Static Analysis of SystemC/SystemC-AMS System and Architectural Level Models

Karsten Einwich, Thilo Vörtler
COSEDA Technologies
Outline

• Motivation
• Generic Concept
• Framework Implementation in SystemC
• Analyzer Examples
• Application Example
• Result Visualization
• Summary
Motivation

• Numerous properties can be analyzed/estimated statically based on structural information using simple formulas

• Static analysis is extremely fast compared to extracting the property by a dynamic simulation

• Very often complex excel sheets are used to perform static analyzes

• Difficult to maintain consistency between excel sheets and system implementation

• Supporting of different configurations, operating modes or settled states becomes extremely complex

• Increasing complexity of the structure of the system limits the excel approach
Examples for Static Analyses

- Power Consumption (in different operating modes)
- Chip Area
- Timing Analysis
- Gain / Signal level
- Noise
- Interception Points (IP2/IP3)
- ...
Generic Concept - Principle

• Attach the static properties to the modules of a SystemC/SystemC-AMS based simulation model
• Analyze the properties using the hierarchical structure, connectivity and the current state of the model
Generic Concept – Mapping to SystemC

• Static properties are represented by a class derived from a static analyzer attribute base class which is derived from the SystemC `sc_core::sc_attribute`.
• The SystemC standard mechanism is used to attach the attributes to objects of class `sc_core::sc_module`.
• The ports of the module are categorized as in- or outport and referenced to the static property.
• The attached properties are analyzed using the SystemC hierarchy exploration methods (e.g. `get_parent_object`) and for analyzing the connectivity the port base class method `get_interface`.
• The static property contains a callback to calculate and propagate the property value in dependency of pre-decessor values and the current module state.
Generic Concept – Classes of Analyzers

• Analyzers can be principally categorized in three categories:
  1. Hierarchic Analyzer
     • Analyzes and combines properties of a hierarchic level and propagates them through the hierarchy
     • Examples: Power, Chip Area
  2. Dataflow analyzer
     • Analyzes/combines values along a datapath
     • Examples: gain, noise, IP2/IP3
  3. Equation system based Analyzer
     • An equation system is setup and solved based on the system structure, the properties representing contributions to the equation system
     • Examples: Settled value, AC-analysis

• The presented framework supports directly analyzers of class 1 and 2
class cos_analyzer_attr_base : public sc_core::sc_attribute<cos_analyzer_value>
{
    public:
        cos_analyzer_attr_base(sc_core::sc_module* parent);

        // access to number of predecessors / successors for a dataflow analysis
        unsigned long get_number_of_inports() const;
        unsigned long get_number_of_outports() const;

        // predecessor/successor value access/propagation
        template<class T>
        void set_outport_value(const T& val, long o);

        template<class T>
        T get_inport_value(unsigned long i);

        template<class T>
        T get_module_value() const;

        // value callback function to propagate the value
        virtual void value_function(){}
};
class cos_static_analyzer_base
{
    public:
        virtual void analyze() = 0; //callback to perform the analysis

        //start dataflow analysis
        void analyze_datapath(const std::vector<std::string>& starting_points);
        void analyze_hierarchy();  //start hierarchic analysis

        //utility functions for generic analysis result access
        //for hierarchic analyzis
        const vector<tree_analysis_result_element>& get_tree_analysis_result();

        //for dataflow analyzis
        const vector<std::vector<cos_analyzer_attr_base*> >& get_clusters();

        //callback function to recognize attributes which belong to
        //a concrete analyzer
        virtual bool is_type(cos_analyzer_attr_base*) = 0;
};
Analyzer Example – Power Analyzer

• Calculates/Estimates the power consumption in the current system state
• A power analyzer sums the annotated power values of one hierarchy level and attaches it to the corresponding instance of the next hierarchy level
• -> Hierarchic Analyzer
Analyzer Example – Power Analyzer - Attribute

class cos_analyzer_attr_power : public cos_analyzer_attr_base

public:
    cos_analyzer_attr_power(sc_core::sc_module* mod, double val);

    void set_value(double val);
    double get_value() const;

    // datastructure to hold the values for the analysis
    struct cos_budget_value : public cos_value_base
    {
        double value=0.0;
    } value;

    // call back function for calculating the power
    void value_function(cos_value_base& previous)
    {
        auto res=dynamic_cast<cos_power_value*>(&previous);
        // sum power values
        res->value+=this->get_value();
    }

© Accellera Systems Initiative
Analyzer Example – Power Analyzer - Implementation

class cos_static_analyzer_power :
    public cos_static_analyzer_base
{
    public:
        //perform analysis
        void analyze() override;

    //print results to shell
    void print_results(std::ostream& str=std::cout);

    private:

        //callback for type recognition
        bool is_type(cos_analyzer_attr_base*) override;
    };

void cos_static_analyzer_power::analyze()
{
    this->analyze_hierarchy(); //perform hierarchic analysis
}

bool cos_static_analyzer_power::is_type(cos_analyzer_attr_base* attr)
{
    return (dynamic_cast<cos_analyzer_attr_power*>(attr)!=NULL);
}

void cos_static_analyzer_power::print_results(std::ostream& str)
{
    auto& result=this->get_tree_analysis_result();
    double overall_power=
        result[0]->get_value<cos_analyzer_attr_power::cos_power_value>().value;
    str << "Overall power: " << overall_power << " W" << std::endl;
    for(auto& modp : result)
    {
        double mod_value=
            modp->get_value<cos_analyzer_attr_power::cos_power_value>().value;
        str << "\t" << modp->get_object()->name() << " :\t" << mod_value << " W" << std::endl;
    }
}
Application Example – Attaching Attributes

• The Framework supports different mechanism for attaching the attributes (for details see paper)
• Simplest way is to attach the attribute in the module constructor

```cpp
SC_MODULE(temp_sensor)
{
    ...
    SC_CTOR(temp_sensor)
    {
        pattr=new cos_analyzer_attr_power(this);
        pattr->set_value(1e-3); // set power consumption to 1mW
    }
    ...
    cos_analyzer_attr_power* pattr=NULL;
};
```
Application Example – Starting Analyzis in sc_main

• An analysis can be started/re-started any time after DUT instantiation

```c
int sc_main(int argc, char* argv[])
{
    ...
    dut_tb* i_dut_tb;
    i_dut_tb = new dut_tb("i_dut_tb");
    ...
    cos_static_analyzer_power panalyzer;
    panalyzer.analyze();
    panalyzer.print_results();
    ...
}
```
Application Example - Result

Overall power: 4.31e-002 W

\[
\begin{align*}
  i_{\text{hierarchic_toplevel}} & : 4.31e-002 \text{ W} \\
  i_{\text{hierarchic_toplevel}.i_{\text{cw_top_start_1}}} & : 3.00e-003 \text{ W} \\
  i_{\text{hierarchic_toplevel}.i_{\text{cw_top_start_2}}} & : 2.00e-003 \text{ W} \\
  i_{\text{hierarchic_toplevel}.i_{\text{gain_nl_gen1}}} & : 5.00e-004 \text{ W} \\
  i_{\text{hierarchic_toplevel}.i_{\text{gain_nl_gen2}}} & : 1.00e-003 \text{ W} \\
  i_{\text{hierarchic_toplevel}.i_{\text{muls_gen1}}} & : 5.00e-004 \text{ W} \\
  i_{\text{hierarchic_toplevel}.i_{\text{sub_module1}}} & : 3.30e-003 \text{ W} \\
  i_{\text{hierarchic_toplevel}.i_{\text{sub_module2}}} & : 3.30e-003 \text{ W}
\end{align*}
\]
Result Visualization

• The framework supports the creation of proprietary formats for result backannotation to schematics
• Allows coloring in dependency of the result
• Visualization of signal paths, paths with now influence or side paths with influence
Result Visualization – Example Power Analysis
Result Visualization – Noise figure for two states

Control signal : 0

Control signal : 1
Summary

• Static analysis is a powerful method to estimate system properties
• Especially suited for design space exploration
• Can be used to calculate expected „ideal“ values
• The introduced framework allows to establish consistency between static analysis and the model and thus also to the implementation
• The framework supports the analysis in dependency of the current system state and thus the calculation of the properties for this configurations
• The results can be backannotated and visualized in schematics
Questions