Reuse doesn’t come for free - learnings from a UVM deployment

Sumeet Gulati - Senior Technical Leader, NXP Semiconductors
Srinivasan Venkataramanan - Chief Technology Officer, CVC Pvt. Ltd.
Ketki Gosavi - Trainee, NXP Semiconductors
Saumya Anvekar, Senior Design Engineer, NXP Semiconductors
Azhar Ahammad - ASIC Design Verification Engineer, CVC Pvt. Ltd.

accellera SYSTEMS INITIATIVE

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

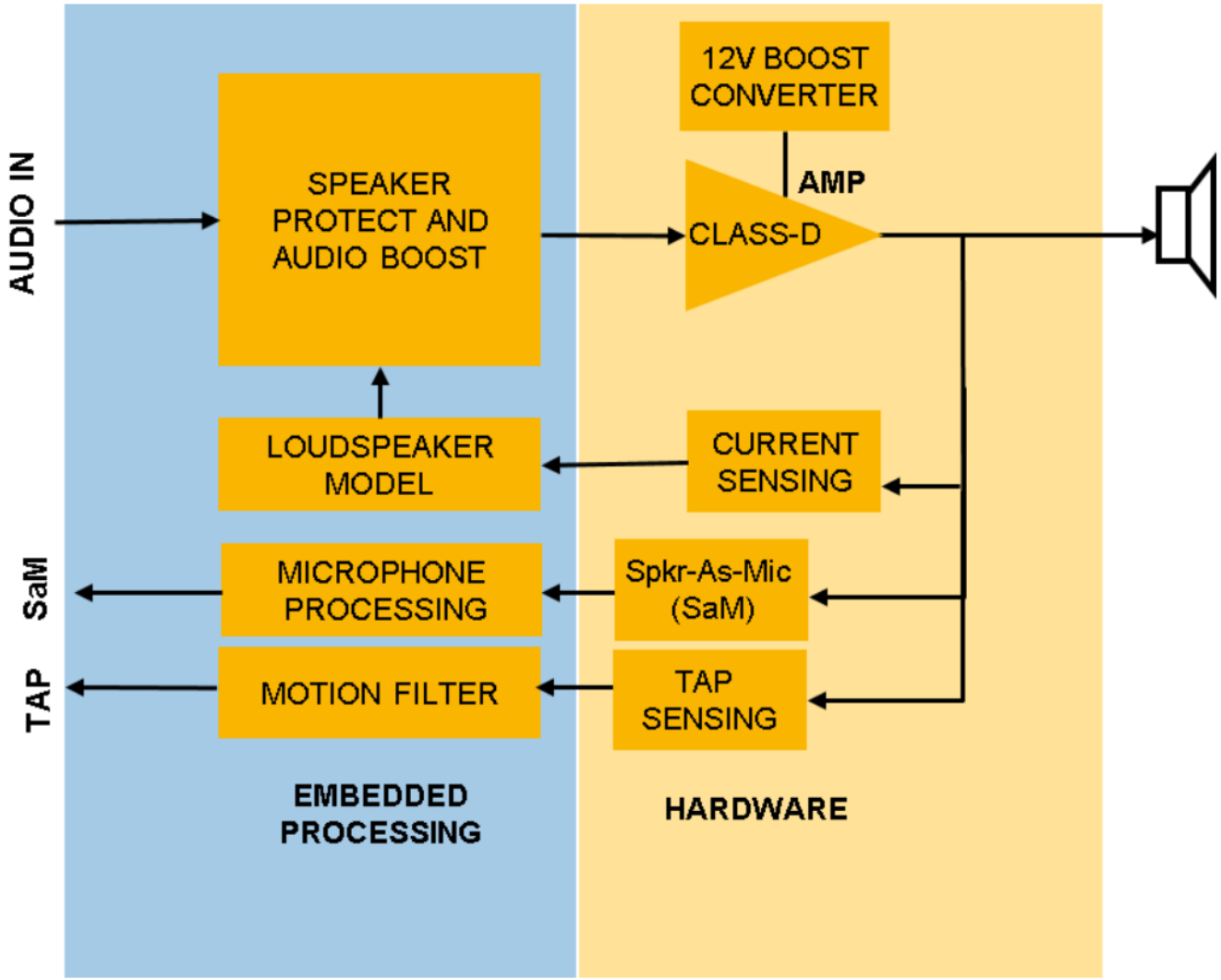
Agenda

- Background and Introduction
- Design Partitioning at IP level verification
- Tweaking UVM RAL Predictors.
- Ensuring Reusability – UVM Factory
- Conclusion

Introduction

- Verification of modern day Mixed Signal SoCs and associated IPs is very challenging and an Innovative task
- UVM Standard has become very widely adopted for verifying complex designs at various levels of abstraction.
- UVM defines a set of templates and a set of coding guidelines to keep verification environments reusable across levels of verification, across projects etc.
  - *But there is more to reuse than just what UVM Prescribes.*
- We will share some of our finer learning during the process of UVM deployment.

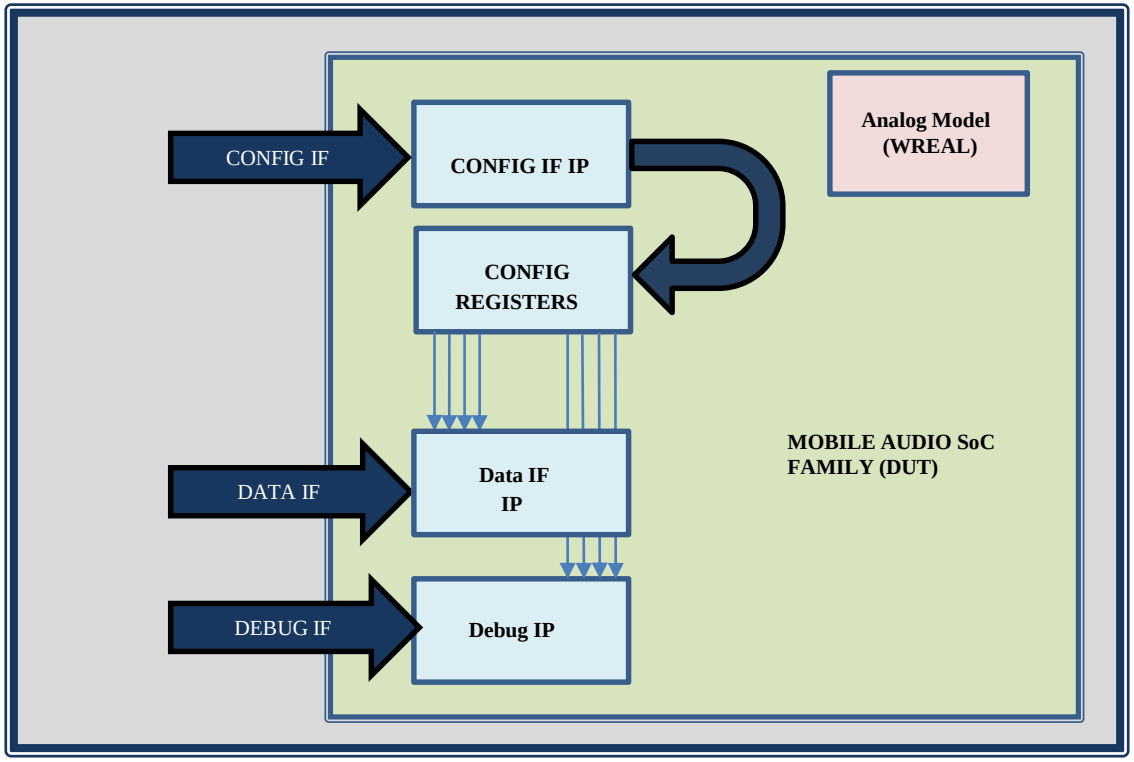
Smart Class-D Amplifier SoC



SoC Architecture

- Our SoC is a Intelligent Class D Amplifier that reuses several IPs along with in-house DSP processor.
- SoC has multiple interfaces for Data, Debug and one for configuration which is through I2C (Inter-Integrated Circuit) interface.
- Design is equally heavy in Analog & Digital
  - A true Mixed Signal SoC in nature.

SoC Architecture



Verification Requirements

- Individual IPs are verified in a stand-alone IP verification environment and then integrated to the top level SoC verification environment.
- **IP level sequences shall be maximally portable**.
- The individual IP owners understand each IP deeply
- Top level integrator may not be fully aware of complex configuration sequences for each IP to configure correctly at top level.

Agenda

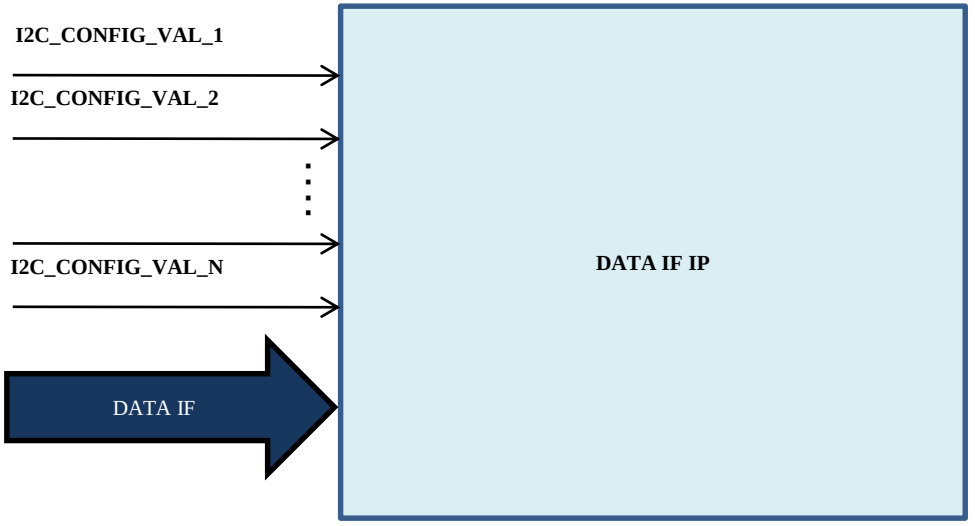
- Background and Introduction
- Design Partitioning at IP level verification
- Tweaking UVM RAL Predictors.
- Ensuring Reusability – UVM Factory
- Conclusion

Design Partitioning at IP level verification

- Significant limitations of legacy environment was the inability to reuse IP level scenarios directly at SoC level.
- There was duplication of all register programming sequences per IP at SoC level leading to:
  - Redundant work at SoC Level
  - Wasted debug cycles due to wrong configurations at SoC level

Initial IP Verification Partitioning

- First cut verification for an IP
- Two important input interfaces to the DUT
  - I2C based configuration values
  - Data interface driving audio data

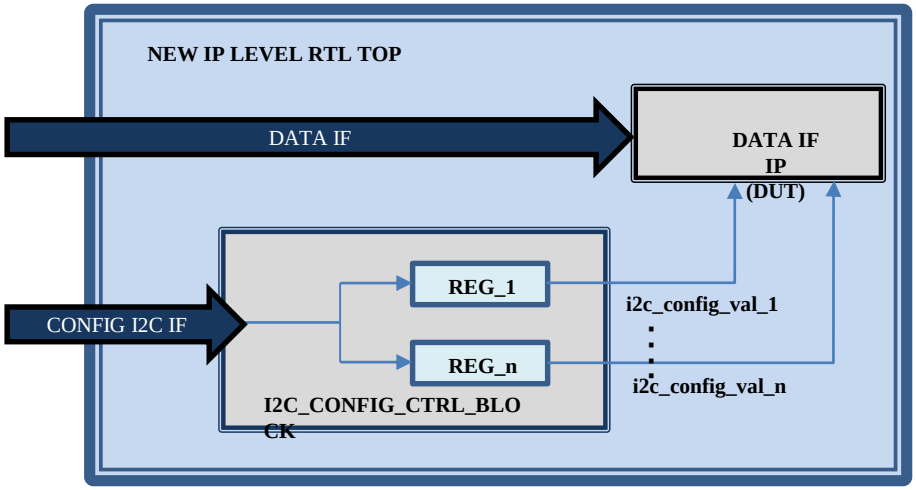


Initial IP Level Sequence

- Sequence body simply specifies the configuration values for needed i2c_cfg_val inputs.
- A simple, dummy UVM driver attached to this sequence (via sequencer) then drives the data interface.
- Since at SoC level there is a real I2C interface, the basic sequences developed at IP level were not reusable at SoC level before.

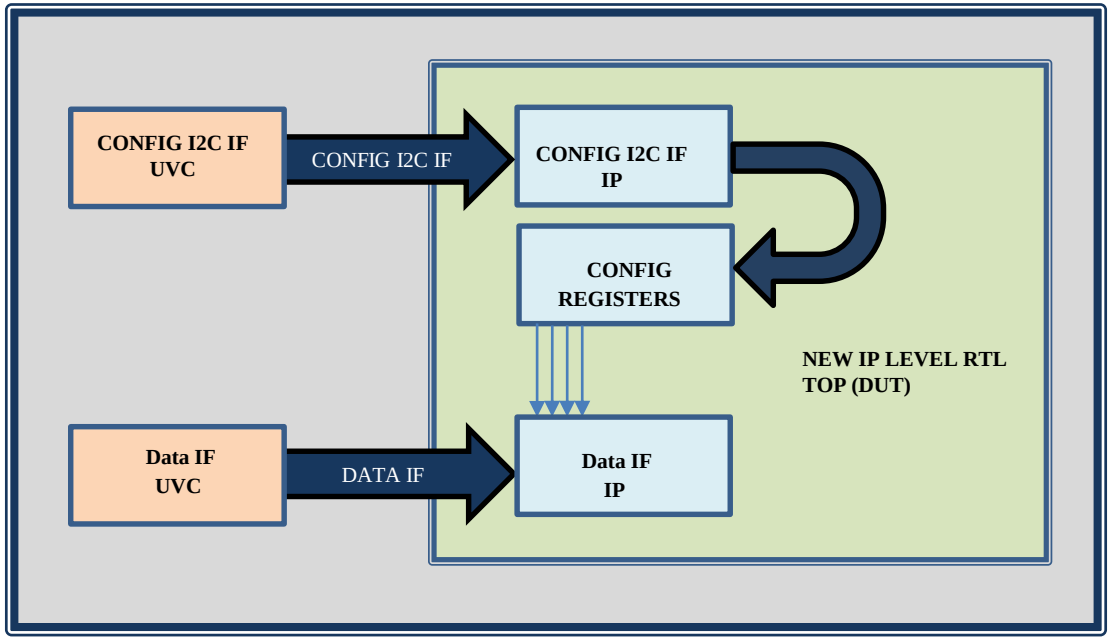
Re-Partitioned DUT

- Just mere use of UVM & SystemVerilog alone doesn't always enable reuse.
- So we re-partition the design for each IP level DUT to include I2C control block with relevant registers



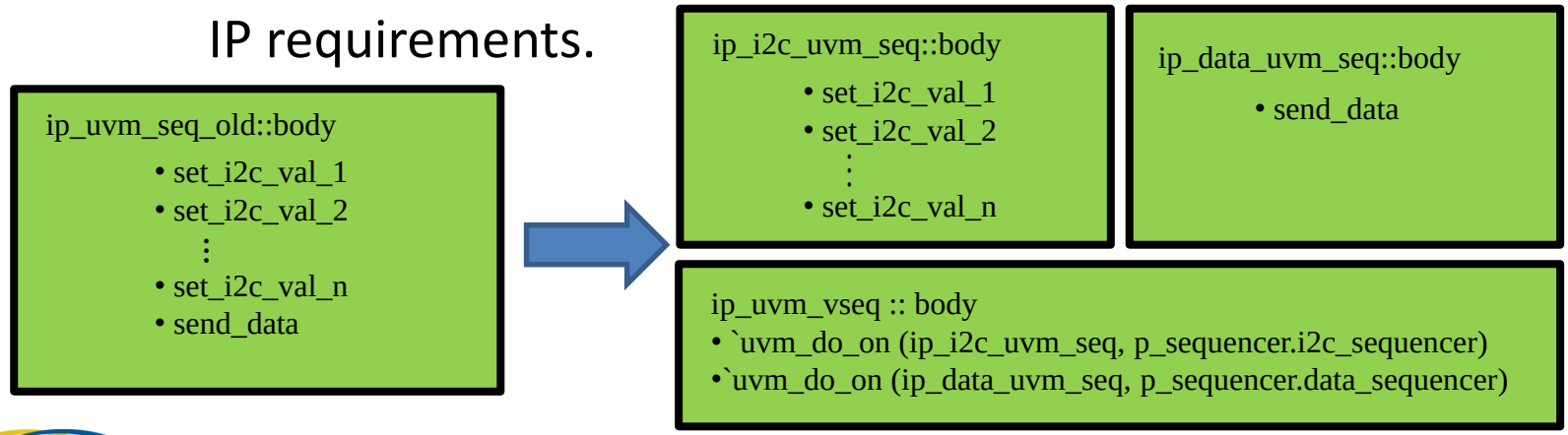
New IP level verification environment

- We used additional I2C UVC along with IP Level UVC/UVCs in the IP level environment.
  - Motivation behind this re-partitioning was reusability of IP level sequences than the UVCs themselves.



New IP Sequences

- New modified IP level verification environment now included register models.
  - UVM RAL model added to individual IP level verification environment
- IP level sequences programmed I2C registers that are needed to configure the specific IP
  - In exact order as per individual IP requirements.



Advantages

- New IP level UVM sequences turned out to be virtual sequences
  - I2C configuration RAL sequence runs on a I2C sequencer
  - Data sequence runs on a data interface UVC sequencer.
- IP level sequences now mimic the programmer's model of the IP
  - Easy for reviewing by system architects as a side benefit
- Advantage at the SoC level
  - I2C RAL sequences were plug-and-playable.
  - RAL provides an easy way to manage the offset of individual register blocks

Agenda

- Background and Introduction
- Design Partitioning at IP level verification
- Tweaking UVM RAL Predictors.
- Ensuring Reusability – UVM Factory
- Conclusion

Tweaking UVM RAL Predictors

- Some hype around UVM that tends to imply
  - UVM knows how to verify your system
- In reality, UVM only provides a reusable framework
- Users need to write good amount of code on top to make it usable.
- A classical case is UVM RAL predictors.
- Only some amount of "auto prediction" is supported by UVM out-of-the-box.
- Auto-prediction does not work for complex modern designs having very advanced register features
  - Volatile registers, Safety controlled registers etc.

Dynamic Register Access-mode

- Registers Accesses are set once during the model generation.
- Dynamic changes to access policy during a simulation needs to be modeled by user code.
- Critical for the predictor to work well in our designs as there are protected or safety related registers that change access privileges based on some control register settings.
- UVM provides API to achieve this
  - *uvm_reg_field::set_access()*
- Our team used post_predict callback in UVM registers to model this.

Safety Lock Callback

```
class lock_field_cb extends uvm_reg_cbs;
    local uvm_reg_field safety_field;

    function new (string name, uvm_reg_field prot);
        super.new (name);
        this.safety_field =prot;
    endfunction

    virtual function void post_predict ( input uvm_reg_field field,
                                         input uvm_reg_data_t previous,
                                         input 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)
        begin : predict
            if (value == VL_RAL_LOCK)
            begin : lock
                void'(safety_field.set_access("RO"));
            end : lock
            else
            begin : unlock
                void'(safety_field.set_access("RW"));
            end : unlock
        end : predict
    endfunction : post_predict
endclass : lock_field_cb
```

Safety Model via UVM RAL Callback

- Safety lock callback models the dynamic control to the field named "safety_field" based on value of another register content.
- This callback is then integrated to the register model during the model configure step as shown in the pseudo-code below:

```
function void configure(vlb_control_regcontrol_reg, vlb_safety_regdata_reg);
  lock_field_cblock_cb;
  lock_cb = new("lock_cb", data_reg.safety_field);
  uvm_reg_field_cb::add(control_reg.ctrl_field, lock_cb);
endfunction : configure
```

Agenda

- Background and Introduction
- Design Partitioning at IP level verification
- Tweaking UVM RAL Predictors
- Ensuring Reusability – UVM Factory
- Conclusion

Ensuring Reusability

- Given the upfront guidelines set by management
  - Keep the code reusable for future changes.
  - Ensure all UVM coding guidelines were followed.
- Two significant phases in the project where these guidelines had to be ensured
  - First during the environment development
  - Second during developing sequences.
- In UVM, factory provides the necessary infrastructure to keep code reusable.

UVM Factory

- UVM Factory involves three steps:
  - Registering the classes with factory table
  - Consulting the factory table during construction
  - Setting overrides on need basis

- First step is achieved through the use of handy macros:
  - for registering components

    `uvm_component_utils_begin (i2c_driver)
  - for sequences/register models

    `uvm_object_utils (data_if_config_reg)

Object Construction

- Second step is to be done during construction of every object within UVM .
- Below is the standard constructor of a uvm_component:
  - *function new (string name, uvm_component parent);*
- UVM recommends not to change the prototype of this constructor in any derived class
  - UVM base class would call this new() internally during object creation.

Using create() instead of new()

- User code should instead call a proxy method named create() that in-turn calls the new() after consulting any overrides set by end user code.
- At a high level, this create() goes and checks in the factory table to see if there was an override and returns derived object or the base object otherwise
  - A pseudo code describing this behavior is shown below:

```
if (factory_override_table.exists(vlb_drvr))
    create = derived_class::new();
else
    create = base::new();
```

Advantages

- A typical usage of create() instead of new() looks as
  - *vl_ctrl_reg = vl_ctrl::type_id::create("vl_ctrl_reg");*
- With the create() routine, we now have a mechanism to set an override in a table/registry and swap a base class with a derived class.
- This is core to reusing components and transactions in UVM including register models.
- At SoC level, a factory override can be done to leverage on individual IP owner's deep know-how on the IP programming sequence and yet tailor to a SoC verification scenario.

Agenda

- Background and Introduction
- Design Partitioning at IP level verification
- Tweaking UVM RAL Predictors
- Ensuring Reusability – UVM Factory
- Conclusion

Preliminary results & Conclusions

- IP level verification has to be thought upfront on requirements of future reusability.
- Some of the IP level work has to be re-factored, re-engineered at a small cost – keeping in view of the bigger benefits that SoC level verification will yield.
- Ability to leverage on IP level knowledge directly at SoC level in the form of UVM sequences.
- Involve all from Design architects early in the verification brainstorming sessions to arrive at optimal choices/partitions for reusing verification.
- We have heard the term “DFV – Design For Verification”, but this experience shows something at a higher level of “architecting IP partitions for easier reuse at SoC level verification”
- Finally, we stand by our title – Reuse isn’t free!

Questions

Thank You!