

VERIFICATION COOKBOOK

Tomáš Kratochvíla
krata@liberouter.org

Easy Communication

When you want to verify any property, you can include a comment to your VHDL code in some of the following form:

- Very elegant form:

```
-- assert "informal description in english"
```

Example of property that the stack never overflows:

```
-- assert The stack never overflows.
```

- For more complicated properties there could be added a lemma with an attached letter (possibly in czech language):

```
-- assert LEMMA unique_mark
```

```
To: ipv6-ver@muni.cz
Subject: LEMMA file.vhd-unique_mark
Content: Description of properties of your VHDL code.
```

Example of verification that the signal named `DATA` in `hfe_core.vhd` change every 8 tick of clock:

```
-- assert LEMMA signal-DATA
```

```
To: ipv6-ver@muni.cz
Subject: LEMMA hfe_core.vhd-signal-DATA
Content: The signal named DATA change its value every 8 tick of clock.
```

- Description of properties in the VHDL boolean expressions (extended by implication, cycle, branching and more), which used to be always true:

```
-- assert globally extended_VHDL_bool_expression
```

Whenever are BUF1 or BUF2 up so needs must be REG6 up:

```
-- assert globally if (BUF1 or BUF2) then REG6
```

- In a boolean expression could be described signals, ports, variables or registers, which never falls to have constant value.

```
-- assert alive boolean_expression
```

To verify that FULL-SIGNAL never falls to be constant:

```
-- assert alive FULL-SIGNAL
```

We take no notice about another occurrence of word `assert`.

Typical Properties we are able to verify

- **Invariants, Reachability and Mutual Exclusion**
- **Liveness** (signal, port, variable or register changes its value forever)
- **Fairness** (every request will be eventually granted)
- **Temporal Behaviour** (every request will be granted in given amount of time)

Invariants, Reachability and Mutual Exclusion

- Globally during computation have to hold WRITE_RDY or READ_RDY:

```
-- assert globally (WRITE_RDY or READ_RDY)
```

- State when WRITE_RDY = 9 and READ_RDY = 1 is reachable via a computation:

```
-- assert in future ((WRITE_RDY = 9) and READ_RDY)
```

- Mutual exclusion of LUP1_ACCESS, LUP2_ACCESS, LUP3_ACCESS:

```
-- assert exclusion LUP1_ACCESS, LUP2_ACCESS, LUP3_ACCESS
```

Liveness and Fairness

- Something eventually happens.
- Globally there is an eventuality that READY will hold:

```
-- assert alive READY
```

- Every request will be eventually granted.
- If WRITE_REQ then WRITE_RDY will hold:

```
-- assert globally (if WRITE_REQ then READ_RDY in future)
```

Temporal Behaviour

- Every request will be eventually granted in 15 steps.
- If WRITE_REQ then WRITE_RDY will hold in 15 steps:

```
-- assert globally (if WRITE_RDY then READ_RDY in 15 steps)
```

- WRITE_REQ until WRITE_RDY or RESET:

```
-- assert WRITE_REQ until (READ_RDY or RESET)
```

Advanced Communication

Are you sure of what you want? Do you wish the best effect? Try to use the following constructions in your VHDL code:

- Command `assert` in VHDL language. We can verify this assertions better than your simulator – we will check every posible run of the system.

```
assert condition [report error_string] [severity severity_value]
```

- Exact description in LTL language, which syntax is thereafter.

```
-- assert LTL ltl_formulae
```

- Exact description in CTL language.

```
-- assert CTL ctl_formulae
```

LTL Syntax

$$\varphi = true \mid a \mid not \varphi \mid \varphi_1 \text{ or } \varphi_2 \mid \varphi_1 \text{ and } \varphi_2 \mid X\varphi \mid \varphi_1 \text{ until } \varphi_2$$

a is of signal, port, variable or register.