Efficient Debugging on Virtual Prototype using Reverse Engineering Method

Sandeep Puttappa, Senior Staff Engineer
Dineshkumar Selvaraj, Lead Principal Engineer
Ankit Kumar, Associate Engineer
Agenda

• Background
• Traditional debugging approaches on VP and its Limitations
• Reverse Engineering Method Implementation
• Proof of Concept
• Next Steps
How to Debug Issues on Virtual Prototype in the absence of SW?
Traditional VP Debugging Approaches

1. **Sharing Simulation Log**
   - Limited Debug Information
   - Significant Manual Effort for longer simulation analysis

2. **Sharing Stripped-down Binary**
   - Complex Dependency among SW components
   - Dependency on External Models

3. **Joint Debugging**
   - Co-ordination challenges
Proposed Solution – Reverse Engineering Method

- Instrumentation of Logging into IP SystemC model
- Generation of Reproducer sequences from Simulation log
- Replicating the behaviour in standalone reproducer environment
Implementation

1. Instrumentation Of Logging into IP SystemC model

2. Creation of Standalone Reproducer environment

Implemented using In-house Automation Framework
Instrumentation of Logging (1/3)

IP Functional Block Model design using
• SystemC
• TLM2.0

MODEL BOUNDARY EVENTS

- Model Configuration Changes
- Register Transactions
- Interface Modifications

ELABORATION PHASE

SIMULATION PHASE
Instrumentation of Logging (2/3)

Model Constructor (Model Name, Data Type1 Parameter Name1, Data Type2 Parameter Name2, ...)
{
    CONFIGURATION, 1, Data Type1, Parameter Name1, Value1
    CONFIGURATION, 2, Data Type2, Parameter Name2, Value2
    .......
}

Register Access Function (Transaction Object)
{
    REGISTER, Simulation Time, Address, Read/Write, Data, Data Length
}

SC_METHOD(reset_in_value_changed)
Sensitive << reset_in;
void reset_in_value_changed ()
{
    INTERFACE, Simulation Time, INTERFACE_NAME, TYPE, CURRENT_VALUE
}
Instrumentation of Logging (3/3)

```cpp
ifx Ir::ifxIr(sc_module_name name, bool virtualization_enabled, bool fast_mode, unsigned int number_of_vms)
{
    unsigned int count = 1;
    cout << " CONFIGURATION, " << count++ << " ", bool, virtualization_enabled, " << virtualization_enabled << " endl;
    cout << " CONFIGURATION, " << count++ << " ", bool, fast_mode, " << fast_mode << " endl;
    cout << " CONFIGURATION, " << count++ << " ", bool, number_of_vms, " << number_of_vms << " endl;

    m_virtualization_enabled = virtualization_enabled;
    m_fast_mode = fast_mode;
    m_number_of_vms = number_of_vms;
}

if (cmd == tlm::TLM_WRITE_COMMAND)
{
    cout << "REGISTER," << sc_time_stamp() << hex << WRITE,"" << adr""."" << numBytes"" << endl;
}
else
{
    cout << "REGISTER," << sc_time_stamp() << hex << READ,"" << adr""."" << numBytes"" << endl;
}
```

---

**IP SystemC Model**

**Configuration Changes**

**Register Transactions**

**Interface Modifications**

---

*Accellera Systems Initiative*
Creation of Standalone Reproducer environment (1/2)

IP SystemC Model

Automation Framework

IP SystemC Model (with record feature)

IP SystemC Model

Standalone Reproducer Top

Reproducer Model
IP SystemC Model

Standalone Reproducer Environment

Reproducer Top
Reproducer Base (interface)
Reproducer (thread)

IP SystemC Model
Creation of Standalone Reproducer environment (2/2)

Reproducer Base with Complementary Interfaces

Reproducer thread to execute Trigger sequences

IP SystemC Model

Standalone Reproducer Environment

Reproducer Top
Reproducer Base (interface)
Reproducer (thread)
Proof of Concept using AURIX™ Interrupt Controller

* ECU like Virtual Platform
Next Steps

• Implementation for all models of Infineon next generation automotive microcontrollers
• Adaptation of the methodology for
  • serial communication interfaces
  • bus master interfaces
• Configuration to enable/disable the instrumentation of logging
  • Instrumentation should be enabled only for debugging purpose
  • For a normal simulation, Instrumentation should be disabled, as it degrades the performance
Questions