LVM RAL PREDICTOR

Version 1.06 (Dec 2025)

Overview

Overview

About LVM RAL Predictor

WHAT IS RAL PREDICTOR FROM LVM

- Enhanced version of UVM RAL predictor
- Coded to be robust, and loaded with easy debug-ability and user-friendly features
- It is free!

OBJECTIVES OF LVM RAL PREDICTOR

- To help user see the effect of bus transaction towards RAL in best detail
- To help user to migrate from hard-coded RAL programming to UVM register model write and read style
- To ease the firmware coding with info like address and data

Before AND After of RAL Predictor

BEFORE UVM RAL Predictor original print

Read

```
[LVM_PRED_READ ] Observed READ transaction to register urm.control_0_00: value='h5cd4b34b
```

Write

```
[LVM_PRED_WRITE] Observed WRITE tourm.control_8: value='h2c252101 : updated value = 'h2c252101
```

Before AND After of RAL Predictor

AFTER LVM RAL Predictor

Read

```
[LVM_PRED_READ ] 4) Observed READ from urm.control_0_00 : was 'hdc6d0012  updated='h87fe7761
[LVM_PRED_READ ]
[LVM_PRED_READ ]    urm.control_0_00.read  (status, reg_data);
[LVM_PRED_READ ]
[LVM_PRED_READ ]    FIRMWARE_READ(0x8000006c);
[LVM_PRED_READ ]
[LVM_PRED_READ ]    ADDR          VECTOR   FIELD_PATH             ACCESS  RESET    BEFORE   MIRRORED   VOLATILE
[LVM_PRED_READ ]    32'h10000000  [31:24]  urm.control_0_00.B3    RW       8'h0     8'hdc    8'h87      0  <------
[LVM_PRED_READ ]    32'h10000000  [23:16]  urm.control_0_00.B2    RW       8'h0     8'h6d    8'hfe      0  <------
[LVM_PRED_READ ]    32'h10000000  [15: 8]  urm.control_0_00.B1    RW       8'h0     8'h0     8'h77      0  <------
[LVM_PRED_READ ]    32'h10000000  [ 7: 0]  urm.control_0_00.B0    RW       8'h0     8'h12    8'h61      0  <------
[LVM_PRED_READ ]
```

1. before != updated if the read transaction is not launched from register model
2. UVM code
3. Address for the register
4. Original mirrored value before the transaction
5. New Mirrored Value after the transaction
6. Arrow to highlight fields that get changed in this transaction
7. Total prediction number (example above is no. 4)

Before AND After of RAL Predictor

AFTER LVM RAL Predictor

Write

```
[LVM_PRED_WRITE] 8) Observed WRITE to urm.control_0_18 : was 'hd641b62d  incoming='h9ff8 (byte_en='h3)  updated='hd6419ff8
[LVM_PRED_WRITE]
[LVM_PRED_WRITE]     urm.control_0_18.B1  .set( 8'h9f      ); // before= 8'hb6
[LVM_PRED_WRITE]     urm.control_0_18.B0  .set( 8'hf8      ); // before= 8'h2d
[LVM_PRED_WRITE]     urm.control_0_18.update(status);
[LVM_PRED_WRITE]
[LVM_PRED_WRITE]     FIRMWARE_WRITE(0x10000018, 0xd6419ff8);
[LVM_PRED_WRITE]
[LVM_PRED_WRITE]     ADDR          VECTOR   FIELD_PATH             ACCESS  RESET      BEFORE     MIRRORED     VOLATILE
[LVM_PRED_WRITE]     32'h10000018  [31:24]  urm.control_0_18.B3    RW       8'h0       8'hd6      8'hd6        0
[LVM_PRED_WRITE]     32'h10000018  [23:16]  urm.control_0_18.B2    RW       8'h0       8'h41      8'h41        0
[LVM_PRED_WRITE]     32'h10000018  [15: 8]  urm.control_0_18.B1    RW       8'h0       8'hb6      8'h9f        0  <------
[LVM_PRED_WRITE]     32'h10000018  [ 7: 0]  urm.control_0_18.B0    RW       8'h0       8'h2d      8'hf8        0  <------
[LVM_PRED_WRITE]
```

1. Final updated value
2. UVM code
3. Address for the register
4. Original value before the transaction
5. New Mirrored Value after the transaction
6. Arrow to highlight fields that get changed in this transaction
7. Total prediction number (example above is no 8)

LVM RAL Predictor Verbosity Control

User may configure the verbosity of the print out, where

Controlled by `LVM_PRD_SUMMARY_VERBOSITY

```
[LVM_PRED_READ ] 8) Observed READ from urm.control_0_00 : was 'hdc6d0012  updated='h87fe7761
[LVM_PRED_READ ]
[LVM_PRED_READ ]     urm.control_0_00.read  (status, reg_data);
[LVM_PRED_READ ]
[LVM_PRED_READ ]     FIRMWARE_READ(0x8000006c);
[LVM_PRED_READ ]
[LVM_PRED_READ ]     ADDR          VECTOR   FIELD_PATH              ACCESS  RESET         BEFORE        MIRRORED      VOLATILE
[LVM_PRED_READ ]     32'h10000000  [31:24]  urm.control_0_00.B3    RW       8'h0          8'hdc         8'h87         0  <------
[LVM_PRED_READ ]     32'h10000000  [23:16]  urm.control_0_00.B2    RW       8'h0          8'h6d         8'hfe         0  <------
[LVM_PRED_READ ]     32'h10000000  [15: 8]  urm.control_0_00.B1    RW       8'h0          8'h0          8'h77         0  <------
[LVM_PRED_READ ]     32'h10000000  [ 7: 0]  urm.control_0_00.B0    RW       8'h0          8'h12         8'h61         0  <------
[LVM_PRED_READ ]
```

Controlled by `LVM_PRD_DETAIL_VERBOSITY

```
+define+LVM_PRD_SUMMARY_VERBOSITY=UVM_HIGH
+define+LVM_PRD_DETAIL_VERBOSITY=UVM_HIGH
```

LVM RAL Predictor Verbosity Control

Meanwhile, the RAL code and FIRMWARE code can be turned OFF totally as well

Can be turned OFF by +define+LVM_PRD_RAL_CODE=0

Can be turned OFF by +define+LVM_PRD_FIRMWARE=0

```
[LVM_PRED_READ ] 8) Observed READ from urm.control_0_00 : was 'hdc6                updated='h87fe7761
[LVM_PRED_READ ]
[LVM_PRED_READ ]    urm.control_0_00.read  (status, reg_data);
[LVM_PRED_READ ]
[LVM_PRED_READ ]    FIRMWARE_READ(0x8000006c);
[LVM_PRED_READ ]
[LVM_PRED_READ ]      ADDR          VECTOR   FIELD_PATH              ACCESS  RESET        BEFORE        MIRRORED      VOLATILE
[LVM_PRED_READ ]      32'h10000000  [31:24]  urm.control_0_00.B3    RW       8'h0         8'hdc         8'h87         0  <------
[LVM_PRED_READ ]      32'h10000000  [23:16]  urm.control_0_00.B2    RW       8'h0         8'h6d         8'hfe         0  <------
[LVM_PRED_READ ]      32'h10000000  [15: 8]  urm.control_0_00.B1    RW       8'h0         8'h0          8'h77         0  <------
[LVM_PRED_READ ]      32'h10000000  [ 7: 0]  urm.control_0_00.B0    RW       8'h0         8'h12         8'h61         0  <------
[LVM_PRED_READ ]
```

LVM RAL Predictor Verbosity Control

Now as default, when the read is reading the same value like previously mirrored value, then the print is simplified to:

```
[ LVM_PRED_READ  ] 322) Observed READ from urm_top.l_urm[4].control_0c : value='hcda52b76 <Unchanged>
```

User can always revert to full print using:

<path to prd>.unchanged_no_print = 1'b0; **// 0 means switch off the trim mode.**

Benefits from the Log

1. **Quick grep of field writing history**
   This grep cmd can quickly let you know the last written field value over time

```
grep urm.control_4.B3_2   <your log file> | grep '<-----'

UVM_INFO @ 3070.000 ns: uvm_test_top.m_ahb_env.prd   [LVM_PRED_WRITE]     32'h1000005c  [31:16]  urm.control_4.B3_2     RW      16'h0         16'h0         16'hd2d0      0  <------
UVM_INFO @ 31690.000 ns: uvm_test_top.m_ahb_env.prd  [LVM_PRED_WRITE]     32'h1000005c  [31:16]  urm.control_4.B3_2     RW      16'h0         16'hd2d0      16'h5555      0  <------
UVM_INFO @ 32330.000 ns: uvm_test_top.m_ahb_env.prd  [LVM_PRED_WRITE]     32'h1000005c  [31:16]  urm.control_4.B3_2     RW      16'h0         16'h5555      16'h1111      0  <------
UVM_INFO @ 33710.000 ns: uvm_test_top.m_ahb_env.prd  [LVM_PRED_WRITE]     32'h1000005c  [31:16]  urm.control_4.B3_2     RW      16'h0         16'h1111      16'h5555      0  <------
UVM_INFO @ 37290.000 ns: uvm_test_top.m_ahb_env.prd  [LVM_PRED_WRITE]     32'h1000005c  [31:16]  urm.control_4.B3_2     RW      16'h0         16'h5555      16'h8888      0  <------
UVM_INFO @ 109590.000 ns: uvm_test_top.m_ahb_env.prd [LVM_PRED_WRITE]     32'h1000005c  [31:16]  urm.control_4.B3_2     RW      16'h0         16'h8888      16'h4444      0  <------
UVM_INFO @ 109950.000 ns: uvm_test_top.m_ahb_env.prd [LVM_PRED_WRITE]     32'h1000005c  [31:16]  urm.control_4.B3_2     RW      16'h0         16'h4444      16'h6666      0  <------
```

Benefits from the Log

2. Upgrade your code to register model style:
By referring to each PREDICTOR activity, the code here can be directly used:

```
[LVM_PRED_WRITE]
[LVM_PRED_WRITE] 88) Observed WRITE to urm.control_0_00 : was 'h136111ad  incoming='h37 (byte_en='h1)  updated='h13611137
[LVM_PRED_WRITE]
[LVM_PRED_WRITE]     urm.control_0_00.B0  .set( 8'h37      ); // before= 8'had
[LVM_PRED_WRITE]     urm.control_0_00.update(status);

[LVM_PRED_WRITE]


[LVM_PRED_READ ]
[LVM_PRED_READ ] Observed READ from urm.control_0_00 : was 'h13611137  updated='h13611137
[LVM_PRED_READ ]
[LVM_PRED_READ ]     urm.control_0_00.read  (status, reg_data);
```

Benefits from the Log

3. **Look out for code that is applicable for C for firmware:**
   By referring to each "FIRMWARE_READ" or “FIRMWARE_WRITE” code can be modified and used:

```
[LVM_PRED_READ ]     FIRMWARE_READ(0x8000001c);
[LVM_PRED_READ ]     FIRMWARE_READ(0x8000001c);
[LVM_PRED_READ ]     FIRMWARE_READ(0x8000001c);
[LVM_PRED_READ ]     FIRMWARE_READ(0x8000001c);
[LVM_PRED_WRITE]     FIRMWARE_WRITE(0x8000002c, 0xfd61cc47);
[LVM_PRED_WRITE]     FIRMWARE_WRITE(0x8000002c, 0xfd61cc47);
[LVM_PRED_WRITE]     FIRMWARE_WRITE(0x8000002c, 0xfd61cc47);
```

RAL Printing

1. LVM encapsulates the API as well:

```
[LVM_MAP_PRINT] Total registers          <env>.cfg.rseq.size     = 160
[LVM_MAP_PRINT] First register address   <env>.cfg.rseq.min_addr = 32'h10000800
[LVM_MAP_PRINT] Last  register address   <env>.cfg.rseq.max_addr = 32'h10000a7c
[LVM_MAP_PRINT] Full  register array     <env>.cfg.rseq.regs
[LVM_MAP_PRINT] ================================================================================================
[LVM_MAP_PRINT] Addr=32'h10000800   Desired=32'h0   Mirrored=32'h0   Reset=32'h0
[LVM_MAP_PRINT] ------------------------------------------------------------------------------------------------
[LVM_MAP_PRINT]     ADDR          VECTOR   FIELD_PATH                                ACCESS  RESET    BEFORE    MIRRORED
[LVM_MAP_PRINT]     32'h10000800  [31:24]  urm_top.l_urm[0].control_0_00.B3          RW      8'h0     8'h0      8'h0
[LVM_MAP_PRINT]     32'h10000800  [23:16]  urm_top.l_urm[0].control_0_00.B2          RW      8'h0     8'h0      8'h0
[LVM_MAP_PRINT]     32'h10000800  [15: 8]  urm_top.l_urm[0].control_0_00.B1          RW      8'h0     8'h0      8'h0
[LVM_MAP_PRINT]     32'h10000800  [ 7: 0]  urm_top.l_urm[0].control_0_00.B0          RW      8'h0     8'h0      8'h0
[LVM_MAP_PRINT] ================================================================================================
[LVM_MAP_PRINT] Addr=32'h10000804   Desired=32'h0   Mirrored=32'h0   Reset=32'h0
```

2. User just need to

Ral's map

```
<predictor path>.map_print(.my_map(urm_top.default_map));
```

Steps to Integrate LVM RAL Predictor

Scenario 1: **If you are already using LVM VIPs**, no action is needed.

Scenario 2: Else **if you already instantiating the uvm_reg_predictor in your UVC**

Step 1: Look for class that instantiate the uvm_reg_predictor, change it to lvm_prd

```
typedef uvm_reg_predictor#(<your seq item class>)    <Your PRD type name>; // before
typedef lvm_prd          #(<your seq item class>)    <Your PRD type name>; // after
```

Step 2: Look for package that include the class at step above and add lvm_prd.svp into your uvc package in the step above, for example:

```
package <your package>;
    ...
    `include "lvm_prd.svp"
    `include "<your class that instantiate uvm_reg_predictor>"
    ...
endpackage
```

Steps to Integrate LVM RAL Predictor

Scenario 3: Else **if you have register model in your TB, have ral adaptor but not using predictor yet**

Step 1: Look for package that include the class at step before.

Step 2: Add lvm_prd.svp into your uvc package in the step above, for example:

```
package <your package>;
    …
    `include "lvm_prd.svp"
    `include "<your class that instantiate uvm_reg_adaptor>"
    …
endpackage
```

Steps to Integrate LVM RAL Predictor

Scenario 3: Else **if you have register model in your TB, have ral adaptor but not using predictor yet**

Step 3: Look for a class that suitable to instantiate the predictor lvm_prd, add the following codes at respective UVM phases

Pls use the seq item type matching your monitor

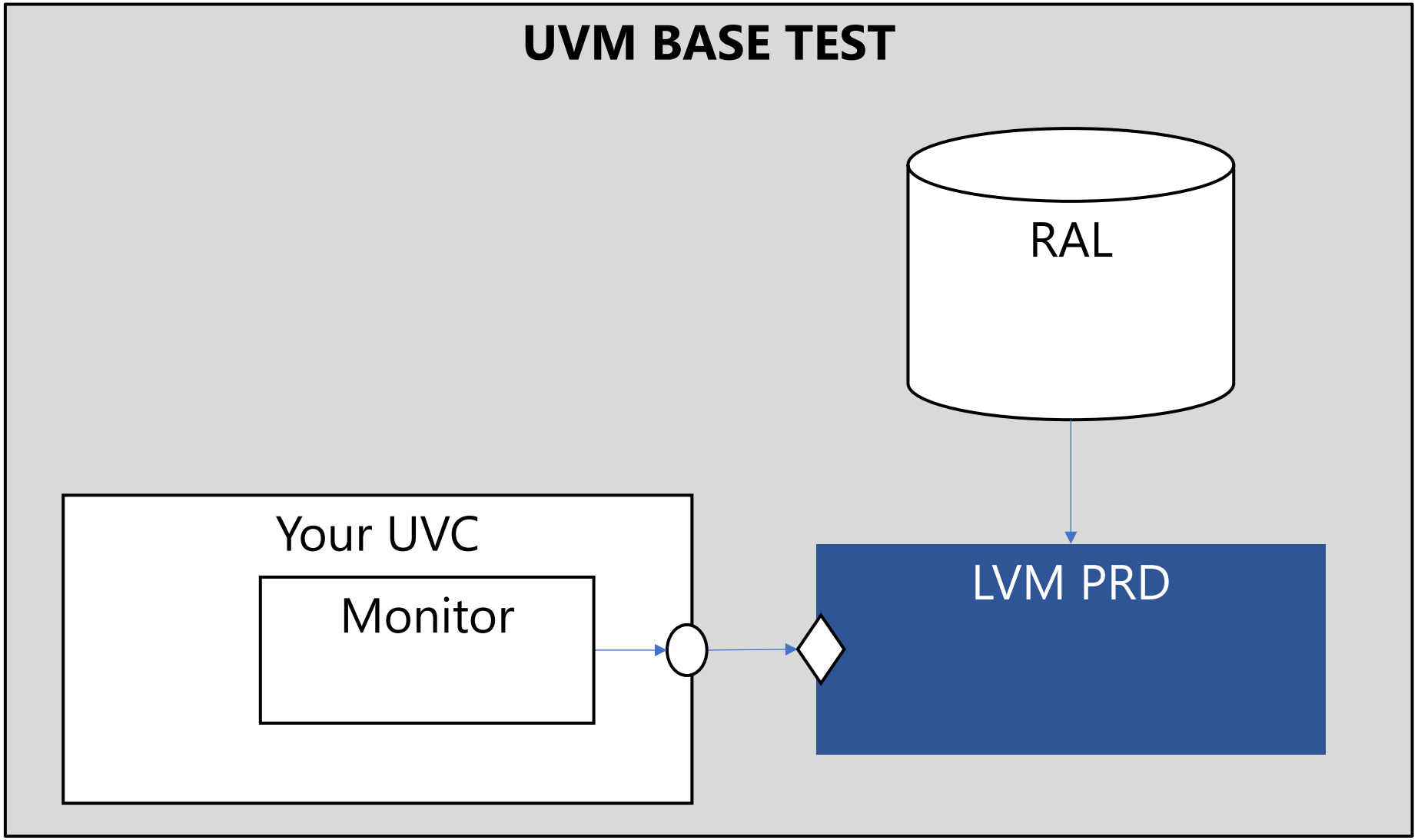
```
typedef lvm_prd#(<your seq item class>)     <Your PRD type name>;
<Your PRD type name>                        prd;         // UVC reg predictor

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        prd = <Your PRD type name>::type_id::create("prd", this);
        ...
    endfunction

    virtual function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        prd.adapter = <your ral adaptor>;
        prd.map     = <your ral>.default_map;
        <your UVC monitor port>.connect(prd.bus_in);
        ...
    endfunction
```

Monitor with seq item type matching your lvm_prd's seq item type

Example LVM RAL Predictor INSTALLATION



Terms and Conditions

IMPORTANT NOTICE

LVM reserves the right to make changes without further notice to any product or specifications herein.

LVM does not assume any responsibility for use of any its products for any particular purpose, nor does LVM assume any liability arising out of the application or use of any its products.

About LVM VIP

1. At LVM, our mission is to enhance the performance and efficiency of design verification work in the industry by providing ultra-high-quality, low-cost VIPs. We offer a range of AMBA VIPs and serial VIPs, including:

   AXI4 / AXI4-LITE, AHB, APB, TCM, USART, JTAG, AXI Stream

2. To learn more about our services, please contact us at

   developer@lvmvip.com.

3. You can also visit our LinkedIn page at
   https://www.linkedin.com/in/lvm-vip-3444b21b5/