LOGGER BLOCK

A Verification Component

Vinay Jain
Pune, India
vinay.jain5@wipro.com

Table of Contents

Contents

Send Feedback

Introduction

Handshaking is an automated process of negotiation that dynamically sets parameters of a communications channel established between two entities before normal communication over the channel begins.

Logger is verification component and data object based on System Verilog – UVM methodology. This component provides handshake mechanism between C code running on processor and top level SV/UVM testcases in SoC verification.

Major Challenge in verifying SoC while running C Code on processor

- Synchronization and Handshake between SV/UVM testcase and C code running on processor
- Printing Debug messages from C code.

Most of time we need synchronization or handshake from C testcase to perform some required action in SV/UVM testcase and vice versa. We used SV events or UVM events in Pure SV environment for synchronization or handshake between communications of two components. But C doesn't not provide such implementation in its library.

Also it's not possible to print debugging messages from your C code as function from 'stdio.h' eg. Printf would not understandable to processor.

To overcome these challenges we introduced the logger block which is pure verification component and provides below feature.

Features of Logger:

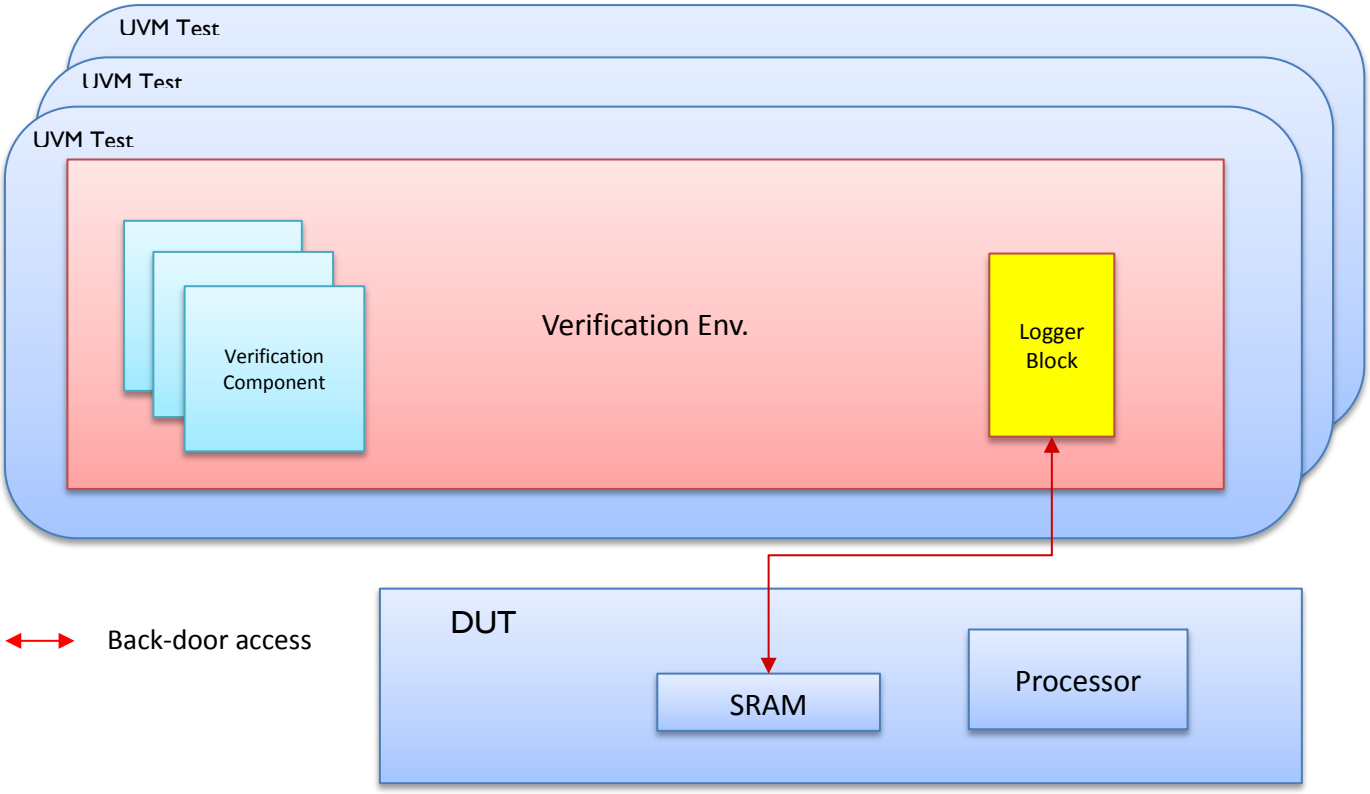
- Provide Synchronization and handshake mechanism between C code running on processor and top level SV/UVM testcases.
- Allow to Print Debug messages.
- More Faster as compare to legacy logger.
- No-timing issue in printing long & lengthy messages
- Standard Printing Methodology used.
- SV-UVM based
- 4 steps integration in any SoC Environment.
- In-build C macros

This logger block will access specific range within the SRAM address space. Two way communications between C testcase running on processor and SV/UVM testcase is established through in-build macros/method of this block.

Example Architecture

Block Diagram

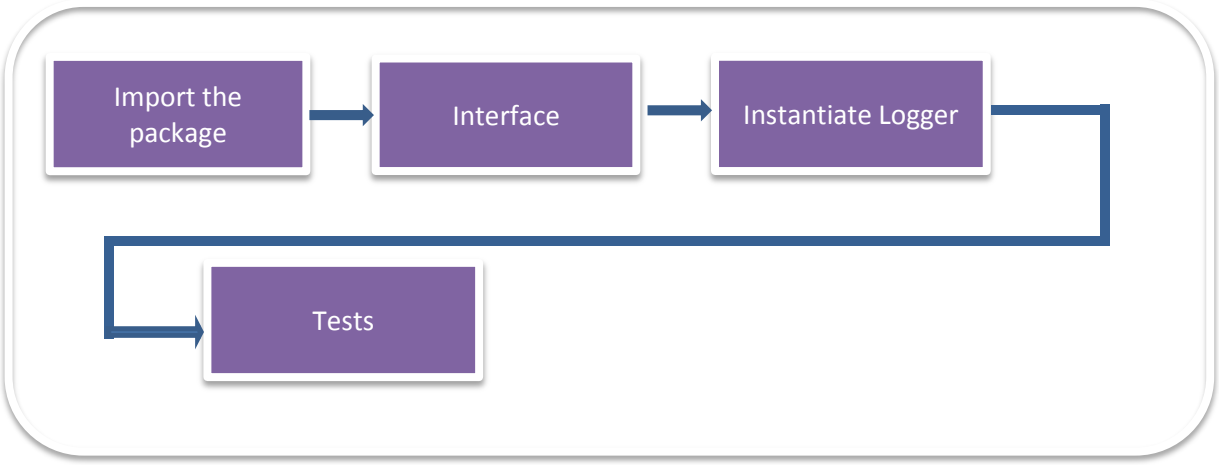
The following diagram explains the basic Testbench architecture.



The basic example is designed only to demonstrate the usage of Logger block in SoC SV verification Environment.

Note that as it's not possible to provide example with real processor so we demonstrate this example with SV based functions. We also keep sample example which is proven example in real SoC Verification environment.

Click on each box for more details.



Quick start Guide

Logger packages are used in this step

Steps:

1. Include and import the "logger_top_package.sv" in Testbench top package.
   top_package.sv

```
//include logger top package
`include "logger_top_package.sv"
package top_package;
.
.

// Import Logger package
import logger_top_package::*;
```


Send Feedback

2. Define the hierarchical path for memory (SRAM) which will use as shared memory between C tests running on processor and SV/UVM testcases in top module.
   top_tb.sv

```
module top_tb();

.
.
//Define the Memory Instance Hierarchical path
`define MEM_INST top_dut.sram_ins
//Derived class implementation.
`include "bd_mem_derived.svh"
```

Logger Interface is used in this step

Steps:

1. In Top level module, instantiate the interface for logger by using in-build *logger_if.*
   top_tb.sv

```
//Instantiate the Interface for logger.
logger_if logger_if(clk);
```

2. Set the virtual logger interface for logger components.
   top_tb.sv

```
//Set Logger interface
initial begin
  #0ns;
  uvm_config_db#(virtual logger_if)::set(uvm_root::get(),"*","LOGGER_IF",logger_if);
end
```

Note that string name should be "LOGGER_IF".

Auto System Verilog Creation

In this Step will we will use SVV tool which comes with logger package i.e. bin/svv To create the system Verilog-UVM base testcase.

Steps:

1.perl <logger_home_dir>/bin/svv -c <c_src_file_name> -s <sram_addr>

Usage:

svv [-c/csrc <c source file>] [s/sram_addr<logger base address>] [d/debug <1/0>]

-c/csrc	Input C source code file
-s/sram_addr	Logger base SRAM address which will be same through the application.
-b/base_test	Base testcase name from which SV testcase supposed to be extend. By Default it will extend from uvm_test.
-d/debug	To enable Debug Mode.

Logger classes are used in this step.

Steps:

1. Define the 'base_class' which extends from uvm_test, and instantiate the logger.

base_test.sv

```
class base_test extends uvm_test;
      `uvm_component_utils(base_test)
  .
  .

  // Instantiate the logger
  logger_base logger;
  .
  .

endclass : base_test
```

2. Construct the logger object in new method.

Send Feedback

base_test.sv

```
// Class Constructor
  function new(string name ,uvm_component parent);
    super.new(name,parent);
    logger =  new("Logger",parent);
  endfunction : new
```

3. Initialize the logger with the base address. This address will be used as shared memory location.
   base_test.sv

```
// Build Phase
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    // Initialization of logger block
    logger.init(`LOGGER_HANDSHAKE_ADDR);
  endfunction : build_phase
```

Logger classes & method will be used in this step

Steps:

As this example is just to demonstrate the functionality of logger block, there are two threads cm3_process & sv_process

- Cm3_process will stand for tests which will run on processor
- Sv_process will stand for SV/UVM testcases.

Case I :-

In this case a simple info message needs to print after performing some operation on processor.

```
virtual task cm3_process();
    int data;

    #10; // some processor process is ongoing.
    data = 1  << `FIRST_MSG;
    // Note: In C code for processor we will be using macro UPDATE_LOGGER_BIT(id)
    // eg. UPDATE_LOGGER_BIT(data).
    logger.bd_access.bd_access_write(`LOGGER_HANDSHAKE_ADDR,data);.

     .
     .
endtask : cm3_process
```

In SV/UVM testcase we can print user specific message with the 'id' which gets set with 'UPDATE_LOGGER_BIT' macro in C testcase.

```
virtual task automatic sv_process();
    .
    .

    //Basic User Info after Write/read/instruction execution has been performed.
    logger.wait_for_state(`FIRST_MSG); //User can use SV define for better
                                       readable code.
    `uvm_info(get_type_name,"First Write/read/instruction has been
executed"),UVM_LOW)

   .
   .
  endtask : sv_process
```


Send Feedback

Case II:-

In this case SV/UVM testcase is waiting for handshake/sync signal before executing some operation from C testcase.

```
virtual task cm3_process();
    int data;
    .
    .

    #50; //Some operation which required to process before execution of sequence
         // in SV/UVM testcase.
    data = 1 << `DATA_SEQ;
   // Note: In C code for processor, we will be using macro UPDATE_LOGGER_BIT(id)
   // eg. UPDATE_LOGGER_BIT(data).
    logger.bd_access.bd_access_write(`LOGGER_HANDSHAKE_ADDR,data);

    .
    .
endtask : cm3_process
```

SV/UVM testcase can start sequence one 'id' which gets set with 'UPDATE_LOGGER_BIT' macro in C testcase.

```
virtual task automatic sv_process();
    .
    .
    `uvm_info(get_type_name(),"Waiting for C testcase to Pass handshake to SV
              testcase",UVM_LOW)
     logger.wait_for_state(`DATA_SEQ); //User can use SV define for better readable
                                       //code.
   `uvm_info(get_type_name(),"Pretending to run UVM base sequence here.",UVM_LOW)

   .
   .
  endtask : sv_process
```


Send Feedback

Case III:-

In this case user would like print the debug info message along with the data which he is using while performing operation in C.

```
virtual task cm3_process();
    int data;
    .
    .

    //Note: In C code for processor ,we will be using
    //C macro LOGGER_UPDATE_DATA(id,data1,data2…).
    // eg : LOGGER_UPDATE_DATA(HANDSHAKE_WITH_DATA,data)
    data = 1 << `HANDSHAKE_WITH_DATA;
    logger.bd_access.bd_access_write(`LOGGER_HANDSHAKE_ADDR,data);
    data = 32'hdeadbeef;
    logger.bd_access.bd_access_write(`LOGGER_DATA_ADDR,data)
    .
    .
endtask : cm3_process
```

In SV/UVM testcase we can print user specific message with the ‘id’ which gets set with ‘UPDATE_LOGGER_DATA’ macro in C testcase.

```
virtual task automatic sv_process();
    bit [31:0] rd_data [];
    .
    .
    // Note : As C macro LOGGER_UPDATE_DATA has variable argument
    // in SV/UVM we can print the number of argument as follow.
     // eg : `uvm_info(get_type_name(),$psprintf("Printing Data0 : %0h Data1 :%0h
     //        Data2 :%0h",rd_data[0], rd_data[1],rd_data[2]),UVM_LOW)
    logger.get_data_with_state(`HANDSHAKE_WITH_DATA,data);
    `uvm_info(get_type_name(),$psprintf("Handshake Received from C testcase with
                                Data : %0h",rd_data[0]),UVM_LOW)
    .
    .
  endtask : sv_process
```

Both thread cm3_process & sv_process will start at same time in Run phase of testcase.

base_test.sv

```
//Run Phase
virtual task run_phase(uvm_phase phase);

    .
    .
    .
  // This is dummy process where we tried to show two treads i.e CM3 execution
  // and SV testcase runs concurrently in SoC verification Env.
  fork
    cm3_process();
    sv_process();
   join

    .
    .
    .
endtask : run_phase
```

To run the example please follows steps.

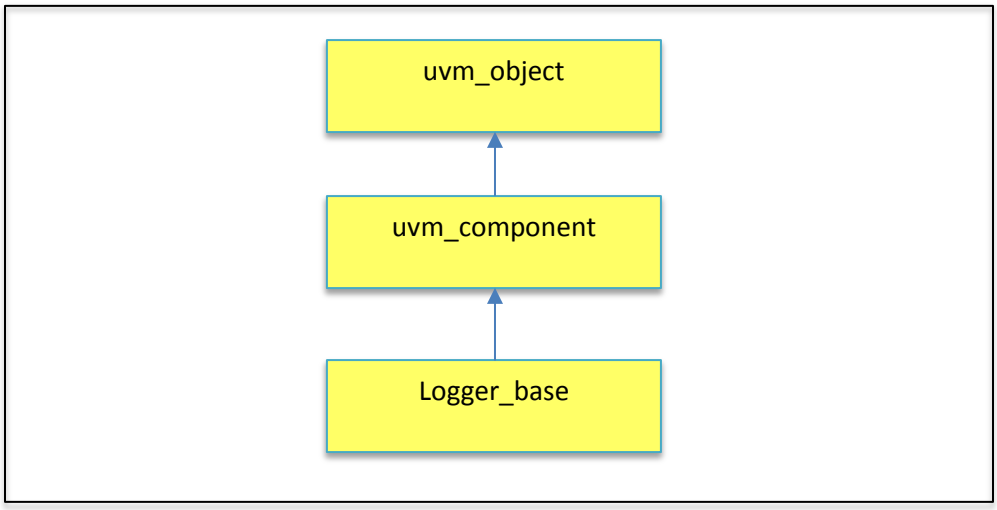
```
1. Go to Logger/example
2. source sourceme.cshrc
3. run the base_test
   make run TEST=base_test

       vcdplus.vpd --> Wavefrom dump
```


Send Feedback

Class Reference

➢ Logger_base

- **Inheritance diagram**



- **Detail Description**

  This class defines the AGENT for logger block. This agent contains the implementation of logger methods.

- **Public member functions**

Function Type	Function name
function void	new (string name, uvm_component parent)
function void	build_phase(uvm_phase phase)
function	init(bit [(`ADDRESS_WIDTH -1):0] saddr)
task automatic	wait_for_state(int s)
task automatic	get_data_with_state(input int s,output bit[(`DATA_WIDTH-1):0] ret_data[])
task automatic	release_state()
task automatic	readcheck()

Send Feedback

- **Member function details**

  - `function void new (string name, uvm_component parent)`
    Class constructor: This creates the new instance of logger agent class.

    name – *The name of instance*
    parent - *The component that contains this instance*

  - `function void build_phase(uvm_phase phase)`
    UVM Build phase construct the sub-component.

  - `function init(bit [(`ADDRESS_WIDTH -1):0] saddr);`
    Initialize the logger agent with Base address of shared memory as argument.
    `ADDRESS_WIDTH is by default 32, can be change by passing +define+ADDRESS_WIDTH=<width> during compilation.

  - `task automatic wait_for_state(int s)`
    This task waits for particular state which is set from C testcase. Argument 's' is ID which is passed during C macro 'UPDATE_LOGGER_BIT(id)'

  - `task automatic get_data_with_state(input int s,output bit[(`DATA_WIDTH-1):0] ret_data[])`
    This task returns the array of data which passed during C Macro UPDATE_LOGGER_DATA(id,data1,data2,…)

  - `task automatic release_state();`
    Resume the C testcase.

  - `task automatic readcheck();`
    SV function which should call in SV testcase correspond to C macro READCHECK(Base_Address, Offset_address, Expected data) in C testcase.

- **Logger_if**

  This is system Verilog interface define the top-level interface used by logger agent. Clocking block cd is used as sample event in logger agent.

C Macros

This Macro's will be used in C base testcase which will be running on Processor.

Note: Refer sample_example present in example directory.

Macro Name	Arguments
LOGGER_UPDATE_BIT	(id)
LOGGER_UPDATE_DATA	(id, data1,data2, …)
READCHECK	(BASE_ADDR,OFFSET,exp_data)
LOGGER_HOLD_STATE	()

- **LOGGER_UPDATE_BIT(id)**

  This macro used to set respective message id in C testcase.

- **LOGGER_UPDATE_DATA(id,...)**

  This macro having one fixed argument which is message id and rest are variable arguments. This macro is used to pass different set of data to SV testcase.

  Eg: `LOGGER_UPDATE_DATA(id,data1,data2)`

- **READCHECK(BASE_ADDR,OFFSET,exp_data)**

  This macro can be used to check the expected data from specific address. Can also be used as C checker to confirm Write operation is happened or not.

- **LOGGER_HOLD_STATE()**

  This macro can be used to Hold the C testcase execution until SV/UVM testcase release the hold state.