Accelerate Your Verification with LVM Verification IP

TRY GO THRU THE QUESTIONNAIRES

WHY IS IT IMPORTANT TO HAVE A GOOD QUALITY PROTOCOL VIP?

- To have best silicon health!
  - you need a thoroughly randomization capable VIP
  - you need a detailed and strict checker with the VIP
  - The whole point of design verification is to catch RTL bug

WHY IS IT IMPORTANT TO HAVE A GOOD QUALITY PROTOCOL VIP?

- To save time!
  - Time is the key to win in a fast pace company
  - Is your VIP require a lot of time on the following item?
    - Integrating, instantiating
    - Installing with RAL where you need to code adaptor or predictor
    - Create your own scoreboard to check response / data
    - Create a lot of uvm sequence to drive stimulus
  - Besides, you want a CPU efficient VIP!
  - Our VIP has unlimited license thus can achieve fast check out.

WHY IS IT IMPORTANT TO HAVE A GOOD QUALITY PROTOCOL VIP?

- To save money!
  - Our VIP is having most reasonable price in the industry with the great quality / price
- To have minimal code while using the VIP
  - Using our VIP will help "clean up" your testbench! Our VIP can absorb a lot of codes!
  - Always remember, the lesser the code, the better your testbench
- To have junior engineer can verify the RTL right away!
  - Junior DV or even designer who is not UVM-experienced, can use VIP to do all kinds of stimulus

HOW LONG DO YOU SPEND TO INTEGRATE A VIP THEN SEND THE FIRST PACKET?

- Have you wonder that this job can be done by a junior engineer with very less uvm knowledge?
- Our senior engineer clients use <4 hours in average
- Our junior engineer clients use 1 day max.
- Our VIP is designed to be “UVM-blind” that designer / junior DV also can do the job!

UNLIMITED LICENSES? UNLIMITED INSTANCES?

- Is your VIP limit your regression where only n number of instances * number of parallel run are allowed at 1 time?

- Our VIP does not do that! We provide the unlimited instances and also unlimited parallel run with 1 license purchase!

IS YOUR VIP READY FOR SOC MEMORY VERIFICATION?

- Our VIP are specialized to verify sram / rom in your SOC.

- Embedded checker for data and response are ready inside VIP. Most importantly, they can be fully configured:
  - Ignore certain memory range
  - Customized expected response checker, for example, write will have error response, read ok response.

HOW DO YOU VERIFY THE CORRECTNESS OF DATA AND RESPONSES DURING A CPU INSTRUCTION FETCH?

- Your SoC has one or more CPUs that boot firmware from SRAM/ROM.
- When the CPU begins fetching instructions after booting, a mid-reset, or waking from sleep or deep sleep, can you ensure that every single read data is free from corruption and with the correct response?

HOW DO YOU VERIFY YOUR ROM?

- Can your VIP verify ROM automatically?
  - Your ROM is read-only thus designed to be hresp=1 / bresp=SLVERR when write
  - Write data shall not have effect
  - Read back to ensure all data is untouched.
  - b2b random write / read towards ROM are having the correct response and data from slave

HOW DO YOU VERIFY YOUR INTERCONNECT?

- Can your VIP verify AHB / AXI interconnect automatically?
  - Range A to B will be having HRESP=0 or BRESP/RRESP=OK
  - Range B+1 to C will be having HRESP=1 or BRESP/RRESP = SLVERR or DECERR and return data all 0 or all 1?
- Have you exercised parallel traffic that fully stress your interconnect?
- Can your VIP tells you which slave is having data corruption easily?

MULTI-MASTER SOC: SUSPICIOUS PACKET CHECK AVAILABLE?

- In your system, says that you have a module that has 3 masters interfaces.
- In this test, expected to have traffic on master 0 and master 1 only.
- Due to a bug in the module, master 2 is running and do some illegal accesses.
- Can your VIP catch this bug?

HAVE YOU EXERCISED RANDOM PACKETS – VALID AND INVALID ADDRESSES

- Have you covered a scenario where you put an access of invalid memory region in between the good packets, where the bad packets are responded with bad hresp while good one are correct with data and hresp?

- Have you cover all the packet types, wrap, incr, single randomly, for good and bad memory region , with full blast of random packet?

CPU MEMORY ACCESS PATTERNS COVERAGE CAN BE FULFILLED?

- Have you covered some CPUs memory access patterns, where it may read and write same or adjacent address in pipeline mode?

IS YOUR VIP COMES WITH RAL READY?

- Yes for LVM VIP!
- You do not need to code any predictor or adapter.
- You just need to connect the RAL via register block or register map, our VIP will do the rest!

CAN YOUR VIP PRINT RAL MAP AND DETAILED REGISTER ACCESS?

- Can your VIP print out whole ral map?
- Can your VIP do predictions which clearly show which field are changed (and which fields are not) per every register write packet.

IS YOUR VIP READY FOR REGISTER PARTIAL OR BURST PROGRAMMING?

- UVM bit bash helps verifying register access in 32 bits, and single packet.

- Says that your register map support 8 bits and 16 bits access, with the field access varies from RW, RO, W1C, WO etc, have you assured that these scenarios are covered at your tests with correct data returned?
  - partial write followed by full read
  - Full write followed by partial read
  - Partial write followed by partial read.

- What about burst programming of these registers?
  - Write burst has effect data written, while burst read return correct data.
  - While burst access can be 8 / 16 / 32 bits

- Our VIP does all in <5 lines of code

CAN YOUR VIP SUPPORT REGISTER ALIASING CHECK AND RESERVED CHECK?

- When you have a register of a slave range from 400h to 7ffh, while used registers are just valid from 400 to 4ffh, have you confirmed that writing register 500, or 600, or 700 won't impact your original register at 400h?
- For the above, can you confirm that 500h till 7ffh is ready only and return 0 every time?
- Our VIP does this in <5 lines of code.

CAN YOU CONTROL SETUP AND HOLD TIME OF YOUR VIP?

- Can your VIP control the setup and hold time with 2 lines of code?

CAN YOU CONVENIENTLY TURN OFF A CHECKER?

- While it is confirmed / justified, you may sometimes want to turn OFF the our strict checker for a small window of time within the simulation, or the whole test.

- Our error messages are embedded with codes to turn OFF, so that you can basically do it without referring the VIP manual.

CAN YOUR VIP CHECK WHETHER THE CLOCK SUPPLY HAS A GLITCH?

- It can be turned OFF/ON on the fly like entering clk gating, or before PLL clk stabilized.

IS YOUR VIP CONTAIN ARM SVA CHECKER?

- Is your AMBA VIPs activating the arm sva checker?
- Can it be easily disabled?
- Is it being flagged as uvm error?

IS YOUR VIP CAN EASILY MIX DIFFERENT PACKET SENDING STYLES IN SINGLE TEST?

- Can your VIP configured to be waiting data before proceeding and then configured to be pipeline, easily in single test?

- How many lines of code to do that?

IS YOUR AHB VIP GOT THESE FEATURES?

- Can your VIP allow you easily
  - cancel on error
  - cancel on not ready
  - Busy injection?
- How many lines of code to make it happen?
- Do you have a testsuite that exercise all the above randomly?

DO YOU HAVE JTAG VIP WITH OPENOCD EQUIPPED?

- We can run openocd to your DUT via our JTAG VIP in EDA simulation.
  - First in the industry!

- It also provides user friendly printing for activities done by openocd, including the DMI register access activity, for RISC V Debug Module.

IS YOUR VIP "EDUCATIONAL"?

- Have you wondered that before using the VIP, you need to fully understand the protocol first?
- Do you know that LVM VIP let you learn the data byte lane concept easily just by reading the traffic log files?
- Do you know the interface signal help you translate the signals to let you visualize the packet?

VALUABLE SELF VERIFICATION TESTBENCH FROM YOUR VIP?

- Is your VIP comes with copy-paste ready sample testbench codes?

- Is the TB built with rough structure? Or demonstrate not only how to integrate and use the VIP, but also showing some useful code on
  - Makefile
  - systematic filelist
  - Ral example code and ral usage
  - Uvm testbench simple but robust structure
  - etc

DO YOUR VIP PROVIDE YOU A VALUABLE SELF VERIFICATION TESTBENCH?

- Is your VIP comes with sample codes that showcase on how to use the VIP to send write or read?
- Is the TB dump waveform to help you see the traffic?
- Is the sample testbench able to do uvm randomization?
- Do you know it has sample uvm scoreboard that fetch seq item from VIP monitor, with all connections in place?

CAN YOU RETRIEVE SEQ ITEM FROM VIP MONITOR EASILY?

- Is your VIP monitor gives you seq item per traffic captured on bus?
- Do you know how is your seq item looked like? Is there a sample code to show that?

FROM WHICH VENDOR CAN YOU FIND A FREE REINDENTATION SCRIPT ATTACHED?

- Do you know that a well indented code for testbench can improve read-ability of the code thus improve efficiency.

- What VIP vendor can provide a free code reindentation / alignment script as a free tool?

IS YOUR VIP VENDOR GIVES YOU FREE EVALUATION PERIOD UP TO 3 MONTHS?

- Sometime you just want to have a feel and experience on using our VIP, and yes, it is totally free of charge!

INTRODUCTION TO LVM VIP

LVM VIP - OVERVIEW

- Currently having multiple customers, tape out >4 SOC chips
- Current products:
  - AMBA
    - AXI
    - AXI Stream
    - AHB
    - APB
    - TCM
  - JTAG – first JTAG VIP in the industry with openOCD
  - USART

INTRODUCTION

- Our VIP are SV UVM compatible.
- It is robust, reusable, measurable, systematic and efficient, with easy debug-ability and user-friendly features.
- Support for multiple EDA platforms
- Support Custom Solutions: Ability to provide tailored solutions

- **Objectives:**
  - To simplify complex SoC/ASIC design verifications.
  - To create various stimulus with ease (minimal codes).
  - To shorten the time to bring up UVM testbench with RAL.
  - To provide high quality industry standard protocol checkers.
  - To create an ideal platform for new UVM users.
  - To provide a **low cost solution** for verification in the industry.

COMPARISON WITH THE TOP VENDORS

LVM VERSUS OTHERS

No	Criteria	𝕃	Others
1	*Price*	Cheap	Expensive
2	*License*	Unlimited instances, unlimited parallel run	Limited, may need to query when parallel jobs are run. Extra cost for extra license.
3	*Speed of compilation, simulation*	Faster (30s)	Slower (55s)
4	*Customer Service*	Fast	Slow
5	*Useful Tracker*	Helpful for debug	NA
6	*Memory Tracker*	Available	NA
7	*Ease of sending packet*	Very easy as it is API based, can copy paste from example code	Troublesome, not much example code, need to use UVM sequence
8	*RAL readiness (predictor, adaptor)*	Already embedded	Need to install and embed by user.
9	*RAL content printing*	Ready	No such API
10	*AHB/AXI Burst programming register API*	Available	No such API
11	*AHB/AXI Partial programming register API*	*Same as above*	No such API
12	*LVM Bit bashing*	Advanced RAL bit bashing	No such API
13	*Auto Data Check (Data and Response signal)*	Yes	No such API

LVM VERSUS OTHERS

No	Criteria	𝕃	Others
14	*Speed from integrating to sending first packet*	Fast, even junior engineer (1 year experience) can do it	Slow, need more senior engineer to do the job
15	*RAL full register printing facility*	Clear	No such API
16	*ROM (AXI / AHB / APB) verification*	*Auto verify ROM in the chip, where write has no effect, and if response is expected error, VIP will auto check*	No such API
17	*Unused register map range verification*	Yes	No such API
18	*Register aliasing check*	Yes	No such API

LVM VERSUS OTHERS

No	Criteria	𝕃	Others
19	*Embedded HRESP/BRESP/RRESP check*	Ready, user configures and it will be activated	No such facility.
20	*AHB Busy stimulus, AHB Cancel when not ready / Cancel on error*	Can be easily enabled via plusarg/cfg	New sequence to code
21	*Performance*	*Clear and simple score, retrievable*	“
22	*Testsuite ready*	Ready to be run, *fully focus on data integrity (very useful for SRAM / ROM verification in bus), total test=82 for AHB, 80 for AXI*	No such facility or need to pay extra to get it
23	*AHB HREADYIN checker*	Yes	No
24	*Character printer facility*	Yes	No
25	*Signal skew insertion*	Ready	No
26	*Self Verification TB learning values*	The example TB of the VIP is built with high quality, which can demo Makefile, RAL, module vs class world, as a mini UVM testbench.	Not much value, in fact, confusing new UVM users

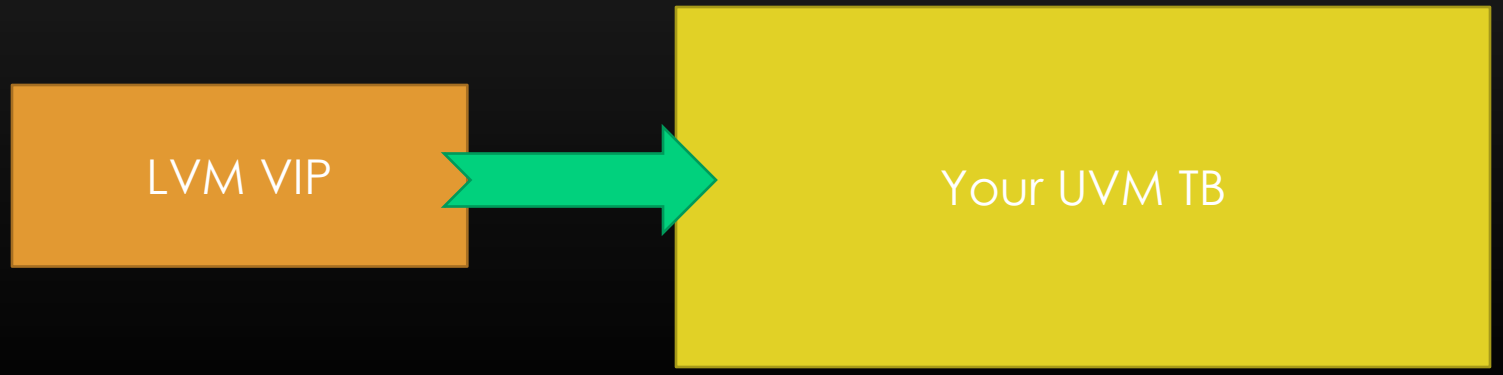
STRENGTHS

USER FRIENDLY

Integration and Configuration

INTEGRATION & CONFIGURATION

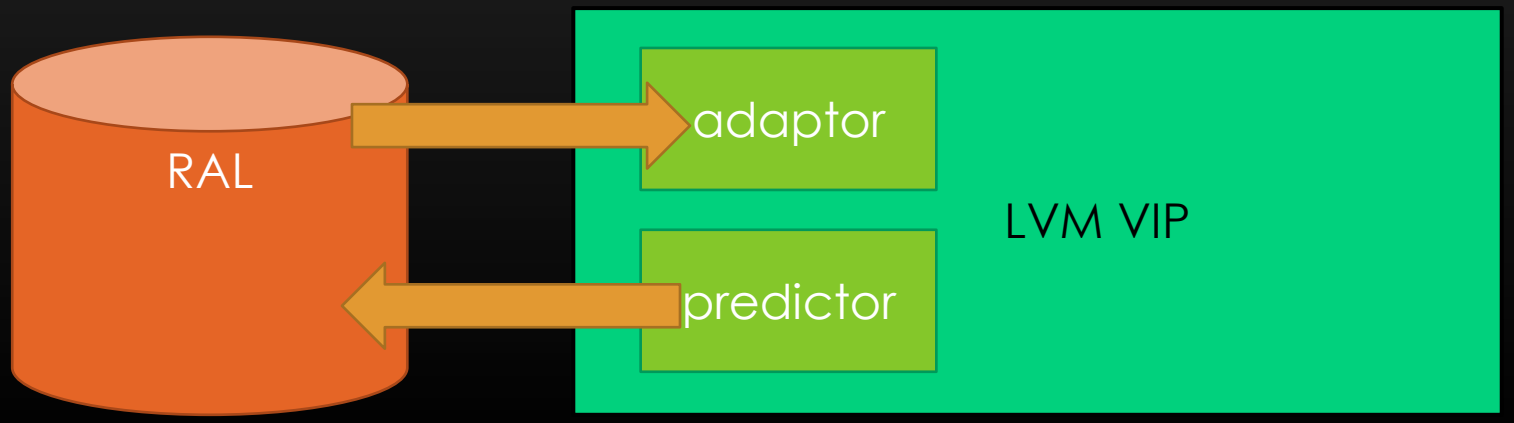
- Very easy to integrate, simpler than other vendors
  - All steps are demonstrated in the self verification testbench.
- Can configure different protocol per instance
- Can configure different signal width per instance
- Easily can on-off components on the fly.
- Just need to instantiate the UVC, no need to do anything on cfg class, sequence, sequencer, adaptor, predictor etc



DRIVING PART

ADVANCED TECH WITH RAL

- Predictor and Adaptor are built in, connections syntax is ready.
- Support partial and full register access (individual accessible modes) for driving
  - 8b accesses
  - 16b accesses
  - 32b accesses
- Predictor works for burst packets, and partial accesses packets



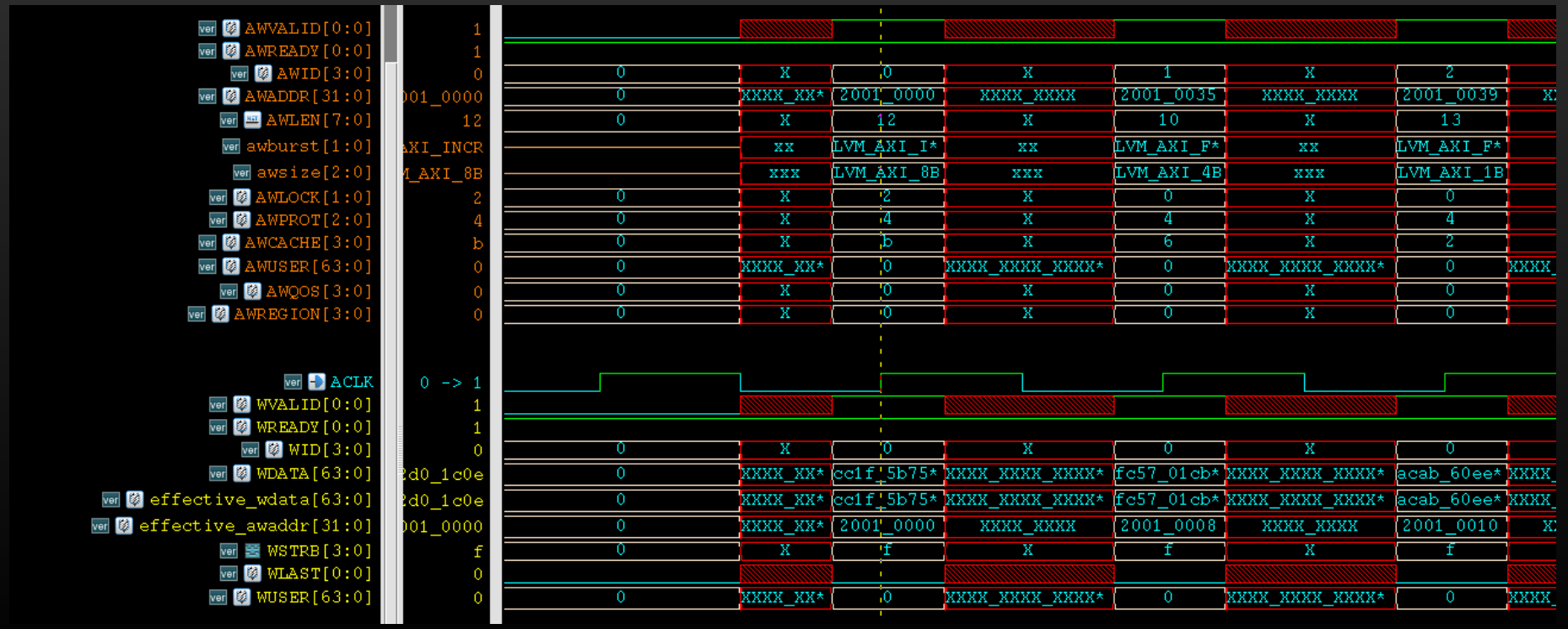
EASE OF DRIVING STIMULUS

- All fully pipelined.
- API based.
- Various API to do more stimulus style
  - Ease of driving AW and W sequences
    - AW first
    - W first
    - Parallel
  - Parallel AW and AR, busiest traffic
  - Wait and driving capabilities.
  - Ready signals, power signals driving.
- Most API can be done in 1 line of code.
- Full examples at self verification uvm tests.


```
m_axi_env.s_write(.AWADDR(32'h2000_0000),.wdata(32'h1111_2222),.BID(BID),.BRESP(BRESP));
```

SETUP AND HOLD X INJECTION

- Already supported X injection for window outside setup and hold time.
- Can be easily configured and can be turned OFF too.



REUSABLE CODE

- As stimulus mostly done using UVC's API, the code is very reuse friendly, where just the UVC handle is needed.

```
m_axi_env.drv_wait_output(LVM_ON);


m_axi_env.s_write(.AWADDR(32'h2000_0000),.wdata(32'h1111_2222),.BID(BID),.BRESP(BRESP));
`uvm_info(msg_tag, $sformatf("BID=%0h BRESP=%0h", BID, BRESP), UVM_DEBUG)


m_axi_env.s_read (.ARADDR(32'h2000_0000),.rdata(rdata)        ,.RID(RID),.rresp(rresp));
`uvm_info(msg_tag, $sformatf("RID=4'h%0h, RRESP=2'h%0h, RDATA=32'h%0h", RID, rresp, rdata), UVM_DEBUG)
```

SIMPLE SEQ ITEM RETRIEVAL

- Full code for seq item retrieval for all info needed is already part of example user scoreboard
- Already can be used for various high level scoreboard.

```
`uvm_info(msg_tag, $sformatf("AWID='h%h  AWADDR='h%h  AWPROT='h%h  AWLEN='h%h  AWSIZE='h%h  AWBURST='h%h  AWLOCK='h%h  AWCACHE='h%h",
    captured_item.AWID,
    captured_item.AWADDR,
    captured_item.AWPROT,
    captured_item.AWLEN,
    captured_item.AWSIZE,
    captured_item.AWBURST,
    captured_item.AWLOCK,
    captured_item.AWCACHE
), UVM_MEDIUM)
```

```
        foreach(captured_item.WDATA[i]) begin
            // effective read data per address
            write_impact = "";
            foreach (captured_item.mem_impact[i].addr[j])
            write_impact={$sformatf(" %h<-%2h", captured_item.mem_impact[i].addr[j],      captured_item.mem_impact[i].data[j]),write_impact};


            `uvm_info(msg_tag, $sformatf("[Beat %3d %2d:%2d] ADDR='h%h  EFF='h%h  WDATA='h%h  WUSER='h%h  WSTRB='h%h",
                i,
                captured_item.msb[i],     captured_item.lsb[i],
                captured_item.ADDR[i],
                captured_item.effective_wdata[i],
                captured_item.WDATA[i],
                captured_item.WUSER[i],
                captured_item.WSTRB[i]
            ), UVM_MEDIUM)


            // effective write data per address
            write_impact = "";
            foreach (captured_item.mem_impact[i].addr[j])
            write_impact={$sformatf(" %h<-%2h", captured_item.mem_impact[i].addr[j],      captured_item.mem_impact[i].data[j]),write_impact};
            `uvm_info(msg_tag, $sformatf("                IMPACT=%s",write_impact), UVM_MEDIUM)
        end
```

MONITORING PART

COMPREHENSIVE TRACKER

- Full info for each packet in the AXI bus can be observed via tracker.
- Make the debug handy

```
[Packet 547: WRITE 537]     23190.000 ns
       ------------------------------------------------------------------------------------------------------------------------------------------------
       AWID=4'h8  AWADDR=32'h20010012  AWLEN=8'h04  AWSIZE=LVM_AXI_1B  AWBURST=LVM_AXI_INCR   AWLOCK=LVM_AXI_LOCKED     AWCACHE=4'hb  AWPROT=3'h0

                           WID   ADDR          EFFECTIVE            WDATA                  WSTRB  IMPACT
       [Beat   0   23:16]  4'h8  32'h20010012  8'h11                64'h18c50a6c14118e19   8'h04  20010012<-11
       [Beat   1   31:24]  4'h8  32'h20010013  8'h74                64'ha610cf3c74ab84b3   8'h08  20010013<-74
       [Beat   2   39:32]  4'h8  32'h20010014  8'hff                64'he9d4ceffa68b78e1   8'h10  20010014<-ff
       [Beat   3   47:40]  4'h8  32'h20010015  8'h71                64'hdf6971101f6bff2a   8'h20  20010015<-71
       [Beat   4   55:48]  4'h8  32'h20010016  8'hb6                64'h69b65808ae1b36a5   8'h40  20010016<-b6

       BID=4'h8  BRESP=LVM_AXI_OKAY

       +-------------------------------------------------------------------------------------------------------+
       |  AWCACHE[3]=1          AWCACHE[2]=0                AWCACHE[1]=1               AWCACHE[0]=1              |
       |  LVM_AXI_ALLOCATE      LVM_AXI_NO_OTHER_ALLOCATE   LVM_AXI_MODIFIABLE         LVM_AXI_BUFFERABLE        |
       |                                                                                                       |
       |                        AWPROT [2]=0                AWPROT [1]=0               AWPROT [0]=0              |
       |                        LVM_AXI_DATA_ACCESS         LVM_AXI_SECURE_ACCESS      LVM_AXI_UNPRIVILEGED_ACCESS  |
       +-------------------------------------------------------------------------------------------------------+
```

END OF TEST MEMORY PRINTER
END OF TEST SCOREBOARD PRINTER

- It prints all memory contents exercised in the test to ease the debug of the AXI traffic impacts in the test.
- Also print total read and total write, ensure test is not empty.

MEMORY TRACKER

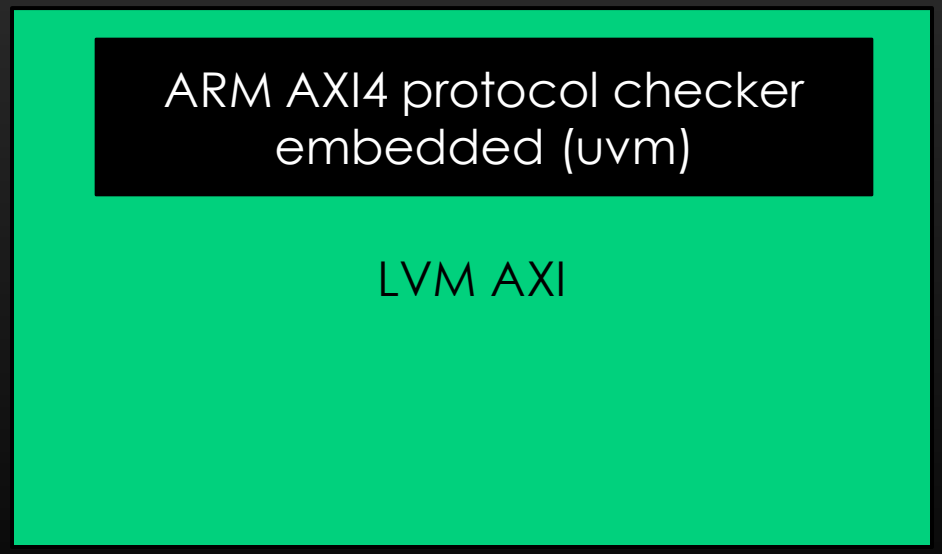
NO	ADDR	f e d c	b a 9 8	7 6 5 4	3 2 1 0
1	20000000	ED 6A 4E B4	E6 28 B7 D5	D2 36 3D 17	13 61 11 08
2	20000010	BF F7 3A 51	A1 39 DA D5	AB E0 64 DD	B6 8B 59 B4
3	20000020	2A 4C C2 48	5A 58 BE C7	E2 66 EB 77	C2 D9 49 47
4	20000030	DD 43 5A 89	56 C7 60 94	1D 55 0F 69	DB 66 96 1A
5	20000040	EA EE 15 B9	53 A3 F7 A3	58 07 79 53	FF A4 22 31
6	20000050	75 5E 47 8B	88 79 A9 2D	02 D5 BF 71	ED F4 71 75
7	20000060	46 5F CC F4	58 BF BB 84	17 0D 31 C1	5C E5 B6 10
8	20000070	7B 5E E4 DB	A8 B1 53 2B	A5 3C E6 EE	18 E9 A7 59

```
--------------------------------
     AXI Scoreboard Report
--------------------------------
    Total AXI Reads :      10
    Total AXI Writes:     538
    Total Checked   :      10
--------------------------------
```

CHECKING PART

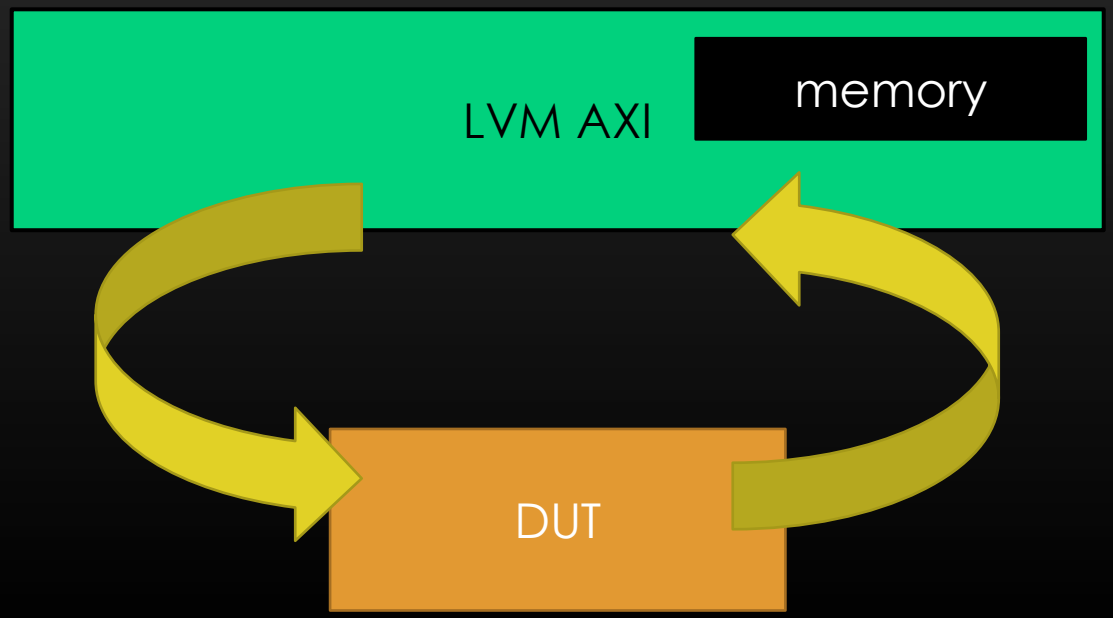
EMBEDDED ARM PROTOCOL CHECKER

- Already integrated SVA from ARM for AXI4 protocol checker.
- Enhanced to become UVM_ERROR when assertions fail.



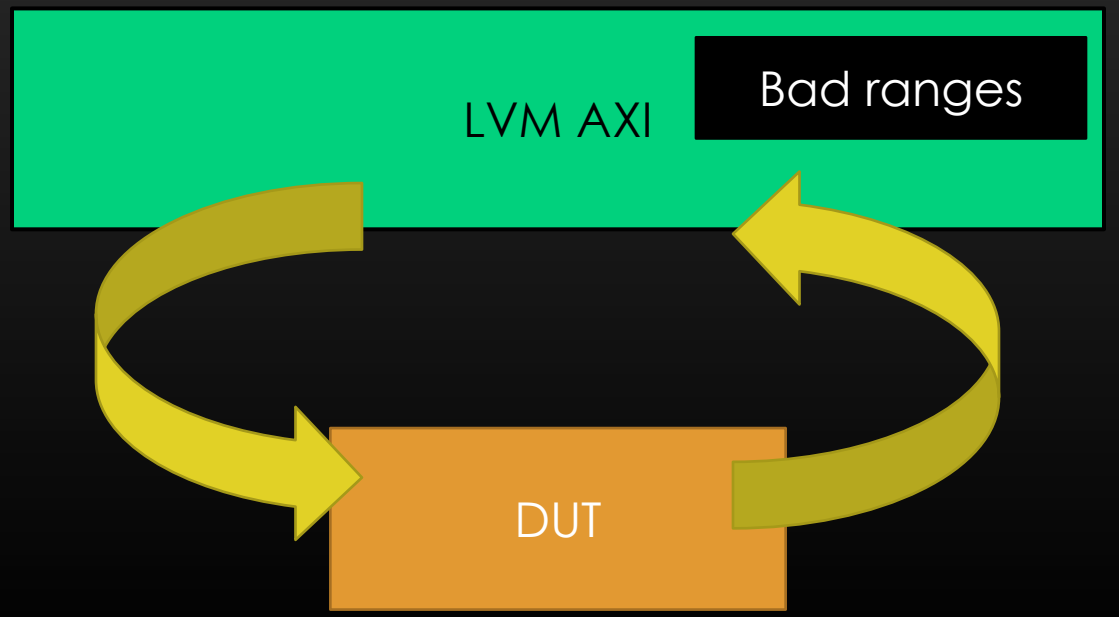
EMBEDDED MEMORY CHECKING

- Built in memory verification scoreboard, where
  - all writes within memory range (configurable) will be keep tracked as new data (fulfil the conditions)
  - All reads within memory range (configurable) will check data matching expectation (per last written data)



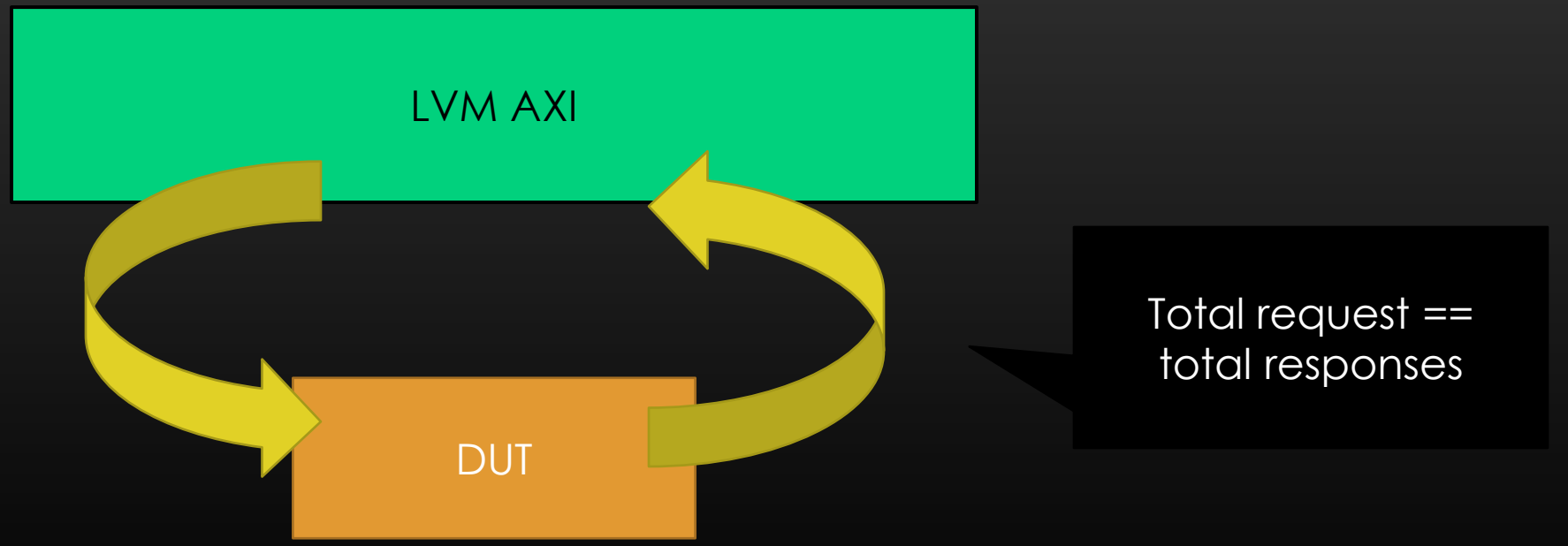
OUT OF BOUND ADDRESS CHECKING

- Built in out of bound address verification scoreboard, where
  - all writes outside valid address range will get BRESP=DECERR/SLVERR (user configurable)
  - All reads outside valid address range will get RRESP=DECERR/SLVERR (user configurable)



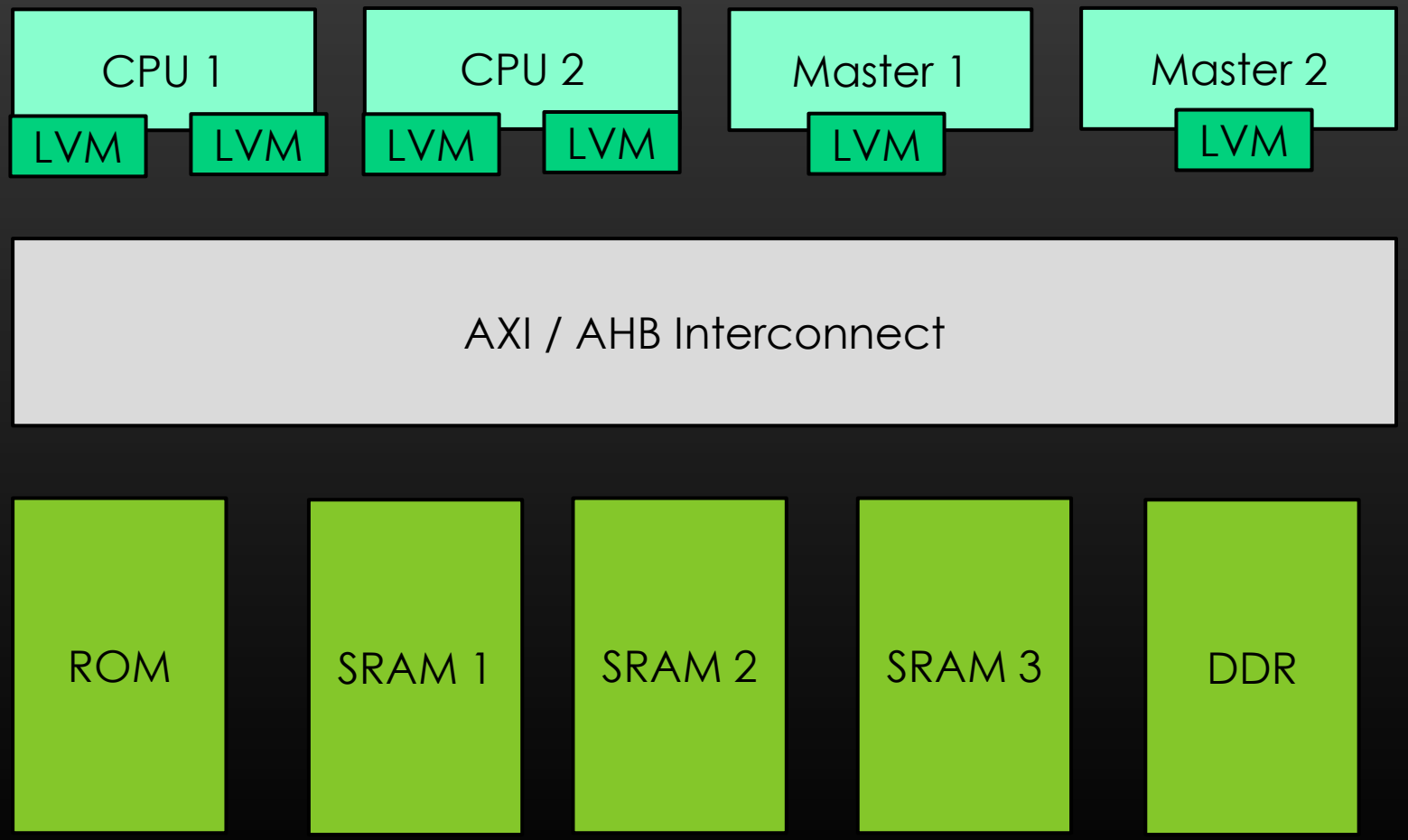
AUTO ENSURE SLAVE COMPLETES ALL REQUESTS

- At the end of test, UVC will ensure slave does not missed out any read / write.



POWERFUL TOOL TO VERIFY DATA PATH IN SOC

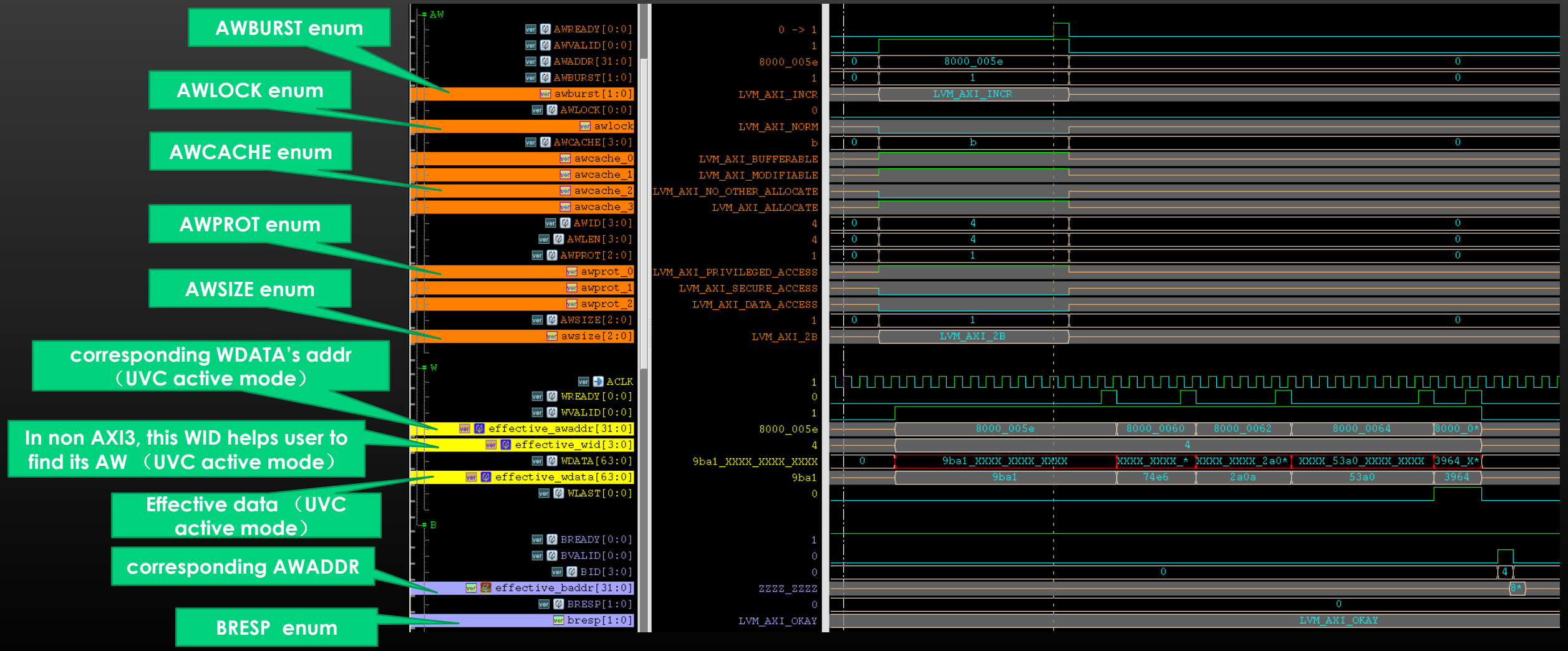
- Says that you have 4 masters and 5 target memories
- Applying LVM testsuite can let you confirm:
  - All valid paths are correct
  - All invalid paths are returned with error responses (configurable)
  - All random packet are exercised / covered in pipeline with all data are checked.
  - Testsuite that launched in parallel will ensure your system can handle busiest traffic



DEBUG PART

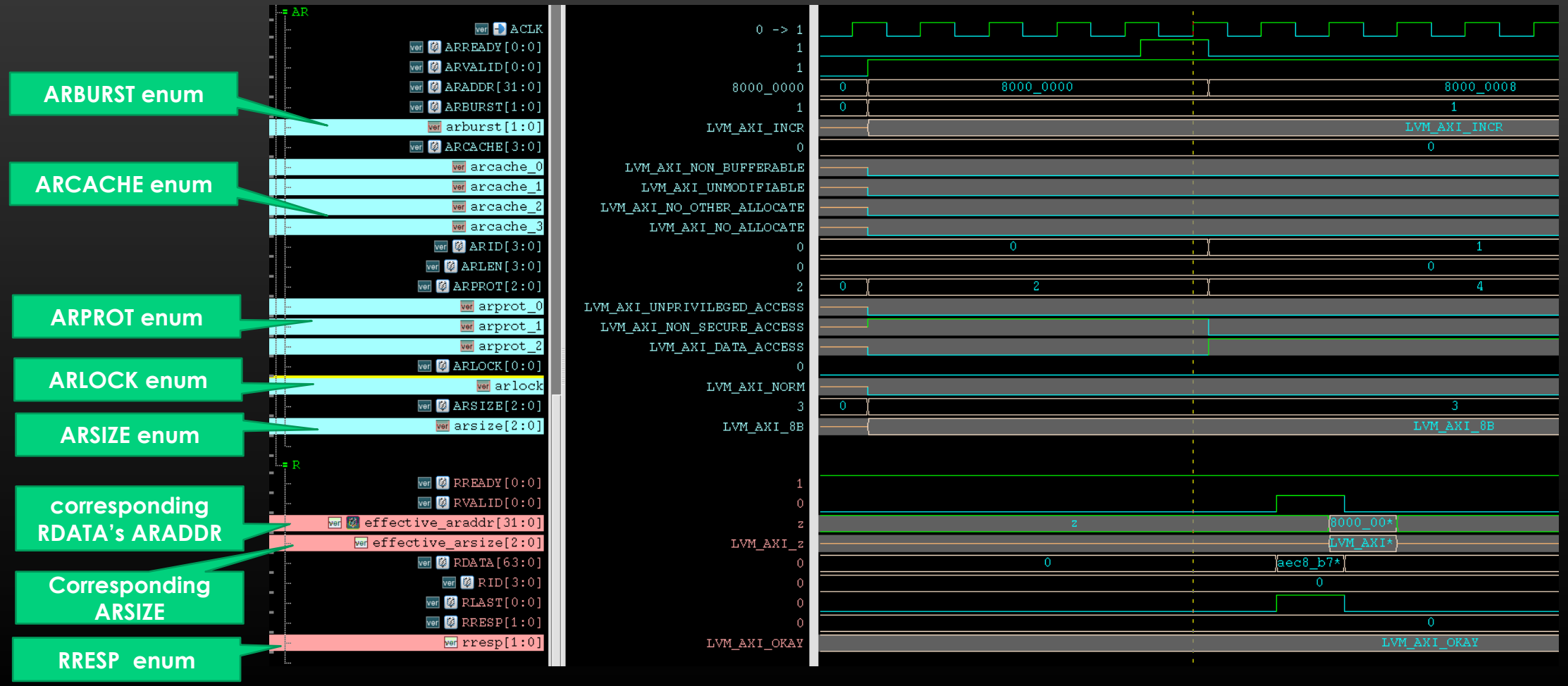
ENUM & DEBUG SIGNALS

- awburst, awsize, bresp, and more



ENUM & DEBUG SIGNALS

- arburst, arsize, rresp and more



TESTSUITE

THE BENEFITS FROM TESTSUITE

- Can verify various aspects, for example:
  - Verify data path of every master to all possible slaves in an SOC
  - Verify the data integrity, with full blast of randomized AXI packets
  - Verify the response of every packet to meet expected value.
  - Verify the every channel outstanding depth.
  - Verify all types of BREADY, RREADY conditions (for stressing slaves/interconnect)

PERFORMANCE ANALYZER

PERFORMANCE ANALYZER

- Performance per AXI channels

AXI Performance Report

Chnl	Total thru	Total waits	Average wait wait / beat	% Bus utilization	Performance
AW	472	3390	7.182	12	WORST
W	472	3352	7.102	12	WORST
AR	924	7939	8.592	10	WORST
R	924	0	0.000	100	EXCELLENT
B	472	0	0.000	100	EXCELLENT

PARTIAL REGISTER ACCESS VERIFICATION

PARTIAL REGISTER ACCESS VERIFICATION

- Can verify register partial access
  - Full access is fully verified at uvm's bit bashing sequence
- LVM adds the coverage for partial accesses

BURST REGISTER ACCESS VERIFICATION

BURST REGISTER ACCESS VERIFICATION

- Can verify register burst access
  - Single beat full access is fully verified at uvm's bit bashing sequence
- LVM adds the coverage for burst accesses
- This enable the bus read / write more than 1 registers in burst modes, where randomized burst size will be covered as well.
- Example like below, where there are 4 burst packets (in 4 different colors) in 1B to program all the 6 x 32bits registers.

Register 0	BYTE 3	BYTE 2	BYTE 1	BYTE 0
Register 1	BYTE 3	BYTE 2	BYTE 1	BYTE 0
Register 2	BYTE 3	BYTE 2	BYTE 1	BYTE 0
Register 3	BYTE 3	BYTE 2	BYTE 1	BYTE 0
Register 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0
Register 5	BYTE 3	BYTE 2	BYTE 1	BYTE 0

WHY CHOOSING LVM VIP?

Strengths

User friendly
Minimum lines of code to send packet.
Friendly for new UVM engineers.
API based UVC.

Reusable
Codes on lvm VIP is highly reusable.

Ease of Integration
RAL-ready with adaptor and predictor.
Minimum steps to integrate.

Light weight
CPU efficient UVC.

Robust
Highly configurable.
Parameterized signal width per instance.

High debug-ability
Useful tracker log, interface signals.

Performance Analyzer
Performance analyser for each channel

Strong & Strict Checker
Industry standard checker embedded.
Support X injection at Read and Write DATA for inactive lanes.
Embedded memory checker, RAL access OK checker

Reset aware
Support reset events.

Ready testsuite
Provided multiple useful tests to verify AXI slaves DUT

Register Burst Access
Embedded register burst access where user just need ~4 lines of codes to start it

Register Partial Access
Embedded register partial access where user just need 2 lines of codes to start it

Setup and Hold X injection
Inject X outside setup and hold window.

THANK YOU !!

- Thank you for your time!
- Let our VIPs talk for themselves.
- LVM will provide free consultation on VIP installation and usage model.
- LVM can provide free 1 month evaluation with no obligations