Modelling of UVC Monitor Class as a Finite State Machine for a Packet-Based Interface

Djordje Velickovic, Verification Engineer, Veriest Solutions
Milos Mitic, Verification Engineer, Veriest Solutions
Agenda

• How was the idea born?
• Concept of modelling of UVC monitor Class as an FSM for a packet-based interface
• Implementation of UVC monitor class for exemplary interface
• Benefits
• Results
• Conclusion
How was the idea born?

- Best quality VIPs need time and resources
- Packet Based Interfaces could be difficult to model
  - Large Data Packets
  - Interface Signals Behavior
- Packet-based Interface and FSM – Natural connection
- Everybody knows about FSMs
- Development time
- Debug tool
Concept

- FSM states -> Packet frames
- State “jumping”
- Coding simplicity
- Protocol violations checks
- Knowledge of interface functionality
- Debug benefits
Implementation

- UVM
- Definition of interface used for showcasing the implementation
- Coding the FSM model inside the monitor class
- Implementation of protocol checkers
## Definition of interface – signal list

<table>
<thead>
<tr>
<th>Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRxDV</td>
<td>1 bit</td>
<td>input</td>
<td>Valid indication</td>
</tr>
<tr>
<td>MRxD</td>
<td>4 bits</td>
<td>input</td>
<td>Data bus</td>
</tr>
<tr>
<td>r_Pro</td>
<td>1 bit</td>
<td>input</td>
<td>Promiscuous mode control signal - all frames are received regardless of their destination address</td>
</tr>
<tr>
<td>r_Bro</td>
<td>1 bit</td>
<td>input</td>
<td>Broadcast mode control signal - all frames containing broadcast addresses are rejected</td>
</tr>
<tr>
<td>MAC</td>
<td>48 bits</td>
<td>input</td>
<td>MAC address of the used Ethernet MAC IP Core</td>
</tr>
<tr>
<td>MRxClk</td>
<td>1 bit</td>
<td>input</td>
<td>Clock</td>
</tr>
<tr>
<td>Reset;</td>
<td>1 bit</td>
<td>input</td>
<td>Reset</td>
</tr>
</tbody>
</table>
## Definition of interface

- **Frames in the packet**

<table>
<thead>
<tr>
<th>4 bytes</th>
<th>6 bytes</th>
<th>6 bytes</th>
<th>2 bytes</th>
<th>46.. 1500 bytes</th>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREAMBLE</td>
<td>DESTINATION ADDRESS</td>
<td>SOURCE ADDRESS</td>
<td>PACKET LENGTH</td>
<td>DATA</td>
<td>CRC</td>
</tr>
</tbody>
</table>
Task for packet detection

- Started in the run phase
- New packet detection
- Create a new item
- Collect address and other control signals
- Call packet collection task

```plaintext
01 virtual task detect_packet();
02 fsm = IDLE;
03 forever begin
04 if (vif.slv_cb.MRXDV == 1'b1) begin
05 fsm = PREAMBLE;
06 item = uvm_eth_vip_item::type_id::create("item");
07 item.MAC = vif.slv_cb.MAC;
08 item.r_Pro = vif.slv_cb.r_Pro;
09 item.r_Bro = vif.slv_cb.r_Bro;
10 collect_packet();
11 end
12 else @vif.slv_cb;
13 end
14 endtask
```
• Collect frame by frame
• Each frame – unique FSM value
• Conditions for “state jumping”
• End of packet condition
Task for protocol violations checks

- Started in the run phase
- Tracks each signal behavior separately
- Do not check the value of the signal
- Check if detected activity on the signal is in valid frame of the packet

```verilog
01 virtual task check_signals();
02 fork
03 forever begin
04 @(posedge vif.r_Pro, negedge vif.r_Pro);
05 r_Pro_signal_chk: assert(fsm == IDLE) else
06 `uvm_error("check_signals", $sformatf("Illegal r_Pro change detected during packet state : %s", fsm.name));
07 end
08 end
09 forever begin
10 @(posedge vif.r_Bro, negedge vif.r_Bro);
11 r_Bro_signal_chk: assert(fsm == IDLE) else
12 `uvm_error("check_signals", $sformatf("Illegal r_Bro change detected during packet state : %s", fsm.name));
13 end
14 end
15 forever begin
16 @(vif.MAC);
17 MAC_signal_chk: assert(fsm == IDLE) else
18 `uvm_error("check_signals", $sformatf("Illegal MAC change detected during packet state : %s", fsm.name()));
19 end
20 end
21 join_none
22 endtask
```
Benefits

• Gains in debug
• Usage of modeled packet state on a waveform
• Compare modelled state behavior to interface behavior
Test environment

Master agent
- Traditional/FSM monitor
- Driver
- CFG
- Seqr

Interconnect

Slave agent
- Traditional/FSM monitor
- CFG
Traditional vs. FSM monitor results

**CPU usage**

![CPU usage graph]

**Simulation time**

![Simulation time graph]
Conclusion – Why FSM monitor is good?

• Standardized monitor coding
• Easier implementation based on a well-known concept
• Lower threshold for Interface behavior expertise
• Significant boost in debug
• On par or better performance when compared to the traditional approach
Questions?
Thank you for your attention.