February 28 – March 1, 2012

Exquisite Modeling of VIP

by
Adiel Khan; Anuradha Tambad;
Imran Ali; Prashanth Srinivasa; Shivani Upasani; Subashini Rajan
Engineering Team
LSI and Synopsys
Agenda

• The presentation covers

  - Different Challenges faced developing an efficient VIP
    • Complex Randomization
    • High bandwidth data transfer
    • Protocol dependency

  - Case study
    • Describe each challenge through an example
    • Recommendations in developing any efficient VIP

  - Summary and Results
Context of Verification

- LSI designs AXI subsystems
- Need to verify DMA controller for correct data transfer
- Performance of DMA controller is important
  - This requires large amounts of data to be moved
  - Pre-generation of more than one descriptor/data is needed as DUT reads multiple packets based on DUT fifo size and interface bandwidth
- Pre-generation is a common need for verification of many design blocks
Challenges faced in developing VIP

Three main obstacles encountered developing the verification IP.

- **Constraint solver**
  - Large set of inter-related constraints
  - Performance of solver

- **Pre Generating payload-data for each transaction**
  - Large quantities of payload per transaction
  - Consumes significant runtime memory
  - Transaction stored in ref-models until scoreboard check is complete

- **Stabilizing and coding Verification IP**
  - Frequent changes in protocol, architecture reasons: better performance etc
  - Highly configurable for reuse in higher level of chip hierarchy
  - Verify the scalability of design (e.g.: configurable no. of DMAC channels)
Challenges faced in developing VIP

Three main obstacles encountered developing the verification IP.

- **Constraint solver**
  - Large set of inter-related constraints
  - Performance of solver

- Pre-Generating payload-data for each transaction
  - Large quantities of payload per transaction
  - Consumes significant runtime memory
  - Transaction stored in ref-models until scoreboard check is complete

- Stabilizing and coding Verification IP
  - Frequent changes in protocol, architecture reasons: better performance etc
  - Highly configurable for reuse in higher level of chip hierarchy
  - Verify the scalability of design (e.g.: configurable no. of DMAC channels)
Data class - complex constraints

```plaintext
constraint rate_c {
    (jd_D.mode == MAX_MODE) -> (rate inside {0,2,3,4,6});
};

constraint blk_size_c {
    if(jd_D.mode == MAX_MODE && rate == 0)  (jd_C.blk_size/2 inside {108,120,144,180,192,216,240,480,960,1440,1920,2400}) ;
    if(jd_D.mode == MAX_MODE && rate == 2)  (jd_C.blk_size inside {48,96,144,192,240, 288, 384, 432, 480}) ;
};

constraint s_c {
    ((rep == 3) || (rep == 1)) -> jd_D.mode == TE_MODE;);

constraint s_rm_c {
    (jd_C.so == 1) -> (jd_D.byp != 0);}

constraint dyn_stop_c {
    (dyn_stop == DYN_STOP) -> (jd_C.so == 0); }
```

Adiel Khan, Synopsys UK & LSI India R&D Pvt Ltd
Randomization - List of complex objects

- With a smaller queue size, constraint solver easily solves the queue of objects
- With large queue size, constrain solver times out

Adiel Khan, Synopsys UK & LSI India R&D Pvt Ltd
Randomization in Phases – Objects List

- Randomization in phases for queue of class objects
- Complex class objects solved one by one and inserted in queue
- Knowledge of dependency of constraints required
Parameterized base class

class base_class #(type child_pkt = vmm_data)
  extends vmm_data;
  rand int q_size; //size of queue for class
  constraint q_size_c { q_size[31] == 0;};
  child_pkt pkt_obj_q[$];
  rand child_pkt pkt_obj;
  virtual function void post_randomize();
    pkt_obj_q.delete();
    $cast(pkt_obj, pkt_obj.copy(pkt_obj));
    for(int i = 0; i < q_size; i++) begin
      .................
      pkt_obj.randomize();
      pkt_obj_q.push_front(pkt_h); end
endfunction
endclass

User defined base class
List of Class objects
Randomization and insertion in queue
Recommendations - constraints

- Split constraints into
  - Variables that must be solved together
  - Partition non-related variables for solving independently
  - Per sub-object randomization called from post-randomize

- Reduction in time taken to solve complex constraints for multiple packets

- Hierarchical parameterised base class was reused in DMA and different types of decoder
Challenges faced in developing VIP

Three main obstacles encountered developing the verification IP.

- **Constraint solver**
  - Large set of inter-related constraints
  - Performance of solver

- **Pre Generating payload-data for each transaction**
  - Large quantities of payload per transaction
  - Consumes significant runtime memory
  - Transaction stored in ref-models until scoreboard check is complete

- **Stabilizing and coding Verification IP**
  - Frequent changes in protocol, architecture reasons: better performance etc
  - Highly configurable for reuse in higher level of chip hierarchy
  - Verify the scalability of design (e.g.: configurable no. of DMAC channels)
Simplified DMAC Descriptor

**Simplified Descriptor format**

- **SRC**: 0x0000_4000
- **DEST**: 0x2000_F000
- **A**, **B**, **C**: 16bit fields
- Theoretically, $2^{48}$ bytes can be read/write.

- **X**, **Y**: Controls the memory read/write pattern.

Another 2 fields **X**, **Y** controls the pattern i.e. how memory will be read/write.

3 fields named **A**, **B**, **C** controls the total number of bytes to be read/write. **A**, **B** & **C** all are 16bit field. So, Theoretically $2^{48}$ bytes can be read/write.
Basic DMA Functionality

SRC 0x0000_4000  DEST 0x2000_F000

No. of bytes to be read and write i.e.
A * B * C = 500

Simplified Descriptor format

Memory 1
0x0000 4000 0xE8
0x0000 4001 0x56
0x0000 4002 0x4A

Memory 2

Data Fifo
0x2000_F000 0xE8
0x2000_F001 0x56
0x2000_F002 0x4A

Adiel Khan, Synopsys UK & LSI India R&D Pvt Ltd
DMA Verification Strategy

SRC
0x0000_4000

DEST
0x2000_F000

No of bytes to be read and write i.e.
A * B * C = 500

Simplified Descriptor format

Memory 1
0x0000_4000
E8

Memory 2
0x0000_4000
E8

Data Fifo

Mon

Reference Model

Read addresses

Write addresses

Data list

Mon

compare

0x0000_4000

0x2000_F000

E8

Score Board

compare

0x0000_4000

0x2000_F000

E8

compare

0x0000_4000

0x2000_F000

E8

compare

0x0000_4000

0x2000_F000

E8
## Issue in Handling Large Queues

<table>
<thead>
<tr>
<th>SRC</th>
<th>DEST</th>
<th>Number of bytes to read and write i.e. $A \times B \times C = 655070000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000_4000</td>
<td>0x2000_F000</td>
<td></td>
</tr>
</tbody>
</table>

### Simplified Descriptor format

#### Reference Model

<table>
<thead>
<tr>
<th>Read addresses</th>
<th>Write addresses</th>
<th>Data list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000_4000</td>
<td>0x2000_F002</td>
<td>4A</td>
</tr>
<tr>
<td>0x0000_4001</td>
<td>0x2000_F001</td>
<td>56</td>
</tr>
<tr>
<td>0x0000_4000</td>
<td>0x2000_F000</td>
<td>E8</td>
</tr>
</tbody>
</table>
Water Mark Concept

Once upper threshold level is reached, stop filling it more.

Once lower threshold level is reached, tank needs to fill again.

Once list of addr/data reached upper threshold level, stop pushing more in the queue.

Once queue size reaches lower threshold level, addr/data need to generate again to fill the queue.
Recommendations for Pre-generation

• DMA reads/writes data as a chunk of bytes. Hence we always need to keep some pre generated expected address/data ready in the queues.
• Water mark concept helps to use system memory in an efficient manner.
• Upper and lower threshold level can be set to any value by the user depending upon the design.
• Without using Water mark concept tool runs out of memory and gives the following message:

```
Error-[OUTMEM] The tool has just run out of memory:
Memory allocated = 3684 MB, Request size = 923885568 bytes.
make: *** [run] Error 3
```
Challenges faced in developing VIP

Three main obstacles encountered developing the verification IP.

- **Constraint solver**
  - Large set of inter-related constraints
  - Performance of solver

- **Pre Generating payload-data for each transaction**
  - Large quantities of payload per transaction
  - Consumes significant runtime memory
  - Transaction stored in ref-models until scoreboard check is complete

- **Stabilizing and coding Verification IP**
  - Frequent changes in protocol, architecture reasons: better performance etc
  - Highly configurable for reuse in higher level of chip hierarchy
  - Verify the scalability of design (e.g.: configurable no. of DMAC channels)
VIP dependent on Interface protocol

- **Scenario Layer**
  - Dma generator
  - Dma Driver
  - Dma Ref Model
- **Dmac Functional Layer**
  - Dma Driver
  - Dma Ref Model
  - Bus Interface Modports and clocking blocks
- **Command Layer**
  - Dma generator DMA VIP
  - Dma Driver DMA VIP with AHB
  - Dma Ref Model DMA VIP with AXI Interface
- **Signal Layer**
  - DUT

Dma VIP components which need updates are:
- Driver and Ref Model other than command layer components.
Interface Independent VIP

**Scenario Layer**
- Dma generator
- Dma burst class
- Dma Driver
- Dma Ref Model
- Converter class
- Dma burst list

**Command Layer**
- AXI H/W VIP
- AHB H/W VIP
- AHB S VIP
- AXI H/W VIP
- Dma burst
- Dma burst
- AHB burst
- AHB beat

**Signal Layer**
- Bus Interface Modports and clocking blocks
- DUT

Dma VIP components work with it's in-house developed 'dma burst data'

Only file which needs modification is converter class.
**Interface Independent VIP**

- One more advantage we find with this approach is when verification is moved from block level to higher level the data integrity is checked not with immediate interface but with end interface hitting full path coverage.

![Diagram of verification process](image)

Data integrity check is between Bus X0 and Bus X1.

Data integrity check is between Bus A0 and Bus A1.

Adiel Khan, Synopsys UK & LSI India R&D Pvt Ltd
VIP Interface independent benefits

- Time to migrate between VIP’s significantly reduced
  - 6X faster than traditional VIP changes for DUT modifications

- Block to Top Reuse thought out upfront

- Data integrity covering any end to end data path
Summary

• Following above recommendations, a robust VIP is created to generate random packets which are constrainable and easily controlled from test case without burdening the constraint solver.
• This reusable VIP is efficient, faster, and immune to interface changes.
• Also reduces run time memory consumption.
  - Without using water mark concept tool runs out of memory when queue size reaches around 149314080 value.
Q&A
Usage - Parameterized base class

program test();
    initial begin
        base_class #(turbo_pkt) tr;
        turbo_pkt tpkt;
        tr.pkt_obj = tpkt; // assigning constrained value
        // to baseclass handle
        ............
        tr.randomize();
    end
endprogram

User defined base class