2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Portable Stimulus Models for C/SystemC, UVM and Emulation

Mike Andrews, Mentor Graphics

Boris Hristov, Ciena

Why Portable Stimulus

- Re-writing the stimulus model for multiple environments is a big challenge
- Non SV options for constrained random have not been satisfactory
  - E.g. SCV had too many limitations
- In 2014, Accellera Proposed Standard Working Group approved formation of a new standard
- Primary standard requirements:
  - Self-contained
  - Independent of any specific language (SV/SC)

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2015 UNITED STATES

Portable Stimulus Features

- Abstracted rule-based (declarative) description of legal stimulus scenarios
  - Blend of Backus-Naur Form and algebraic constraints
- Able to be mapped to any specific verification environment:
  - SystemVerilog UVM
  - SystemC SCV / CRAVE
  - Software driven verification
  - Post-silicon validation

Backus Naur Form is a "context-free grammar"

Note: These are features of an existing portable stimulus application, not the TBD Standard

Portable Stimulus Example

- Uses existing graph-based rule language

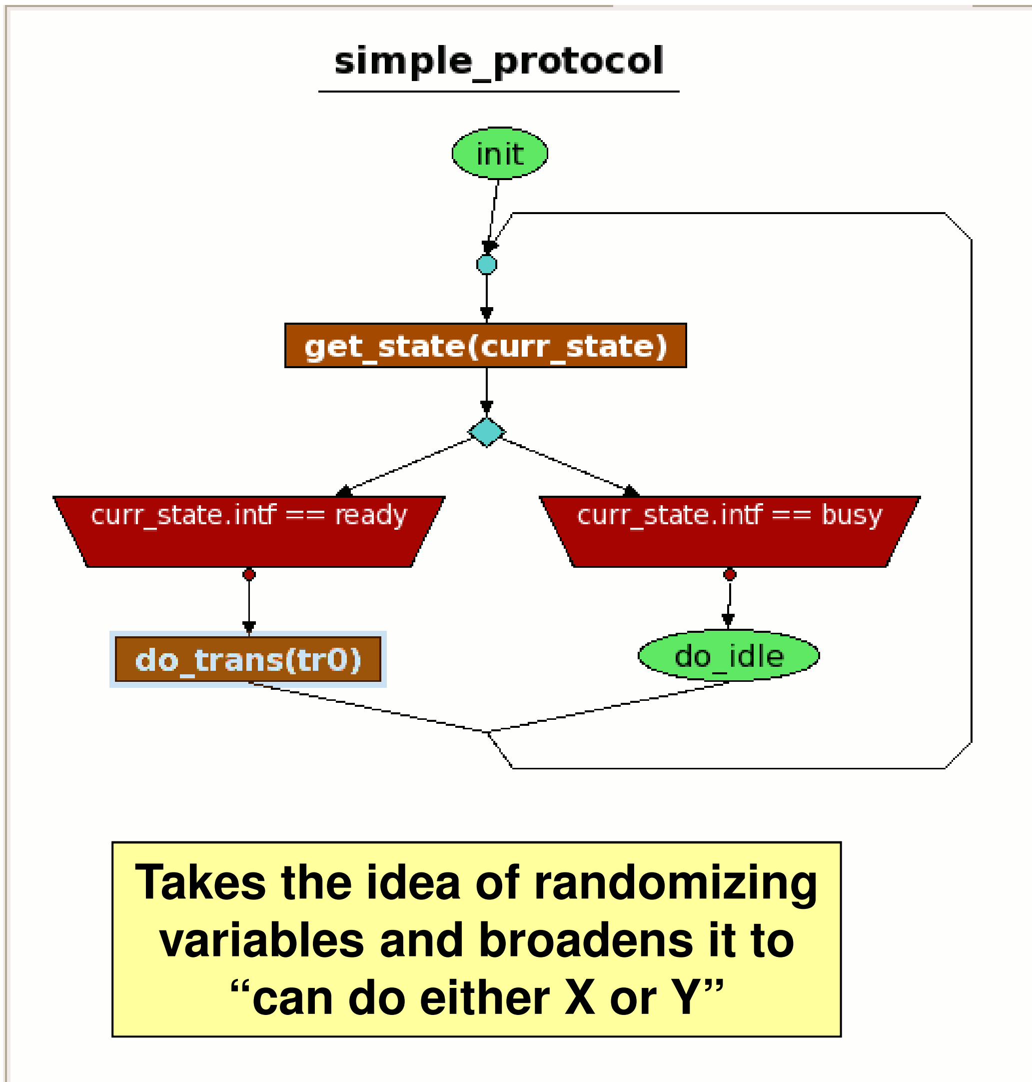
```
rule_graph simple_protocol {
    import "decls.rseg";
    action init, do_idle;

    trans tr0;
    interface do_trans(trans);

    input state_vars curr_state;
    interface get_state(state_vars);

    simple_protocol = init repeat {
        get_state(curr_state)
        (if {curr_state.intf == ready} do_trans(tr0)) |
            (if {curr_state.intf == busy} do_idle)
    };
}
```

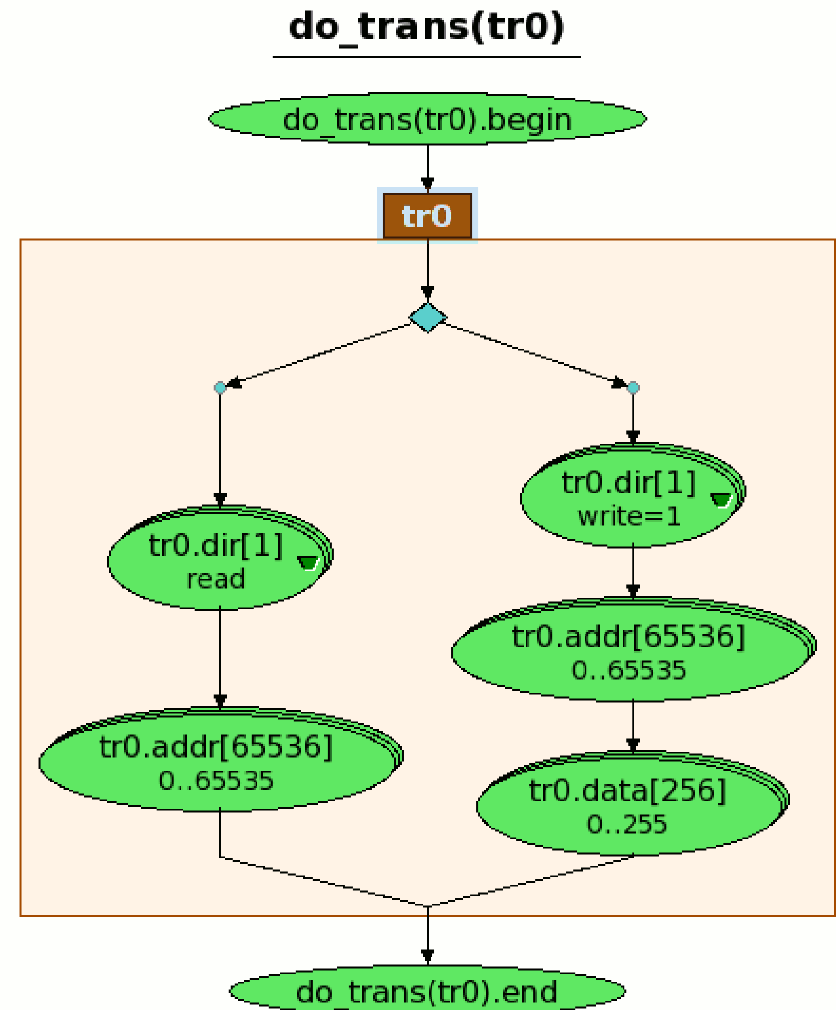
Interfaces here indicate data transfer to or from the testbench

Repeats can be inside the sequence, nested if desired

'|' is choice operator

Example as a Graph





DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2015

Sequential Dependencies

- Uses instances and constraints on instance fields
- Generates a simple scenario of:
  - write to random address
  - then read from same address

```
rule_graph write_read_test_seq {
    import "decls.rseg";
    action init;

    interface do_trans(trans);
    trans tr0, tr1;

    constraint do_write_read_c {
        tr0.dir == write;
        tr1.dir == read;
        tr0.addr == tr1.addr;
    }

    write_read_test_seq = init repeat {
        do_trans(tr0) do_trans(tr1)
    };
}
```

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2015

HLS Flow Application 1

- Growth in high-level synthesis drives need for advanced verification methodology for C/C++ (ESL)
  - Random stimulus generation
  - Functional coverage collection
- The stimulus model is developed at system architecture stage
  - More functional verification earlier in design cycle
  - Re-usable at implementation-level
- Common stimulus model is a communication bridge

HLS Flow Application 2

- Methodology requires:
  - Alignment of stimulus architecture
  - E.g. input variables collected into a class

```
class fir_filter_ld_stimulus
{
public: // data (one class member per DUT function argument)
  ac_int<8, true > inp;
  ac_int<8, true > coeffs[8];
  ac_int<3, false > addr;
  bool ld;

public: // interface
  fir_filter_ld_stimulus() {}
```

Analogous to SystemVerilog UVM Sequence Item

HLS Flow Application 3

- Methodology requires:
  - SystemC analog to class.randomize()
  - E.g. using interface to the stimulus graph to populate stimulus class fields

```
// call graph to obtain next values
fir_filter_ld_gen->ifc_fill_fir_filter_ld_stimulus(&stimulus);

// Call C++ design
fir_filter_ld(&stimulus.inp, stimulus.coeffs, &stimulus.addr,
                                           &stimulus.ld, &output);
```

New Flow Possibilities

- Same stimulus model can drive:
  - Abstract ESL model
  - SystemVerilog UVM simulation
  - Fast interface to emulation / other hardware acc.
- What if a solver could provide random stability?
  - Portable stimulus produces same scenarios
  - Determined by a seed independent of language
  - Problems found in one domain can be debugged in any other…

Summary

- Portable stimulus model provides:
  - Enhanced power in verification scenario description
  - Another opportunity for re-use of significant coding investment
  - Eases project bottleneck by allowing more functional verification at abstract level