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)

Texas Instruments

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

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

SoC	System on Chip
IP	Intellectual Property
MRV	Memory Register Verification
UVM	Universal Verification Methodology
GIT	means *unpleasant person* in British English slang
DesignSync	Data Management tool

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:

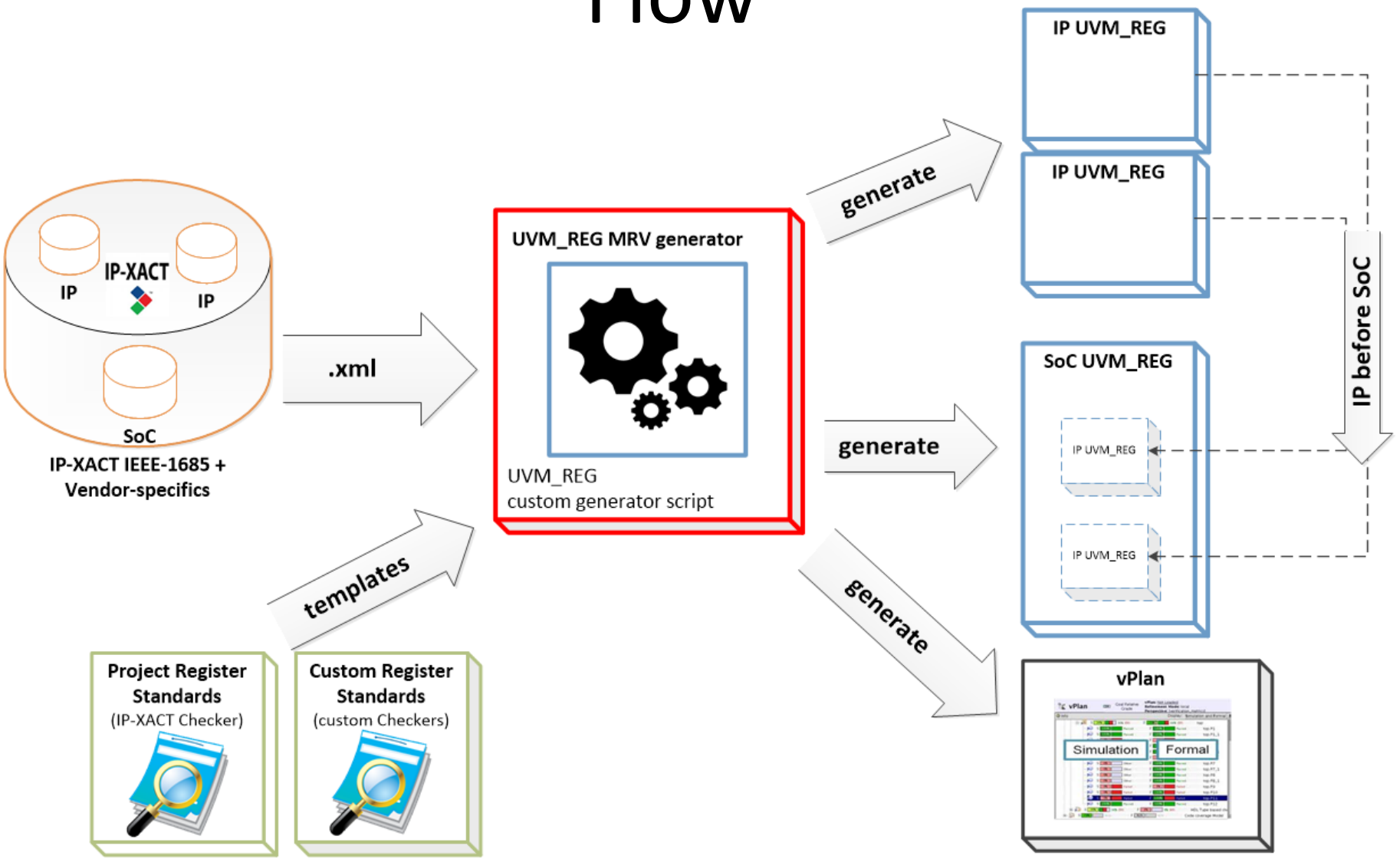
IP-XACT register description

UVM register generators

UVM RAL

(indirect registers, lock, shadow, alias, alternate register, interrupts, counters, FIFOs, wider registers, atomic registers, register arrays, etc.)

Flow



Register Requirements

Register IP/SoC Requirements

- UVM Access policies
- TI access field polices
- Optional Bit Coverage
- Optional Register Coverage
- Optional Address Coverage
- Naming Options
- Addressing Options: base + offset vs absolute addressing

TI Access Field Polices

- 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

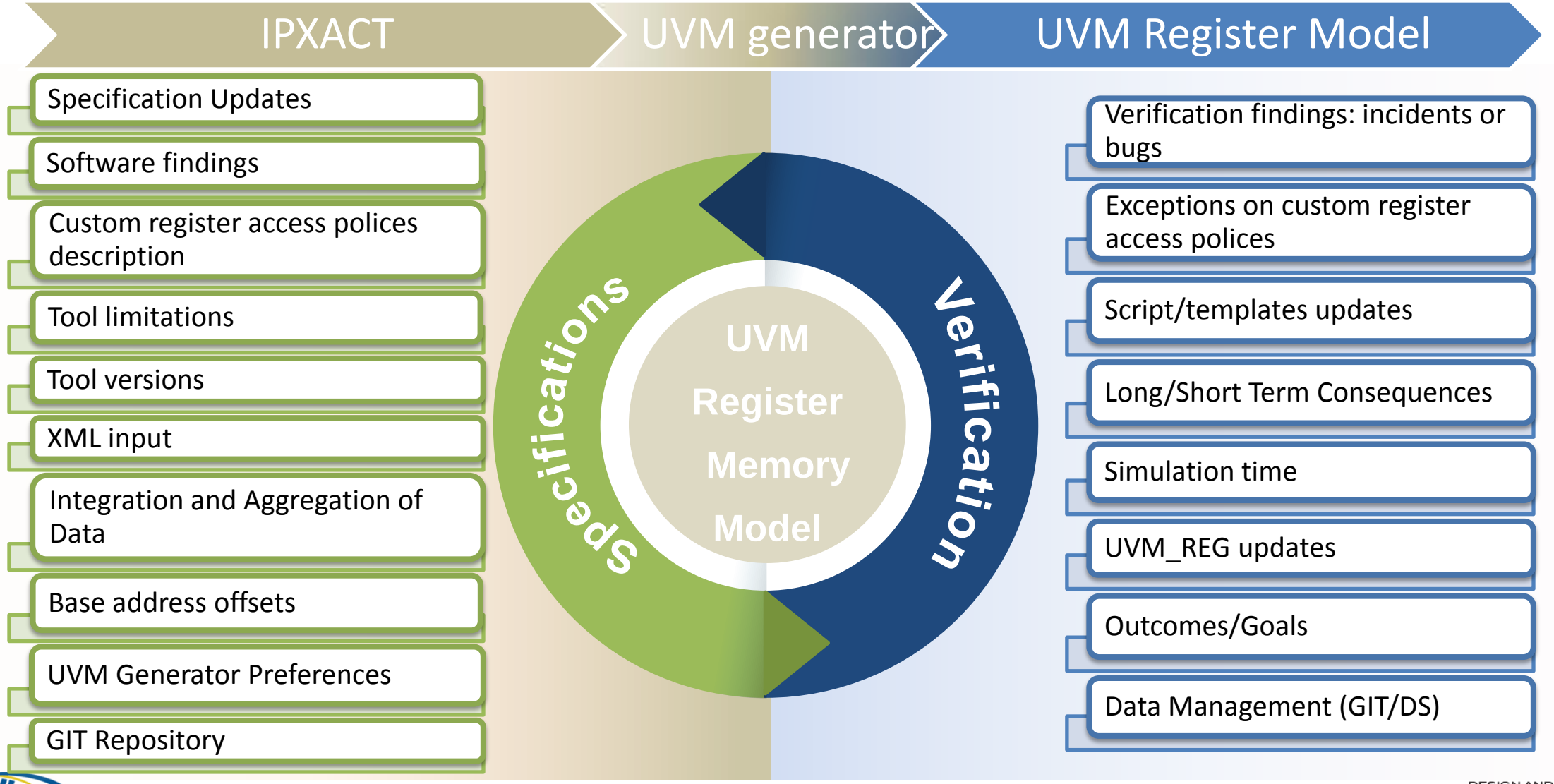
Standardizing Peripheral Rules

- TI access field polices 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



Effort Savings

Manual	**Generation**	**Generation**	**Generation**	**Generation+Manual**
Always	Effort 1st time	Effort 2nd time	3rd time	2nd time
UVM_REG manual written	Input preparation	Input preparation		
	Generation scripts	Generation scripts		
	Output update			

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 ??

BACKUP

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
- Advanced features of the UVM Register model: http://www.verilab.com/files/litterick_register_final_1.pdf

What is IP-XACT?

IEEE 1685, "Standard for IP-XACT, Standard Structure for Packaging, Integrating and Re-Using IP Within Tool-Flows," describes an XML Schema for meta-data documenting Intellectual Property (IP) used in the development, implementation and verification of electronic systems and an Application Programming Interface (API) to provide tool access to the meta-data.

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.

UVM Custom Field Access Policy Example

```
class zeroToSet_cbs extends uvm_reg_cbs;
 `uvm_object_utils(zeroToSet_cbs)

 function new(string name = "zeroToSet_cbs");
  super.new(name);
 endfunction

 virtual function void post_predict(input uvm_reg_field  fld,
                                    input uvm_reg_data_t previous,
                                    inout uvm_reg_data_t value,
                                    input uvm_predict_e  kind,
                                    input uvm_path_e     path,
                                    input uvm_reg_map    map);
  if (kind == UVM_PREDICT_WRITE && fld.get_access() == "RWOS" && value == 0)
    value = 1;
 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");

 //Cunstructor
 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;

 //Cunstructor
 function new(string name = "<IP/SoC>_reg");
  super.new(name);
 endfunction

 virtual function void build();
  field1 = uvm_reg_field::type_id::create("field1");
  field2 = zeroToSet_reg_field::type_id::create("field2");

  field1.configure(this,  16, 16, "RW",   0, 0, 1, 1, 0);
  field2.configure(this,  16,  0, "RWOS", 0, 0, 1, 1, 0);

  // register callback
  zeroToSet_cb field2_cbs = new("rwos_cbs");
  uvm_reg_field_cb::add(field2, rwos_cbs);
  ...
 endfunction

endclass
```