The Universal Translator

David Cornfield
dcornfield@apm.com
Traditional Agents

- Unit I/O exposed in both scopes
- Agent shared across both scopes
• Unit I/O not exposed in both scopes
• Can’t connect virtual interface to a port
Sequence Layering

- Advanced by Tom Fitzpatrick of Mentor
Sequence Layering

- Asymmetry
- Peripheral Clutter
- Packaging Ambiguity
- Semantic Dependency
• PCS ENC VIP has **PULL** Semantic at Top Level
• PCS DEC VIP has **PUSH** Semantic at Top Level
Semantics – Unit Level

- Encoder DUT reversed by Decoder VIP
- Decoder VIP has **PUSH** Semantic
  - Same Semantic as Top Level Context
- **WRONG** – The DUT is a one-way function!
Semantics – Unit Level – Correct

- Encoder DUT compared against Encoder VIP
- Encoder VIP has **PUSH** Semantic
  - Opposite Semantic as Top Level Context
- One function, two semantic contexts!!
If Only ...

- Component based Architecture
- Connected with Ports
- Semantic Independence
The Translator Class
The Translator Class

- A **Translator** is a `uvm_component` that translates a stream of *inbound items* into a stream of *outbound items*.

```plaintext
virtual class translator #(
    type t_inbound_item = uvm_sequence_item,
    type t_outbound_item = uvm_sequence_item
) extends uvm_component;

pure virtual task translate();

endclass
```
Outbound items are **PULLED** out the seq_item_export
Inbound items are **PULLED** in the seq_item_port
Translate from inbound to outbound

is_active = UVM_ACTIVE
UVM_PASSIVE Translation

- Inbound items are **PUSHED** in the analysis_export
- Outbound items are **PUSHED** out the analysis_port
- Translate from inbound to outbound

\[
\text{is_active} = \text{UVM\_PASSIVE}
\]
The Translation API

• Derivatives implement the translate task calling:

get_inbound_item ( output t_inbound_item item );
try_inbound_item ( output t_inbound_item item );
put_outbound_item ( input t_outbound_item item );
put_uncloned_outbound_item ( input t_outbound_item item );

• Always follow a get-transform-put pattern
• Can be periodic 1:1, 1:M, M:1, M:N or aperiodic
• Same task called in both semantic contexts
class pcs_encoder extends translator #(t_mii_transfer, t_block);

    task translate();
        t_mii_transfer t1,t2;
        t_block block;

        get_inbound_item(t1);
        get_inbound_item(t2);

        block = encode(t1,t2);
        put_outbound_item(block);
    endtask
endclass

From t_mii_transfers To t_blocks

2 :1 Periodicity
Inline Sequencing

- Outbound items no longer directly controllable
- No possible input sequence to result in the desired output sequence
- Generally only an issue for stimulus generation
Debug Hooks

- Optional Inbound/Outbound item analysis taps
- Optional Inbound/Outbound item logging
Orthogonal Sequencing
Control Knob Pollution

- Control knobs, $Z'$, for $Z$ items show up in $X$ items
- A $Z'$ has nothing to do with an $X$ item
- Translator B must be $Z'$ aware to pass them through
Control Knob Explosion

- Control Knobs accumulate with each link and with each usage context
Orthogonal Sequencing

- Control sequenced separately from Data
- X timed, Y timed or independent
Dynamic Translation

- Why be limited to Control Knobs for Error Insertion?
- Example: Encapsulation
Adaptive Translation

- Response Channel used to tune the Dynamic translation.
- Example: IPG requested vs IPG actual
Package Isolation

• Helps resolve package dependency
• Package boundaries have are one of four data types:
  – A packet  \text{bit [7:0] data[]};
  – A frame  \text{bit [0:FL-1][7:0] data};
  – A bitstream  \text{bit [BW-1:0] data};
  – A bundle  \text{bit [0:LC-1][BW-1:0] data};
The Layered Architecture
• A **Layer** translates from *low abstraction* to *high abstraction* in the *analysis path*, **AND** from *high abstraction* to *low abstraction* in the stimulus path.
Layer Implementation

optional

one or more

optional

is_active
Layer Example – Ethernet PCS

ordered sets
RX RS
TX RS
link state + ipg
mii
blocks
DEC
ENC
BER
slip/skip/flip
block lock
bitstream
packets

© Accellera Systems Initiative 2014
Attachment Agents

- An **Attachment Agent** is a Traditional Agent without a sequencer.
Chains

- A **Chain** connects a sequencer to an Attachment Agent and has zero or more intervening Layers.
- A Chain is **simple** if it has only one layer.
A **Chainable Agent** is a Chain with no Layers

- Degenerate case similar to a Traditional Agent
Usage Contexts
Edge Unit Context

• An **Edge Unit** has I/O exposed in both scopes
• The Attachment Agent is ported
An **Internal Unit** has no I/O exposed at the Chip

The Layer is ported
End Unit Context

- An **End Unit** is the Internal Unit adjacent to the protocol divide
- The Layer and Sequencer is ported
Conclusion
It’s in the Numbers

- 300 lines of code
- ~400 extensions
- 16 Layers, 3 Attachment Agents, 2 utility Translators
- ~240,000 simulation runs
- ~16,000 tests
- The work horse of Unit and Chip level tests for ~2½ years
Questions

David Cornfield
dcornfield@apm.com