KB Verification Library and Coding Guidelines

1/31/2018

Verification Team

KANDOU BUS

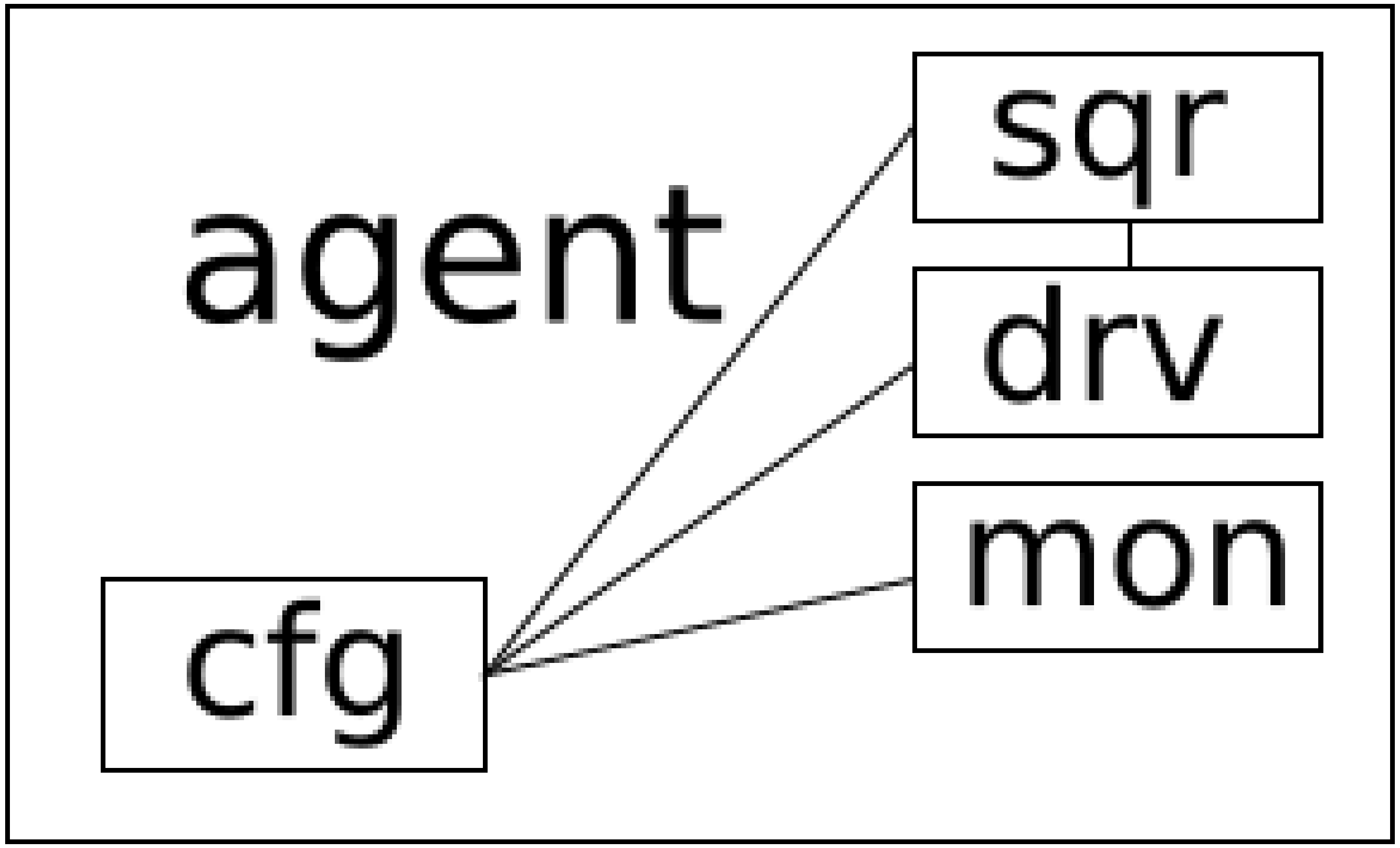
Outline

- Presentation of existing base classes
- How to use them
- Future developments
- Coding guidelines

KANDOU BUS

Existing base classes : agent structure

- Kb_agent
  - Kb_mon
  - Kb_drv
  - Kb_sqr
- Kb_agent_cfg



Existing base classes : agent

- Kb_mon functionality
  - Instantiate analysis port
  - Create transaction handle
  - Get agent config handle
  - Handle reset : TODO
- kb_drv
  - Get agent config handle
  - Handle reset : TODO
- Kb_agent
  - Create drv/sqr/mon type as specified in config
  - Connect/create driver and sequencer if ACTIVE
  - Get virtual interfaces and connect them to drv/mon

Existing base classes : agent config

- Kb_agent_cfg
  - Extends from uvm_object
  - Encapsulate all the config for the agent
    - Check_en, coverage_en, is_active, …
  - Define mon/dvr/sqr types
  - Keep track of number of transactions
  - Contains generic control fields such as m_ctrl[string]
- Having one config per agent allows great vertical/horizontal reusability

Existing base classes : clock agent

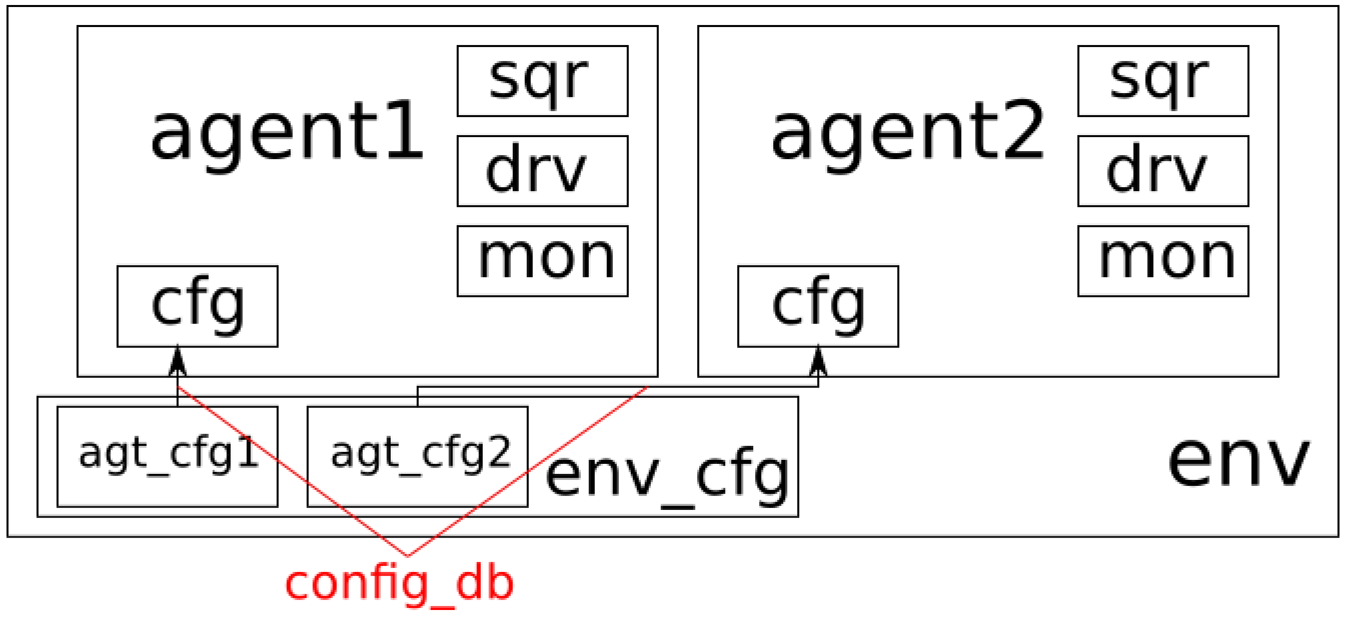
- Let's see use case of agent base classes with kb_clock_agent
  - Clock_agent_config contains all the information
    - Mon_type_name = kb_clock_monitor
    - Drv_type_name = kb_clock_driver
    - Set_active
    - Printing format code
    - Clock_cfg
      - Defines clock names to drive and their config
        - Period
        - Duty cycle
        - Jitter
        - …
- This config is created in the env/test and then given to agent through config_db

Existing base classes : reset agent

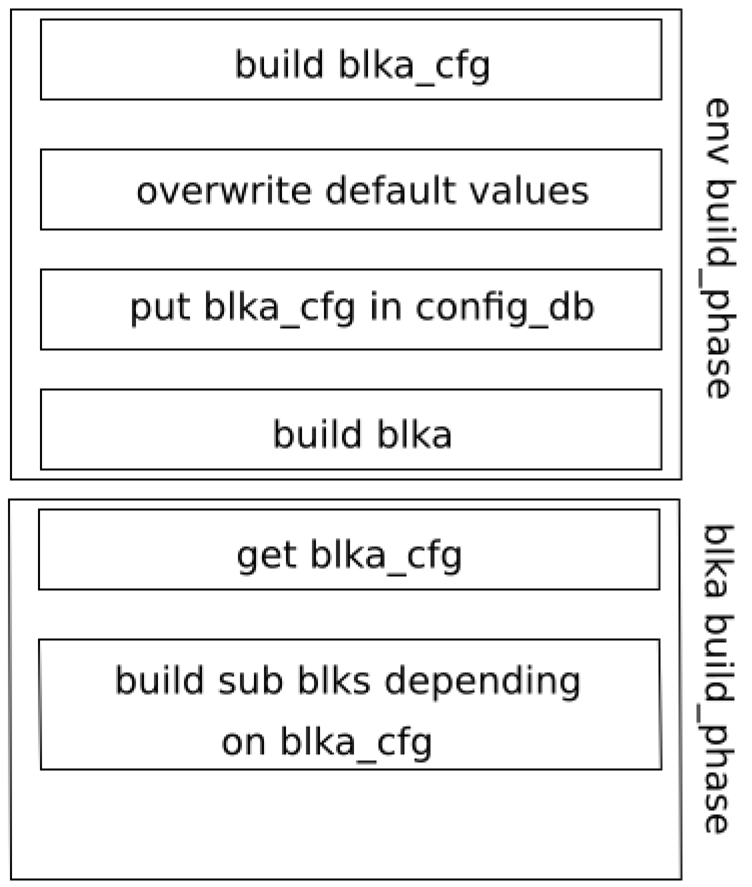
- reset_agent_config contains all the information
    - Mon_type_name = kb_reset_monitor
    - Drv_type_name = kb_reset_driver
    - Set_active
    - Printing format code
    - Reset_cfg
        - Defines clock names to drive and their config
            - Polarity
            - Pulse width
            - Init value
            - …
- This config is created in the env/test and then given to agent through config_db

How to use base classes

- In general case, we have the following architecture
  - One config at the top, that contains all the sub configs.
  - Top config is built in the base test
    - Then all sub configs are built as well
  - Then each extended test can overwrite some top/sub config parameters
  - Sub env/agents are then built and get correct updated configs
- This scenario has been tried successfully on glasswing environment



How to use base classes



Existing base classes : base test

- Contains generic UVCs
  - Custom report server
  - Report catcher for handling report errors
  - Watchdog for timeout handling
  - Assert monitor for checking assertions/cover/assume are covered for specific tests

- Handles end-of-test mechanism
- Add debug mechanisms : print factory, field override
- Hooks for config object program
  - Used for overwriting default values of config objects before the objects that use these config are built

Common macros

- Factory registration
  - For parameterized object/classes
    - Kb_param_object/component_registry
- For reporting : use get_type_name() for ID field and string formatting
  - `kb_info((Tests %d, 1), UVM_NONE)
- For dynamic upcast
  - `kb_dcast(dest, src)
- For assertion definitions
  - Automatic property/assertion naming
- For config_db get/set
  - Automatic fatal report if get fails

KANDOU BUS

Existing base classes : RAL

- Kb_reg_block
  - Define shared init method
    - Configure, build, set_base_addr, lock_model and reset
- Kb_reg_model_cfg
  - Contains configuration of the reg model
    - Register mode : EXPLICIT/IMPLICIT
    - Adapter_type_name
    - Register coverage enable
    - Reg block name
  - It is provided to the reg model and predictor
- Kb_reg_model
  - Build adapter
  - Provides method for connecting adapter
- Kb_reg_predictor
  - TODO : provide mechanism for automatic connection to bus agent

How to use base classes

- Build phase
  - RAL objects are built in base test : TBC
    - Fields can be set/randomized in test and then there is automatic update of the RAL in the configure phase of the running sequence.
  - Reg block is first built with specified base address
  - Reg model config is built and programmed
    - Set adapter type name, reg mode, …
  - Set reg model config in config db
  - Build reg model and predictor
- Connect phase
  - Connect bus agent with reg model method
  - Connect bus agent with predictor

Future developments

- Automatic generations of UVCs
  - Use templates for agents/mon/drv … that extends base classes

- Develop methodology/base classes for sequences in order to ease vertical/horizontal reuse : sequence_cfg ?
- Complete features for clock, reset agents
- Use Doxygen for base classes documentation
- Coverage base class encapsulating covergroups
- Wavedrom json timing diagram for interface
- Develop common header
- Add example for base classes
- Add description in header of each test/sequence

Common coding Guidelines

- Common indentation level = 2
- No TABs
- Use one file per module/class/interface except for specific cases
- Identifiers should use lower_case_with_underscores except for parameters, enum literals, constants and `defines which use UPPER_CASE_WITH_UNDERSCORE
- Macros may use lower_case_with_underscore if they provide procedural code
- Add label for endclass delimeters, begin end blocks
- All base classes should start with prefix kb_
- All base macros should start with prefix kb_macro_
- Use prefix m_ before the name of user-defined class members (properties). Exception for ports/exports and vif
- Use prefix _ ? for private members
- Use prefix h_ when you get handles from config_db or when you upcast components/objects
- These guidelines are consistent with UVM base class library

Common coding Guidelines

- Use suffixes _env, _agent, _test, _drv, _mon, _sqr, _vsqr, _seq, _vseq, _sb, _trans for corresponding classes except base classes
- Use suffix _cfg for user-defined configuration classes
- When configuration objects are referenced from config_db, field name should be "cfg"
- Use suffix _port / _export for corresponding port/export instance. They do not need m_ prefix since they are always class member variables.
- Use suffix _vif for virtual interfaces
- Use suffix _t for user-defined typedef
- Use suffix _e for enumerated types
- Use suffix _cb for clocking blocks
- Use suffix _mp for modports
- Use suffix _cg for covergroups
- Use suffix _ctr for constraints
- Use suffix _pkg for user-defined packages
- Do not use UVM deprecated features
- Do not use internal features of UVM that are not documented
- Use conditional guards to avoid compiling the same include file more than once

KANDOU BUS

Common coding Guidelines

- Primary focus should be reusability, we should avoid introducing dependencies that would prevent subsequent reuse
- Each class should be defined within a package
- Do not use wildcard import at compilation unit scope (outside module/package declaration)
- Env should be written so that they can be integrated in another top env
- All class should be registered in the factory with corresponding macros
- Always create objects/components through the factory
- Handle name should match string instance name of factory
- Use new constructor for covergroup and port/exports creations only
- Use build_phase for building other elements

Coding Guideline file structure

- File structure
- Common header
- Include guards
- Possible include files
- Class declaration
- All class should be registered in the factory with corresponding macros
- Covergroup definition
- Declare ports, exports, and virtual interfaces
- Declare any member variables
- Constraints declaration
- After any member variables, define constructor/ covergroup creation
- If class implements multiple phase methods, keep their order in the file. First declare build_phase, connect, …
- Other functions use extern if many lines of code

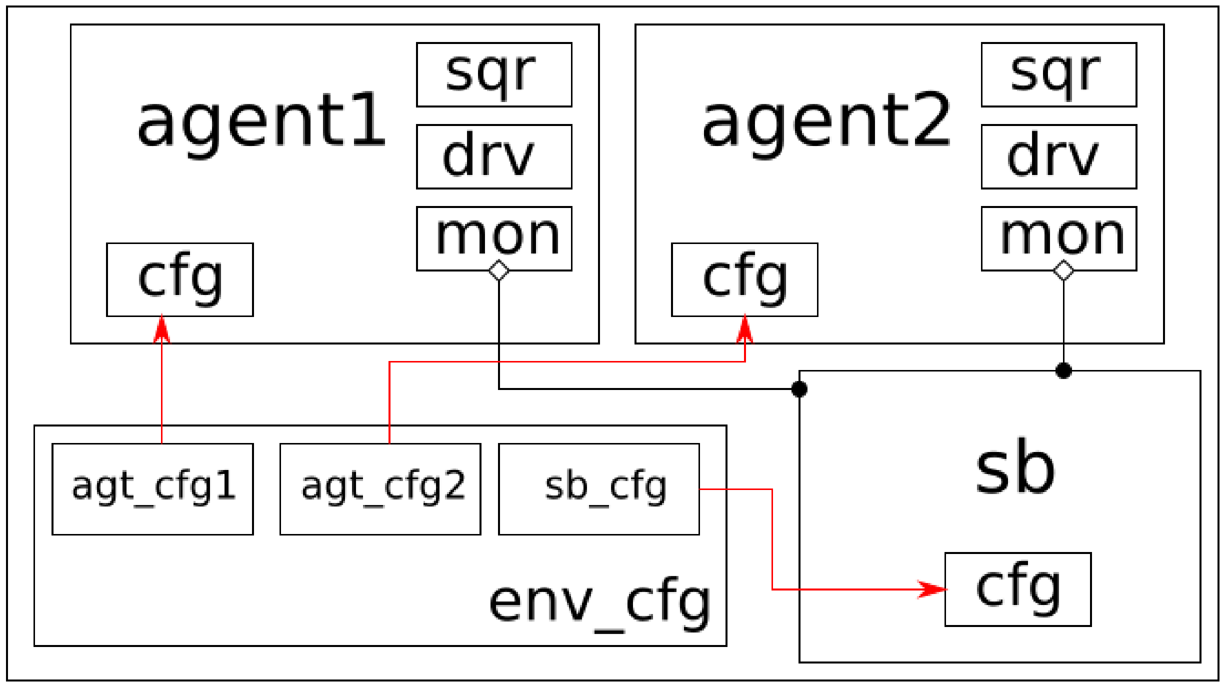
- Enclass
- `endif

KANDOU BUS

Coding Guidelines : env

- Inside env we can find following structure
  - Agents
  - Scoreboards
  - Configs
  - Exports/Port forwarding
  - Subscribers
- Each env may be reused at higher level so only put components that can be reused : no virtual sequencer, no bus agent.
- Scoreboard and their connections are placed inside the env.
- Env has also its own config containing sub configs for the agents and scoreboards

Coding Guidelines : env



Coding Guidelines : agent

- For agent base classes use following naming style
  - Kb_VIP_monitor
  - Kb_VIP_driver
  - Kb_VIP_sequencer

Coding Guidelines : packages

- Group includes for typedef, sequences, components in svh files

Coding Guidelines : uvm_sequence

- For any common code use pre/post_start instead of pre/post_body because you are not sure if the latter method will actually be called
- Use macro declare_p_sequencer only when the sequence needs to access members of the sequencer on which it is running

Backup

KANDOU
reinventing the
BUS

KANDOU BUS