WLAN UVM Verification environment architecure

Verification plan and architecture

Document Reference

Version 1.0

2018-10-08

RivieraWaves
http://www.rivierawaves.com/

Revision History

Version	Date	Revision Description	Author
0.1	2016-05-27	Initial release	SR
0.2	2016-06-02	Diagram updates and rename of modem_config object. Appendix added with coding guidelines.	SR
0.3	2018-09-26	Added chapter for simulation flow and detailed verification component description	SR
0.4	2018-10-02	Added agents and UVC detailed description	SR
0.5	2018-10-03	Added MAC RX testcase descriptions	LM
0.6	2018-10-05	Added MAC TX testcase descriptions	LM
0.7	2018-10-05	Added scoreboard description	SR
0.8	2018-10-08	Added WLAN testcase descriptions	LM
0.9	2018-10-08	Overview update	SR
1.0	2018-10-08	Simulate bash script described	LM

Table of Contents

List of Figures

List of Tables

1 Overview

1.1 Document overview

This document describes the UVM testbench for RivieraWaves's RW-WLAN-nX 802.11a/b/g/n/ac/ax modem and HW MAC core design. The purpose of this document is to describe the testbench architecture and configuration. The goal is to give sufficient information for users who are not familiar with UVM to be able to run simulations and make changes to or configure the testbench.

Second chapter describes the basic UVM verification components, overall verification environment, sequencers and test case organisation.

Third chapter describes the testbench architecture; PPDU frame modelling and PPDU frame object descriptors, data path verification structure. Tables that are used in the TB (e.g. keyRAM, DMA descriptors) are described.

Forth chapter describes verification strategy and goals of this testbench.

Fifth chapter gives detailed description of every agent and UVC, common components and there usage in verification environment.

Sixth chapter contains full overview of all test cases and sequences used in testbench, every test case is described in table with its detailed scenario description and configuration.

Seventh chapter explains simulation run phase and simulation flow, directory structure of UVM testbench and how to run test cases or regression lists.

2 Basic UVM verification components

This chapter will give basic description of verification components that are going to be used in testbench architecture. Basic UVM agent and UVC component will be described, with block diagrams and interconnection between components inside them.

Basic components that are also part of testbench are UVM environment, scoreboard, configuration, virtual sequencer, sequence item, register model and register model adapter.

Coding guidelines are also provided in this chapter. These guidelines will be followed throughout development phase and future test/sequence development phase.

2.1 UVM agent

Agent is a main basic component in verification environment. Purpose of agent component is to manipulate with signals on some given interface. Agent has an active part responsible for driving signals on given interface and a passive part responsible for monitoring signals on given interface (collect them into meaningful transactions called sequence items). Basic parts or components in agent are (shown on Figure 2-1):

- Configuration
- Sequence library
- Sequencer
- Driver
- Monitor
- Coverage
- Interface

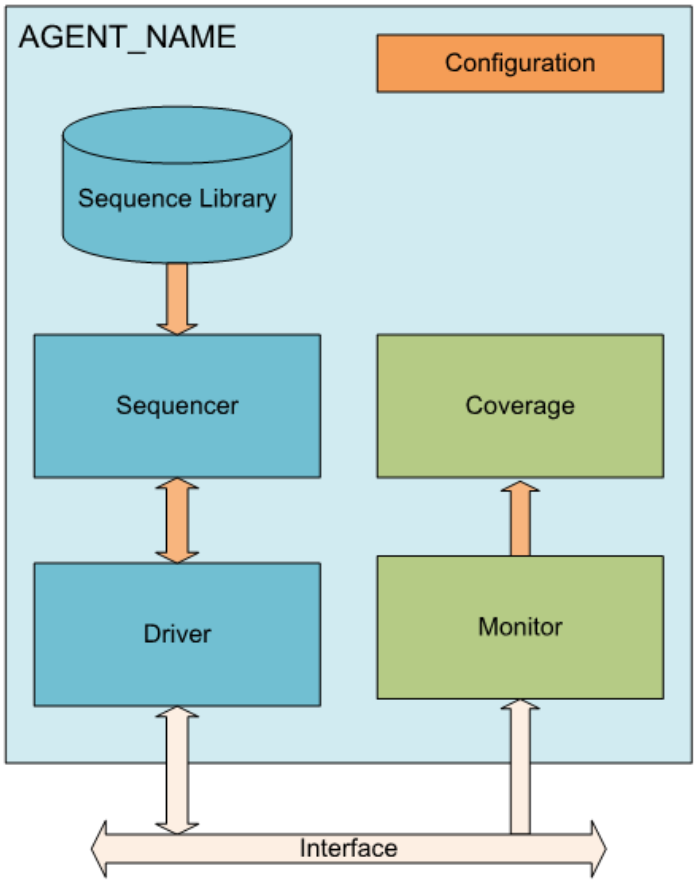

Figure 2-1 Active agent block diagram

Every active agent has a configuration object that contains fields that will setup agent in some specific functionality manner, for example to be active-passive, bus width, number of monitors, number of drivers, type of transactions and many more other settings. Configuration is used also for configuring top level verification components, it can contain fields that determine whether some agent, scoreboard, sequencer will be instantiated in the system.

When agent is active, meaning that it will drive signals on interface, it has 3 components on active path: driver, sequencer and sequence library. Driver is responsible to drive transactions on interface signals when it receives sequence item from dedicated sequencer. After driver gets its item, item will contain relevant information what driver needs to drive on interface signals and with that create transactions. The sequencer serves as an arbiter for controlling transaction flow from multiple stimulus generators. More specifically, the sequencer controls the flow of sequence item (transactions) generated by one or more sequences. Sequence library is a collection of different sequences that can be executed on driver component. Those sequences can have different constraints making sequence items behave in different manner when needed.

Besides active path, agent has passive path components: monitor and coverage collector. Monitor is responsible to observe interface signals and collect transactions in order to create a sequence item. This collected sequence item can them be sent to checkers or scoreboards for further analysis. Coverage collector is a component responsible for collecting functional coverage of transaction. Agent can be set to contain only passive components and agent is called “passive” agent (shown on Figure 2-2). This type of agent wont drive any signals on interface it only collects transactions on interface signals.

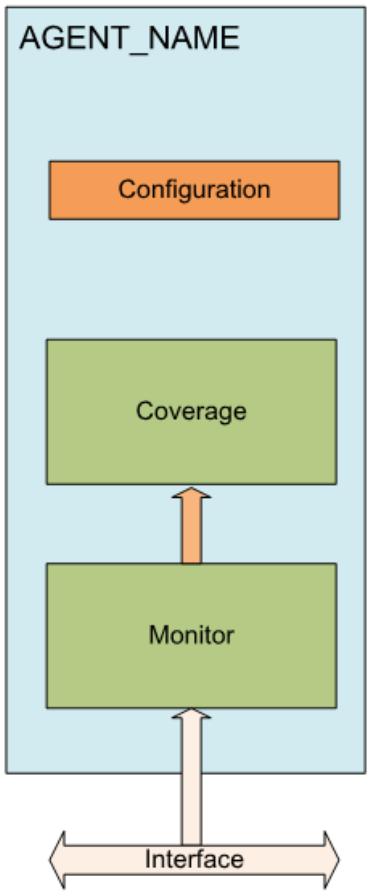

Figure 2-2 Passive agent block diagram

2.2 UVC – Universal Verification Component

A UVC (UVM Verification Component) is a verification component designed for use in UVM. It is a multi-faceted definition and has different layers of meaning in different contexts. Each verification component connects to a single DUT interface and speaks a single protocol, optionally either as a master or a slave endpoint. UVC is a verification component that contains a configurable number of instances of the above Protocol UVC, either in master or slave or passive mode, and configured and hooked up coherently as a unit. The purpose here is to verify a structured fabric with multiple interfaces of the same protocol.

UVC architecture contains these components:

- N number of agents
- Configuration
- Scoreboard (optional)
- Bus monitor (optional)
- Virtual sequencer
- Sequence library

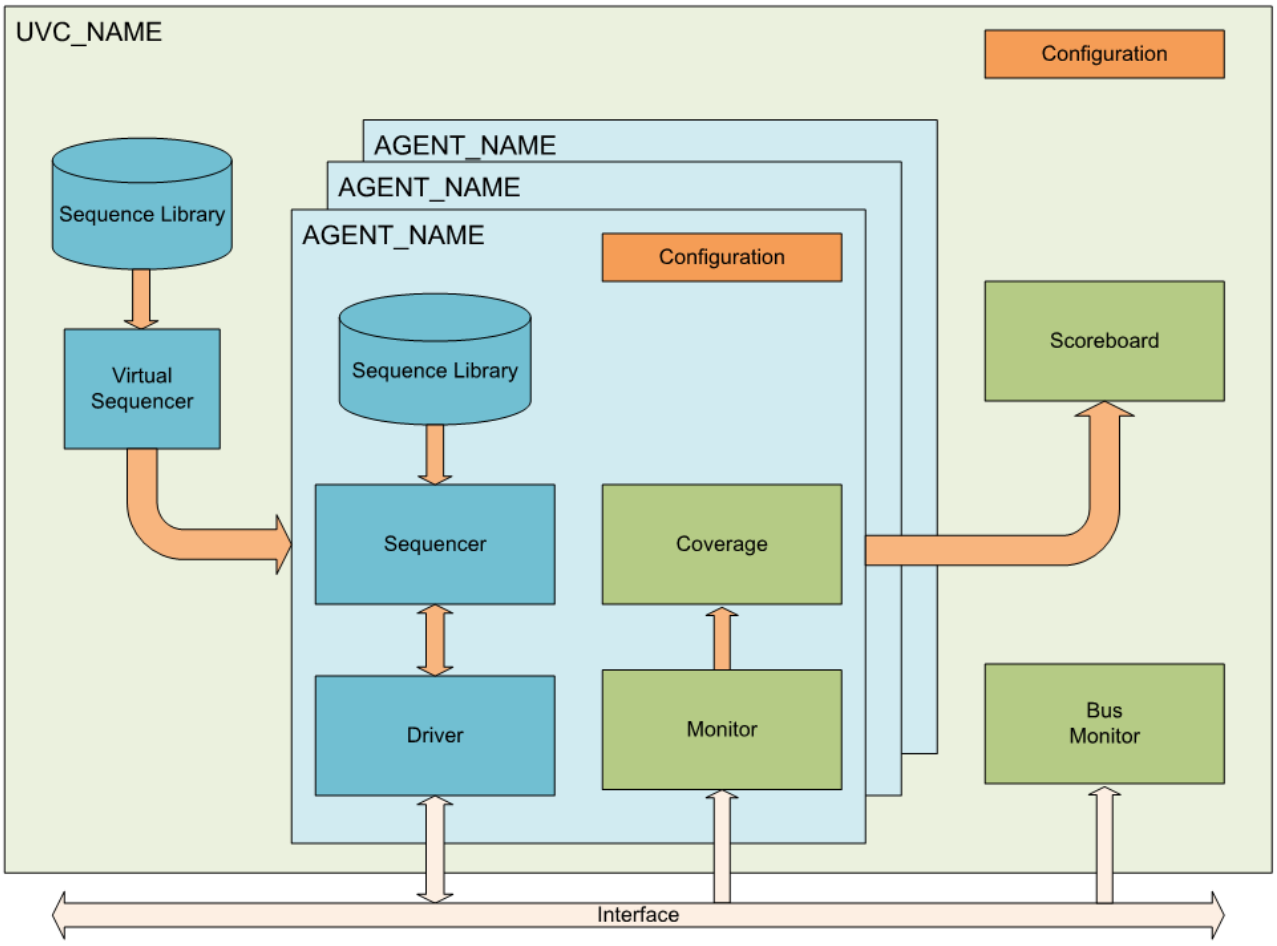

Figure 2-3 UVC block diagram

Main characteristic of UVC is that it can contain multiple numbers of agents inside, that work on the same bus interface. Interface is divided into several functional parts that are connected to dedicated agent which will operate on that interface. With this organization we can divide several driving (active) paths in the system, also divide several collecting parts in the system. To be able to setup UVC properly, configuration object is introduced inside UVC. Main goal of configuration object is to setup whole UVC environment, how many agents it will have and of which type will the agents be. Basically to control what component instances are going to be created during build phase.

There are 2 optional components that can be part of UVC, scoreboard and bus monitor. Scoreboard is used to receive sequence items from different agents and to do some checking of transactions that occurred on interface signals. Bus monitor is a passive component used for collecting interface signals, but the difference in comparison with agent monitor collecting is that interface signals that it monitors are not functionally related (are not part of the same protocol for example) and bus monitor can do some checking on collected signals.

Virtual sequencer and sequence library are used for driving paths of agents to control which sequence will be executed on which sequencer-driver. Virtual sequencer has a pointer to every agent sequencer in the UVC so the active paths are controlled from one component.

2.3 Coding guidelines

Detailed description of coding guidelines and folder structure is given in appendix.

3 WLAN testbench architecture – overview

In this chapter overview of WLAN testbench is presented. Testbench organization is divided into 3 parts or 3 types of testbenches. In testbench there is a wrapper around WLAN IP core that can be configured to be:

1. Modem standalone TB
2. MAC platform TB
3. WLAN platform TB (MAC + Modem TB)

With these switches all 3 configurations of testbench can be tested as standalone. For better coverage it is better to have possibility to divide IP as Modem or MAC standalone and test it like that. This kind of architecture will give possibility of verification component reuse in all 3 testbenches. Only thing that differs is which components are instantiated and how are they configured to behave in the system (to be active/passive, master/slave).

One UVM environment component is used to wrap all agents, UVCs, scoreboards, bus monitors, sequence library, virtual sequencer and configuration into one object/component. With this kind of organization modularity is high, because environment can be easily configured in any shape and form that is needed for current testbench. It is easy to create and configure agents and UVCs to behave in desired manner.

Test case scenarios will be controlled from virtual sequencer which has handles on all sequencers in the configured TB architecture. This allows virtual sequencer to set which driver will be active in the system and which sequence will be executed on which driver.Sequence library contains sequences dedicated to standalone andMODEM, MAC or for WLAN testbench. Reuse of some sequences is possible on all 3 types of testbenches.

Main sequence item in testbench is PPDU frame (PLCP protocol data unit) as a main object of communication in WiFi network. This item is interpreted in different way depending on the interface on which it occurs. It can be interpreted on Modem side, MAC side, Host side, RUI side etc. When PPDU frame is created, in test case, it can be used for both transmit or receive testing path. More details about this topic refer to chapter 3.4. PPDU frame object modeling.

Regarding passive components in testbench architecture, there are dedicated scoreboards for protocol and packet checking (also intermediate checkpoints), assertions in bus monitor components and coverage models (functional coverage collecting). All these components are configurable via main configuration object. Also a passive component that is part of the environment are register model, UVM register layer component containing all register definitions and register maps from WLAN IP (Modem, MAC, DMA, Radio, AGC etc.).

On the top of the testbench architecture is the base test case object that contains instance and creation of environment component and testbench configuration object inside. Base test case will configure testbench architecture to proper setup of verification components in relative to one of the 3 types of TB architecture. All test cases will extend this base test case so reconfiguring environment is possible from test case to test case.

3.1 Modem standalone TB

In Modem standalone TB, functionality of Modem IP is tested. On input of the Modem is radio interface and on the other side is MAC-PHY interface through which Modem is connected with the MAC CORE IP. Modem is tested in Rx and Tx path respectively and for this purpose MATLAB shared library is used. MATLAB is used to provide stimuli arrays, also golden reference data for checking inside scoreboards, reference data for checking intermediate signal and data processing inside Modem logic (FFT, time domain, frequency domain, bit domain phases). MATLAB model is called from test case when new stimuli data and reference data is needed (more detail about modem data model in 5.2.3).

Components that are part of Modem TB:

- Radio UVC – this component is responsible to get stimuli data from MATLAB model, drive Rx data on ADC interface and collect Tx data from DAC interface. Also this verification component has agent for configuring AGC and agent for serial communication with Modem.
- MAC-PHY UVC – this component is used to behave as MAC IP on this interface. To drive Tx data on MAC-PHY interface and collect data received from Modem side. It is configured to have MAC active agent and PHY passive agent instantiated.

- AHB master agent – this component behaves as master device on AHB bus to communicate with Modem AHB slave block. AHB master drives register RD/WR transactions on bus and collects transactions from bus. Agent is connected to register model component which contains all Modem registers definition, this helps when RD/WR transactions are called from test cases (calling basic functions is simplified).
- Modem data model – this is shared object class used for encapsulating all DPI calls for MATLAB model. Modem configuration implements all call functions that handle MATLAB model, execution of stimuli data, preparation of golden reference data, parsing and generating *sysparam.txt* and *payload.txt* files that MATLAB model uses when triggered.
- Beamforming UVC – this verification component is used to mimic MAC behavioral when beamforming is performed. In Modem standalone TB this UVC is configured to be active from MAC point of view, it reads from Modem beamforming FIFO and send data back to Modem via MAC-PHY interface.
- Bus monitors – passive components used to monitor internal signals and streams and do checking and comparing with MATLAB golden reference data.
- Modem scoreboards – main checking components in Modem TB. Used for checking Rx and Tx path packets/frames, timing checks, interrupt checks.
- Configuration – component used to configure all verification components in environment.
- Virtual sequencer – component responsible for executing and managing sequences on all active components in environment.

Figure 3-1 shows block diagram of Modem standalone testbench architecture.

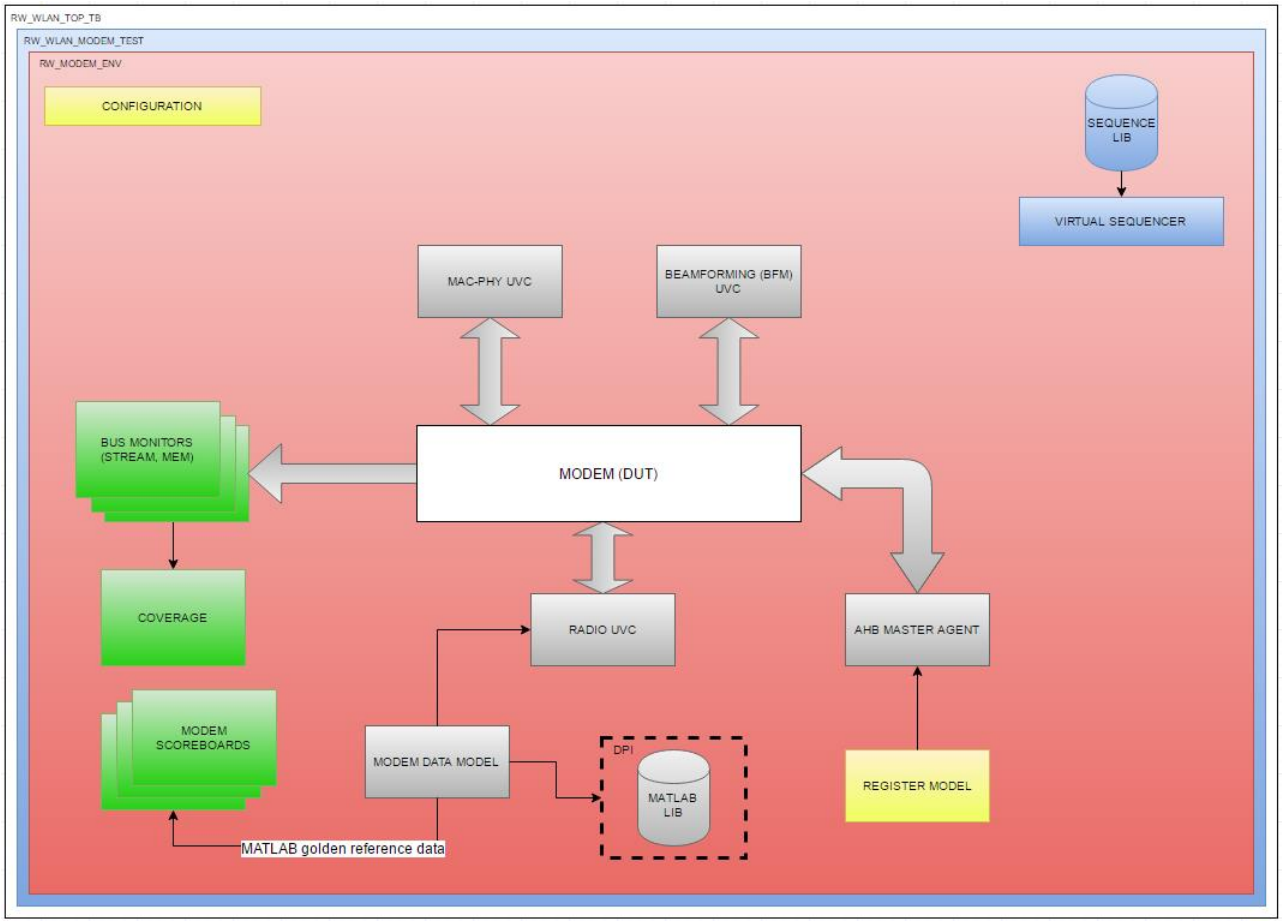

Figure 3-1 Modem standalone TB architecture

3.2 MAC platform standalone TB

In MAC standalone testbench functionality of MAC platform is tested. MAC IP has several interfaces connected to it from host machine, CPU subsystem, shared memory (SRAM), MAC-PHY interface with Modem, interrupt controller, IPC, beamforming interface with Modem. MAC is tested in Rx and Tx path from different sides:

- Rx/ Tx path from/to host device from/to SRAM
- Rx/Tx path from/to SRAM to/from MAC-PHY interface
- Rx/Tx path from AHB master to SRAM
- Rx/Tx path with beamforming

Components that are part of MAC TB:

- MAC-PHY UVC – this component is used to behave as Modem IP on this interface. To drive Rx data on MAC-PHY interface and collect data received from MAC side. It is configured to have PHY active agent and MAC passive agent instantiated.
- AHB master agent – this component behaves as master device on AHB bus to communicate with MAC AHB slave block. AHB master drives register RD/WR transactions on bus and collects transactions from bus. Agent is connected to register model component which contains all MAC, DMA, IPC, interrupt, SYSCTRL register definitions, this helps when RD/WR transactions are called from test cases (calling basic functions is simplified).
- SRAM agent – this component is responsible to provide backdoor access to SRAM memory array, so storing data inside is much faster than through front door access. Also reading data through backdoor is faster than front door access. This agent is able to store Tx descriptors in memory so MAC can read them and send frame to the MAC-PHY interface.
- AXI slave agent – this component is the main interface to the host device and DMA platform. Purpose of this agent is to read from SRAM via DMA access. This agent collects Rx descriptors from memory and creates MPDU frame.
- Interrupt agent – this component is passive and it is responsible to collect toggling of interrupt line in MAC interface.
- IPC agent (Inter Process Controller) – this component is passive and it is responsible to collect toggling of IPC signal in MAC interface.
- Interrupt raw agent – this component is passive and it monitors and collects all interrupt signals (maximum 64 sources) in MAC IP. It is attached on interrupt signals before they are connected to MUX and propagated to output signal. This agent provides information from which block in MAC has interrupt occurred and sends that information (item) to scoreboard for checking.
- Beamforming UVC – this verification component is used to mimic Modem behavioral when beamforming is performed. In MAC standalone TB this UVC is configured to be active from Modem point of view, to provide FIFO data for MAC to read out.
- Bus monitors – passive components used to monitor internal signals and streams and do checking and comparing with MATLAB golden reference data.
- MAC scoreboards – main checking components in MAC TB. Used for checking Rx and Tx path packets/frames, timing checks, interrupt checks, DMA access.
- Configuration – component used to configure all verification components in environment.
- Virtual sequencer – component responsible for executing and managing sequences on all active components in environment.

Figure 3-2 shows block diagram of MAC platform standalone architecture.

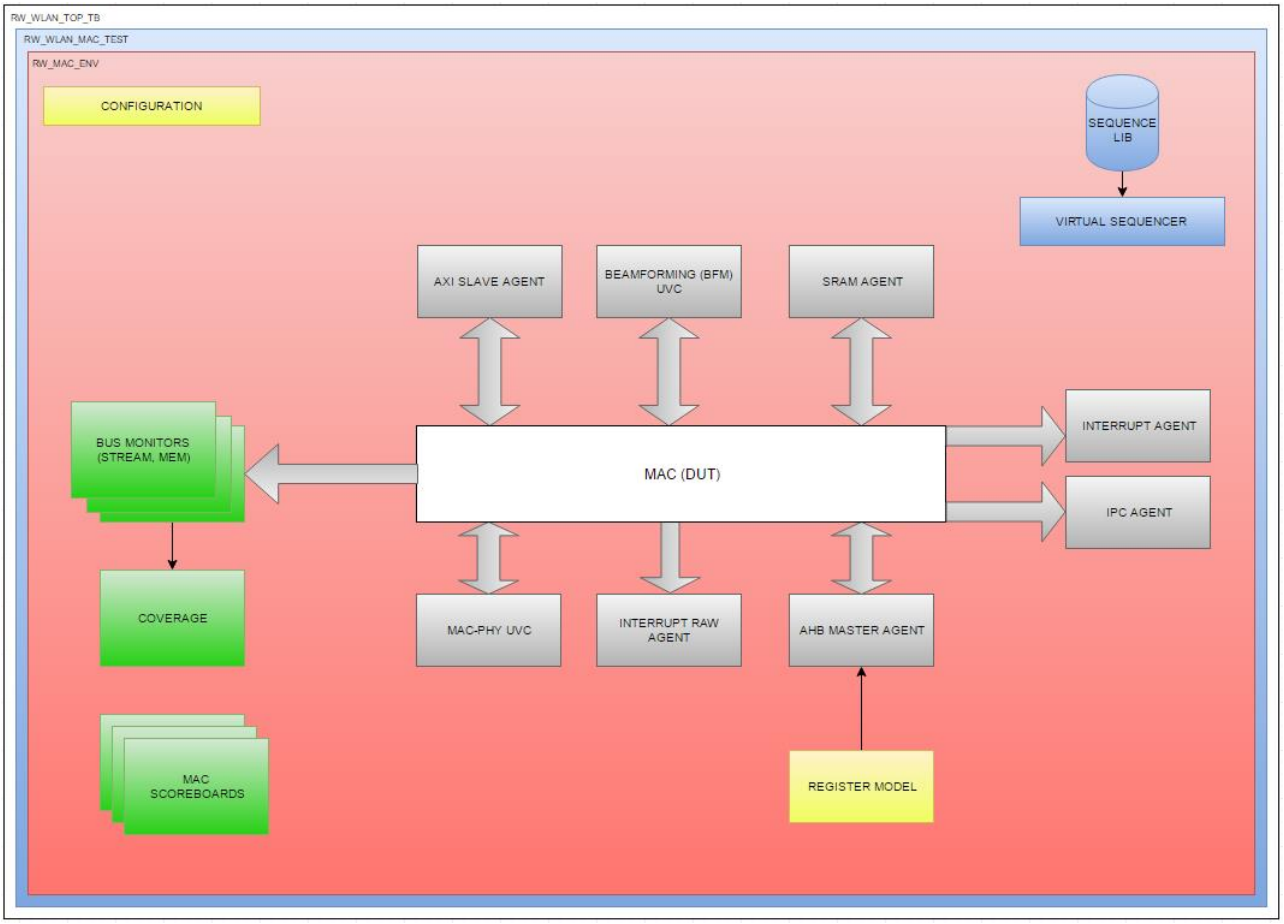

Figure 3-2 MAC standalone TB architecture

3.3 WLAN platform TB (MAC + Modem)

In WLAN platform TB Modem and MAC functionality is tested together, so every feature that is tested in standalone testbenches needs to be tested in combined WLAN TB also. Modem and MAC are connected via MAC-PHY interface. All verification components that are used on Modem and MAC standalone TB are reused on this testbench, only difference is that some of agents are reconfigured to be passive because now MAC and Modem are communicating together. Like in Modem standalone, on input of Modem is radio interface and on other side is MAC-PHY interface through which Modem is connected with MAC IP. Modem is tested in Rx and Tx path respectively and for this purpose MATLAB shared library is used. MATLAB is used to provide stimuli arrays, also golden reference data for checking inside scoreboards, reference data for checking intermediate signal and data processing inside Modem logic (FFT, time domain, frequency domain, bit domain phases). MATLAB model is called from test case when new stimuli data and reference data is needed. WLAN is tested in Rx and Tx path from different sides:

- Rx/Tx path from/to host device from/to SRAM
- Rx/Tx path from/to SRAM to/from MAC-PHY interface and to/from Radio interface
- Rx/Tx path from AHB master to SRAM
- Rx/Tx path with beamforming

Components that are part of WLAN TB:

- MAC-PHY UVC – this component is used to behave as passive monitor. UVC only collects transaction on MAC-PHY interface from MAC and Modems side. Collected transactions are sent to scoreboards for checking.
- Radio UVC – this component is responsible to get stimuli data from MATLAB model, drive Rx data on ADC interface and collect Tx data from DAC interface. Also this verification component has agent for configuring AGC and agent for serial communication with Modem.

- AHB master agent – this component behaves as master device on AHB bus to communicate with MAC and Modem AHB slave block. AHB master drives register RD/WR transactions on bus and collects transactions from bus. Agent is connected to register model component which contains all MAC, Modem, DMA, IPC, interrupt, SYSCTRL register definitions, this helps when RD/WR transactions are called from test cases (calling basic functions is simplified).
- Modem data model – this is shared object class used for encapsulating all DPI calls for MATLAB model. Modem configuration implements all call functions that handle MATLAB model, execution of stimuli data, preparation of golden reference data, parsing and generating *sysparam.txt* and *payload.txt* files that MATLAB model uses when triggered.
- SRAM agent – this component is responsible to provide backdoor access to SRAM memory array, so storing data inside is much faster than through front door access. Also reading data through backdoor is faster than front door access. This agent is able to store Tx descriptors in memory so MAC can read them and send frame to the MAC-PHY interface.
- AXI slave agent – this component is main interface to the host device and DMA platform. Purpose of this agent is to read from SRAM via DMA access. This agent collects Rx descriptors from memory and creates MPDU frame.
- Interrupt agent – this component is passive and it is responsible to collect toggling of interrupt line in MAC interface.
- IPC agent (Inter Process Controller) – this component is passive and it is responsible to collect toggling of IPC signal in MAC interface.
- Interrupt raw agent – this component is passive and it monitors and collects all interrupt signals (maximum 64 sources) in MAC IP. It is attached on interrupt signals before they are connected to MUX and propagated to output signal. This agent provides information from which block in MAC has interrupt occurred and sends that information (item) to scoreboard for checking.
- Beamforming UVC – this verification component is used passive monitor. UVC collects transaction on beamforming interface and sends items to scoreboard for checking.
- Bus monitors – passive components used to monitor internal signals and streams and do checking and comparing with MATLAB golden reference data.
- WLAN scoreboards – main checking components in WLAN TB. Used for checking Rx and Tx path packets/frames, timing checks, interrupt checks, DMA access. All scoreboards from Modem and MAC standalone TB are reused in WLAN TB.
- Configuration – component used to configure all verification components in environment.
- Virtual sequencer – component responsible for executing and managing sequences on all active components in environment.

Figure 3-3 shows block diagram of WLAN testbench architecture.

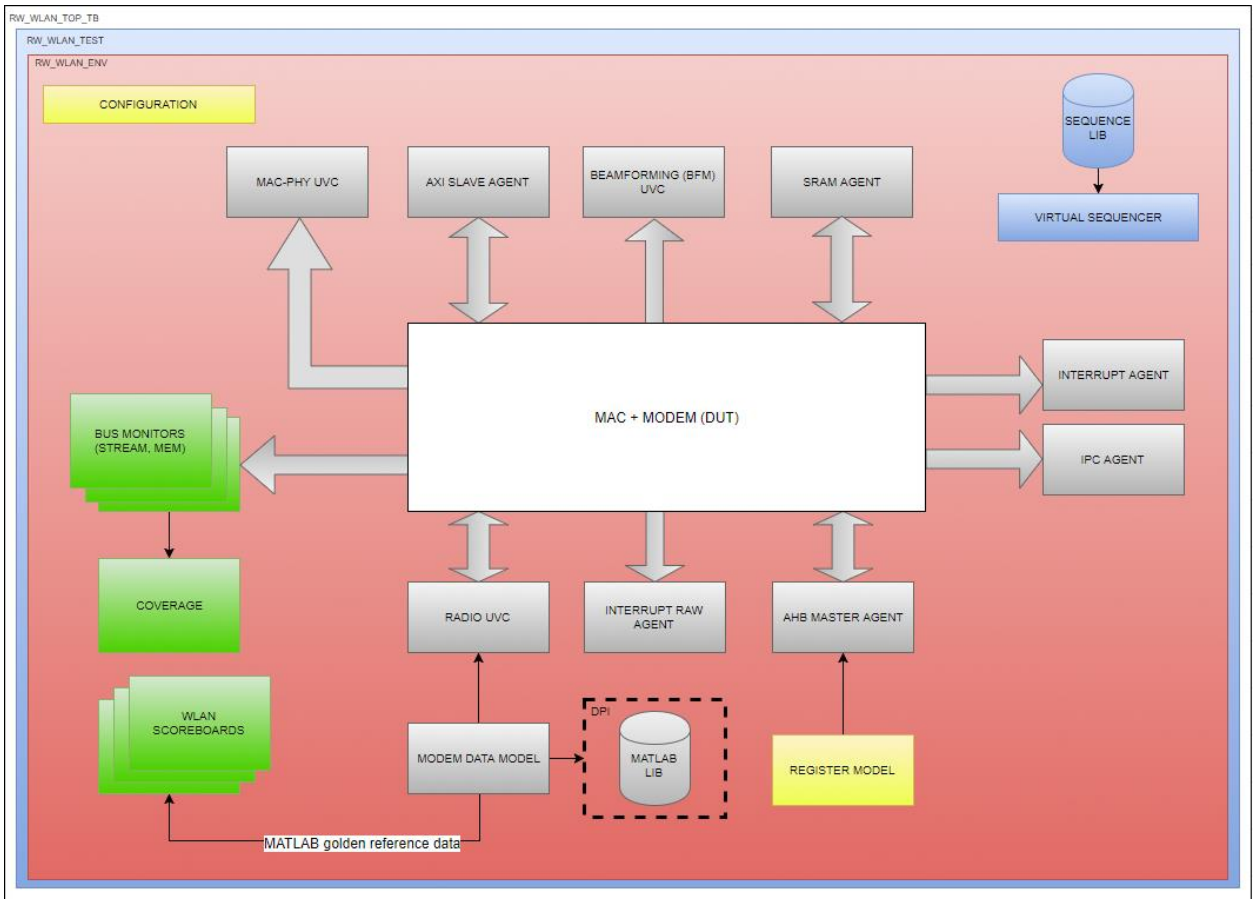

Figure 3-3 WLAN TB architecture

3.4 PPDU frame object modeling

Since WLAN testbench is complex system, there is a need to unify what can be main transaction item that can be interpreted on different blocks in the same way. To have an object that can be interpreted in every layer of testbench flow, from radio side (Radio UVC), from MAC-PHY interface, shared RAM, host device etc. Model of this object has to be modular so it is ease to extend with new features.

Solution for this model is PPDU frame (PLCP protocol data unit). It encapsulates all necessary information that is transmitted or received “from/to the air” and also it can be interpreted by the SW running on host device. PPDU frame object is used on different interfaces and blocks:

- MAC-PHY interface
- MATLAB model
- Radio UVC interface – ADC/DAC interface
- SRAM Tx and Rx descriptors

PPDU frame contains PPDU preamble, PPDU header and PPDU payload. PPDU payload is also known as MPDU frame (MAC Protocol Data Unit). All 3 parts are structures that are complex and are modeled separately with their own types and subtypes. PPDU preamble, header and payload are defined as base class objects that are extended by other class object in which with constraining specific subtypes are written.

In next chapters in more detail is explained how PPDU frame parts are modeled and overview of class hierarchy and relations are given.

3.4.1 MPDU frame object

Like said in previous chapter MPDU frame is basically payload part of PPDU frame. MPDU stands for MAC Protocol Data Unit. Every MPDU frame object contains 3 fields: MAC header, frame body and FCS (Frame Check Sum). In the Figure 3-4 structure of MAC frame is shown. MPDU frame (shown in blue rectangle) represents base class object with 3 fields (that are illustrated in the diagram on right side of MPDU frame block). These 3 fields are structures with defined standard fields like in IEEE 802.11 standard. Structure fields must contain all relevant fields of legacy and new standard features, because we need to be also back compatible in our testbench.

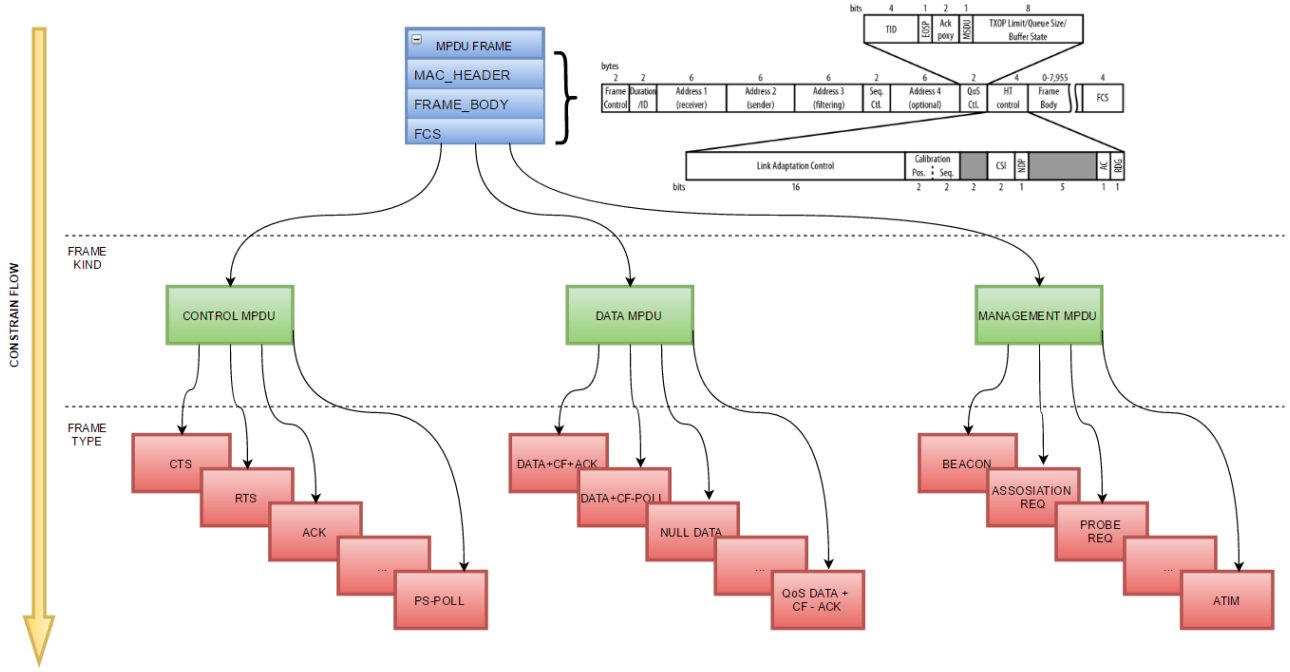

Figure 3-4 MPDU frame object model

MPDU frame has 3 kinds (show in Figure 3-4 with green rectangles):

1. Control MPDU
2. Data MPDU
3. Management MPDU

These 3 frame kinds are derived from MPDU base frame object with applying their own specific constrains to make MPDU object behave as a kind needed in testbench. Fields that are constrained are MAC header and frame body (for example frame body length for control MPDU is 0). Specific frame type is created as one of 3 frame kind objects are derived with new constraints. With this hierarchy flow of deriving and constraining all frame type class objects are made. With this organization it is easy to create new frame for transmit in testbench without any need for further constraining. These frame types are shown in Figure 3-4 in red rectangles.

Since every frame type is derived from base MPDU frame, only instance in the environment is type of MPDU frame and when object is created dynamically (with factory create::) it can be overridden with any subclass frame type and constrains are applied automatically. This makes sequence items easy to manage and create inside test cases.

3.4.2 PPDU frame object – sequence item

Main sequence item in WLAN testbench is PPDU frame object. This frame gives big flexibility and modularity so it can be interpreted by different verification components, active and passive, in any way that suits that verification component. Table 3-1 shows, how PPDU frames are interpreted by each verification component. To trigger MATLAB model to give us stimuli data and golden reference data, first 2 files need to be prepared (*sysparam.txt and payload.txt*). Creation of these 2 files is done by parsing already created PPDU frame and storing relevant information into these 2 files. For MAC-PHY interface, PPDU frame is interpreted and prepared as Tx/Rx vector plus data, so active

components can send them over interface signals. Regarding SRAM Tx descriptors they interpret PPDU frame as policy table plus transmit buffers and stores that information to memory. Same as Tx, for Rx descriptors they are stored in memory as RHD (Receive Header Descriptor) and RPD (Receive Payload Descriptor).

PPDU frame for	PREAMBLE +HEADER	PAYLOAD
MATLAB	sysparam.txt	payload.txt
MAC-PHY IF	Tx/Rx Vector	Data
SRAM Tx DESCRIPTOR	Policy table	Transmit buffer (THDs and TBDs)
SRAM Rx DESCRIPTOR	RHD	Receive buffer (RPDs)

Table 3-1 PPDU frame modeling in relative to block that uses it

PPDU frame can have 4 kinds:

1. **Singleton** – legacy PPDU frame that has only one MPDU frame in payload
2. **Aggregated** – PPDU frame that has more than one MPDU frames aggregated sequentially one behind other, separated with delimiters
3. **MU-MIMO** - (Multiple User – Multiple Input Multiple Output) this PPDU frame is special type of aggregated MPDU frame, it has up to 4 A-MPDU in one PPDU frame
4. **NDP** - (Null Data Packet) this PPDU frame is singleton PPDU frame but with no payload, so it only caries preamble and header information inside frame

Figure 3-5 shows PPDU frame organization and field definitions.

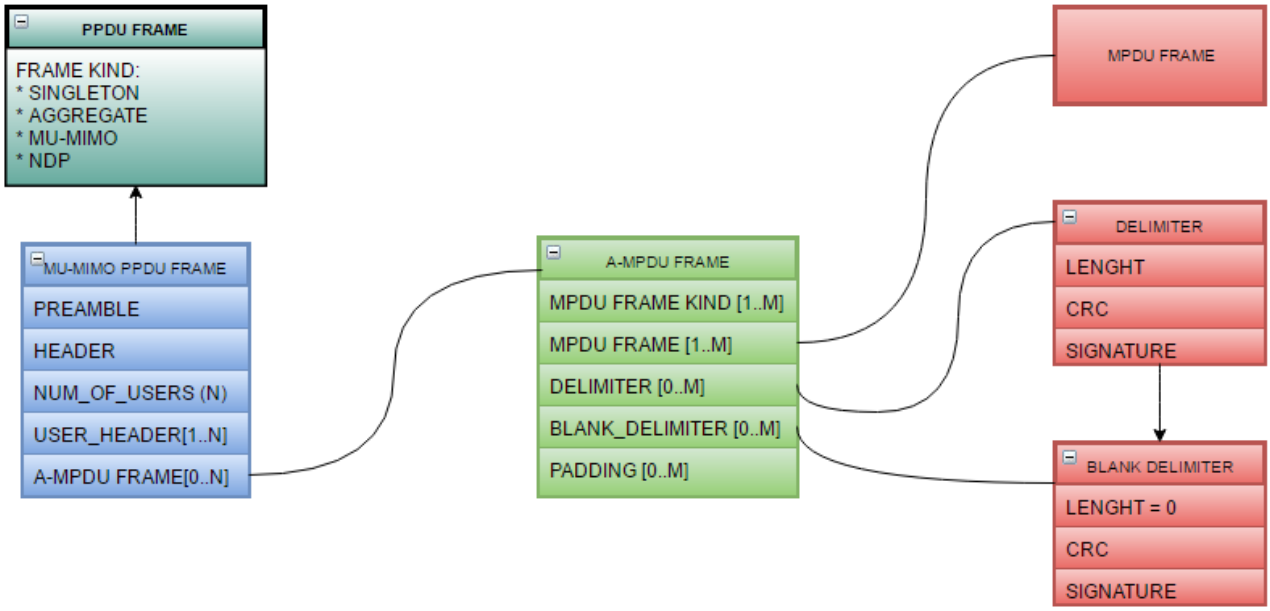

Figure 3-5 PPDU frame object model

Like already said, PPDU frame is divided into 4 types, so base PPDU frame has filed with randomization of frame type. Then new class representing MU-MIMO PPDU frame derives base PPDU frame. This way basically MU-MIMO frame is top in hierarchy and every other PPDU frame type can be get with constraining member fields of class. So from MU-MIMO frame by constraining we can get aggregated, singleton and NDP frames.

Members of MU-MIMO frame are preamble, header, number of users, user header and array of A-MPDUs. All these fields are constrained to be any possible subtype that can occur in IEEE 802.11 standard. This is regarding preamble and header constraining. Number of users can be 1-4, so when PPDU frame needs to be singleton or aggregated this

field is set to 1. User header field is related to MU-MIMO frame when it is generated. Every user has its own specific user header attached to it. Regarding A-MPDU array it represents payload part of PPDU frame. Array can have 0 to 4 A-MPDU packets, array is 0 in case NDP frame is generated and in other cases number of A-MPDU is equal to number of users (maximum 4).

A-MPDU frame is constructed out of several fields:

- MPDU frame kind – enumeration array of several different or same types of MPDU frames that are part of A-MPDU frame (CTS, NULL DATA, BEACON etc.)
- MPDU frame – array of pointers to MPDU frame object (explained in 3.4.1)
- Delimiter – is structure that is part of every A-MPDU frame. When singleton frame is sent, number of delimiters is equal to 0. Delimiter is 4 byte structure with fields for length, CRC and signature. Delimiter is used in A-MPDU frame to set a part singleton MPDU frames inside.
- Blank delimiter – this is a special type of delimiter, it has length set to 0. This object is derived from delimiter with constrain on length filed.
- Padding – every length of A-MPDU frame needs to be multiple of 4 so padding is used for additional bytes in packet. Then don't carry any information.

In test case scenarios when singleton needs to be sent, A-MPDU frame is constrained in a way that delimiter number is 0, blank delimiter number is 0 and padding is 0. MPDU frame kind array has one element, representing singleton MPDU frame type.

3.5 Data path verification structure

The following DUT data paths are verified with the UVM testbench:

- Transmit and receive paths of modem
- Transmit and receive paths of MAC
- Beamforming
- DMA
- IRQ and IPC
- AHB register access

Transmit and receive data paths of both the modem and the MAC are verified in both standalone (only modem or only MAC) and WIFI platform (both modem and MAC instantiated) modes (3.1 and 3.2).

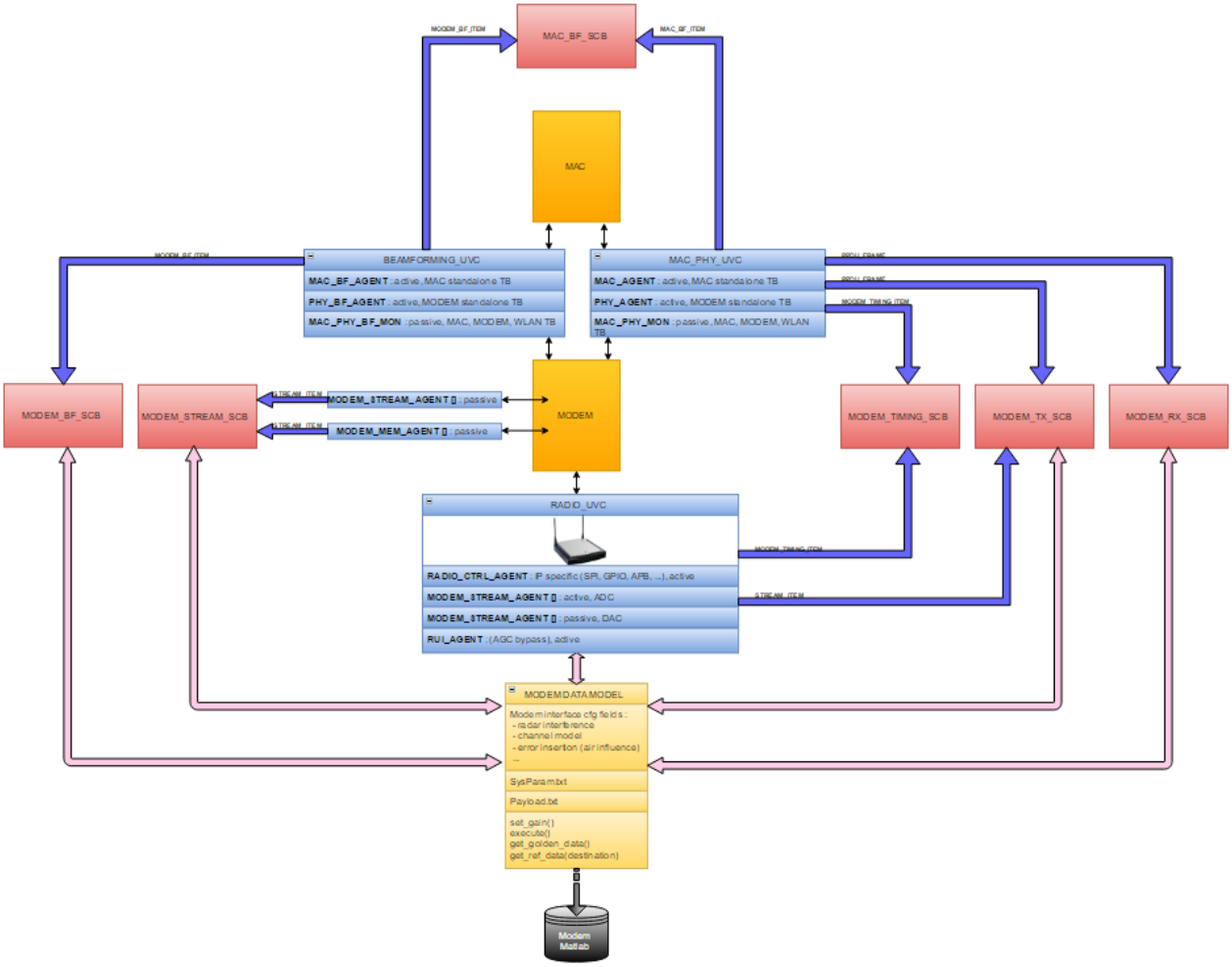

Figure 3-6 Modem and MAC data path verification

Figure 3-6 shows testbench components, MAC/PHY DUT and interconnections between them for data path verification. For better readability, MAC related scoreboards are depicted in a single block with dotted edges in Figure 3-6, and are displayed in more detail in Figure 3-7. IRQ related verification components and connections are shown in detail in Figure 3-8. The testbench is comprised of the following components:

- UVCs:
  - MAC_PHY_UVC
    - MAC_AGENT
    - PHY_AGENT
    - MAC_PHY_MON
  - RADIO_UVC
    - RADIO_AGENT
    - MODEM_STREAM_AGENT
  - BEAMFORMING_UVC
    - MAC_BF_AGENT
    - PHY_BF_AGENT
    - MAC_PHY_BF_MON
- Agents:

    - SRAM_AGENT, active;
    - AXI_SLAVE_AGENT; active
    - IRQ_OUT_AGENT; passive
    - AHB_MASTER_AGENT; active
    - IRQ_RAW_AGENT; passive
- Scoreboards:
    - MODEM_TX_SCB
    - MODEM_RX_SCB
    - MODEM_TIMING_SCB
    - MODEM_STREAM_SCB
    - MODEM_BF_SCB
    - MAC_BF_SCB
    - MAC_TX_SCB
    - MAC_RX_SCB
    - MAC_CTRL_SCB
    - DMA_SCB
    - REG_SCB
    - IPC_OUT_SCB
    - IRQ_OUT_SCB
- Data generator
    - MODEM_DATA_MODEL

All listed components are explained in detail in chapter 3. It should be noted that both MAC and PHY and all of their accompanying verification components have been depicted in Figure 3-8 and Figure 3-7. If the UVM verification environment is used for standalone MAC or standalone PHY verification, parts of the testbench will be left out during UVM build phases (e.g. for standalone MAC verification, the PHY DUT and RADIO_UVC will be left out, and the appropriate scoreboards - MODEM_*_SCB - will not be used either).

It should also be noted that agents inside UVCs change their mode (active or passive) depending on what is being verified. For example, in a standalone MAC verification mode, the MAC_PHY_UVC has one active agent, MAC_AGENT, which is actively driving the interface connected to the MAC and one passive monitor, MAC_PHY_MON which is collecting data for the scoreboards. When the UVM verification environment is used in WIFI platform mode and both MAC and PHY are present, MAC_PHY_UVC doesn't have any active agents, only the passive MAC_PHY_MON. All of these verification possibilities are implemented using the UVM configuration database, which is read during the UVM build phases [1], when the entire verification environment is built and connected. The verification engineer only needs to set the desired configuration (MAC standalone, PHY standalone, WIFI platform) in the configuration database and UVM takes care of the rest.

3.5.1 Modem standalone data path verification structure

3.5.1.1 TB configuration

In this verification mode, the MAC is not present; MAC_PHY_UVC is configured to have one active agent, PHY_AGENT, which is actively driving the modem DUT on its MAC-PHY-IF [2]. RADIO_UVC is present with two mandatory MODEM_STREAM_AGENT agents. One of the agents is active and drives the ADC samples into the modem DUT (Rx path), while the second one is passive and collects the samples sent from the modem DUT to the DAC (Tx path). An IP specific active agent can also be present in RADIO_UVC for driving the optional SPI/GPIO/APB interfaces.

The optional BEAMFORMING_UVC has one active agent, PHY_BF AGENT which is responsible for supplying beamforming data to the modem DUT in beamformer mode. In beamformee mode, the UVC only collects data from the modem (compressed beamforming report, see [2] for details) for use by MODEM_BF_SCB. This UVC is included only if the modem DUT is configured with beamforming support.

All modem-related scoreboards (MODEM_*_SCB) are present in the verification environment and are connected as shown in Figure 3-6. MAC related scoreboards (MAC_*_SCB) are not present.

RF communication is modelled using a MATLAB model. For use in the UVM verification environment, a standalone MATLAB executable library is built and is connected to the testbench via the MODEM_CFG class. All verification components interact with the MATLAB library using this class and 4 functions:

- set_gain() – used by RADIO_UVC to set new gain level for AGC
- execute() – run the MATLAB model
- get_golden_data() – used by RADIO_UVC to calculate new input data for Rx path verification
- get_ref_data(destination) – used by UVM component *destination* to calculate reference data

MODEM_CFG communicates with the MATLAB library using DPI calls. For data exchange, two files are used:

- payload.txt – contains PPDU frame payload
- SysParam.txt – contains RF communication parameters

3.5.1.2 Data flow

For Tx path verification with Radio Interface Unit (RIU) [2] present, the MAC_PHY_UVC generates a constrained random PPDU frame and passes it along to the modem DUT for processing. As a result of this processing, data samples appear on the NTx output of the modem DUT. RADIO_UVC collects these samples and sends them to MODEM_TX_SCB scoreboard. The scoreboard uses MODEM_CFG to call the MATLAB model to generate reference data and then compares this data to data received from RADIO_UVC.

For Tx path verification without RIU, the MAC_PHY_UVC generates a constrained random PPDU frame and sends it to the modem DUT. Scoreboard MODEM_STREAM_SCB uses two passive agents, MODEM_STREAM_AGENT and MODEM_MEM_AGENT to collect data at various locations inside the modem (e.g. memories used for frequency - time domain crossings with IFFT). Reference data is again calculated using the MATLAB model with MODEM_CFG.

For Rx path verification, RADIO_UVC calls function get_golden_data() from MODEM_CFG to generate data which is then sent to the ADC interface of the modem DUT. MODEM_CFG also receives the referent PPDU frame from the MATLAB model which corresponds to the data sent to the modem DUT. This referent PPDU frame is sent to MODEM_RX_SCB scoreboard. After the processing inside the modem DUT, a PPDU frame will appear on the MAC-PHY-IF which is collected by MAC_PHY_UVC and sent to MODEM_RX_SCB, where it is compared to the referent PPDU frame.

If the modem DUT has Beamforming support enabled, one additional UVC, BEAMFORMING_UVC and one scoreboard, BEAMFORMING_SCB are present in the verification environment. If the DUT is acting as a Beamformee, the scoreboard verifies the correctness of the compressed beamforming report upon reception of an NDP frame, sent by RADIO_UVC using data obtained from the MATLAB model. If the DUT is acting as a Beamformer, then the BEAMFORMING_UVC supplies the compressed beamforming report using PHY_BF_AGENT. The correctness of the transmission is verified using MODEM_TX_SCB.

Scoreboard MODEM_TIMING_SCB is in charge of timing checks for the modem DUT. It receives timestamped PPDU frames from MAC_PHY_UVC as well as timestamped data from RADIO_UVC. By analyzing frame type information from the PPDU frame, the scoreboard is able to verify if the modem DUT has finished transmission or reception of the packet on time or not.

3.5.2 MAC standalone data path verification structure

3.5.2.1 TB configuration

In this configuration, none of the MODEM_* components displayed in Figure 3-6 are present in the testbench. MAC_PHY_UVC has one active agent, MAC_AGENT, which is actively driving the MAC-PHY-IF. BEAMFORMING_UVC also has one active agent, MAC_BF_UVC which is actively driving the beamforming report input to MAC DUT. A more detailed display of the UVM verification environment for MAC DUT verification is shown in Figure 3-7. In this figure, the modem DUT is not present for standalone MAC verification.

An active agent, SRAM_AGENT is used to load the PPDU frame object to SRAM memory. Instead of AHB accesses which are relatively slow, in order to speed up the simulation this agent forces the PPDU frame object into the SRAM memory during MAC Tx path verification, or loads it directly from the SRAM memory for MAC Rx path verification.

AXI_SLAVE_AGENT is used for DMA access verification. It is an active agent as handshaking is required on AXI bus. The agent collect all the data received from platform DMA and send it to DMA_SCB scoreboard.

IPC_OUT_AGENT is a passive agent which collects data from the IPC DUT and sends it to the IPC_OUT_SCB scoreboard for verification.

AHB_MASTER_AGENT is an active agent that drives transactions on the AHB bus, which is mainly used for DUT configuration. Directly related to this agent is the UVM register model, displayed in Figure 3-7 as *REGModel+AHB adapter*. This model is created from the register specification using a script. The purpose of the model is to verify the correctness of RTL registers with the help of REG_SCB scoreboard. When a register write is detected on the AHB bus, the register model updates the value of the register being accessed. After this, when the same register is read the scoreboard compares the value stored in the register model to the value that was received on the AHB bus. If there is a difference, an error will be reported. The register model and the scoreboard can also be used to verify the reset value of registers.

MAC_CTRL_SCB is used for verification of 802.11 management frames (frames that originate from the MAC and do not carry payload data from higher layers).

IRQ related verification components are shown in Figure 3-8. IRQ_RAW_AGENT is a passive agent that collects IRQ signals from their DUT sources and sends them to all scoreboards that need to have this data. The interrupt controller inside the DUT is verified by IRQ_OUT_SCB scoreboard, which receives raw IRQ data from IRQ_RAW_AGENT and controller data from the passive agent IRQ_OUT_AGENT.

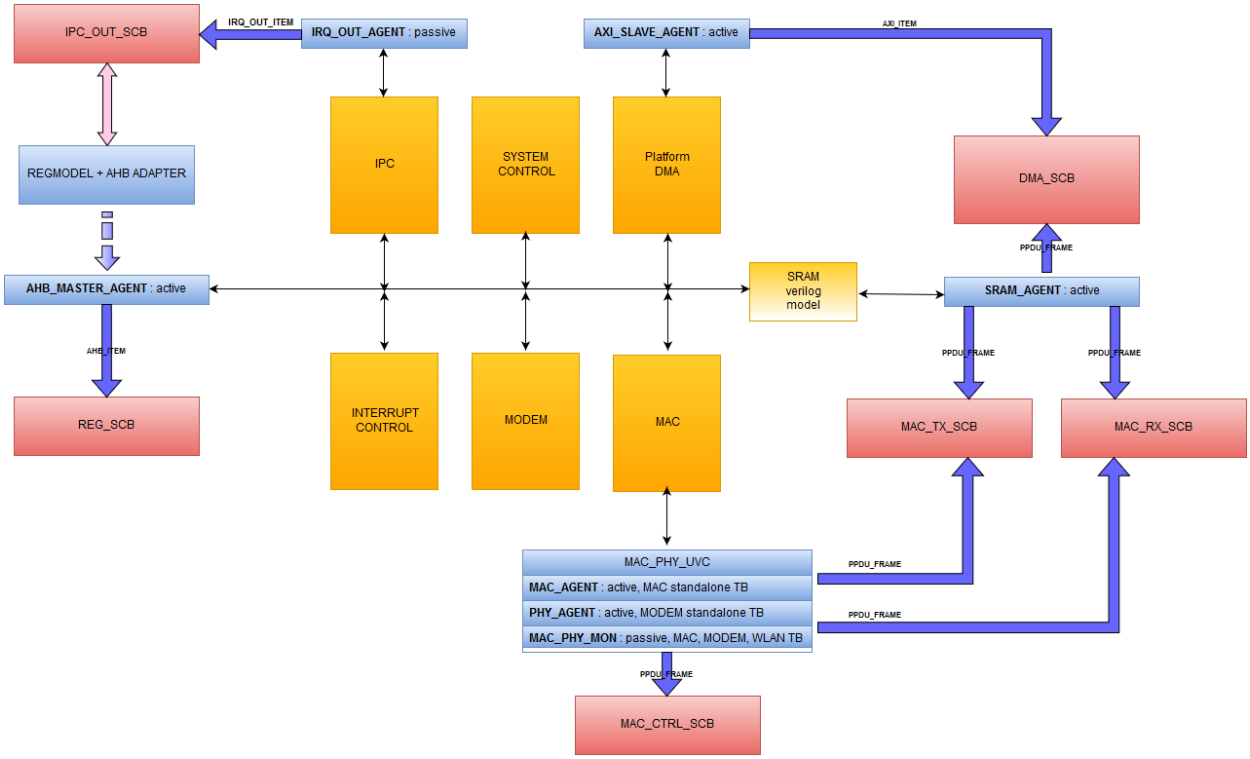

Figure 3-7 Detailed MAC verification environment

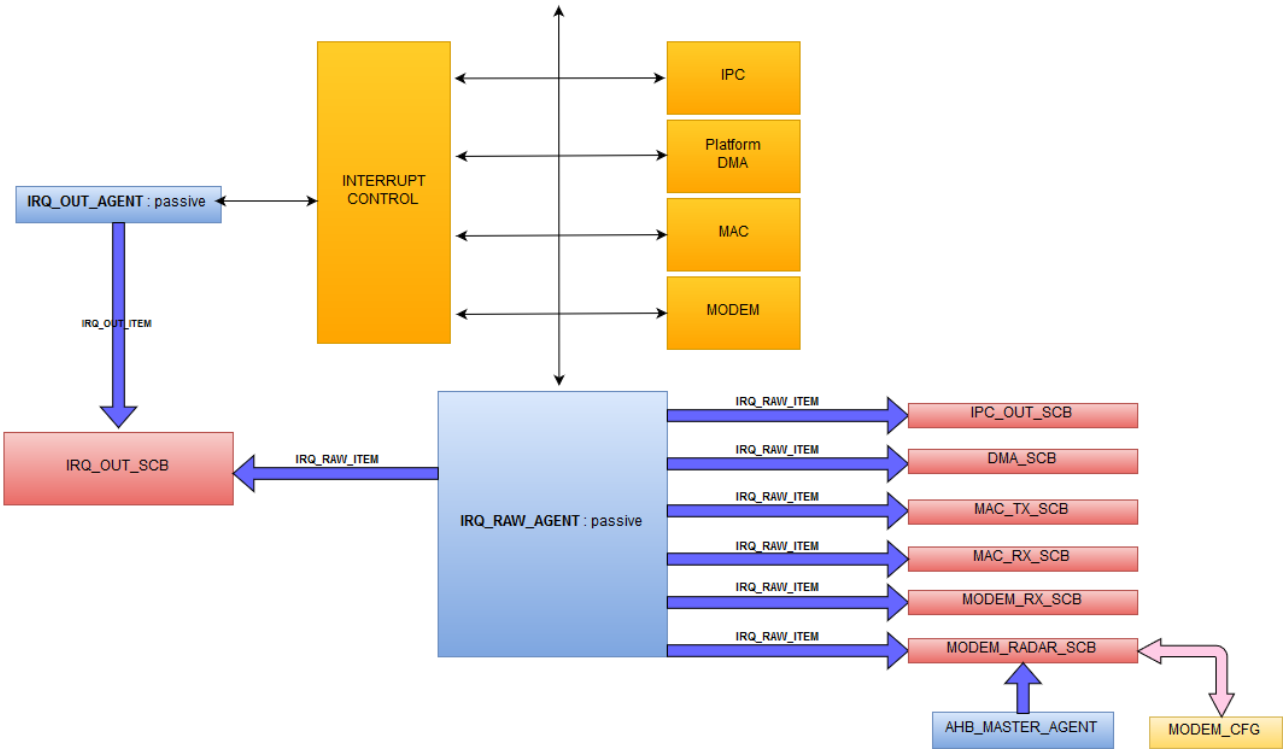

Figure 3-8 IRQ related agents and scoreboards

3.5.2.2 Data flow

For 802.11 data frame Tx path verification, the SRAM_AGENT (Figure 3-7) generates a constrained random PPDU frame object, loads it into the SRAM model and signalizes the MAC DUT to start processing it. The frame is also sent to MAC_TX_SCB for verification. The DUT processes the frame, and the resulting data appears on the MAC-PHY-IF interface, where it is collected by the MAC_PHY_UVC and sent to MAC_TX_SCB for verification.

For Rx path verification, it is the MAC_PHY_UVC which generates a constrained random frame and sends it to the MAC DUT using the MAC-PHY-IF. The same frame is sent to the MAC_RX_SCB for verification. The DUT processes the frame and writes the results to the SRAM memory. The SRAM_AGENT collects this data and sends it to MAC_RX_SCB for comparison and verification.

802.11 management frames are verified using MAC_CTRL_SCB, which receives collected PPDU frames from MAC_PHY_UVC.

For DMA data path verification, the MAC DUT is configured to transfer a random number of random data. AXI_SLAVE_AGENT collects data from the AXI bus and sends it to DMA_SCB, where it is compared to SRAM contents obtained by SRAM_AGENT.

IPC and IRQ are verified with their scoreboards using data collected by their passive agents. The UVM verification environment doesn't contain any active agents that seek to proactively trigger interrupt requests. The chosen approach was to collect IRQ and IPC data and distribute it to all scoreboards that participate in verification of DUT features that can cause interrupt requests to occur. It is down to the scoreboards to know when and IRQ is expected and to verify if it has occurred.

3.5.3 WIFI platform data path verification structure

3.5.3.1 TB configuration

In WIFI platform verification mode, all verification components shown in Figure 3-6, Figure 3-7 and Figure 3-8 are present. The only exception is for features that are optional. If the optional feature is not present in the DUT, then the appropriate verification components are not present either (e.g. beamforming, RIU).

MAC_PHY_UVC contains only the passive MAC_PHY_MON monitor. Likewise, BEAMFORMING_UVC contains only MAC_PHY_BF_MON. RADIO_UVC contains all agents as described in the standalone modem verification mode (page 22). Other agents shown in Figure 3-7 and Figure 3-8 are also present with their active/passive mode as described in the MAC standalone chapter (page 23).

3.5.3.2 Data flow

In WIFI platform verification mode, the interface between the MAC and modem DUTs is connected. No UVM verification component will generate random data on this interface.

For verification of the Tx data path, SRAM_AGENT generates a constrained random PPDU frame object and loads it into the SRAM, as in the standalone MAC verification mode and starts the MAC DUT. The MAC processes the data, loads the resulting frame to the MAC-PHY-IF and starts the PHY. The PHY processes the data and outputs its results to the DAC interface, where it is collected by the RADIO_UVC. All scoreboards that are used in the standalone modes are used here as well, so the data is checked during the entire path, as described in previous chapters for standalone modes of verification: on the MAC-PHY-IF by MAC_TX_SCB and on the radio interface by PHY_TX_SCB.

Similarly, the Rx path verification is started by RADIO_UVC generating constrained random data and starting the PHY DUT. The DUT processes the data, sets the MAC-PHY-IF and starts the MAC DUT. The MAC processes the frame and stores the result in the SRAM model. Active scoreboards for this data path are MODEM_RX_SCB and MAC_RX_SCB.

4 Verification strategy and overview

The purpose of this chapter is to give an overview of the RW-WLAN-nX IP verification scope and strategy.

4.1 Verification scope

4.1.1 Objective

The objective of the RW-WLAN-nX IP verification effort is

1. To achieve 100% RTL code coverage
2. To achieve 100% functional coverage
3. To achieve 100% assertion coverage

Verification of the entire RW-WLAN-nX IP will be separated in several distinctive phases:

1. Development of all verification components and documentation
2. Development of testbench environment and documentation
3. Functional modem standalone verification environment and documentation without implementation of some features (e.g. beamforming, MUMIMO support, RF error injection). Code coverage should be above 80%, interface and register coverage at 100%
4. Functional MAC standalone verification environment and documentation without implementation of some features (e.g. MUMIMO support). Code coverage should be above 80%, interface (AXI, AHB slave, DMA, interrupt controller, IPC and system controller) and register coverage at 100%
5. Functional WIFI platform (MAC with the modem) verification environment without features and code coverage as specified in points 3 and 4.
6. Implementation of missing features (MUMIMO support, beamforming and RF error injection). Code coverage above 95%, full functional coverage.

4.1.2 Device under test

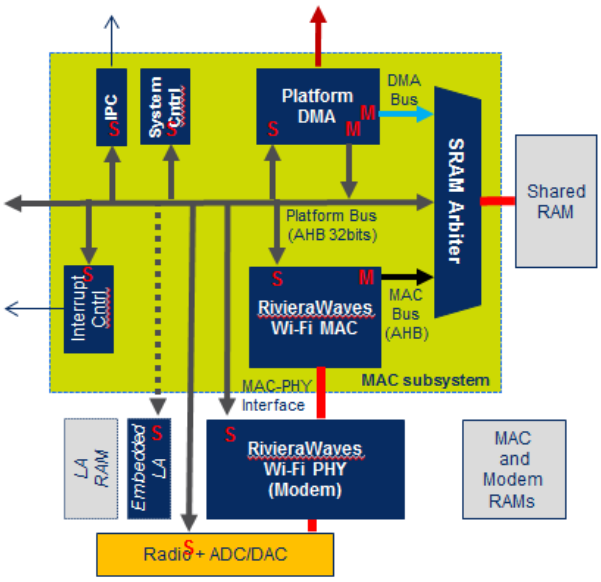

Figure 4-1 RW-WLAN-nX DUT

The Device under Test (DUT) is shown in Figure 4-1. It comprises of the MAC subsystem and the WiFi PHY. These IP cores form a comprehensive Wi-Fi IP family, compliant with 802.11a/b/g/n/ac/ax standard, tailored for various applications including IoT, wearable, mobile and gateway.

The MAC subsystem is a hardware accelerator core that is to be connected to a processor bus (AHB). It contains embedded on the fly encryption engine (WEP, TKIP, CCMP and WAPI) and supports up to 72 Mb/s (20MHz) or 150Mbps (40MHz) data rates. Along with the RivieraWaves's MAC firmware it implements the complete 802.11 MAC specification.

The RW-WLAN-nX-Modem implements the PHY level of the 802.11nac standard including beamforming support. It has support for 20MHz, 40MHz and 80MHz channel bandwidths, up to 2 transmit and 2 receive paths and up to 2 spatial streams.

4.2 Verification strategy

RW-WLAN-nX verification is performed using SystemVerilog language and Universal Verification Methodology (UVM), with coverage driven constraint random verification approach. Functional and RTL code coverage metrics are used to measure the completeness and quality of the verification. Collected RTL code coverage metrics are:

- **Statement coverage**; also known as line coverage identifies which lines in the RTL source code have been executed. It considers only executable statements, i.e. the *module-endmodule* Verilog statements are ignored, so are comments, timescale directives, define etc.
- **Branch coverage**; also known as decision coverage identifies if both true and false branches of conditional statements (e.g. *if-else*) have been evaluated during verification.
- **Conditional coverage**; also known as expression coverage identifies how the variables or sub-expressions in conditional statements are evaluated (only logical operators are considered). For example, if the logical expression is *(a&b),* there are 2 possible values for *a* and 2 for *b*, giving 2*2=4 possible logical conditions. Branch coverage only checks if the entire expression evaluates to both 0 and 1 during verification, but it doesn't check if all values for *a* and *b* have been evaluated. That is what conditional coverage does
- **FSM coverage**; identifies which states of an FSM has been reached and if all transitions between all states are covered

Functional coverage goals are defined and implemented in the verification environment as coverage groups and assertion coverage.

4.2.1 Verification IP reuse

The current verification environment for RW-WLAN-nX is written in Systemverilog with VMM verification methodology, therefore none of the existing verification components will be reused in the UVM testbench. All verification components used in the UVM testbench will be developed from the ground up.

One part of the existing verification environment is developed in MATLAB and it's used to model RF channel and PHY data processing in both time and frequency domain. This model will be reused in the UVM verification environment.

4.2.2 DUT and verification environment connections

For the MAC, a black-box verification approach has been chosen. This means that the verification IP (VIP) is connected to MAC IPs outside interfaces. The advantage of this approach is that the VIP depends only on the functional specification of the MAC IP and not on the actual implementation of the RTL (e.g. there is no need to update the VIP if a signal is renamed inside the RTL).

A different approach had to be chosen for the Modem IP, as it will have to be verified using internal signals and FIFO/memory contents (e.g. to verify crossings between time and frequency domain using the MATLAB model). This is known as a white-box approach.

For the verification of the entire WIFI platform, a mix of these approaches is needed – this is known as a grey-box approach. MAC can be interfaced to using its outside interface, but for the modem we still need access to its internals.

4.2.3 Verification environment customization

High customization possibility of the verification environment is achieved by using the UVM configuration database.

For customization of the testbench architecture, each agent's configuration object contains "is_active" fields which configures if the agent is active or passive. The UVC configuration object contains "has_*_agent", "has_scb" and "has_monitor" fields so that these components can be excluded from the testbench if necessary.

For coverage collecting customization, each agent's and UVC's configuration object contains "has_coverage" fields so the entire coverage model can be turned on or off. Each configuration objects also contains "has_*_cvg" fields so that individual covergroups can be turned on or off.

For checking customization, each agent's and UVC's configuration object contains "has_checks" fields so the entire checker model can be turned on or off. Each configuration object also contains "has_*_chk" fields so that individual checkers can be turned on or off.

Interfaces also contain references to configuration objects so that SystemVerilog Assertions (SVA) can be easily turned on or off.

For message logging purposes (reporting information, warnings and errors) only UVM report macros *`uvm_info*, *`uvm_warning*, *`uvm_error* and *`uvm_fatal* are used. This approach also increases the portability of the VIP as these UVM macros are supported by all flavours of mainstream EDA tools, so there is no need to adapt the VIP to use simulator-specific (built-in) reporting functions. Each message contains an ID obtained by UVM *get_type_name()* method. This ensures high readability of the presented information as the source of the message is easily identifiable. Message verbosity is also handled by UVM, as the *`uvm_info* macro can be set to use the following verbosity levels: *UVM_DEBUG, UVM_FULL, UVM_HIGH, UVM_MEDIUM, UVM_LOW* or *UVM_NONE*. As the verbosity level can be changed from the command line with a command line switch (+UVM_VERBOSITY=), we can control the number of displayed messages during simulation without recompiling the code. By controlling UVM verbosity levels for all UVM messages, regression logs are kept with minimal content.

4.3 Verification risk

An identified verification risk is the interface between the UVM verification environment and the MATLAB model. While the model has been in development and use for a while and can be considered as stable, the model is called using DPI calls. The SystemVerilog Direct Programming Interface (DPI) is basically an interface between SystemVerilog and a foreign programming language, in particular the C language. The implementation of DPI call mechanisms is down to the EDA tool vendor, and in the past DPI calls have turned out to be source of simulation crashes (internal tool errors).

5 WLAN testbench architecture – detailed

In this chapter verification components will be explained in detail, there purpose, organization, configuration, sequence library if component is active, connection with scoreboards, assertions if present. Verification components are divided into separate logical groups:

- agents and UVCs - explain agents and UVCs which are composed of agents
- common components that are reused all over the verification environment and verification components
- checking and validation components – scoreboards

5.1 Verification components – agents and UVCs

This chapter will give detailed description and overview of all agents and UVCs used in verification environment. Sequences from active agents will not be described in detail.

5.1.1 AHB master agent

This agent implements AHB bus master functionality and it is used for register access through AHB interface. Besides the standard UVC components, it also contains a register adapter, which is used as a bypass between the register model and sequencer inside the UVC. This UVM RAL mechanism makes sequence running easier and more SW like.

It can be also used for LDPC, AGC, BFMER, and SRAM direct memory access (frontdoor access). This way memory access like CPU/HOST IP are accessing can tested.

5.1.2 AXI4 Lite slave agent

This agent is reactive slave component on AXI4 Lite interface. AXI4 Lite slave agent is implemented as memory model which communicates with the DMA engine, through AXI4 lite interface and is responsible to store data in memory model in case of write transaction or to read from memory model and provide data on outputs in case of read transaction. AXI memory model is developed as associative array which is stored in UVM configDB so it can be accessed from verification components (sequences, scoreboards).

Supported data transfers that are implemented in sequences:

- Upstream DMA transfer –data transfer from Shared RAM to HOST memory (AXI memory model)
- Downstream DMA transfer – data transfer from HOST memory (AXI memory model) to Shared RAM
- Midstream DMA transfer – data transfer from Shared RAM to platform

Communication with DMA is done over Link List Item (LLI) which is a descriptor what is source address, what it destination address, what is the length of transfer and enables fields for interrupt generation. LLI node is shown in Table 5-1 LLI structure.

Address offset	Range	Name field	Description
0	31-0	SADDR	Source address of the data fragment. It is a host address if the node belongs to a list attached to the downstream channel or an embedded address is the node belongs to a list attached to a upstream channel.
4	31-0	DADDR	Destination address of the data fragment. It is a host address if the node belongs to a list attached to the downstream channel or a host address is the node belongs to a list attached to a downstream channel.
8	31-29	-	Reserved
	28	IRQ_EN	If the bit IRQ_EN is set, the LLI interrupt corresponding to field IRQ_NO ias asserted once the fragment processed.
	27-24	IRQ_NO	
	23-21	-	Reserved
	20	COUNTER_EN	If the bit COUNTER_EN is set, the DMA counter pointed by the COUNTER_NO field is incremented once the fragment processed.
	19-16	COUNTER_NO	
	15-0	LENGTH	The byte length of the fragment to be transferred. Valid range values starts from 1 byte to 65535 bytes.

Table 5-1 LLI structure

5.1.3 Beamforming UVC

WLAN platform can have configuration that supports beamforming feature (transmit of beamformed frames and reception of beamformee frames). The support of beamformee feature is mainly managed by the Core. The MAC SW shall indicate in the association request the supports of beamformee. Once done, the SW is not involded anymore.The Core supports the two beamforming calibration procedures (SU and MU) defined in the 802.11ac-2013 based on NPDA and NPD. Regarding the reception of beamformed frame, the MACCore does not perform specific task. It reports as is the beamformed bit of the rxVector into the RHD.

In order to verify this feature Beamforming UVC is developed. This UVC has two agents, MAC BF and PHY BF agents. When standalone MAC is tested then PHY FB agent is active, when standalone PHY is tested then MAC BF agent is active. In case of WLAN platform testing this UVC is fully configured to be PASSIVE and it only collects data on interfaces and provides items to scoreboards.

In chapters 5.1.3.1 and 5.1.3.2 both agents are described in detail.

5.1.3.1 MAC BF agent

This agent is configured to be ACTIVE when standalone PHY beamforming feature is enabled. Responsibility of agent is to mimic behavior of MAC core in case when beamforming report needs to be created. MAC BF driver has execution of these commands/sequences:

- First configure beamforming block with parameters received in NDPA/NDP frames
- Trigger start of creating of beamforming report by PHY
- Read out beamforming report from BFMER memory

MAC BF monitor will collect configuration data and beamforming data and send it to scoreboard for checking.

5.1.3.2 PHY BF agent

This agent is configured to be ACTIVE when standalone MAC beamforming feature is enabled. Responsibility of agent is to mimic behavior of PHY core in case when beamforming report needs to be created. PHY BF driver has execution of these commands/sequences:

- When MAC initiates start of creation of beamforming report, driver will start providing random data which MAC core will read out
- Size of beamforming report will be calculated in relative to configuration interface signals, provided by MAC core

PHY BF monitor will collect configuration data and beamforming data and send it to scoreboard for checking.

5.1.4 Bus monitor

This is a PASSIVE component, which is used for collecting information from interfaces or internal signals in RTL, which does not require a creation of a new UVC. Bus monitor can be used for writing SVA to check some properties which are not logically part of any existing interface. With bus monitor signals get probed and provided to checking components for analysis.

5.1.5 COEX BT agent

The COEX BT agent is used together with the PTA WLAN agent. In case, Bluetooth coexistence is supported, using this agent the following parameters are set on the coexistence interface:

- bt_tx – Bluetooth transmission is ongoing
- bt_rx – Bluetooth reception is ongoing
- bt_event –Bluetooth event ongoing
- bt_tx_abort – Bluetooth transmission abort request
- bt_rx_abort – Bluetooth reception abort request
- bt_pti – Bluetooth Packet Traffic Information
- bt_channel – Used Bluetooth channel
- bt_bw – Used bandwidth

Based on these parameters, and the parameters that are written in the COEXCONTROLREG register, the PTA module does arbitration. Depending on the priority of the WLAN and BT, transmission or reception can be aborted, for WLAN or BT.

5.1.6 IRQ RAW agent

This component doesn't have an ACTIVE side, only the monitoring of the IRQ RAW line is done. It detects any change on the interrupts, and transmits this information to the scoreboard. Using this agent, the testbench is aware of some of the errors that are detected inside the hardware. For example if there is an overflow or underflow on the MAC-PHY interface, an error is reported by the MAC_IRQ_SCOREBOARD.

5.1.7 MAC-PHY UVC

MAC-PHY UVC consists of two agents, MAC and PHY agent. Depending on the current configuration of the testbench these agents are configured in the following way:

- MODEM standalone testbench – The MAC agent is used, which mimics the behavior of the MAC responding to the PHY, while monitoring the MAC-PHY interface
- MAC standalone testbench – The PHY agent is mimicking, the behavior of the PHY responding to the MAC requests, while monitoring the MAC-PHY interface
- WLAN platform tesbench (MAC + Modem) – In this case, only the monitoring is active, so both components are configured as PASSIVE.

Both of the agents are using the same MAC-PHY monitor.

5.1.7.1 MAC agent

Figure 5-1 shows the tasks that are called during the transmission of a frame.

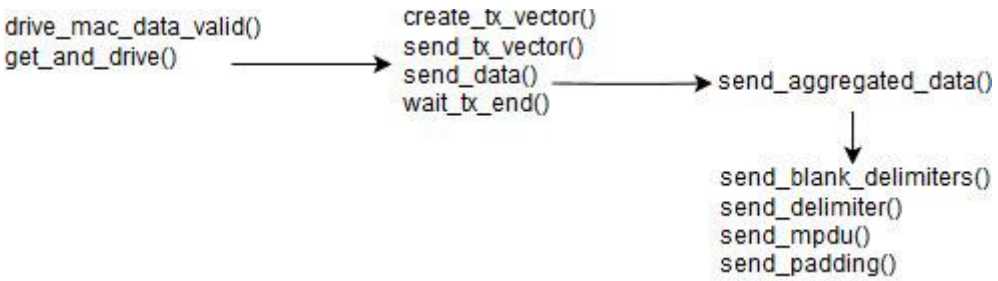

Figure 5-1 Task flow in MAC agent

The *drive_mac_data_valid()* generates the mac_data_valid signal, in the same way, as the PHY generates the phy_rdy, only it is delayed by two clock cycles.

The main task is the get_and_drive(). The item that is received by the driver contains a PPDU frame, which is described in chapter 3.4, the transaction type (RX, TX). Based on these parameters and the description in [5], TX VECTOR is constructed, and sent to the MAC-PHY interface.

After the TX VECTOR is sent, the next item is the actual data, where, if there is any, blank delimiters are forwarded to the interface, before the delimiter.

On every mac_data_valid, one byte of the MPDU is sent, and at the end of the frame, PAD bytes are added if needed. The final task is waiting for the TX_END appearance, before the whole process starts again.

5.1.7.2 PHY agent

Figure 5-2 shows the task flow in the PHY driver. The task flow is divided into the reception side and transmission side.

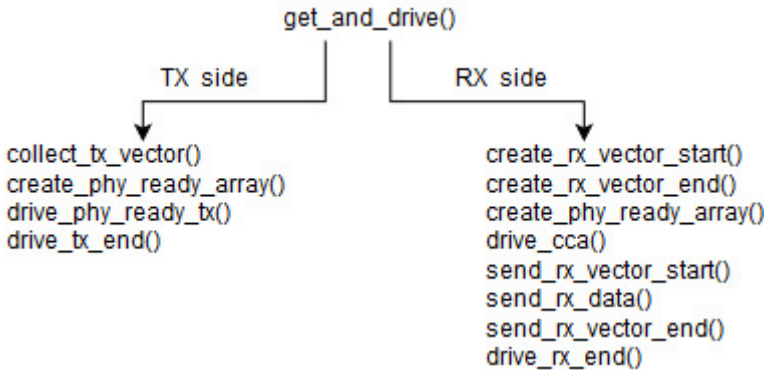

Figure 5-2 PHY agent task flow

During a transmission of a frame, the TX VECTOR is needed to be collected from the MAC-PHY interface in order for the driver, to know how many of phy_rdy should be generated. At the end of the frame, TX END line of the interface is asserted for one clock cycle.

When the reception side of the flow is active, the PPDU frame from the sequence item is used as a base to construct the algorithm, which is used for driving the interface. In [5] is a detailed description on how the RX VECTOR and RX VECTOR END should be formed.

The drive_cca is responsible, for asserting the CCA based on the channel bandwidth that is used for the current frame, and deasserting it at the end of the frame.

Based on the RX VECTOR, a separate array is prepared with values, which will be driven to interface on phy_rdy signal, together with the PPDU frame. At the end of the PPDU frame, RX VECTOR END is sent. The last step is to assert rx_end, indicating that the reception is done.

5.1.8 Memory agent

Memory agent is a generic agent used for driving and monitoring memory interface signals. This agent is used for PHY memory observation, memories with same interface signals. It is configured as generic so data width is set by agent parameter.

In case of standalone PHY testing this agent can be used as ACTIVE to force and drive memory interface signals, in order to test a specific path of processing like time domain, frequency domain and bit domain. When agent is active it can be configured to be read or write agent since some memories need to be written or read, depending on a testing path.

Interfaces that use this agent in PHY are FFT input and output memory, H memory.

5.1.9 Modem stream agent

For purpose of verification of PHY modem stream agent is developed. This is serial interface agent used for driving ADC inputs and collecting data on DAC outputs of modem. This is parametrized agent, parameter that can be set in data width.

Modem stream agent can be configured to be pure PASSIVE stream agent used for collecting stream data from processing blocks (both on Rx and Tx path). Stream interface signals are consistent through the design so this agent is reusable whenever monitoring is needed. Agent provides collected item over analysis port to any checking component.

5.1.10 PTA WLAN agent

PTA WLAN agent is a simple agent, which collects data from the PTA interface, so it is configured as a PASSIVE component. As it was mentioned in 5.1.5, this agent is used together with the COEX BT agent, if the coexistence is enabled in the current configuration of the DUT. Opposite to COEX BT agent, the following signals are collected:

- wlan_tx – WLAN transmission ongoing

- wlan_rx – WLAN reception ongoing
- wlan_tx_abort – WLAN transmission aborted
- wlan_rx_abort – WLAN reception aborted
- wlan_pti – WLAN packet priority
- wlan_chan_freq – WLAN primary channel frequency
- wlan_chan_bw – WLAN bandwidth
- wlan_chan_offset – WLAN secondary channel offset

5.1.11 Radio UVC

Modem core has Radio Interface Unit (RUI) block which communicates with analog RF unit which is not part of WLAN system. In order to verify this part of modem core Radio UVC is developed. Main features for this UVC is to drive ADC inputs of RUI, monitor DAC outputs, setup Radio controller block and monitor AGC block. Agents that are part of this UVC:

- Radio Control agent (5.1.11.1)
- RUI agent (5.1.11.2)
- Modem stream agent – ADC interface
- Modem stream agent – DAC interface

5.1.11.1 Radio Control agent

Modem core has Radio controller unit inside which is responsible to communicate with RF transceiver. Radio Control agent is developed to communicate with this block over multiple interfaces, since RADIO block can be changed from setup to setup.

Radio Control agent can behave as SPI master or slave, monitor GPIO pins, perform RF reset.

5.1.11.2 RUI agent

RUI agent is used in test cases which are configured to perform in AGC bypass mode. In this case RUI module is bypassed in Modem core so RUI agent needs to drive RX data on input of Modem Core (OFDM or DSSS modem). There are test cases for Modem standalone where only Time or Frequency domain blocks are tested. Because of this RUI agent in those configurations needs to force frame parameters which are inputs signals to Time and Frequency domain blocks. In tests with AGC on, RUI agent is not present.

5.1.12 RST agent

Reset agent is always configured to be ACTIVE since it drives only one signal which is main system reset. Reset sequence is called in the beginning of every test sequence.

Other feature in this agent is sequence that drives idle cycles in µs time units and it is used in test sequence when some delay needs to be introduced.

5.1.13 SRAM agent

Shared RAM is memory where SW stores PPDU frame prepared for transmit and it is used for storing received PPDU frame from PHY. Shared RAM has its interface bus for access but in order to accelerate storing and reading frames from memory SRAM agent is developed. Interface of SRAM is accessed via backdoor, so interface signals are not driven. Data is stored directly memory. With this approach storing and reading PPDU frame is much faster.

SRAM driver is always ACTIVE in MAC standalone environment and in WLAN platform environment. SRAM monitor does nothing since there are no SRAM bus transactions going on. Monitor is just a placeholder. SRAM driver implements commands:

1. WRITE_TX_FRAME command stores PPDU frame in memory, it created linked list of THDs and TBDs like SW would do
2. READ_RX_FRAME command reads PPDU from memory, reading RHDs and RPDs like SW would do
3. PREPARE_RD_BUFF command prepares RHD and RPD buffers for reception
4. RD_TX_STATUS command reads THD status information fields
5. GET_NEXT_RHD_PTR command reads address of next RHD pointers

6. WRITE_MEM command writes data at given address – backdoor
7. READ_MEM command reads data from given address - backdoor

5.1.14 SRAM bus agent

SRAM bus agent is PASSIVE agent used to monitor and collect items on SRAM bus interface when it is accessed frontdoor.

5.2 Common components

Common components are used and shared over verification environment, between different components in order to be consistent and reusable as much as possible. This library contains necessary components and data models like frame model which is main data object in TB, SRAM descriptors which are SW like linked lists, register model, MATLAB data model used for interaction with MATLAB shared object library, key storage RAM representing memory model of key RAM from MAC core.

5.2.1 MAC descriptors

In MAC core both transmit and receive paths go through shared RAM memory as some sort of buffer between SW running on host IP and MAC HW. PPDU frame object can be interpreted also from DMA and SRAM side. Way how SW stores Tx frame in SRAM is through descriptors and Rx frame is stored in SRAM by MAC IP also with descriptors. This topic is described in detail [1] document.

PPDU frame object needs to be stored in structure shown in **Error! Reference source not found.**. This structure is s uitable for SRAM agent to use it when Tx frame needs to be stored inside SRAM (via backdoor access). Structure hierarchy looks like this: MU-MIMO PPDU frame for Tx SRAM is base object with up to 4 aggregated PPDU frames for Tx SRAM inside. Aggregated PPDU frame has 3 fields A-THD (Aggregated Transmit Header Descriptor) that contains all relevant information about Tx frame, PT (Policy table) which contains information about PHY device and singleton PPDU frame array which contains information about one MPDU frame that is being sent by MAC IP.

Singleton PPDU frame Tx SRAM structure is composed of fields:

- THD (Transmit Header Descriptor) – structure that has information about how MPDU data is stored in memory
- PT (Policy table)
- THD DATA – Tx descriptor can have pointers to data stored in shared memory, this array contains that data
- TBD (Transmit Buffer Descriptor) – link list of Tx data spread throughout shared memory
- TBD DATA – array of data on which TBDs point

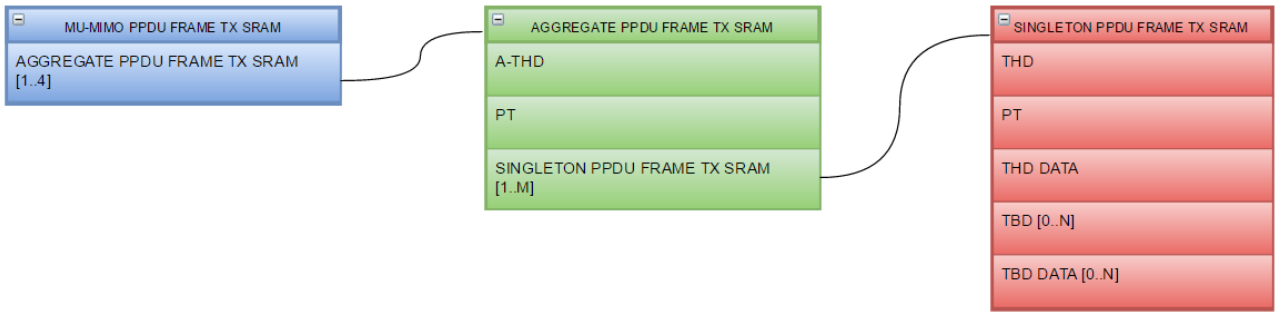

Figure 5-3 Structure for storing transmit PPDU frame in SRAM

Regarding Rx side, when MAC IP stores received frame to shared memory, there is also a structure that is obtained during storing of received frame. This structure is called PPDU frame Rx SRAM and it's shown in **Error! Reference s ource not found.**. SW prepares this structure in memory like circular buffer linked list of RHDs (Receive Header Descriptor) and RPDs (Receive Payload Descriptor). When MAC IP receives PPDU frame it stores it in SRAM by filling these 2 linked lists. SRAM agent can read via backdoor, the received frame and fill structure (PPDU frame Rx SRAM).

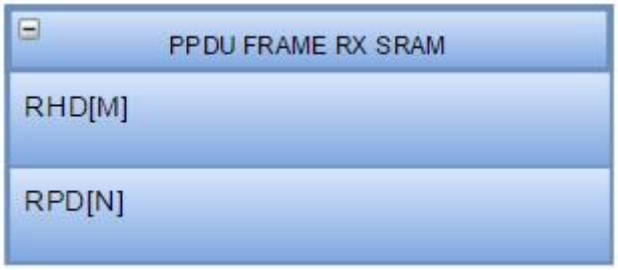

Figure 5-4 Structure for storing received PPDU frame in SRAM

5.2.1.1 Memory organization for Tx frame

SRAM agent is also responsible for memory management and organization. When Tx frame needs to be stored in SRAM there is an algorithm that needs to be followed during that process. One approach is that content is stored continually in memory, because this is the easiest and fastest way. No additional logic and arbitration is needed when memory content is stored in that way. In **Error! Reference source not found.** is shown how this memory is organized.

Figure 5-5 Transmit PPDU frame organized in memory

5.2.2 Key storage RAM

One thing that needs to be created before any test case is run is keyStorageRAM. The per-MAC address and group address encryption keys are stored in an embedded HW RAM, called the KeyStorageRAM. The address locations 0 - 23 of the KeyStorageRAM are reserved for storing 4 default key IDs for 6 Virtual LANs in sequence. The remaining locations of the RAM holds address mapped (pairwise) encryption keys. The structure of the KeyStorageRAM for 64 locations is illustrated in Table 5-2 Key storage RAM fields. For a KeyStorageRAM having 64 addresses, up to 40 Device addresses and corresponding keys can be stored. Note that all these numbers (VLAN number, size of KeyStorageRAM,

number of devices) are dependent of RTL configuration and can change from configuration type, but KeyStorageRAM object is also configured in relative to RTL so no additional workaround is needed in this case.

KeyStorageRAM is implemented as a shared object in TB, randomized at the beginning of scenario to contain at least one station that supports encryption type for each PPDU frame type. During PPDU randomization, station (ID, MAC address, encryption type and encryption key) will be chosen according to encryption type supported by generated PPDU frame type. KeyStorageRAM is used in MAC standalone and full system testing.

keyIndex RAM	cType RAM [2:0]	vlanID RAM[3:0]	sppRAM[1:0]	useDef Key RAM[0]	cLen RAM [0]	macAddrRAM [47:0]	encrKeyRam [127:0]	intWPIKeyRam [14] [127:0]
0		Don't care				Don't care	Default Key ID 0 for VLAN 0	
1		Don't care				Don't care	Default Key ID 1 for VLAN 0	
2		Don't care				Don't care	Default Key ID 2 for VLAN 0	
3		Don't care				Don't care	Default Key ID 3 for VLAN 0	
4		Don't care				Don't care	Default Key ID 0 for VLAN 1	
5		Don't care				Don't care	Default Key ID 1 for VLAN 1	
. . .		Don't care				Don't care		
22		Don't care				Don't care	Default Key ID 2 for VLAN 5	
23		Don't care				Don't care	Default Key ID 3 for VLAN 5	
24						Device 1 MAC Address	Device 1 Secret Key	Device 1 WPI Integrity Key
25						Device 2 MAC Address	Device 2 Secret Key	Device 2 WPI Integrity Key
26						Device 3 MAC Address	Device 3 Secret Key	Device 3 WPI Integrity Key
. . .						. . .	. . .	. . .
62						Device 39 MAC Address	Device 39 Secret Key	Device 39 WPI Integrity Key
63						Device 40 MAC Address	Device 40 Secret Key	Device 40 WPI Integrity Key

Table 5-2 Key storage RAM fields

5.2.3 Modem Data Model – MDM

For purpose of verifying Modem (PHY) in TB, MATLAB model is used. MATLAB provides easier way of calculating and modeling data path inside Modem core (ACG, FFT, time domain block, frequency domain block, bit domain block). MATLAB is suitable for all mathematical calculations which will be used as either stimuli samples or as referent data when checking processing blocks. All checkpoints and referent data for testing Modem is provided by MATLAB library. MATLAB provides shared object which is used in UVM environment (MATLAB libdump).

For communication with MATLAB shared object Modem Data Model is used (MDM). This component is wrapper and communication buffer between test sequences and MATLAB calls. Responsibility of this component is to fetch PPDU frame, golden frame, generated in test sequence and to execute call of MATLAB function which will simulate provided PPDU frame and give dump samples for stimuli, referent data and all intermediate checkpoints needed. MDM in order for MATLAB to know what to execute, needs to prepare input files for MATLAB. Those files are divided in two logical groups, one file contains all parameters regarding frame preamble and header settings (RX/TX vector parameters following WiFi standard definitions) and second file which contains PPDU payload bytes. Input file organization and parameter naming is different in two versions of MDM component so transition from one version to other is not 1 on 1. When input files are prepared MDM triggers, via DPI calls, MATLAB execution of test case. MATLAB as output will provide golden reference data, stimuli samples, intermediate checkpoint data and referent register values. After output data is generated MDM provides wrapper functions which are used in test sequence and scoreboards to get stimuli or referent data samples in order to drive or check data. In Figure 5-6 use case of MATLAB call is shown.

Mechanism for communication between MDM and MATLAB shared object is done over DPI (Direct Programming Interface) is an interface which can be used to interface SystemVerilog with foreign languages, in our case C code.

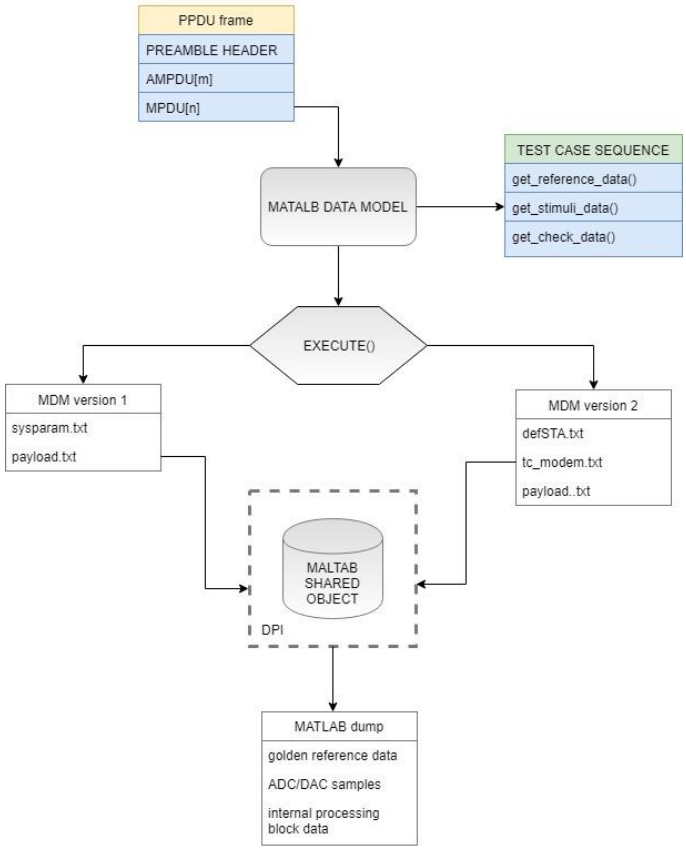

Figure 5-6 Use case for PPDU frame execution

List of files used for MDM (dependencies):

- ***mdm_ref_func.c*** - contains function calls of MATLAB shared object
- ***mdm_import_func.sv*** - contains DPI function import prototypes
- ***mdm_data_model.sv*** - contains legacy code of MDM with functions and data structures (supports up to AC standard).
- ***mdm_data_model_ver2.sv*** - contains version 2 of MDM code with functions and data structures that support AX standard, this component derives legacy MDM version 1 and upgrades it

5.2.4 Register model

Register model is an UVM register abstraction layer (RAL) of HW registers contained in WLAN platform. Register model wraps all register blocks into one component which is shared in UVM environment (through config DB) so all components can get a handle to it (sequences, scoreboards).

Registers and register fields are described in XLS files, which are provided with RTL release (inside Documentation directory). From these XLS files scripts can generate register model. Advantage of register model is that access of registers/fields is more SW like, read/write transactions are provided over set/get tasks developed in base sequence. How to generate register model will be explained in detail in **Error! Reference source not found.**.

5.3 Checking and validation components – scoreboards

Checking and validation of WLAN platform is mainly divided into receive and transmit paths. Both paths are tested in all 3 configurations of testbench, Modem and MAC standalone and WLAN platform (Modem + MAC). This is the list of scoreboards and description:

- MAC Rx scoreboard: checks does MAC core behave correct during reception of PPDU frame. MAC core needs to receive frame and store in SRAM (into pre-prepared RHD/RPD linked list). Checks performed:
  - Frame will be read out from SRAM and compared with PPDU frame collected on MAC-PHY interface.
  - Rx vector will be checked and frame payload.
  - Status flags from RHD will be checked.
  - Value of TSF register will be checked in case when BEACON/PROBE_RESPONSE frame is received.
  - Collected frame from MAC-PHY interface will be decrypted before checking and with this MAC core decryption engine will be checked.
  - Rx events are checked in register
  - Inside scoreboard NAV counter mimic task is executed
- MAC Tx scoreboard: checks does MAC core transmit correctly PPDU frame stored in SRAM. Frame is collected on MAC-PHY interface. Checks performed:
  - NAV protection check is it performed like it was set in policy table (self CTS, RTS-CTS, STBC or no protection). MPDU frame type is checked and preamble header parameters.
  - Checking is timestamp value in PPDU frame set to value that is in TSF register.
  - Checking of frame duration in transmitted frame and in protection frames that are potentially sent.
  - Checking of encryption engine in MAC core, did MAC encrypt frame correctly.
  - Checking if prepared PPDU frame is identical Tx vector and payload with one collected on MAC-PHY interface.
  - Checking of status bits in THDs.
- MAC immediate response scoreboard: MAC core has a possibility to immediate respond to received frame if requested by standard. CTS-RTS, ACK, BFR_REPORT. Checking performed:
  - Respond to singleton frame received relative to MPDU subtype (Tx vector, frame duration, response delay)
  - Respond to aggregated frame received (Tx vector, frame duration, response delay, BLOCK ACK bitmap)
  - Respond to NDP frame received with BEAMFORMING REPORT which is checked
- MAC IRQ scoreboard: scoreboard will check if unmasked event occurred and is set in register field
- MAC PTA scoreboard: This scoreboard is used for checking the coexistence controller between WLAN and Bluetooth. Based on the configuration only one of the protocols can be active at a time.
- Platform IRQ scoreboard: check if platform interrupt occurred if unmasked.
- MAC DMA scoreboard: scoreboard will check read and write transactions on AXI bus and SRAM bus.
- Modem Rx scoreboard: checks modem core does it receive PPDU frame from air and it provides received frame on MAC-PHY interface. Referent data is provided by MATLAB libdump. Checks performed:
  - PPDU frame driven on ADC inputs (from test) with PPDU frame on MAC-PHY interface.
  - Time domain block output check.
  - Frequency domain block output check.
  - FFT output check.
  - Compressed and uncompressed soft bits check.

    - Checking of status registers.
- Modem Tx scoreboard: checks modem core transmit of PPDU frame driven on MAC-PHY interface and transmitted on DAC outputs. Referent data is provided by MATLAB libdump. Checking performed:
    - Check DAC samples.
    - Check TXCORE data.
    - Check TX mapper data.
    - Check L/HT/VHT/HESIG data.
- Modem beamforming scoreboard: if beamforming is enabled in modem core, this scoreboard will check did modem properly create beamforming report. MATLAB provides referent beamforming report expected to be created. Checking done:
    - Check beamforming report.
    - H-memory data.
- Modem stream scoreboard: checks byte stream from MAC-PHY interface which includes Rx vector and payload data. MATLAB provides referent data.

6 Test cases and sequences

6.1 MAC standalone test cases

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_ampdu_frame	In this testcase the following AGGREGATED frames are received by the MAC: HT_GF, HT_MF, VHT, HE_SU. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response CONTROL frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-1 test_mac_rx_ampdu_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_ **\basic_trigger** **\beamforming_report_poll_trigger** **\bqrp_trigger** **\bsrp_trigger** **\mu_bar_trigger** **\mu_rts_trigger**	In these testcases a SINGLETON HE_SU Trigger frame is received by the MAC. The following types are supported: • BASIC TRIGGER • BEAMFORMING REPORT POLL TRIGGER • BQRP TRIGGER • BSRP TRIGGER • MU BAR TRIGGER • MU RTS TRIGGER The AID is set to a random generated value, in the User Info subfield of the Trigger frame. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM. For the following frames, these frames should appear as a response: • BASIC TRIGGER, BSRP TRIGGER – HE_TB frame, where the TX VECTOR parameters are extracted from the received Trigger frame • MU RTS – CTS response • MU BAR – BACK response For the HE TB response frame, the he length is constrained based on the length extracted from the received Trigger frame and the addresses are set based on the used KSR index. If it is a protected frame, it will be also encrypted. When the frame is prepared it is written to the SRAM, with a special access category AC_TB, and sent to the MAC-PHY interface.
TEST CONFIGURATION	
The following scoreboards are active during these testcases:	

- MAC RX SCOREBOARD
- MAC IMMEDIATE RESPONSE SCOREBOARD
- MAC IRQ SCOREBOARD
- PLATFORM IRQ SCOREBOARD

Table 6-2 test_mac_rx_*_trigger

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_data_frame	In this testcase the following SINGLETON DATA frames are received by the MAC:NON_HT, NON_HT_DUP_OFDM, HT_GF, HT_MF, VHT. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response CONTROL frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-3 test_mac_rx_data_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_control_frame	In this testcase the following NON HT SINGLETON CONTROL frames are received by the MAC. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. For ACK, CTS, BA and CONTROL WRAPPER the RA is set to a Unicast address. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response CONTROL frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-4 test_mac_rx_control_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_data_frame_ \he_mu \he_su	In this testcase HE SU SINGLETON DATA frames are received by the MAC. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response CONTROL frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-5 test_mac_rx_data_frame\he_mu\he_su

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_error_data_frame	In this testcase the following SINGLETON DATA frames are received by the MAC: NON_HT, NON_HT_DUP_OFDM, HT_GF, HT_MF, VHT, HE_SU, and HE_MU. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive this frame through MAC-PHY interface where some error scenarios can appear on the reception such as: PHY_ERR on the RX VECTOR, PHY_ERR before RX_END. If the frame is received it will be stored in the SRAM, and if the configuration of the frame is such, a response CONTROL frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-6 test_mac_rx_error_data_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_fcs_error	In this testcase the following SINGLETON QOS DATA frames are received by the MAC: NON_HT, NON_HT_DUP_OFDM, HT_GF, HT_MF, VHT, HE_SU, HE_MU. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. Randomly, FCS error will be inserted in some of the frames. During the next phase the MAC will receive this frame through MAC-PHY interface, and if there was no error in the received frame, it will be stored in the SRAM, and if the

	configuration of the frame is such, a response CONTROL frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-7 test_mac_rx_fcs_error

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_he_su_ctrl_id0	In this testcase HE SU SINGLETON QOS DATA frames are received by the MAC. The HT control is field is set in such way, that when received, the MAC will respond to it, in form of an HE TB frame. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response HE TB frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-8 test_mac_rx_he_su_ctrl_id0

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_mgmnt_frame	In this testcase NON HT MANAGEMENT frames are received by the MAC. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR for BEACON frame the RA is always set to BROADCAST address, for PROBE RESPONE, ASSOCIATION RESPONSE and REASSOCIATION REPSONE, the RA is set to device address, while for other MANAGEMENT frames, the RA can be set to BROADCAST, MULTICAST or device address. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response CONTROL frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD	

- MAC IMMEDIATE RESPONSE SCOREBOARD
- MAC IRQ SCOREBOARD
- PLATFORM IRQ SCOREBOARD

Table 6-9 test_mac_rx_mgmnt_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_mgmnt_he_su	In this testcase only HE SU MANAGEMENT frames will be sent. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive this frame through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response frame will be sent.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-10 test_mac_rx_mgmnt_he_su

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_ndp_mu_calib	In this testcase NDP MU calibration is simulated. First PHY behavior is emulated, where BFM report data will be created, when BEAMFORMING frame is sent as a response. Based on the generated Key Storage RAM, the addresses of the frames are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive first VHT NDP ANNOUNCEMENT and VHT NDP frame, if the STAID is different than 0, a BEAMFORMING REPORT POLL will also be received, through MAC-PHY interface and they will be stored in the SRAM. As a response a BEAMFORMING frame will be sent by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-11 test_mac_rx_ndp_mu_calib

TEST CASE NAME	TEST DESCRIPTION
test_mac_rx_ndp_su_calib	In this testcase NDP SU calibration is simulated. First PHY behavior is emulated, where BFM report data will be created, when BEAMFORMING frame is sent as a response. Based on the generated Key Storage RAM, the addresses of the frames are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the MAC will receive first VHT NDP ANNOUNCEMENT and VHT NDP frame through MAC-PHY interface and they will be stored in the SRAM. As a response a BEAMFORMING frame will be sent by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-12 test_mac_rx_ndp_su_calib

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_ampdu_frame	In this testcase the following AGGREGATED frames are sent by the MAC: HT_GF, HT_MF, and VHT. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. If there is protection in form of RTS/CTS, RTS frame is sent by the MAC, to which in response a CTS is received. During the next phase the MAC will send the AGGREGATED frame. Based on the expected ACK the following scenarios are possible: • NORMAL_ACK – An ACK frame is received for the AGGREGATED frame, a BAR is sent by the MAC, which is again acknowledged by the testbench • COMPRESSED_BA – BACK is received by the MAC • NO_ACK – only BAR is sent by the MAC All the received frames are stored inside the SRAM.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-13 test_mac_tx_ampdu_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_beacon_frame	In this testcase, transmission of BEACON and DATA frames are tested. The MAC is configured as an access point. BROADCAST address is set as RA of the BEACON frame. After it is prepared together with the DATA frames, they are written in the SRAM, and sent by the MAC through MAC-PHY interface.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-14 test_mac_tx_beacon_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_data_frame	In this testcase the following SINGLETON frames are sent by the MAC: NON_HT, NON_HT_DUP_OFDM, HT_GF, HT_MF, VHT and HE_SU. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. If there is protection in form of RTS/CTS, RTS frame is sent by the MAC, to which in response a CTS is received. During the next phase the MAC will send the SINGLETON frame and based on the expected response, and ACK frame can be received by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-15 test_mac_tx_data_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_data_frame_he_su	In this testcase only HE_SU SINGLETON frames are sent by the MAC. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. If there is protection in form of RTS/CTS, RTS frame is sent by the MAC, to which in response a CTS is received. During the next

	phase the MAC will send the SINGLETON frame and based on the expected response, and ACK frame can be received by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-16 test_mac_tx_data_frame_he_su

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_mgmnt_frame	In this testcase NON-HT SINGLETON MANAGEMENT frames are sent by the MAC, except BEACON frame. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. If there is protection in form of RTS/CTS, RTS frame is sent by the MAC, to which in response a CTS is received. During the next phase the MAC will send the SINGLETON frame and based on the expected response, and ACK frame can be received by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-17 test_mac_tx_mgmnt_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_mumimo_frame	In this testcase MUMIMO frames are sent by the MAC. During the generation of Key Storage RAM, WAPI encryption type is prevented. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. If there is protection in form of RTS/CTS, RTS frame is sent by the MAC, to which in response a CTS is received. During the next phase the MAC will send the AGGREGATED frame. Based on the expected ACK the following scenarios are possible: • NORMAL_ACK – An ACK frame is received for the AGGREGATED

	frame, a BAR is sent by the MAC, which is again acknowledged by the testbench • COMPRESSED_BA – BACK is received by the MAC • NO_ACK – only BAR is sent by the MAC All the received frames are stored inside the SRAM.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-18 test_mac_tx_mumimo_frame

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_ndp	In this testcase VHT NDP ANNOUNCMENT and NDP are linked together, and written into the SRAM. They both have be sent on the same channel bandwidth. After these frames are sent, as a response a BEAMFORMING REPORT will be received by the MAC, and saved in the SRAM.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-19 test_mac_tx_ndp

TEST CASE NAME	TEST DESCRIPTION
test_mac_tx_with_blank_delimiter	In this testcase the following AGGREGATED frames are sent by the MAC: HT_GF, HT_MF, and VHT. Blank delimiters are inserted between the MPDU frames inside an AMPDU. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. If there is protection in form of RTS/CTS, RTS frame is sent by the MAC, to which in response a CTS is received. During the next phase the MAC will send the AGGREGATED frame. Based on the expected ACK the following scenarios are possible: • NORMAL_ACK – An ACK frame is received for the AGGREGATED frame, a BAR is sent by the MAC, which is again acknowledged by the testbench • COMPRESSED_BA – BACK is received by the MAC • NO_ACK – only BAR is sent by the MAC

	All the received frames are stored inside the SRAM.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-20 test_mac_tx_with_blank_delimiter

6.2 PHY standalone test cases

TEST CASE NAME	TEST DESCRIPTION
test_modem_bf_tx	At first NDP parameters are randomized and these parameters will be used to set the configuration of the mdm_data_model. Based on the supported channel bandwidth control channel will be selected. The created NDP frame is then sent by the PHY. After some delay a new VHT AGGREGATED or SINGLETON frame will be prepared, with the same channel bandwidth as it was set in the NDP frame. Using this frame, a beamforming report will be stored in the memory through AHB, and the previously prepared frame will be sent by the PHY.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM BF SCOREBOARD	

Table 6-21 test_modem_bf_tx

TEST CASE NAME	TEST DESCRIPTION
test_modem_tx_ **\he_su** **\he_tb** **\ht_gf** **\ht_mf** **\non_ht** **\vht**	For the transmission testcases, register PRIMARYIND shall be set, which represents the value of the primary channel. Based on the configuration of the testcase the following PPDU formats can be set: NON_HT, HT_MF, HT_GF, VHT, HE_SU, HE_TB. If the randomized frame's format is HE_TB, the he_length in the frame needs additional constraining based on the leg_length. When all is setup the PHY is ready for frame transmission.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM TX SCOREBOARD	

Table 6-22 test_modem_tx_*

TEST CASE NAME	TEST DESCRIPTION
test_modem_rx_tx	In this testcase both transmission and reception of the following frames are simulated: NON_HT, HT_MF, HT_GF, VHT, HE_SU. At first a random number of frames are received through ADC interface, by the PHY, before it transmits a random number of frames through DAC interface.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM TX SCOREBOARD • MODEM RX SCOREBOARD • MODEM STREAM SCOREBOARD	

Table 6-23 test_modem_rx_tx

TEST CASE NAME	TEST DESCRIPTION
test_modem_rx_tx_dsss_frame	In this testcase both transmission and reception of NON_HT DSSS frames are simulated. At first a random number of frames are received through ADC interface, by the PHY, before it transmits a random number of frames through DAC interface.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM TX SCOREBOARD • MODEM RX SCOREBOARD • MODEM STREAM SCOREBOARD	

Table 6-24 test_modem_rx_tx_dsss_frame

TEST CASE NAME	TEST DESCRIPTION
test_modem_rx	In this testcase reception of the following frames are simulated: NON_HT, HT_MF, HT_GF, VHT, and HE_SU. A random number of frames are received through ADC interface, by the PHY.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM RX SCOREBOARD	

• MODEM STREAM SCOREBOARD

Table 6-25 test_modem_rx

TEST CASE NAME	TEST DESCRIPTION
test_modem_rx_fd_ **\he_mu** **\he_su** **\he_tb** **\ht_gf** **\ht_mf** **\non_ht**	In these testcases reception of the following frames are simulated: NON_HT, HT_MF, HT_GF, VHT, and HE_SU. A random number of frames are received through ADC interface, by the PHY, while the AGC is turned off. Using these tests the proper behavior of the frequency domain is checked.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM RX SCOREBOARD • MODEM STREAM SCOREBOARD	

Table 6-26 test_modem_rx_fd_*

TEST CASE NAME	TEST DESCRIPTION
test_modem_rx_ **\he_mu** **\he_su** **\he_tb** **\ht_gf** **\ht_mf** **\non_ht**	In these testcases reception of the following frames are simulated: NON_HT, HT_MF, HT_GF, VHT, and HE_SU. A random number of frames are received through ADC interface, by the PHY.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM RX SCOREBOARD • MODEM STREAM SCOREBOARD	

Table 6-27 test_modem_rx_*

TEST CASE NAME	TEST DESCRIPTION
test_modem_rx_td_ **\he_mu** **\he_su** **\he_tb** **\ht_gf** **\ht_mf** **\non_ht** **\vht**	In these testcases reception of the following frames are simulated: NON_HT, HT_MF, HT_GF, VHT, and HE_SU. A random number of frames are received through ADC interface, by the PHY, while the AGC is turned off. Using these tests the proper behavior of the time domain is checked.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM RX SCOREBOARD • MODEM STREAM SCOREBOARD	

Table 6-28 test_modem_rx_td_*

6.3 WLAN platform test cases

TEST CASE NAME	TEST DESCRIPTION
test_wlan_bf_mu_rx	In this testcase Beamforming MU is simulated on WLAN level. At first NDP parameters are randomized and these parameters will be used to set the configuration of the mdm_data_model. These parameters will be written in the necessary registers, and used for NDPA randomization purposes. Based on the generated Key Storage RAM, the addresses of the frames are set to the corresponding values from the KSR. First the NDPA is received through the ADC, and after a SIFS time the NDPA also. In MU calibration when AID in STA info is not matched, then Beamforming Report Poll is sent to the STA, in order for the STA to send a Beamforming Report. When the calibration process is finished, MU-MIMO frames are sent to the DUT through ADC, to which the DUT sends an ACK frame.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-29 test_wlan_bf_mu_rx

TEST CASE NAME	TEST DESCRIPTION
test_wlan_bf_su_rx	In this testcase Beamforming SU is simulated on WLAN level. At first NDP

	parameters are randomized and these parameters will be used to set the configuration of the mdm_data_model. These parameters will be written in the necessary registers, and used for NDPA randomization purposes. Based on the generated Key Storage RAM, the addresses of the frames are set to the corresponding values from the KSR. First the NDPA is received through the ADC, and after a SIFS time the NDPA also, to which an ACK is received on the DAC interface. When the calibration process is finished, AGGREGATED frames are sent to the DUT through ADC, to which the DUT sends an ACK frame.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-30 test_wlan_bf_su_rx

TEST CASE NAME	TEST DESCRIPTION
test_wlan_rx_ampdu_frame	In this testcase the following AGGREGATED frames are received by the MAC and PHY: HT_GF, HT_MF, VHT, HE_SU. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the PHY will receive this frame through ADC interface, and pass it on to the MAC through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is such, a response CONTROL frame will be sent, by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-31 test_wlan_rx_ampdu_frame

TEST CASE NAME	TEST DESCRIPTION
test_wlan_rx_dsss_frame	In this testcase DSSS frames are received by the MAC and PHY. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the PHY will receive this frame through ADC interface, and pass it on to the MAC through MAC-PHY interface, store it in the SRAM, and if the configuration of the frame is

	such, a response CONTROL frame will be sent, by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-32 test_wlan_rx_dsss_frame

TEST CASE NAME	TEST DESCRIPTION
test_wlan_rx_mumimo_frame	In this testcase MU-MIMO frames are received by the MAC and PHY. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. During the next phase the PHY will receive this frame through ADC interface, and pass it on to the MAC through MAC-PHY interface, store it in the SRAM. A response CONTROL frame will be sent in form of an ACK, by the MAC.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC RX SCOREBOARD • MAC IMMEDIATE RESPONSE SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-33 test_wlan_rx_mumimo_frame

TEST CASE NAME	TEST DESCRIPTION
test_wlan_tx_ampdu_frame	In this testcase the following AGGREGATED frames are sent by the MAC and PHY: HT_GF, HT_MF, and VHT. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. If there is protection in form of RTS/CTS, RTS frame is sent by the MAC, to which in response a CTS is received. During the next phase the DUT will send the AGGREGATED frame, through the DAC interface. Based on the expected ACK the following scenarios are possible: • NORMAL_ACK – An ACK frame is received for the AGGREGATED frame, a BAR is sent by the MAC, which is again acknowledged by the testbench • COMPRESSED_BA – BACK is received by the MAC • NO_ACK – only BAR is sent by the MAC

	All the received frames are stored inside the SRAM.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-34 test_wlan_tx_ampdu_frame

TEST CASE NAME	TEST DESCRIPTION
test_wlan_tx_dsss_frame	In this testcase DSSS frames are sent by the MAC and PHY. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. When the frame generation is done, the required MAC CONTROL INFO fields are set for THD, and the frame is written into the SRAM. During the next phase the DUT will send the DSSS frame, through the DAC interface.
TEST CONFIGURATION	
The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD	

Table 6-35 test_wlan_tx_dsss_frame

TEST CASE NAME	TEST DESCRIPTION
test_wlan_tx_mumimo_frame	In this testcase MU-MIMO frames are sent by the MAC and PHY. During the generation of Key Storage RAM, WAPI encryption type is prevented. Based on the generated Key Storage RAM, the addresses of the frame are set to the corresponding values from the KSR. If the frame is protected, encryption of the frame will be done. Before the transmission VHT NDP ANNOUNCEMENT and NDP are linked together, and written into the SRAM. They both have be sent on the same channel bandwidth. After these frames are sent, as a response a BEAMFORMING REPORT will be received Through the ADC and saved in the SRAM. Based on these parameters, the beamforming report will be saved in the memory. During the next phase, the required MAC CONTROL INFO fields are set for the MUMIMO frame, and it is written into the SRAM. In the following the MAC and PHY will send the MU-MIMO frame to which BAR frame is received as a response.
TEST CONFIGURATION	

The following scoreboards are active during this testcase: • MODEM STREAM SCOREBOARD • MODEM RX SCOREBOARD • MODEM TX SCOREBOARD • MAC TX SCOREBOARD • MAC IRQ SCOREBOARD • PLATFORM IRQ SCOREBOARD

Table 6-36 test_wlan_tx_mumimo_frame

7 SIMULATION FLOW – User manual

This chapter will give overview of how UVM TB is organized (directory structure) and how to run test case and regression lists. Script for running simulation will be described with all flags explained. Directory structure of UVM testbench is shown in Figure 7-1.

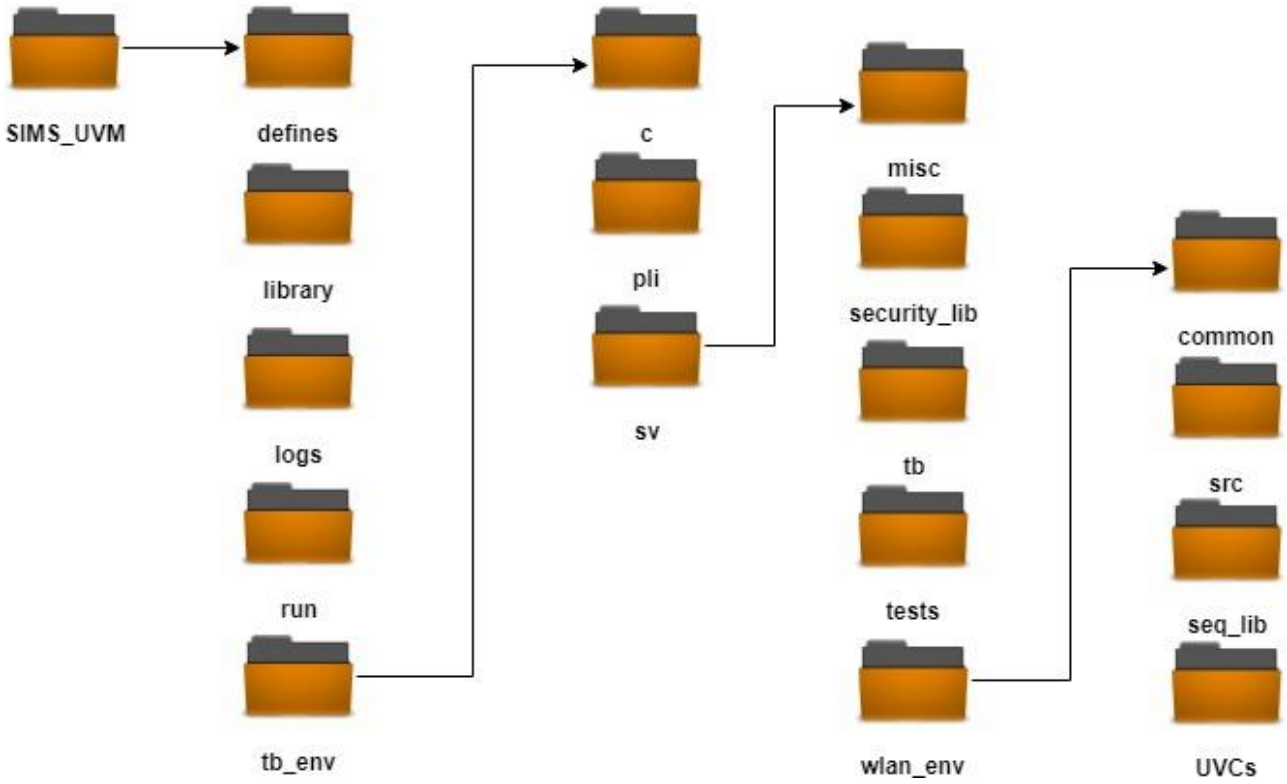

Figure 7-1 Verification folder structure

SIMS_UVM directory is one that contains all components needed for running test cases, these are sub directories:

- define: files used for common testbech defines needed during simulation (timescale, register base address definition, RTL paths definition and signal width definition).
- library: symbolic link with rw_wlan_nx_ovl file
- logs: place where test case log files are stored after simulation run
- run: for running simulation for all 3 types of testbench (MAC, MODEM and WLAN)
- tb_env: contains verification environment files like monitors, drivers, sequences, tests. Directories inside are grouped by language type.
    - C files: contains all C source files used in UVM environment
    - PLI: contains source files and make script for generating shared object (SO) which is PLI (Programming language interface) which is used to create system call tasks for systemverilog from C functions.
    - SV: contains all systemverilog source files and UVM components. It will be described in next paragraph.

Systemverilog directory has sub directories which are:

- misc: contains definitions of common functions that are used in verification environment but are not directly linked with any verification components already existing.
- security_lib: contains files used for encryption/decryption engine in frames (AES, CCMP, TKIP, WEP and WAPI).

- tb: contains top testbench file with instantiation of DUT, clock generation, UVC interfaces instantiation and wiring of verification components. Task run_test() is called in top testbench.
- tests: contains test case files grouped by system that they are testing (MAC, MODEM and WLAN).
- wlan_env: contains all verification component files, including register model, scoreboards, agents, sequences, frame object, descriptors. Chapter WLAN testbench architecture – detailed gives detailed overview of all components inside this directory.

The simulate.bash is placed in the run folder, from where the simulation can be run. The following commands are available in this script:

- -clean : Clean all temp files and logs
- -nodb : Run simulation without database
- -db : Run simulation with database
- -d | -define : Add define for compilation
- -up_reg | -ur : Update UVM register model
- -seed <n> : select seed randomization value. To get a random seed set to: random
- -all : Run simulation of all the testcases
- -f <filename> : Run simulation of all the testcases defined in tests/<filename>
- -t <test1> <test2> : Run simulation of the listed testcases
- -l : list the testcases available
- -gui : Start interactive session
- -loc : compile LOCALSOURCESLIB instead of SOURCESLIB
- -noverifdb : Disable verif database logging
- -simu.run <filename> : Provide the simulation command script to be used by irun
- -verbose : Setup UVM verbosity level
- -inst_verbose : Set verbosity level of specific component in UVM environment
- -err_cnt : set error count before simulation stops
- -fsdb : Enable FSDB generation support
- -cov | -coverage : Enable code coverage
- -config <name> : Select configuration of the DUT (for example config_STA_1x1_CBW20)
- -prof : Enable profiling
- -cpu : Select the top level IP including CPU (RISCV) and run simulation
- -jen : Enable Jenkins log
- -comp_dpi : Enable compilation of DPI code
- -keep_matlab : Instead of deleting Matlab config files, move the files to ./Pattern_backup for later debugging
- -for | -forensic : Creates a tarball named simu_backup.tgz of the working directory

Examples:
./simulate.bash –config config_STA_1x1_CBW20 –t test_wlan_rx_dsss_frame –err_cnt 1
will run the test_wlan_rx_dsss_frame testcase, with error count set to 1

./simulate.bash –config config_STA_1x1_CBW20 –t test_wlan_rx_dsss_frame –inst_verbose
.m_mac_phy_env.,_ALL_,HIGH,run –seed 2
The testcase is run with seed 2, and the verbosity is set to high to a specific component

Appendix

Agent files and folders organization

Example: ahb_master_agent, axi_slave_agent, ipc_agent, sram_agent, etc.

name_agent

- src
  - ***name***_agent.sv
  - ***name***_config.sv
  - ***name***_common.sv
  - ***name***_coverage.sv
  - ***name***_driver.sv
  - ***name***_if.sv
  - ***name***_item.sv
  - ***name***_monitor.sv
  - ***name***_pkg.sv
  - ***name***_sequencer.sv
- seq_lib
  - ***name***_seq_list.sv
  - ***name***_seq_base.sv
  - ***name***_random_seq.sv

UVC files and folders organization

Example: mac_phy_uvc (mac_agent, phy_agent), radio_uvc (radio_agent, spi_agent, gpio_agent etc.)

name_uvc

- src
  - ***name***_config.sv
  - ***name***_common.sv
  - ***name***_coverage.sv
  - ***name***_env.sv
  - ***name***_scoreboard.sv
  - ***name***_virtual_sequencer.sv
  - ***name***_pkg.sv
- ***name***_agent
  - src
    - ***name***_agent.sv
    - ***name***_config.sv
    - ***name***_common.sv
    - ***name***_coverage.sv
    - ***name***_if.sv
    - ***name***_monitor.sv
    - ***name***_driver.sv
    - ***name***_item.sv
    - ***name***_pkg.sv
    - ***name***_sequencer.sv
  - seq_lib
    - ***name***_seq_list.sv
    - ***name***_seq_base.sv
    - ***name***_random_seq.sv

Testcases files and folders organization

Example: wlan_tests, mac_tests, modem_tests

name_tests

- src
  name_test_pkg.sv
  name_test_base.sv
  test_***name***_random.sv
- regression_lists
  rlist_***name_rlistname***.rl

Testbench files and folders organization

Example: wlan_tb, mac_tb, modem_tb

name_tb

- src
  name_tb_top.sv

CODING GUIDELINES

1) General:
   - ✓ No TABs, use 2 spaces instead
   - ✓ Trim trailing spaces
2) All files should have same header with following information:
   - ✓ Copyright
   - ✓ Description
3) Code in all files should be bounded with following directives (example for name_monitor.sv) :

```
`ifndef  NAME_MONITOR_SV
`define NAME_MONITOR_SV
:
:
`endif // NAME_MONITOR_SV
```

4) Component instances in environment should have prefix m_*:

```
ahb_master_agent          m_ahb_master_agent;
spi_slave_agent           m_spi_slave_agent;
rx_modem_scoreboard       m_rx_modem_scb;
radio_uvc_config          m_radio_uvc_cfg;
modem_virtual_sequencer   m_modem_vsqr;
```

5) *_config.sv

```
//Configuration fields
bit         is_active;
bit         has_checks;
bit         has_coverage;
bit         has_name_agent;
:
bit         has_name_scoreboard;
bit         has_bus_monitor;
bit         has_bus_coverage;
```

   - ✓ Configuration object class should extend uvm_object
   - ✓ Register all configuration fields and handles using UVM factory registration macros
   - ✓ In constructor, set all configuration fields default values
   - ✓ In constructor, create all lower level configuration objects using factory type_id::create() method
6) *_common.sv
   a) Defines section: Capital letters should be used for defines

```
`define          MY_DEFINE          100
```

b) Parameters section
   - Defined type
   - Capital letters with _P suffix

```
parameter int unsigned MY_PARAMETER_P = 1;
```

c) Constants section
   - Capital letters with _C suffix

```
const int unsigned MY_CONSTANT_C = 1;
```

d) Types section
   - Use typedef to defined struct, enum or any other type
   - Enum types _e suffix
   - Struct types _s suffix
   - Union types _u sufix
   - Capital letters for enums
   - Typedef for others _t suffix

```
typedef enum byte {
  MY_ENUM_1 = 0,
  MY_ENUM_2 = 1
} my_enum_e;
```

References

	Title	RW-WLAN-nX-MAC-HW-UM		
	Reference	DMA descriptors		
	Version	3.01	**Date**	
	Source	Riviera Waves		

[1]	**Title**	RW-WLAN-nX-MAC-HW-FS		
	Reference			
	Version	3.00	**Date**	
	Source	Riviera Waves		

[2]	**Title**	RW-WLAN-nX Modem Functional Specification		
	Reference			
	Version	1.4	**Date**	
	Source	Riviera Waves		

[3]	**Title**	Universal Verification Methodology (UVM) 1.2 User's Guide		
	Reference			
	Version	1.4	**Date**	
	Source	Accellera (http://accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf)		

[4]	**Title**	RW-WLAN-nx-MAC-PHY-IF-FS		
	Reference			
	Version	3.01	**Date**	
	Source	Riviera Waves		