Complete Functional Verification

Joerg Bormann
joerg.d.bormann@web.de
Contents

- Characterisation of Complete Functional Verification
- Technical Components
- Methodology
- Application Experience

Literature: Bormann: Vollständige funktionale Verifikation (Complete functional Verification), Dissertation, University of Kaiserslautern, 2009
Overview

Complete Functional Verification is a self-dependent verification approach.

- Industrially applied
- For digital synchronous modules (up to ~ 200 k LOC)
- Formal only
- Verifies entire functionality of a module
- Proves that functionality is completely verified

- Alternative to simulation based verification approaches for modules, e.g., coverage driven random pattern simulation
Quality/Cost Metrics For Verification

- Final circuit quality
- Verification cost
  - Human effort
  - Hardware and software usage
- Integration into the industrial environment
  - Processes
  - Mindset
Components of Complete Functional Verification

- Methodology
- Interval Property Checker
- Completeness Checker
- Compositional Completeness Checker
- Operation Properties
**Operation Properties**

- **Observation:** Inputs at specific points in time determine circuit behavior over several succeeding clock cycles.
  - Controller: In the idle state the inputs determine which type of transaction is to be executed next.
  - Pipelines: Inputs to first pipe stage determine behavior of other stages at later clock cycles.

- **Example operations:**
  - Requests of a bus bridge
  - Arbitration cycle
  - Instruction execution of processor
  - In general: Incoming transaction makes circuit switch from one conceptual state to another while producing outgoing transaction(s).
Hardware Model

- Next state function $\Delta$
- Output function $\Lambda$ of primary outputs
- Set $\Sigma$ of reset states

- Traces:
  - Inputs $I$
  - States $S$
  - Outputs $O$

\[ S_0 \in \Sigma, S_{n+1} = \Delta(S_n, I_n) \]
\[ O_n = \Lambda(S_n, I_n) \]

- Automaton $M$ is predicate $M(I, S, O)$ about traces
Operation Properties

A *timed Boolean expression* is a LTL formula using only the operators

\[ P \land Q \mid P \lor Q \mid \neg P \mid X(P) \]

An *operation property* is a LTL formula of the following form

\[ G (SC \land IC \Rightarrow OC \land X^{t_{end}} EC) \]

with SC, IC, OC and EC being timed Boolean expressions.

The sub-formulas of an operation property specify the following conditions:

- **SC**: Start State Condition
- **IC**: Input Condition
- **t_{end}**: duration of the operation
- **EC**: End state Condition
- **OC**: Output Condition

An operation property specifies a behavior where the design makes a transition between conceptual states specified by SC and EC within \( t_{end} \) clock cycles. This transition is triggered by inputs fulfilling IC and produces outputs fulfilling OC.
User-level Language

Syntactic sugar: bounded operators, such as

Time points
\[ t \]
\[ T_i = T_{i-1} + n_i .. m_i \text{ awaits } p_i \]

(uni-versally quantified time point)
(Time of external event \( p_i \), bounded)

Bounded operators
\[ \text{at } T_i + k: \ p; \] (\( p \) holds at time \( T_i + k \))
\[ \text{during } [T_i + k, T_j + v]: \ p; \] (\( p \) holds always in the interval)
\[ \text{within } [T_i + k, T_j + v]: \ p; \] (\( p \) holds once in the interval)
\[ \text{prev(expr), next(expr)} \] (evaluate expr at previous / next time point)

Can be combined with additional techniques for unbounded operators introduced by
\[ T_i \geq T_{i-1} + n_i \text{ awaits } p_i \]
(\( p \) holds at time \( T_i + k \))
(\( p \) holds always in the interval)
(\( p \) holds once in the interval)
(evaluate expr at previous / next time point)

(Time of external event \( p_i \), potentially \( \infty \))
Example

timepoints $T_1 \geq t + 2$, awaits $ack_i = 1$;

| SC  | at $t$: state = idle; | during $[t+1,T_1]$: req_o = 1; |
| IC  | at $t$: start_i = 1;  | at $T_1$: $d_o = prev(d_i)$; |
|     | at $T_1$: error_i = 0; | during $[T_1+1,T_1+2]$: req_o = 0; |
|     |                           | at $T_1+2$: state = term; |

Example

<table>
<thead>
<tr>
<th>state</th>
<th>t</th>
<th>t+1</th>
<th>T1</th>
<th>T1+2</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>start_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>req_o</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ack_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>error_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_o$</td>
<td></td>
<td></td>
<td></td>
<td>$\text{= prev}(d_i)$</td>
</tr>
</tbody>
</table>
COMPLETENESS CHECKER

**Transaction and Operation Automata**

- **Automata with transitions formed by operations**
  - Conceptual states = start and end states
  - The conceptual reset state contains $\Sigma$

- **Transaction Automaton**
  - Transactions treated atomically
  - Similar to transaction level models in simulation

- **Operation Automaton**
  - Transitions = operation properties
  - Cycle accurate representation
  - Less abstraction but more structure

- Complete Functional Verification is an equivalence verification between RTL model and the operation automaton
Completeness Criterium

A set of properties is called complete, if any two circuits fulfilling the properties are sequentially equivalent.

Sequential equivalence check is computationally hard, exacerbated by the indirect description of the circuits by properties.

Operation properties allow to use inductive argument that can be checked quickly and implies completeness.
### Basic Idea

- **Basic idea:** For every input trace, there must be a chain of operation properties $P_0, P_1, \ldots$ that uniquely determines the output trace:

  \[
  \sum \land \prod_{i \geq 1} X^{t_{i-1}} P_i, \text{ where } t_0 = 0, t_i = t_{i-1} + t_{i \text{ end}}
  \]

- **User input for ensuring the existence of chains:**
  - Properties $P_i$ and their duration $t_{i \text{ end}}$
  - Operation Graph
Chain Building Checks

- **Successor Test:** For every property $P$ that is followed by $Q$ in the Operation Graph

\[ EC_P \Rightarrow SC_Q \]

- **Case Split Test:** For every property $P$ and all properties $Q_1, Q_2, Q_3, \ldots$ that follow $P$ in the Operation Graph check that the input conditions of $Q_1, Q_2, Q_3, \ldots$ cover all possible input traces, i.e.,

\[ IC_{free}^{Q_1} \vee IC_{free}^{Q_2} \vee IC_{free}^{Q_3} \vee IC_{free}^{Q_4} \vee \ldots \]

- It follows by induction that a chain of properties exists for every input trace.
Determination Check

- User specifies determination window $w_{\text{outsig}}$ for every output signal and for every property.
- For every property $P$ in the Operation Graph and every output signal, prove on two sets of free variables

$$I = \mathcal{T} \land OC_{\text{free}}^P \land \overset{\sim}{OC}_{\text{free}}^P \Rightarrow \forall \tau \in w_{\text{outsig}}^P : X^\tau(\text{outsig}_\text{free} = \overset{\sim}{\text{outsig}}_\text{free})$$
- Prove that the determination windows are adjacent for every property $P$ and every property $Q$ that can follow $P$ in the operation automaton.
Proof of Assertions

- Assertions capture important verification goals about all operations.
- Use complete property set as design-specific induction scheme.
- Prove assertion \( A \) by

\[
IC^P \land OC^P \Rightarrow \forall_{\tau \in W^P} X^\tau A
\]

for user-defined time windows \( W^P \) that are adjacent in every property chain.
Refined Completeness Criterion

- Use constraints to restrict completeness checking to practically relevant input stimuli.
- Use determination constraints (DC) and determination assumptions (DA) to specify valid data inputs or valid data outputs.
- Notation:
  
  \[
  \text{if } g \text{ then determined}(\text{expr}); \text{ end if; } = (\neg g \land \neg g) \lor \text{expr} = \text{expr}
  \]
- Protocol description by constraints and determination constraints / assertions and determination assertions can describe data transport.

![Diagram](image-url)
INTERVAL PROPERTY CHECKING

„Interval Property Checking (IPC)“: Proving Operation Properties on Bounded Circuit Model

IPC handles large designs (e.g. complete processor core)
- Operation properties can be proven efficiently by SAT/SMT
- Operation properties lead to less difficult reachability problems
Characteristics of IPC

- Operation properties can be proven efficiently by SAT/SMT
  - Proof with arbitrary initial state guarantees unbounded validity
  - Size of iterative circuit model depends on inspection window only
  - Conceptual start state and input conditions simplify SAT model significantly

- Operation properties lead to less difficult reachability problems
  - Reachability information related to conceptual states is intuitive and known to the designer
  - Approximate reachability computation can exploit conceptual states
  - User may safely provide additional reachability information
  - … but there are doomed circuits!
COMPOSITIONAL COMPLETENESS CHECKER

Literature: Bormann: Vollständige funktionale Verifikation (Complete functional Verification), Dissertation, University of Kaiserslautern, 2009
Problem

Given two circuits $M_1$ and $M_2$, each completely verified with
- a property set $\Pi_1$ and $\Pi_2$,
- constraints $C_i \land C'_i$, determination constraints $DC_i \land DC'_i$, $i = 1,2$
- assertions $A_i \land A'_i$ and determination assertions $DA_i \land DA'_i$.

Question: When is the composite circuit completely verified by
- the property set $\Pi_1 \cup \Pi_2$
- constraints $C'_1 \land C'_2$, determination constraints $DC'_1 \land DC'_2$
- assertions $A'_1 \land A'_2$, determination assertions $DA'_1 \land DA'_2$?
Assume/Guarantee Basics

- Subcircuits verified.
  - Constraints C1, C'1, C2, C'2
  - Assertions A1, A'1, A2, A'2
- When is the composite circuit completely verified?
  - Constraints C'1, C'2
  - Output Assertions A'1, A'2
- Problem: Cyclic reasoning
- Assume-Guarantee-Theory allows cyclic reasoning
  - If constraints depend only on input signals
  - No combinatorial loops
  - A1 \rightarrow C2 and A2 \rightarrow C1

How to deal with reactive constraints (depend on outputs)?

How to deal with Completeness?
Implementable Constraints

- Many constraints restrict inputs depending on outputs.
  - e.g. protocol constraints
- Assume-Guarantee-Reasoning allows constraints that depend on outputs, but the constraints must satisfy some requirements:
  - No restriction of outputs
  - No examination of output values in the future
- Application specific characterization: Constraint must be implementable
  - Existence of reference circuit that implements the constraint
  - No combinatorial feedback loop through real and reference circuit
Compositional Complete Verification

- Reduce to the usual Assume-Guarantee-Problem
- Consequence: $C_i \land \tilde{C}_i \land DC_i$ must be implementable ($i = 1,2$)
- No combinatorial loops through reference circuit and $M_j$
- Moreover, $A_i \land \tilde{A}_i \land DA_i \Rightarrow C_i \land \tilde{C}_i \land DC_i$, $i = 1,2$
METHODOLOGY

Operation Properties and the Methodology

- Simple structure
  - Simple language eases formalization of intuition
  - Guidance by the 4 timed predicates
- General applicability
  - Only a focussed set of skills required
- Structuring of the verification task
  - Detailed general examination of one operation at a time
  - Counter examples show different aspects of the same mechanism
- Tool advantages
  - Proof times of 5 min. or less allow interactive use
  - Partitioned completeness check allows to obtain partial results
How a Verification Proceeds

For a representative subset of functionality

- Identify sequences of states in central controller
  - Ideas for start and end state conditions
- Identify conditions to primary inputs for these sequences
  - Input conditions
- Examine the output behavior
  - Output conditions

Use already developed properties as templates for the verification of the remaining functionality
Characteristics of the Methodology

- Follows a common process to familiarize with RTL code
- Quick validation of guesses by property checking
- Counter examples help refining guesses
- Unambiguous yet intuitive description of aspects of functionality

- Automated examination of coverage
- Provision of unexamined situations by completeness checker
- Automated non-heuristic termination criterion

- Allows reliable planning
- Verification planning is usual project planning. No need for comprehensive lists of verification goals
PRACTICAL EXPERIENCE

Application Experience

**Processors**
- TriCore2 (superscalar, 32 Bit)
- Multithreaded network processor
- IEEE floating point processor
- Weakly programmable IP

**Bus Interfaces**
- AHB (master IF, slave IF, bridges, multilayer)
- CAN, LIN, Flex Ray, AXI, SRC Audio bus IF
- Network-On-Chip
- HDLC Controller

**Peripherals**
- USB master interface, Counter/Timer
- UART, Interrupt Controllers, A/D Converter Controller, Flash Card Data Port
- configurable Arbiter, DMA Controller

**Memory Interfaces**
- SDRAM Controller, SATA, Caches
- Flash Memory Interface

**Telecom**
- AAL2 Termination Element
- Address management in ATM Switch
- Sonet / SDH Frame Alignment
- Path Overhead Processing of Multi-Gigabit-Switch
- DSP coprocessor ASIC for correlation computation
Quality & Cost

<table>
<thead>
<tr>
<th>Bugs missed because</th>
<th>Simulation</th>
<th>Complete FV</th>
</tr>
</thead>
<tbody>
<tr>
<td>not stimulated</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>no checker</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>duplication in RTL and verification code (function or constraints)</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

- **Human effort:** 2-4 kLOC RTL code per person month for an expert.
  - May require 2 years to become expert.

- **Hardware / Software cost:**
  - Simulation: Software System consisting of simulator, testbench automation tool, bus functional models, application specific software, …
  - Simulation: Occupies compute farms over weeks
  - Complete functional verification: 1 property Checker, 1 completeness checker, possibly a debugging support tool
Integration Into Industrial Processes

Approach provides:
- Good error localization
- Non-heuristic termination criterion – sign off documentation
- Tolerance wrt. specification quality
- Verification planning / monitoring is usual project planning / monitoring
- Integration to system level verification by checking constraints during system level simulation
- Operations provide coverage base for system verification.
- Provable Design Documentation

Approach demands:
- Provision of white box information, e.g., by designers
Summary

**Methodology**
- Termination criterion, high productivity, high quality, user guidance

**Interval Property Checking (IPC)**
- Special treatment of reachability problems

**Completeness Checker**
- Verification without gaps

**Compositional Completeness Checker**
- Unbounded circuit sizes

**Operation Properties**
- Transaction oriented assertions