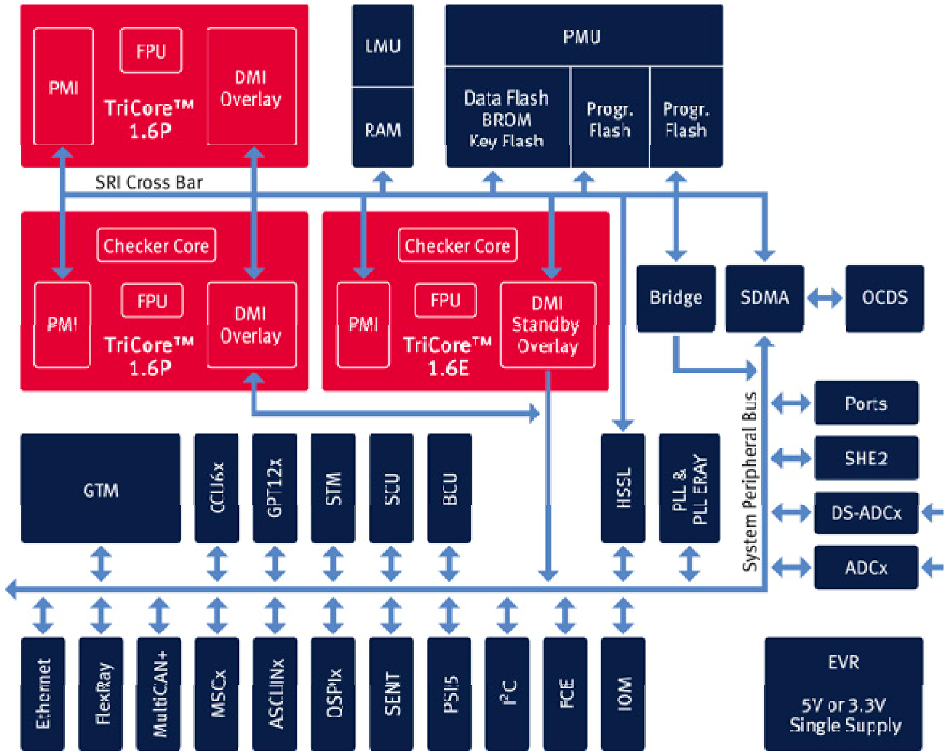
- Quality control ?
  - Each single property 100% checked for all inputs!
    But: specific function potentially uncovered
  - ➢ Assessment of formal property sets needed!
- Formal verification management
  - Progress control
  - Sign-Off Criteria
- Handling of mixed verification tool landscape
  - Directed & constraint driven simulation
  - Formal property checking
- Safety-compliance to ISO26262
  - Traceability of requirements
  - Reproducibility of design and verification process

Overview

- Goals
- Application
- Quantification Approaches
- Onespin's Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin
- Results
- Conclusions

Application: AURIX® µC Family

- Multicore architecture
  - Up to three 32-bit TriCore™ CPUs (up to 300 MHz)
- Single scalable platform for target applications:
  - Powertrain:
    - Engine management
    - Transmission control
    - Hybrid and electrical veh.
  - Safety:
    - Airbag, steering, braking ASIL D (ISO 26262)
  - Driver Assistance:
    - Laser, radar, camera
  - Body:
    - Hardware security



Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin
- Results
- Conclusions

Quantification Approaches

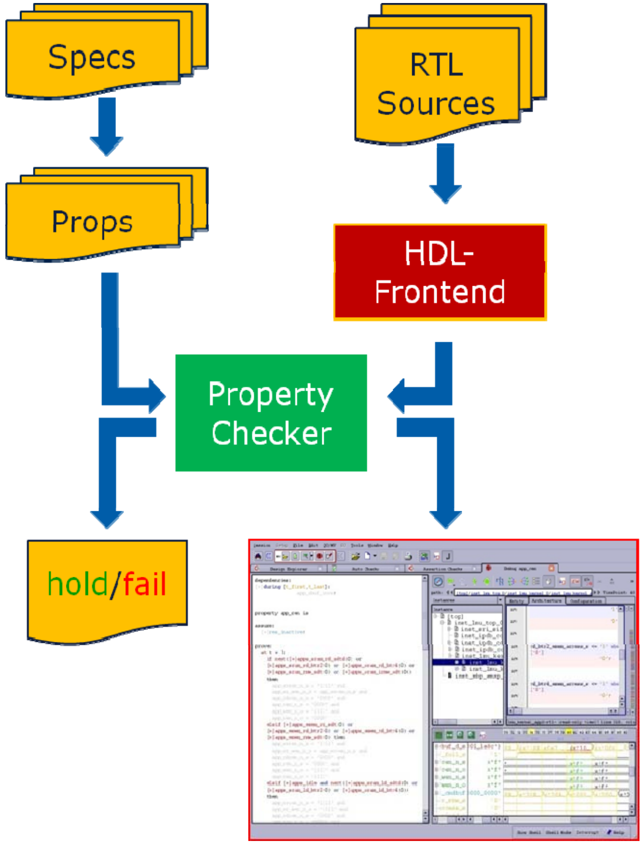
- Manual
  - Review of formal properties
- Formal completeness checks
  - Onespin‘s gap-free verification methodology
  - ➢ Not related to simulation coverage metrics
- Formal witness generation
  - Code coverage for trace: line, branch
  - ➢ Quality of witness ?
- Design mutation
  - Onespin‘s built-in coverage feature Quantify
  - Link to test-bench qualification tool Certitude

Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature
- Certitude
  - – General set-up
  - – Coupling with Onespin
- Results
- Conclusions

Onespin 360°™ MV

- Bounded model-checker
  - Various proof engines
- Property languages:
  - ITL (Interval Language), SVA, PSL
- Consistency checker
  - Dead-code detection, …
- Property debugger
- Coverage
  - Formal completeness checker
  - Line & branch coverage



Onespin‘s Quantify Feature

- Pre-analyses
  - Dead, constrained, redundant code identification
  - Code reachability by witness traces
- Observation coverage:
  - Formal proofs of properties with mutated code locations
  - Code location covered when proof fails
- User-guidance
  - Push-button, focussing possible
- Result
  - XML ~> UCDB-compatible
  - HTML

Onespin's Quantify Feature

Quantify MDV File Result - Konqueror (on vihlc595)

Location Edit View Go Bookmarks Tools Settings Window Help

Location: /home/holger/AIM/AURIX/tc27xb/qs0_smu_html/smu_sw_cntl-rtl-a.vhd.html

Quantify MDV File Result

Overview | Structural Coverage Overview | Structural Coverage by File | Assertion Coverage | File Status | Additional Information

Structural Coverage for smu_sw_cntl-rtl-a.vhd

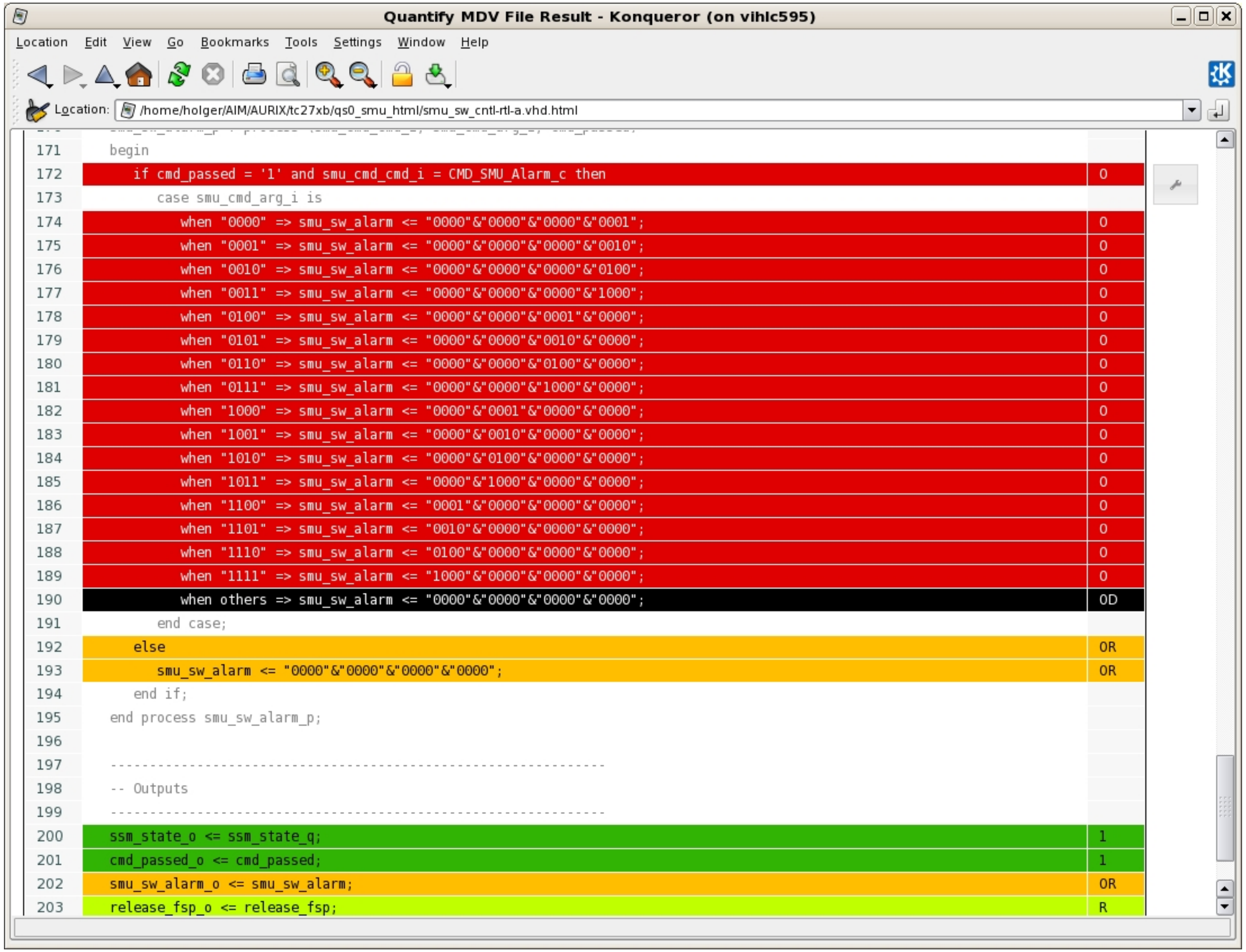
Status		Statements		Branches	
1	covered	8	17.02%	7	16.67%
R	reached	13	27.66%	14	33.33%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	7	14.89%	1	2.38%
0	uncovered	16	34.04%	17	40.48%
0C	constrained	2	4.26%	2	4.76%
0D	dead	1	2.13%	1	2.38%
Sum	quantify targets	47		42	

Annotated Source for smu_sw_cntl-rtl-a.vhd

```
--#-------------------------------------------------------------------
--# File     : smu_sw_cntl-rtl-a.vhd
--# Date     : 16 March 2011
--# Author   : Melody Shi
--# (Modified and improved from smu_ax_kernel v101 from Richard Norwood)
--#
--# Comments :
--#            Design for Aurix TC27x
--#            Copyright(c) Infineon AG 2011 all rights reserved
--#-------------------------------------------------------------------

library work;
use work.p3_smu_fpi_pack.all;
```

Onespin's Quantify Feature



Design & Verification Conference 2013, San Jose, Feb. 26

Quantification of Formal Properties for Productive Automotive Microcontroller Verification

Holger Busch

Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin
- Results
- Conclusions

Certitude

RTL → Certitude → RTL*

```
entity ip is
port (
  a_i: in bit;
  ...)
```

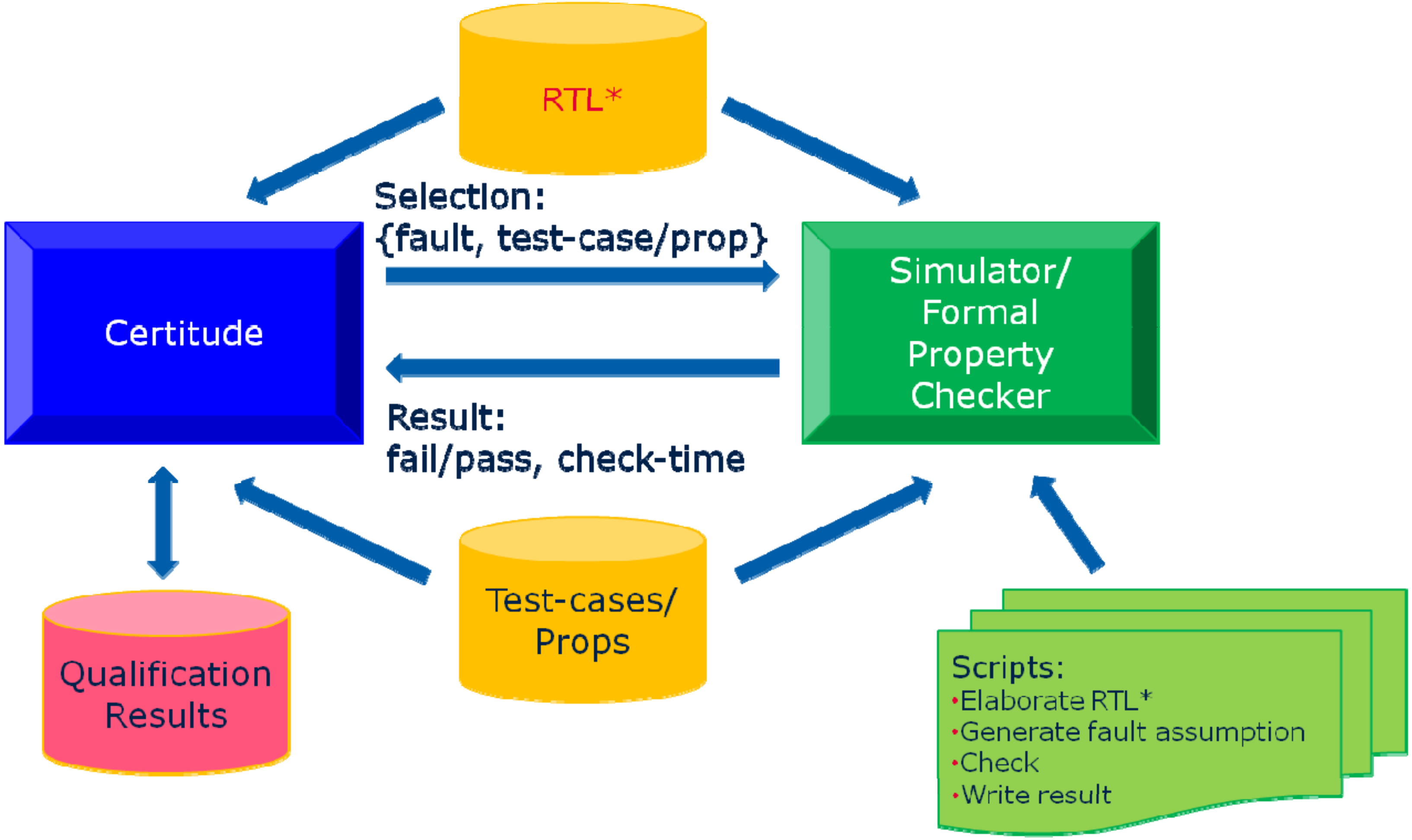
```
...
if x=1 then
  ff <= a_i;
end if;
....
```

```
entity ip is
port (
  f: in bit_vector(0 to n);
  a_i: in bit;
  ...)
```

```
...
if f = 0 then -- no fault
  if x=1 then
    ff <= a_i;
  end if;
elsif f(0) = 1 then
  if x = 0 then -- neg. cond.
    ff <= a_i;
  end if;
elsif f(1) = 1 then
  if x = 1 then
    ff <= not a_i;
  end if;
elsif f(2) = 1 then
  ff <= 0;
...
end if;
...
```

- Principle:
  - Fault-instrumentation of RTL
  - Check fault detection by test-cases

Certitude Qualification Flow



Certitude

- Modeling phase: RTL-code instrumentation by Certitude
  - Different fault models injected into RTL code
  - Top-level entity with additional input vector for individual activation
- Activation phase: Each test-case run once:
  - Activation: test-case stimulus activates fault condition
  - Propagation: fault visible at observation points (DUT interface)
- Detection phase: Analyses for pairs of {fault test-case}:
  - Detection: fail of test-case instead of pass
  - Fault-sets: $F_{\text{injected}} \supseteq F_{\text{activated}} \supseteq F_{\text{propagated}} \supseteq F_{\text{detected}}$
  - Iterative detection controlled by Certitude
- Statistical Approach by Certitude:
  - Metrics computation for statistical samples
- **Application to Formal Properties**
  - Iterative invocation of property checker for formal property instead of simulator for test-case

Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin
- Results
- Conclusions

Certitude <-> Onespin

- Iterative procedure:
  - Let Certitude select:
    - Property `P` from set of qualification properties
    - Fault `c` from current set of non-detected faults
  - Add fault assumption to regular property

Regular Property `P`:	➡	Property `P` with enabling of fault `c`:
`ass(P) |- com(P)`		`1hot(f) ∧ f(c)=1, ass(P) |- com(P)`

  - Check fault-`c`-enabled Property `P` in property checker
  - Return proof result + run-time to Certitude
    - Fail: fault `c` detected by Property `P`
  - Repeat until Certitude is finished:
    - All faults detected

    or

    - All {fault,property}-pairs exercised

Certitude <-> Onespin

- Challenges:
  - Large number of {fault, property}-pairs to be formally checked
    - Estimated full qualification time: $t_{qual} = 0.5 * n_{faults} * n_{props} * t_{check}$
      Example: $n_{faults} = 5000$, $n_{props} = 200$, $t_{check} = 5$ min -> $t_{qual} \approx 9.5$ years
  - Repeated invocation of property checker causes overhead
    - Re-elaboration or loading DUV model
    - Loading properties with current fault assumption
  - Instrumented RTL-design not always clean:
    - Combinational signals become latches for some fault classes
    - Oscillating signals
  - Proof-time for individual check often differs from normal proof
    - More powerful provers invoked if fast prover fails
    - Different provers for counterexample generation

Certitude <-> Onespin

- Reduction of check times
  - Minimization of set-up time before check
    - Keep property checker session open
      - wait for new task sent by Certitude
      - just load new fault constraint
  - Simultaneous property checks (≤ available tool licenses)
  - Selection of proof engines
    - Evaluation of log files
- Reduction of pass-checks
  - Theoretical minimum: $n_{checks} = n_{faults}$ (« $0.5 * n_{faults} * n_{props}$)
  - Selection of {fault, property}-pairs essential
    - Certitude‘s heuristics: Analyze previous results
    - Human knowledge: Relate properties to code partitions
    - Analyses in Formal Property Checker

Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature
- Certitude
  – General set-up
  – Coupling with Onespin
- Results
- Conclusions

Results

Module verification			Quantify		Certitude	
No.	Locs (VHDL)	Props	Code locat.	Days	Faults	Days
1	25563	85	2316	4	1784	7
2	27374	157	1993	5	3732	12
3	57168	253	5309	7*)	4122	17

*) ~ 80% covered, no progress

Comparison

Quantify	Optimized Certitude Qualification
Formal code coverage by internal mutations	Explicit fault injection in RTL design
Closed, indirect controllability	Flexible, full controllability and extensibility
Fast for simple parts, potentially very long computations for more difficult parts; acceptable efficiency for small – medium designs; Open code regions for biggest design, but ongoing improvements by Onespin	Scales to big designs
Restartable, longer setup time	Fast restartability
Onespin session + prover licenses	Onespin session + prover licenses + Certitude license dynamic parallelisation
Code coverage stronger than simulation metrics: formal proofs of observability	Merge with simulation qualification 1:1
Product quality	Packaging of scripts required for wider usage

Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s coverage feature
- Certitude
  - – General set-up
  - – Coupling with Onespin
- Results
- Conclusions

Conclusions

- Two feasible quantification approaches for FPC!
- Both manage big modules with several 10 k loc
- Quantification results largely comparable
- Lots of FPC licenses used
- Long-running properties disadvantageous
- Onespin‘s Quantify:
  - Efficient, but closed
  - Metric similar to simulation code coverage, but stronger
- Certitude-Onespin coupling:
  - Open for optimizations & configuration
  - Exactly same metric for simulation

Thank you!