Transaction Level Modeling and Verification

Adam Rose
VERIFICATION TECHNOLOGIST
DVT
TLM and Verification

- All advanced verification methodologies today are based on transactions
  - Aids reuse
  - Representing transactions as classes makes randomization easier
- Golden Models and Scoreboards are TLMs
  - Verification engineers have been producing TLMs for years without realising it
- Refinement and Abstraction
  - Refinement speeds up verification timescales
  - Abstraction speeds up testbench execution times
Transaction Level Verification and Reuse

Using classes to represent transaction makes randomisation much easier
Golden Models are TLMs of expected behaviour
- Stimulus Generators are (stochastic) models of external environment
- Verification Engineers have been writing TLMs for years
Abstraction and Refinement

- Start Verification before RTL is finished. Refine as RTL comes on line.
- Increase execution speed by abstracting away RTL not being tested.

SoC
- tlm
- tlm
- tlm
- rtl

SoC
- rtl
- rtl
- rtl
- rtl
Driver Design

Stimulus

Driver

Master

Driver Design Diagram:

Stimulus:
- put(t)

Driver:
- tlm_fifo<T>
  - M
  - nb_get(t)

Master:
- rsp = transport(req)

Drivers:
- sc_port
- sc_export
- channel
- SC_METHOD

Data Structures:
- tlm_fifo<req>
- tlm_fifo<rsp>

Channels:
- sc_port
- sc_export
- channel
- SC_METHOD
Monitor Design
Analysis Ports and Recorders

```
analysis_port<trans> ap;
recorder<trans> t;

switch( m_state ) {
    case START_TRANSACTION :
        rec.start(); // start recording transaction
        break;
    .
    .
    case FINISHED_TRANSACTION :
        ap.write( t ); // write to analysis port
        rec.write( t ); // record transaction
        rec.finish();   // finish transaction
        break;
}
```
Scoreboard Design: Analysis Fifos

Analysis fifos are unbounded tlm_fifos that implement the analysis interface:
- No losses even if multiple transactions in one delta cycle.
Transaction Level Functional Coverage

class parity_cov : public coverage_object<int, 2> {
    public:
    parity_cov(sc_module_name nm) :
        coverage_object<int, 2>(nm) {} 
    
    int sort(const int &x) const {
        return x % 2;
    }
};
Transaction Level Cross Coverage

```c
five_cov five;
parity_cov parity;

monitor m;
cross_coverage<trans> cross;
m.ap( cross );
cross.cov_points( parity );
cross.cov_points( five );
```
Components are basically very simple.
Complex verification infrastructures can be built using them.
Components are language independent.
TLM and Verification

- Hidden analysis ports
  - use coverage and some scoreboards with both TLMs and rtl
- Hidden recorders
  - Transaction recording for free, no need to modify code
The Analysis Library

- Mentor has developed the analysis library
  - Analysis ports, Recorders
  - Coverage, configuration interfaces, general purpose verification utilities
- We want to partner with verification and TLM teams to refine it further
- If there is sufficient demand, we will donate to OSCI
- Contact adam_rose@mentor.com for further information
Notes on Graphical Notation

- Dataflow and Flow of Control both necessary to understand verification
  - TLM has often ignored dataflow (like eg UML)
  - Verification has often ignored flow of control (like rtl)
- Squares represent an sc_port (and therefore the “caller”)
- Circles represent an sc_export
  - No circle represents a direct (channel) implementation
- Arrow represents data direction
  - Possibly bidirectional
- RTL
  - Don’t show sc_signal
  - Colour rtl ports black
- Process represented by a curved arrow
  - With T or M for thread or method, if known
- A TL sc_port connected directly to a TL sc_port indicates unknown flow of control
  - Most likely interpretation is that there is an implicit fifo
Examples

- LHS puts (sc_port) to RHS (sc_export)
- RHS (sc_port) gets from LHS (sc_export)
- LHS (sc_port) initiates bidirectional communication in RHS (sc_export)
- Thread in LHS (sc_port) talks to method in RHS (sc_port) via a channel
Examples

Signal Level Communication

Generic Hierarchcial sc_port to sc_export connection (no specified dataflow)

Left to Right Dataflow, no specified flow of control. Most likely assumption is left to right with a tlm_fifo in the middle.