# UVM based VIP for APB-UART

# Table of Contents:

# Description of Configuration Files for APB UART Environment

## I. Uart Configuration

The Configuration settings are implemented inside `uart_config.sv`. This file contains all fields related to the configuration such as frame length, baud rate, parity, and the stop bit. This configuration file is registered inside the uvm configuration database using the `uvm_config_db` command inside `the apbuart_base_test` class. All other test classes extend from the apbuart_base_test base class, in this way all tests can be configured.

Inside the base test class as mentioned above there is a method named `set_config_params`. This method is responsible for setting the Configuration settings are implemented inside uart_config.sv. This file contains all fields related to the configuration such as frame length, baud rate, parity, and the stop bit. This configuration file is registered inside the uvm configuration database using the uvm_config_db command inside the apbuart_base_test class. All other test classes extend from the apbuart_base_test base class, in this way all tests can be configured. Configuration across the testbench environment. Users can configure this method from the test class and the remaining environment will set itself according to the configurations specified inside the method. Configuration settings are registered through `uvm_config_db` at uvm root, which enables its access in every component of the testbench environment. Currently, we only need its access inside our UART monitor and driver classes. The specified configurations will be set on DUT as mentioned by the user in the test class.

Inside the `set_config_params` method, there is a flag that decides whether to have directed configuration settings or randomized configuration settings. The configuration fields are implemented according to the DUT specifications. The number of generated tests can be configured using the Loop Time variable declared inside the configuration file.

## II. APB Configuration

The Configuration settings are implemented inside `apb_config.sv`. This file contains all fields related to the configuration such as address and pselect (slave select pin in APB). This configuration file is registered inside the uvm configuration database using the `uvm_config_db` command inside `the apbuart_base_test` class. All other test classes extend from the apbuart_base_test base class, in this way all tests can be configured.

Inside the base test class as mentioned above there is a method named `set_apbconfig_params`. This method is responsible for setting the Configuration settings to be implemented inside apb_config.sv. This file contains all fields related to the configuration such as address and pselect. This configuration file is registered inside the uvm configuration database using the uvm_config_db command inside the apbuart_base_test class. All other test classes extend from the apbuart_base_test base class, in this way all tests can be configured. Configuration across the testbench environment. Users can configure this method from the test class and the remaining environment will set itself according to the configurations specified inside the method. Configuration settings are registered through `uvm_config_db` at uvm root, which enables its access in every component of the testbench environment. Currently, we only need its access inside our APB monitor and driver classes. The specified configurations will be set on DUT as mentioned by the user in the test class.

Inside the `set_apbconfig_params` method, there is a flag that decides whether to have directed configuration settings or randomized configuration settings. The configuration fields are implemented according to the DUT specifications. The number of generated tests can be configured using the Loop Time variable declared inside the configuration file.

# Description of Agents and scoreboard of configurable UVM environment for APB UART Verification

There are two agents in the environment, APB agent for APB interface and UART agent for UART interface.

## I. APB agent:

APB agent drives and monitors the APB interface of DUT. The agent drives the DUT for read/write to configuration registers, transmission operation, and receiving operations. While the agent monitors "PRDATA", "PREADY", and "PSLERR" signals mainly during read operation on configuration registers and receiving operation on UART receiver.

### A. APB Driver class

Driver class is a parameterized class that is inherited from the "uvm_driver" base class and is registered with the UVM factory. Driver class drives DUT by converting transactions into pin level information for DUT. The class also performs handshaking with the sequencer using "get_next_item" and "item_done" API methods in the "run_phase". There are three phases that are built on in this class i.e. build_phase(), connect_phase and run_phase().

- **build_phase()**

In the build_phase, we get the handler for the "uart_config" class from the UVM configuration database ("uvm_config_db") using the "get()" call.

- **connect_phase()**

In this phase, we get the virtual interface handle from the UVM configuration database ("uvm_config_db") using the "get()" call.

- **run_phase()**

In the run_phase phase, transactions have been collected through the "get_next_item" method and after getting a transaction, the "Drive()" task has been called to drive that transaction (The task is explained below). After driving the transaction into pin level, a response has been sent to the sequencer using the "item_done" method.

- **Drive() task:**

"Drive()" is a user-defined task that drives the DUT through the interface according to the respective transaction. To drive the DUT for any transaction (whether it is read/write operation on configuration registers or transmission/receiving operation), we have to mimic the APB master behavior that is we need to set "PSELx" HIGH and after one clock cycle, we have to set "PENABLE" HIGH and provide DUT with "PADDR", "PWDATA" and "PWRITE" signal. We also have to keep these signals unchanged until we receive "PREADY" asserted from the DUT.

We provide configuration settings to DUT from the "uart_config" file. This file stores the configuration settings for UART and these settings can be set or randomized. So, for read/write operation on DUT configuration registers, "PWDATA" is provided from the configuration file. It makes it easy to drive multiple transmission and receiving operations every time with random configuration settings.

(NOTE: All the Interface signals are explained in the "interface description" file)

## B. APB Monitor class

Monitor class extends the "uvm_monitor" base class and is registered with the UVM factory. The class is responsible for collecting the signal information through the interface and to export the captured data to the scoreboard via analysis port. A virtual interface handle is used to perform signal capturing and TLM Analysis Port is used to broadcast the collected data. As the driver, there are two UVM phases updated in the monitor class i.e. build_phase() and run_phase().

- **build_phase()**

In the build_phase, we get a virtual interface handle from the UVM configuration database ("uvm_config_db") using the "get()" call. We can also create an instance of TLM analysis port here but in this particular environment, we have done this in the "new" construct of monitor class.

- **run_phase()**

During run_phase(), data from the interface is sampled and then exported to the scoreboard for comparison via TLM analysis port. In this phase, monitoring of "PREADY" and "PSLVERR" get started right upon the assertion of "PSELx" and "PENABLE". The error gets sampled if the "PSLVERR" is monitored HIGH, but the value of "PADDR", "PRDATA", and "PREADY" is sampled into the transaction when DUT sets the "PREADY" HIGH. And when the DUT de-asserts the "PREADY" to LOW, the transaction has been sent to the scoreboard in a non-blocking manner using the "write()" call.
All this monitoring follows the APB protocol and the behavior of DUT.

## II. UART agent

UART agent drives and monitors the UART interface of DUT i.e TX and RX pin. The UART agent drives the RX pin when there is a receive request on DUT while it monitors the TX pin for transmission operation.

### A. UART Driver class

Driver class is a parameterized class that is inherited from the "uvm_driver" base class and is registered with the UVM factory. Driver class drives DUT by converting transactions into pin level information for DUT. The class also performs handshaking with the sequencer using "get_next_item" and "item_done" API methods in the "run_phase". There are three phases that are built on in this class i.e. build_phase(), connect_phase and run_phase().

- **build_phase()**

In the build_phase, we get the handler for the "uart_config" class from the UVM configuration database ("uvm_config_db") using the "get()" call. We also created an instance of TLM analysis port using the new() method here, this transaction is being sent to the scoreboard later.

- **connect_phase()**

In this phase, we get the virtual interface handle from the UVM configuration database ("uvm_config_db") using the "get()" call.

- **run_phase()**

In the run_phase phase, we only call the "get_and_drive()" task and two more tasks in the get_and_drive().

In the get_and_drive() task, the transactions have been collected through the "get_next_item" method and after getting a transaction, cfg_settings() task and then "drive_rx()" task has been called. After that, a response has been sent to the sequencer using the "item_done" method.

In the cfg_settings() task, we determine the value of loop time ("LP") that determines the number of frames to be sent according to the value of frame length.

drive_rx() is a task that drives the DUT RX pin during receiving operation on UART. RX-pin has been driven here according to the respective configuration settings in the configuration file i.e. baud rate, frame length, parity enable/disable, even/odd parity, and several stop bits.

### B. UART Monitor class

Monitor class extends the "uvm_monitor" base class and is registered with the UVM factory. The class is responsible for collecting the signal information through the interface and to export the captured data to the scoreboard via analysis port. A virtual

interface handle is used to perform signal capturing and TLM Analysis Port is used to broadcast the collected data. As the driver, there are two UVM phases updated in the monitor class i.e. `build_phase()` and `run_phase()`.

- **build_phase()**

In the build_phase, we get a virtual interface handle from the UVM configuration database (`"uvm_config_db"`) using the `"get()"` call. We also get the handler for the `"uart_config"` class from the UVM configuration database `("uvm_config_db")` using the `"get()"` call. We can also create an instance of TLM analysis port here but in this particular environment, we have done this in the `"new"` construct of monitor class.

- **run_phase()**

During `run_phase()`, data from the interface is sampled and then exported to the scoreboard for comparison via TLM analysis port. For this purpose `"cfg_settings()"` and `"monitor_and_send()"` tasks have been called here.

In the `"cfg_settings()"` task, we determine whether parity is enabled/disabled according to configuration settings. The value of loop time (`"LP"`) is also determined here according to the value of frame length. The value of `"LP"` tells the number of frames to be received from the TX pin of the UART interface.

`"monitor_and_send()"` is the task where actual monitoring is implemented. Only data value (payload) is sampled according to the baud-rate, parity bit and stop bit/s are omitted and not sampled. First payloads for every frame are temporarily stored in a local register and after receiving all the frames the value of the register is passed to the transaction and then sent to the scoreboard via analysis port.

## III. APB-UART Scoreboard class

UVM Scoreboard is an important component inside a UVM based testbench environment, it primarily acts as a checker and verifies the functionality of a design. It receives transaction-level information captured in the monitor via the TLM analysis port. Like driver and monitor classes, the scoreboard class is again extending the "uvm_scoreboard" base class and is registered with the UVM factory. There are necessary TLM analysis ports to receive transactions from other components like monitors and drivers in this case. These TLM ports are instantiated in the "build_phase" of the scoreboard class. There are also additional *queues* in our scoreboard to store received transactions from monitors and drivers, and "write()" functions that receive packets from monitors and drivers and push into the respective queues respectively.

There are two UVM phases built-on in the scoreboard class i.e. "build_phase" and "run_phase".

- **build_phase()**

UVM analysis ports are instantiated using a "new()" call because after all these are class objects. We also get the handler for the "uart_config" class from the UVM configuration database ("uvm_config_db") using the "get()" call.

- **run_phase()**

Three different functions named "compare_config()", "compare_transmission()", and "compare_receive()", that are called in the "run_phase()" and perform comparison.

"compare_config()" is called when there is a read operation on configuration registers. It compares the value coming from APB-monitor to the scoreboard with the configuration settings.

If there is a transmission operation on DUT, "compare_transmission()" is called which compares the data written to transmit on DUT by APB-driver with the data monitored on TX-pin of DUT by UART-monitor.

"compare_receive()" is called when there is receive operation, this function compares the data fed to RX-pin by UART-driver with the data received by APB-monitored from DUT. The function also compares the error signal as we have mechanisms for feeding errored data to DUT.