Synchronicity:

Bringing Order to the SystemVerilog/UVM

Synchronizing Chaos

Bryan Morris, P.Eng.
Ciena Corp.
Agenda

Discuss synchronizing constructs in...

• SystemVerilog ... very quick review:
  – fork-join + variants; event, semaphore, mailbox

• UVM classes
  – which use or re-implement the above...
Agenda (cont’d)

• UVM ... whirlwind tour of:
  – uvm_event + uvm_event_callback
  – uvm_barrier
  – uvm_object
  – uvm_subscriber
  – uvm_heartbeat
  – TLM FIFO

• Goal: Demonstrate these are superior to their SV equivalents.
• Paper has more details
  – guidance on use-model for each
  – references other papers with innovative use of each class above
Tour de SystemVerilog Synching

fork-join

fork : MASTER_SCOPE
begin : SLV1_SCOPE
  Slave1();
end
begin : SLV2_SCOPE
  Slave2();
end
begin : SLV3_SCOPE
  Slave3();
end
join
Tour de SystemVerilog Synching

fork-join_none

fork : MASTER_SCOPE
begin : SLV1_SCOPE
Slave1();
end
begin : SLV2_SCOPE
Slave2();
end
begin : SLV3_SCOPE
Slave3();
end
join_none
Tour de SystemVerilog Synching

fork-join_any

fork

fork : MASTER_SCOPE
begin : SLV1_SCOPE
Slave1();
end
begin : SLV2_SCOPE
Slave2();
end
begin : SLV3_SCOPE
Slave3();
end
join_any
disable_fork;
join
Tour de SystemVerilog Synching

event

```verilog
task Master();
fork
    Slave1();
    Slave2();
    Slave3();
join_none
// wait for some condition
// trigger slaves to wakeup
->wakeup_slave_ev;
endtask: Master
```
Tour de SystemVerilog Synching

```
task Slave1();
  // some processing
  @slave_wakeup_ev;
  // or
  wait(slave_wakeup_ev);
  // more processing
endtask: Slave1
```

Slave2() and Slave3() are similar
Tour de SystemVerilog Synching

Semaphore

- Slave1 gets 1 token -- fails: blocks waiting
- Slave1 gets 1 token -- continues
- Slave2 gets 1 token
- Slave2 puts 1 token
- Slave3 gets 2 tokens
Semaphore and mailbox pictures are same – except:

mailbox blocks when it’s full;

semaphore blocks when it’s empty.
Tour de SystemVerilog Synching

mailbox

Uses

• mailboxes hold handles to objects
• synchronization *with* message passing
• so many uses...
Tour de UVM Synching

• UVM 1.2...
  – and IEEE-1800-2 (from what I’ve seen)
• UVM Class is a better choice over its SV equivalent.
  – I’ll *try* and convince you of this for *most* applications...
  – I’m too old and wise(?) to never say “never use SV constructs”
  – Recommend: look at the UVM constructs FIRST.
    Then choose the best design choice for your application.
### Tour de UVM Synching

<table>
<thead>
<tr>
<th>SV Construct</th>
<th>UVM Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>uvm_event, uvm_heartbeat, uvm_subscriber</td>
</tr>
<tr>
<td>semaphore</td>
<td>uvm_barrier, uvm_objection</td>
</tr>
<tr>
<td>mailbox</td>
<td>TLM FIFO, uvm_subscriber</td>
</tr>
</tbody>
</table>
Tour de UVM Synching

uvm_event

Producer triggers event

Indicates process is blocked waiting

Consumer 1 waiting

Consumer 2

Consumer 3

Producer
Tour de UVM Synching

\texttt{uvm\_event}

All consumers "wake up"
Tour de UVM Synching

uvm_event

• Use
  – Broadcast Notification (same as event)
  – Optionally add meta-data to event – can be retrieved by listener
  – Provides
    • introspection (is_on/off, get_num_waiters, get_trigger_time)
    • mode: trigger sensitive (wait_trigger, wait_ptrigger)
    • OR level sensitive (wait_on/off)
    • control (reset, cancel)
Tour de UVM Synching

`uvm_event_callback`

1. Producer calls `uvm_event_trigger`
2. Pre-trigger callback1
3. `uvm_event` generates event
4. Post-trigger callback1
5. Callbacks run sequentially in same delta-cycle
Tour de UVM Synching

`uvm_event_callback`

- Use
  - Filter the event – return value from `pre_trigger()`
  - Process the meta-data associated with event
  - Trigger another event in a daisy-chain
Tour de UVM Synching

```
uvm_heartbeat
```

![Diagram showing the interaction between parent and child components, with events triggered in sequence.

- **Parent** triggers heartbeat event for **Child1**
- **Child2** triggers heartbeat event
- **Child3** triggers heartbeat event

**Time**
Tour de UVM Synching

uvm_heartbeat

Parent checks heartbeat from all children

Child 1 triggers heartbeat event

Child 2 triggers heartbeat event

Child 3 triggers heartbeat event
Tour de UVM Synching

**uvm_heartbeat**
Parent *listens* for heartbeat

```c
uvm_event child_hb_ev;
parent_heartbeat.set_heartbeat( child_hb_ev, ...
                             q of Child uvm_components )
```
Tour de UVM Synching

Child is responsible for generating heartbeat

```verilog
uvm_event child_hb_ev;

fork : HEARTBEAT_FORK
begin : HEARTBEAT_EV_TRIGGER
forever begin
  #(HEARTBEAT_TIMER_IN_NS * 1ns) child_hb_ev.trigger();
end
end
join_none
```
Tour de UVM Synching

uvm_heartbeat

• Uses
  – Allows mode: `UVM_ALL_ACTIVE`, or `UVM_ANY_ACTIVE`, `UVM_ONE_ACTIVE` (or none)
  – (arguably...) Simpler than `uvm_objectection`
  – Kill a ‘zombie’ child: stop wasting resources
  – Use the “lack” of heartbeat to signal something interesting
    • Wait for ALL heartbeats after a reset before proceeding.
Tour de UVM Synching

**uvm_barrier**

- **process1** waiting
- **process2** calling `wait_for()`
- **process3** is blocked waiting
- Indicates process condition is met
- **processN**
Tour de UVM Synching

uvm_barrier

All processes continue together
Tour de UVM Synching

uvm_barrier

• Uses
  – Similar use to a semaphore
  – Provides
    • introspection (get_num_waiters)
    • control (reset, cancel)
    • auto-reset to start the heartbeat automatically
  – When children contribute to a unanimous decision:
    • All scoreboards have completed their expected checks
    • multiple channels “synched up”
  – Components can add/delete themselves from the barrier.
  – Recommend wrapping a project layer around this class:
    • Add debug tracing capability like the uvm_objection
Tour de UVM Synching

uvm_objection
Tour de UVM Synching

`uvm_object`
Tour de UVM Synching

uvm_objection

- Master creates objection
- Slaves raise objection
- Slaves can raise an objection at any time
- Slave1 drops objection
- Slave3 drops objection
- All Slaves Dropped objection

Indicates process is blocked waiting
Tour de UVM Synching

uvm_objectection

Indicates process is blocked waiting

Drain Time

All Slaves Dropped objection
Tour de UVM Synching

uvm_objection

- Uses
  - Familiar use in UVM Phasing
  - Similar use to a semaphore and fork-join
  - Debug tracing NOT available in SV constructs (display_objections)
  - Can add callbacks at various phases
  - Control: clear
  - Introspection: get_objectors, get_objection_count
  - All components at a certain phase: reset deasserted, synched, end-of-test conditions.
Tour de UVM Synching
TLM FIFO

- Uses
  - Similar use to a parameterized mailbox
  - Introspection: `is_empty`, `is_full`, `used`, `size`
  - Control: `flush`
Summary

• Overview of SV Synchronization
• Review of UVM classes providing synchronization
• UVM classes provide better functionality over SV equivalent
  – Control
  – Debug tracing
  – Introspection
  – Extensibility
Summary

- Paper provides guidelines and references to other papers using the UVM synchronization classes.
- Future work:
  - Guidelines
  - Would welcome your thoughts
Thank YOU

Questions??