February 28 – March 1, 2012

Hardware/Software Co-Verification Using Specman and SystemC with TLM Ports

by
Horace Chan
Verification Lead
PMC-Sierra, Inc.
Overview

- Introduction
- Generic Embedded SoC Testbench Architecture
- Existing Co-verification Methods
- Specman to SystemC TLM Software Bridge
  - Overview
  - TLM port and data structure
  - Specman to C software function calls
  - C to Specman system methods calls
- Advantage and Benefits
- Challenges and Future Development
- Q & A
Introduction

• Co-verification benefits
  - Gives software engineer early access to the hardware
  - Provides additional testing for the hardware design
  - Provides better visibility in debugging software and hardware interaction

• Provides an integrated hardware and software solution
• Shortens the overall project schedule
Generic Embbed SoC Testbench Architecture

- Specman Testbench with UVM
  - How to call C functions from the testbench?
  - How to initiate AXI transactions from C functions?
  - How to check the expected behavior of RTL IP?
Existing Co-Verification Methods

- Acceleration/Emulation
  - Very Expensive
  - Hard to reuse existing testbench components
- ISX/ISS
  - Require extra license
  - Slow execution speed
- Socket/CVL
  - Testbench has to relinquish control to the software
  - C code cannot access internal data structure of the testbench
Specman to SystemC TLM Software Bridge
TLM port and data structure

- Sample software function

```c
int foo(int arg1, arg2);
```

- TLM port data structure

```c
struct func_calls_s {
    func     : func_name;
    arg_ptr  : uint;
};
```

- TLM port declaration

```c
e2c : out interface_port of tlm_blocking_put_if of (func_call_s) is instance;
c2e : in interface_port of tlm_blocking_put_if of (func_call_s) is instance;
```

- Function arguments data structure

```c
Extend func_name_t : [foo]
struct foo_arg_s {
    arg1          : int;
    arg2          : int;
    return_value  : int;
};
```
Specman to C software function Call

```plaintext
foo(arg1 : int, arg2 : int) : int @sys.any is {
    var foo_arg : foo_arg_s = new with {
        .arg1 = arg1;
        .arg2 = arg2;
    };
    var func_call : func_call_s = new with {
        .func = foo;
        .arg_ptr = foo_arg.get_pointer();
    };
    e2c$.put(func_call);
    return foo_arg.return_value;
}
```

- C Wrapper

```plaintext
switch (func_call->func) {
    case SN_ENUM(func_name_t, foo) :
        SN_TYPE(foo_arg_s) arg = (SN_TYPE(foo_arg_s))
            func_call->arg_ptr;
        arg->return_value = foo(arg->arg1, arg->arg2);
        break;
}
```

- e Wrapper
C to Specman system methods calls

• Intercept system read/write options in C functions

```c
int sys_read(int addr)
{
    func_call_s func_call;
    SN_TYPE(sys_read_arg_s) sys_read_arg = new;
    sys_read_arg->addr = addr;
    func_call.func = sys_read;
    func_call.arg_ptr = &sys_read_arg;
    c2e->put(func_call);
    return sys_read_arg->return_value;
}
```

• Hook up system functions call to VR_AD sequences in Specman side
• Use Doxygen to auto-generate the wrappers
Advantage and Benefits

• FREE!!
• Fast execution speed
  – The C code is running on the Linux host
• Easy debug support
  – Simvision SystemC debugger (gdb)
  – Simvision stripe chart
• Scalable
  – Swap the SystemC with ISX using WHEN subtype
• Coverage for software calls
  – Define coverage item on the argument data structure
Challenges and Future Development

• Pointer errors in C code
  - Simulator and software running in same memory space
  - Segmentation fault can bring down the simulator

• Debug turn around time
  - SystemC code is statically linked into RTL snapshot
  - Enhance the bridge to use dynamically loaded shared library

• System Verilog Support
Q & A

• Question and Answer