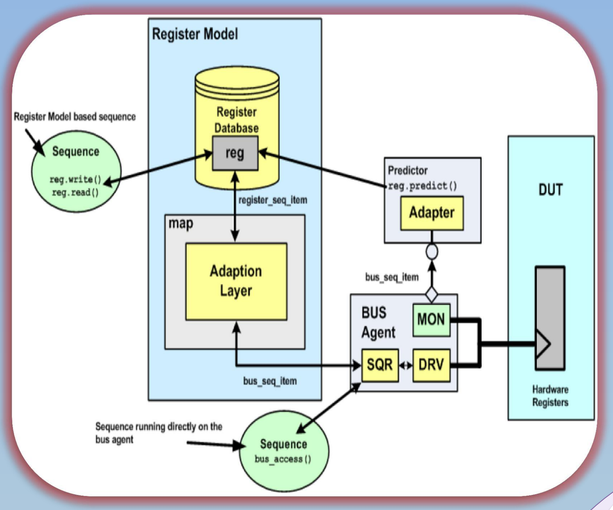
BOOSTING SIMULATION PERFORMANCE OF UVM REGISTERS IN HIGH PERFORMANCE SYSTEMS

Ahmed Yehia

Verification environment with UVM registers integrated



Current do_bus_write() Implementation

```
task uvm_reg_map::do_bus_write (uvm_reg_item rw,
     uvm_sequencer_base sequencer,
     uvm_reg_adapter adapter);
  ...
  //Loop over data values arr in register trans
  foreach (rw.value[val_idx]) begin
    //For each address location
    foreach (addrs[i]) begin
      bus_req = adapter.reg2bus(rw_access);
      //Drive transaction
      bus_req.set_sequencer(sequencer);
      rw.parent.start_item(bus_req, rw.prior);
      rw.parent.finish_item(bus_req);
      ...
    end //foreach (addrs[i])
  end //foreach (rw.value[val_idx])
endtask //do_bus_write()
```

Contribution

- The proposed implementations of *do_bus_write()* and *do_bus_read() below* avoid chopping data in bus width chunks and let the adapter's *reg2bus()* method decide the amount of data it is going to consume in a bus transaction according to its bus capabilities. There is some coding required to workaround the *reg2bus()* prototype which returns only one sequence item.
- A more efficient prototype of *reg2bus()* is to return an array of sequence items.
- Appendix B in the full paper shows a way that can be adopted by the UVM register library to improve the interface of reg2bus() to boost simulation performance and still maintains backward compatibility.

```
task do_bus_write (uvm_reg_item rw,
                   uvm_sequencer_base sequencer,
                   uvm_reg_adapter adapter);
  ...
  //Total num of bits to be sent for all items
  n_bits_total      = n_bits*rw.value.size();
  //Capture the start address
  start_address     = addrs[0];
  //Update rw_access struct
  rw_access.byte_en = byte_en;
  rw_access.kind    = rw.kind;
  rw_access.n_bits  = n_bits_total;
  //Sync mechanism for concurrent writes
  uvm_reg_bus_op_c:: reg2bus_write_sm.get(1);
  //Extract bursts when more bits to write
  while (rw_access.n_bits > 0) begin
    rw_access.addr = start_address;
    //Pass rw to adapter
    adapter.m_set_item(rw);
    bus_req        = adapter.reg2bus(rw_access);
    adapter.m_set_item(null);
    //Push the bus_req to the bus requests queue
    //Workaround adapter.reg2bus() returning only
    //one sequence item limitation
    bus_req_q.push_back (bus_req);
    //Get info about consumed bits by adapter
    //bus2reg(). Workaround rw_access passed as
    //const in bus2reg() prototype
    uvm_config_db #(uvm_reg_bus_op_c)::get(null,
      "", "rw_acc_adapt_write", uvm_reg_bus_op_c_write);
    //update start address of next iteration
    //reflecting num of bits converted to bus
    //transactions
    start_address    += uvm_reg_bus_op_c_write.n_bits
                        /(8*get_addr_unit_bytes());
    rw_access.n_bits -= uvm_reg_bus_op_c_write.n_bits;
  end //while (rw_access.n_bits > 0)
  //Free the semaphore
  uvm_reg_bus_op_c:: reg2bus_write_sm.put(1);
  foreach (bus_req_q[i]) begin
    uvm_sequence_item bus_req = bus_req_q[i];
    bus_req.set_sequencer(sequencer);
    rw.parent.start_item(bus_req,rw.prior);
    ...
    rw.parent.finish_item(bus_req);
    ...
  end
endtask //do_bus_write()
```

- To adopt the proposed implementations above, extend the *uvm_reg_map* class and implement the *do_bus_write()* and *do_bus_read()* methods as shown above. In the test (or when building your register block), just override *uvm_reg_map* with your new register map class name using the *set_type_override()* method.

```
class my_reg_block extends uvm_reg_block;
  virtual function void build();
    uvm_reg_map::type_id::set_type_override(
                my_reg_map::get_type(),1);
  endfunction
endclass
```

UVM register library front-door access

Effect of Executing Unnecessary AMBA AXI Bursts on Simulation Performance writing to a 2KB memory block

The current implementations of *do_bus_write()* and *do_bus_read()* insert undesired bottleneck when performing on high performance buses, by chopping data in bus-width chunks, generating a simple transaction for each chunk, and avoiding making use of bus powerful features when found. Maximizing the number of transactions generated maximizes context switching in simulation, which may have severe consequences on simulation performance imagining a test performing hundreds of these operations.

References

1. www.uvmworld.org
2. www.verificationacademy.com/uvm-ovm
3. AMBA AXI reference, infodenter.arm.com

Or get a copy of the full paper…

Mentor Graphics®