Multi-language Verification requires special considerations: Synchronization of test phases and events and moving data objects between the language’s worlds. In some cases, re-writing part of the code, (either manually or automatically) could be a more reasonable solution. In this paper, we discuss the trade-off between the two alternatives: re-write the code and co-running with original languages.

**The System**

- SOC with peripheral I3C, Driven by ARM through DMA
- Full E environment for SOC, including configurations, registers, interrupt handling, bus drivers, etc.
- I3C UVM VIP from external vendor.

**Introduction**

**Configuration and Execution Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Translate SV-UVM VIP sequences to E and use them within the SoC virtual sequences</td>
</tr>
<tr>
<td>2</td>
<td>Run SV-Sequences and call E methods from SV for each memory accesses.</td>
</tr>
<tr>
<td>3,4</td>
<td>Write from Spec</td>
</tr>
</tbody>
</table>

**Consideration**

Compare Translation/Rewrite vs. co-run

<table>
<thead>
<tr>
<th></th>
<th>Notes</th>
<th>Translation Effort</th>
<th>Error Effort</th>
<th>Test Efficiency</th>
<th>Flexibility</th>
<th>Simulability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Costing Effort</td>
<td>Of Sequences</td>
<td>MOD</td>
<td>LOW</td>
<td>MCH</td>
<td>HIGH</td>
</tr>
<tr>
<td>2</td>
<td>Error Effort</td>
<td>Test flow</td>
<td>MOD</td>
<td>LOW</td>
<td>MCH</td>
<td>HIGH</td>
</tr>
<tr>
<td>3</td>
<td>Tasks: Debug</td>
<td>Error</td>
<td>MOD</td>
<td>LOW</td>
<td>MCH</td>
<td>HIGH</td>
</tr>
<tr>
<td>4</td>
<td>Uncovered</td>
<td>Test suit</td>
<td>RUN-OUT Vendor scenario and checks</td>
<td>MOD</td>
<td>LOW</td>
<td>MCH</td>
</tr>
<tr>
<td>5</td>
<td>Flexibility</td>
<td>Run around Vendor scenario and checks</td>
<td>MOD</td>
<td>LOW</td>
<td>MCH</td>
<td>HIGH</td>
</tr>
<tr>
<td>6</td>
<td>Simulability</td>
<td>Using sequences as reference to SW</td>
<td>MOD</td>
<td>LOW</td>
<td>MCH</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

**Translation**

Sequences translation was done with Python program:
- RAL to Specman register (vr_ad like)
- UVM sequence library to E sequences (mostly body() task to body() TCM)
- It covers only small portion of the Language and uses shortcuts (as macro translation instead of expand)
- Neither all cases nor all commands are supported (less than 10%)
- It took 3-person week to complete
- We estimate a full SV to E Translator to require one person-year

**Synchronization**

Synchronization between sequences and data objects between languages can be done either by E method ports or by TLM (Transfer Level Model) ports.

**Conclusions**

Combining a component with a different language into an existing verification environment depends on the characteristics of the system and the verification requirement. In some cases, partially translation can be more efficient and time/cost-effective than applying a standard language-to-language porting. We found that for sequences, particularly for those which produce memory access command, translation of the code is beneficial.

Using UVM-ML Open Architecture is a valuable way to connect and run together components with different languages. Yet, the implementation of these kinds of systems requires a special attention for synchronization. Ports (TLM) between the components should close this gap and enable effective verification.

---

**Code Examples**

```c
method_type wb_gamal_set_name(name: string);
{3c_set_uvm_test_name: out method_port of wb_gamal_set_name is instance;
 keep {3c_set_uvm_test_name.name.id_path} := 
 ""-.,dte_board_load.get_uvm_test_name_from_e", 
 Test.name: string;
{3c_set_uvm_test_name$}{(test_name)}; 

function void get_uvm_test_name_from_e(string name);
sv_test_name = name;
$display($formatf("I3C DTE Test is %s",sv_test_name));
endfunction
```

---

**Translation**

E to SV : Call in E code and executed in SV (UVM object or TB)

```c
Define in E
```

Assign and linked to SV in E

Called in E function

Executed in SV function

---

**Veriest**

Project was done in collaboration with Nuvoton Technologies