SystemC Assertion Library

SystemC Users Group
22 August 2005

Jacob Smit, Michael Velten, Volkan Esen
Thomas Steininger, Wolfgang Ecker, Michael Mrva
Infineon Technologies

Motivation

- SystemC gains more and more momentum in the SystemLevel domain. It is widely used for “cycle callable”-, “timed transaction”-, “untimed transaction”-, and “functional”-models.
- As shown to be useful for HDLs (Verilog, VHDL, SystemVerilog), assertions are also needed for SystemC to improve quality of the models and to shorten verification time
- Library approach as first step towards SystemC assertions

Related Work

- SystemC Assertions
  - The recent SystemC version provides assertion capabilities
  - They are restricted to immediate boolean checks
- Existing Assertion languages are used in conjunction with SystemC
  - PSL (in a Verilog or SystemC Style), SVA, OVA
  - Designers need to learn an assertion syntax in addition to the HDL language and a language approach is mixed with the class library approach of SystemC
- Temporal Class extension
  - Class library from Jeda as commercial add on
- Libraries
  - OVL (no SystemC support), Checkerware (0in)

Contents

Introduction

- Development of a generic, fully parametrisable assertion library for SystemC (as on OVL) as first and pragmatic approach to provide temporal assertions in SystemC
- Assertions are implemented as SystemC modules
- All assertions have a consistent structure with the same parameters
- Assertions implemented at cycle callable (cycle true) level

Assertion parameters

- Generic parameter examples
  - Clock : falling/rising edge
  - Reset : active low/high, sync/async/none
  - Enable : active low/high
  - X-check : yes/no
  - Severity level : info/warning/error/fatal
- Generics passed in using templates
  - Avoid use of preprocessor commands
  - Necessary to avoid Dynamic Memory Allocation
  - Stricter type checking using templates

Assertion parameters

Header file of the assertion (assertion_header.h)

```
enum RESET_TYPE {NORESET, SYNC, ASYNC};
enum ACTIVE_LEVEL {HIGH, LOW};
enum CLOCK_EDGE {RISING, FALLING};
enum ASSERT_TYPE {NOTE, WARNING, ERROR, FAILURE};

template< int WIDTH >
```

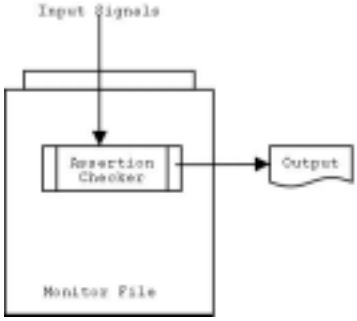
Assertion parameters

Generics using templates

```
#include "systemc.h"
#include “assertion_header.h"

template<int m_width,
        RESET_TYPE m_rst_type,
        ACTIVE_LEVEL m_rst_active,
        CLOCK_EDGE m_clk_edge,
        ASSERT_TYPE m_XCHECK,
        ACTIVE_LEVEL m_enable_active,
        ASSERT_TYPE m_assert_level>
```

Assertion structure

- Structure of the checkers
  - Assertion module as a monitor
  - Signals are passed into the module
  - One thread handles the assertion
  - Assertions written using
    - SC_REPORT_INFO
    - SC_REPORT_WARNING
    - SC_REPORT_ERROR
    - SC_REPORT_FATAL



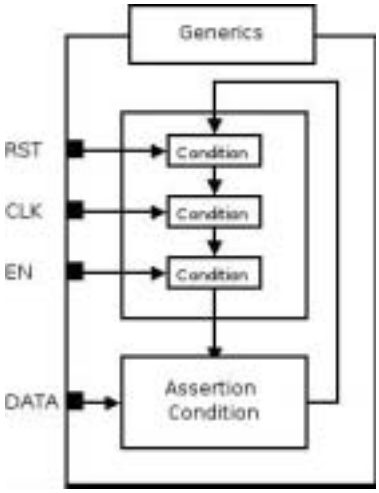
Assertion structure

Interface of the assertion

```
class one_hot : public sc_module
{
public:
        sc_in< sc_lv<m_width> >  data;
        sc_in_clk clk;
        sc_in<sc_logic> rst;
        sc_in<sc_logic> enable;

        void process()
        {
                sc_uint<m_width> tmp_data;
                sc_uint<m_width> n_tmp_data;
                const  sc_uint<m_width> m_ZERO = 0;
```

Assertion structure

- Structure of the checkers



Assertion structure

Handle reset and enable behaviour

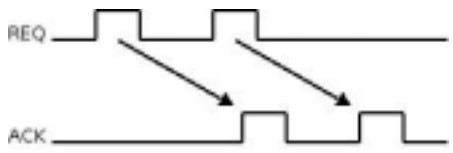
```
while(1) {
    wait(); // Depends on sensitivity set earlier
    if ((m_rst_type != NORESET)
    && (((m_rst_active == HIGH) && (rst == SC_LOGIC_1))
    || ((m_rst_active == LOW) && (rst == SC_LOGIC_0)))) {
code
    }                   // Reset
    else {
          if (((m_enable_active == LOW) && (enable == SC_LOGIC_0))
          || ((m_enable_active == HIGH) && (enable == SC_LOGIC_1))) {

                tmp_data = data;
                n_tmp_data = ~tmp_data;
```

Assertions types

- Boolean
  - Asserts boolean expressions on clock edges
  - Example : one hot
- Transitional
  - Two cycle check using a combination of boolean expressions and past signal values
  - Example : gray code

Assertions types

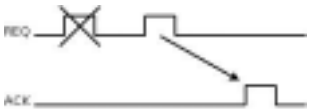
- Multicycle
  - Involves assertions that are dependant on past and future signal values within a variable time window
  - Example : request acknowledge
  - These tests can be pipelined



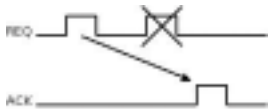
Pipelined : Each request must be acknowledged

Assertions types

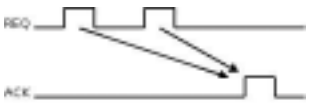
- More Multicycle Modes



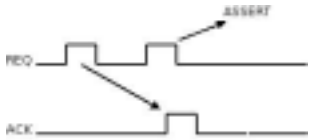
Restart : Ignore first request



Ignore : Ignore second request



Respond all : One acknowledge answers all requests



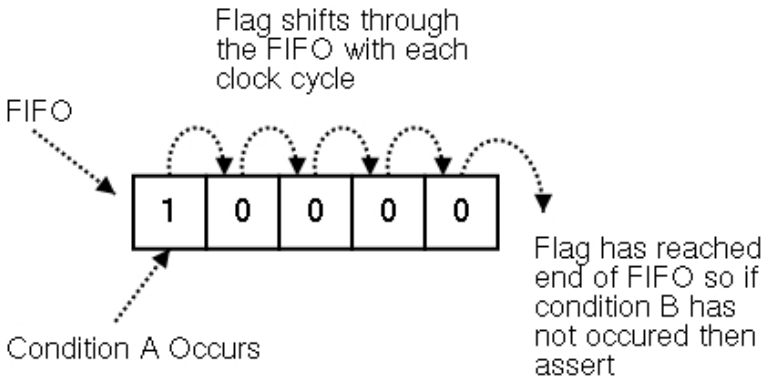
Assert : Second request cannot occur until the first one has been answered thus assert

Assertion Implementation

- Transitional / Multicycle Assertions
  - Past Values
    - Use FIFOs to store previous values to a depth determined at compile time
  - Future Values
    - Assume condition A implies that condition B will occur during some time window
    - When condition A occurs push a check flag onto a FIFO of a suitable length that shifts along with each clock cycle
    - If condition B has not occurred when the check flag exits the FIFO then assert

Assertion Implementation

- FIFO functional diagram



Assertion Implementation

- Implemented Assertions

One hot	Req Ack Pipelined
Zero One Hot	Req Ack Ignore
One Cold	Req Ack Restart
Out of Range	Req Ack Assert
Valid Opcode Pattern	Req Ack Respondall
Valid Pattern	Conditional
Grey Code	Sequence Implication
Past Value	Future Value
Forbidden Sequence	

Assertion structure

Condition of the One Hot checker

```
if (((tmp_data - ((sc_uint<m_width>) 1)) & tmp_data) == m_ZERO) {
}
else {
    if (m_assert_level == NOTE)
        SC_REPORT_INFO("ID", "OneHot Violation");
    if (m_assert_level == WARNING)
        SC_REPORT_WARNING("ID", "OneHot Violation");
    if (m_assert_level == ERROR)
        SC_REPORT_ERROR("ID", "OneHot Violation");
    if (m_assert_level == FAILURE)
        SC_REPORT_FATAL("ID", "OneHot Violation");
}
```

Summary

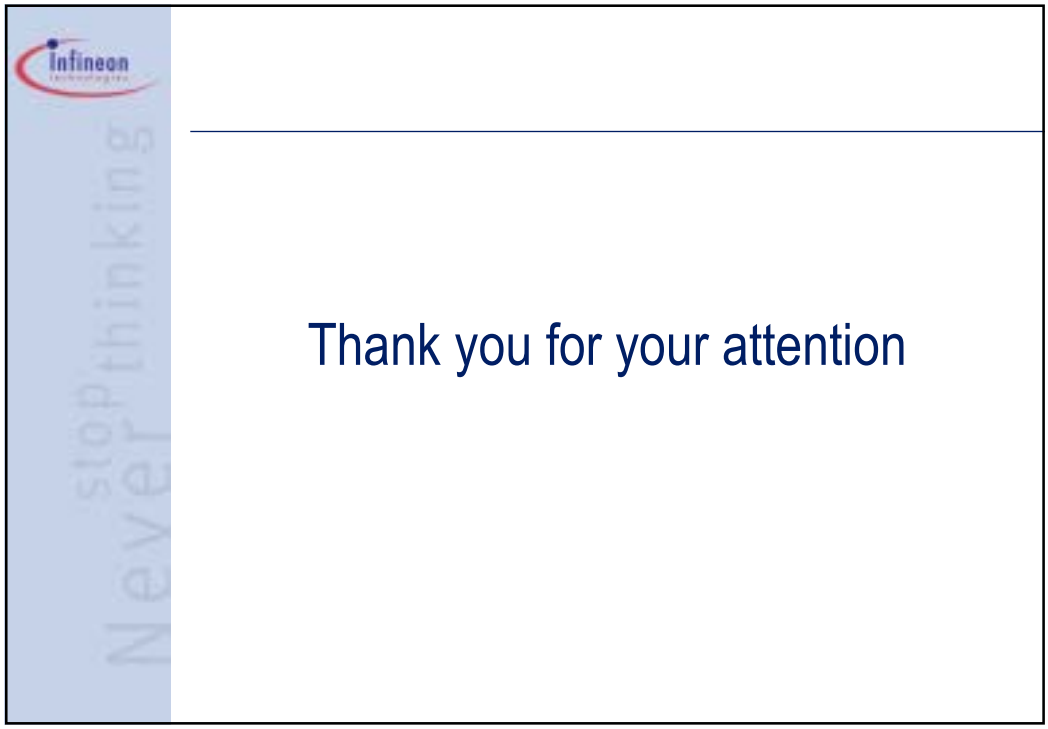
- Library approach to make assertions available in SystemC in an easy way
- Class library approach of SystemC for language definition taken
- A first set of checkers implemented
- Performance …

Future Work / Outlook

- Functional Coverage
  - Post processing Value Change Dump files
  - Analysing collected data (e.g. coverage counters) at the end of simulation
- Level Independence
  - Checkers for all levels of abstraction
  - Synthesisable checkers
  - Monitors for self timed / untimed systems

Future Work / Outlook

- Automatic Assertion Generator
  - The consistent structure of the modules allows an easier development of such a tool
  - Universal Assertion Description Language (UADL)
    - XML based processing
  - Could be used to generate assertions in other HDLs e.g., VHDL, Verilog, SystemVerilog


Infineon
never stop thinking
Thank you for your attention