SimpleLink™ MCU Platform: IP-XACT to UVM Register Model - Standardizing IP and SoC Register Verification

“UVM is a perfect start”

Jasminka Pasagic  (j-pasagic@ti.com)
Frank Donner    (f-donner@ti.com)
Contents

• Abbreviations and Common Terms
• Platform Project and Challenges
• Why changing from manual to automation?
• Flow
• Register Requirements
• UVM Register Model Updates
• Effort Savings - UVM_REG Time Reduction Comparison
• Conclusions
• Further Work
### Abbreviations and Common Terms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>MRV</td>
<td>Memory Register Verification</td>
</tr>
<tr>
<td>UVM</td>
<td>Universal Verification Methodology</td>
</tr>
<tr>
<td>GIT</td>
<td>means <em>unpleasant person</em> in <a href="https://en.wikipedia.org/wiki/Slang">British English</a> slang</td>
</tr>
<tr>
<td>DesignSync</td>
<td>Data Management tool</td>
</tr>
</tbody>
</table>
Platform Project and Challenges

- Multi-site project
- High amount of reuse and standardization
- High amount of automation
- Planning and execution to be shared
- Automotive
Why changing from manual to automation?

Ensure specification and verification data consistency and accuracy

Reduce time spent in developing register verification

Use industry standards:
- IP-XACT register description
- UVM register generators
- UVM RAL (indirect registers, lock, shadow, alias, alternate register, interrupts, counters, FIFOs, wider registers, atomic registers, register arrays, etc.)
## Register Requirements

### Register IP/SoC Requirements
- UVM Access policies
- TI access field policies
- Optional Bit Coverage
- Optional Register Coverage
- Optional Address Coverage
- Naming Options
- Addressing Options: base + offset vs absolute addressing

### TI Access Field Policies
- UVM process of describing custom access policies
- Develop/Implement `ti_reg_field`/`ti_uvm_reg_cb`
- UVM factory: function `set_type_override_type`, limitations, workarounds
- Location

### Standardizing Peripheral Rules
- TI access field policies and naming convention
- Location of PSD – PDF translation to XML document for all
- Base addresses definition and location

### IPXACT Rules
- Optional UVM generator/preferences
- Addressing Options
- Naming conventions
- Location
- Access polices of the registers
- Custom access field polices
UVM Register Model Updates

- Specification Updates
- Software findings
- Custom register access policies description
- Tool limitations
- Tool versions
- XML input
- Integration and Aggregation of Data
- Base address offsets
- UVM Generator Preferences
- GIT Repository
- Verification findings: incidents or bugs
- Exceptions on custom register access policies
- Script/templates updates
- Long/Short Term Consequences
- Simulation time
- UVM_REG updates
- Outcomes/Goals
- Data Management (GIT/DS)
Effort Savings

<table>
<thead>
<tr>
<th>Manual</th>
<th>Generation 1st time</th>
<th>Generation 2nd time</th>
<th>Generation 3rd time</th>
<th>Generation+Manual 2nd time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UVM_REG manual written</td>
<td>Input preparation</td>
<td>Input preparation</td>
<td>Generation scripts</td>
<td>Generation scripts</td>
</tr>
<tr>
<td></td>
<td>Output update</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

Manually developed UVM register Model

- labor intensive task
- introduces range of potential problems and flaws
- difficult to maintain specification change cycles
- difficult to add central changes
- error prone

Automated/Semi-Automated UVM REGISTER Model

- ensures specification consistency
- standardizing inputs and outputs allows data reproducibility, reusability and allows cross industry convergence (eg. if you stay within IEEE 1685 standard but internal reuse of custom register is supported)
- faster cycle time to simulateable register framework
- incremental development
Further Work

- SoC register model testing
- Update loops: tool versions
- Optimization of simulation time
- Checkers improvement for generator and TI register access polices
- Capture speed factor from manually to automated or semi-automated register model generation for our project
- UVM factory type override limitations and workarounds
- Auto generation of register verification plan
Thank you!!

Questions ??
What is UVM_REG? What is IP-XACT?

What is UVM_REG?
UVM_REG is an abstract SystemVerilog model for registers and memories from the DUT. It is built using the UVM methodology.

Ramp-up on UVM_REG model by reading:
- Verification Academy’s Registers: https://verificationacademy.com/cookbook/registers

What is IP-XACT?
What to consider for selecting generator?

1. Scalability - designs may include large numbers of registers and scalability is an important consideration.

2. Support for a standard input format

3. Ability to replace the generator as needed

4. Debug-ability and self-checking of the generated code
Examples of industry UVM register model generators?

1. Cadence **iregGen** is a native IPXACT to UVM generator (supports registers as well as memories, wide range of registers such as immediate, fifo, shared and more, and automatically creates functional coverage)

2. Magillem UVM Generator

3. Synopsis **genSys**

4. AgniSys

5. Etc.
class zeroToSet_cbs extends uvm_reg_cbs;
  `uvm_object_utils(zeroToSet_cbs)
  
  function new(string name = "zeroToSet_cbs");
    super.new(name);
  endfunction
endclass

class zeroToSet_reg_field extends uvm_reg_field;
  `uvm_object_utils(uvm_reg_field_ext)
  
  local static bit m_rwos = define_access("RWOS");

  //Constructor
  function new(string name = "zeroToSet_reg_field");
    super.new(name);
  endfunction
endclass

class <IP/SoC>_reg extends uvm_reg;
  `uvm_object_utils(zts)
  
  rand uvm_reg_field              field1;
  rand zeroToSet_reg_field        field2;
  
  //Constructor
  function new(string name = "<IP/SoC>_reg");
    super.new(name);
  endfunction
endclass