Evaluate Video Performance Using UVM Environment

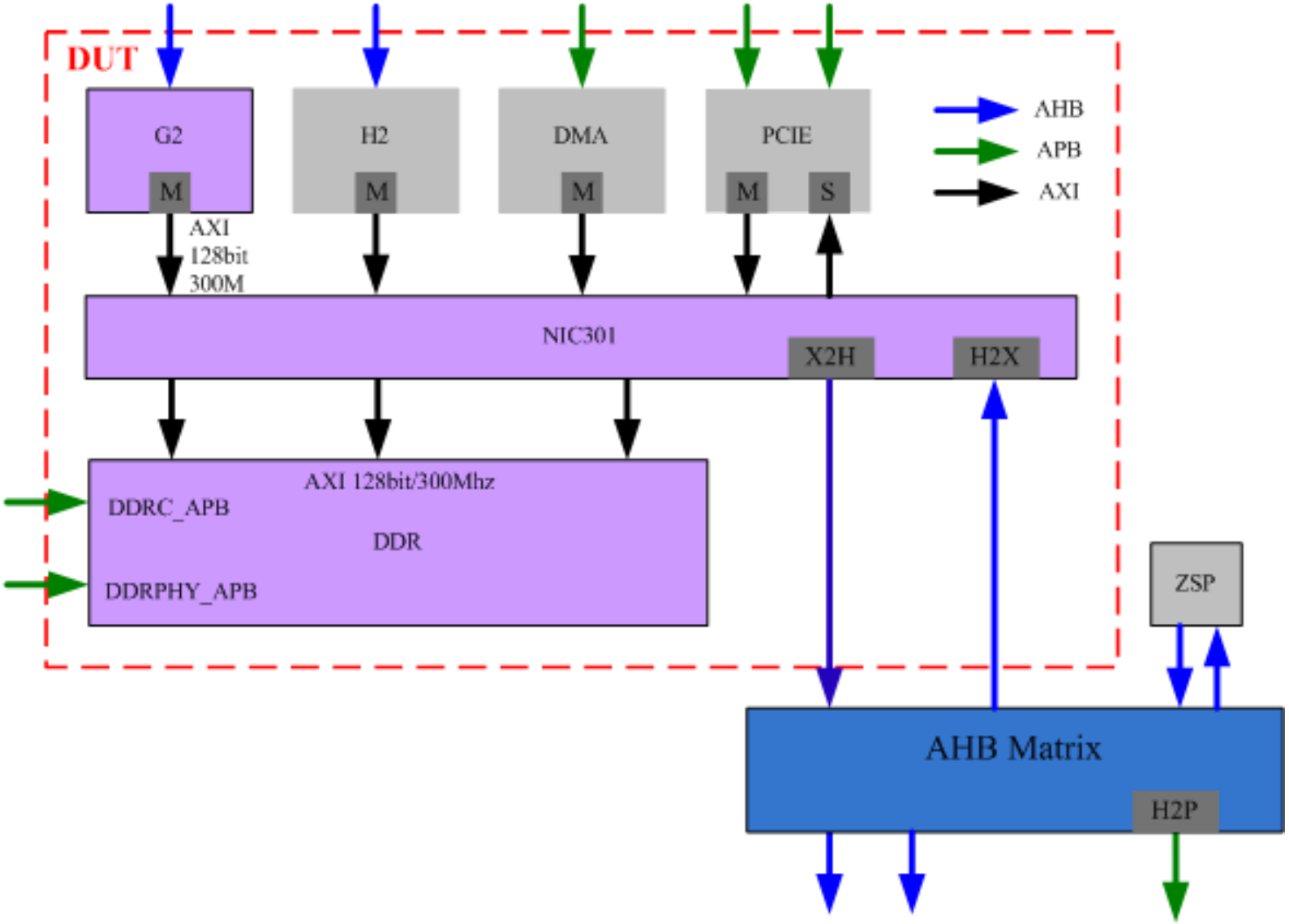
Chunlei Kang

Chunlei.Kang@verisilicon.com

Agenda

- Project Introduction
- Hantro Stimulus File Translation
- DDR Initialization and Backdoor Access

Our DUT


DUT
G2
M
H2
M
DMA
M
PCIE
M
S
AHB
APB
AXI
AXI
128bit
300M
NIC301
X2H
H2X
AXI 128bit/300Mhz
DDRC_APB
DDR
DDRPHY_APB
ZSP
AHB Matrix
H2P

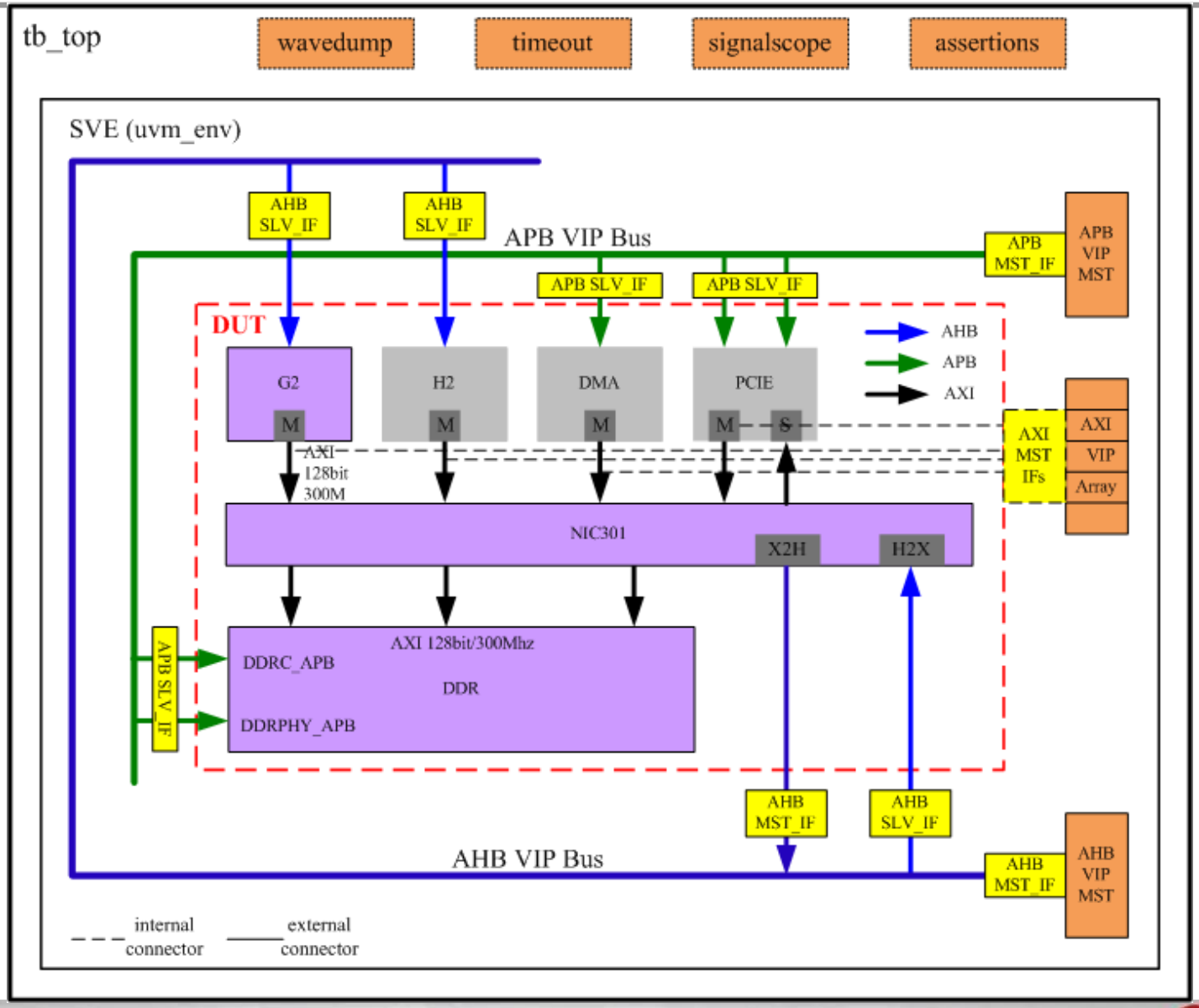
Our Inputs

- DUT
  - AMBA DesignWare
  - Synopsys DDR2 controller and Phy
  - Verisilicon Hantro (G2) Video decoder
- Materials for Test Bench
  - Synopsys discovery AMBA VIP
  - Micron Low power DDR2 verilog model
  - DDR initialization ASM code
  - G2 Command and stimulus file

Our Goals

- G2 performance evaluation
  - With NIC301/AXI bus matrix
  - With Synopsys DDR2 controller
- Generic DV flow to
  - Adopt command file and stream file popular in the video/2D/3D environment
  - Transport bench from IP to system level
  - Unify the software and SV stimulus appearance
    - ASM/C like
  - Build AMBA subsystem DV environment
    - create AHB and APB bus by AMBA VIP
    - trim the system easily by “stub and force”

Test Bench Diagram



Generic Test Bench Elements

- WDUT is wrapped DUT which includes:
  - External interface
  - DUT
- iConnector
  - Internal connector
    - Connect internal design signals by “force” and “stub”
- Simulation Control module
  - Timeout
  - Wavedump
  - Signalscope

Recommended Way for iconnector

- STUB
  - with emacs
- Using force
  - cmd option

```
module axiahbg2dec(/*AUTOARG*/) ;

   /*AUTOINOUTMODULE("axiahbg2dec")*/

   wire                 HREADYoutS_g2s
   wire                 BREADY_g2m
   wire                 RREADY_g2m

   /*AUTOTIEOFF*/
endmodule // axiahbg2dec

// Local Variables:
// verilog-library-flags:("-y . -y ../rtl")
// End:
```

```
`include "svt_axi_master_if.svi"
`include "top_defines.vh"
module g2_iconnector(svt_axi_master_if g2_axi_m_IF);
   parameter IF_TIME=1;
   ///*AUTOINOUTMODULE("axiahbg2dec","^AR")*/
   ///*AUTOINOUTMODULE("axiahbg2dec","^AW")*/
   ///*AUTOINOUTMODULE("axiahbg2dec","^W")*/
   ///*AUTOINOUTMODULE("axiahbg2dec","^R")*/
   ///*AUTOINOUTMODULE("axiahbg2dec","^B")*/
   initial begin
      if($test$plusargs("BFM_MASTER_G2")) begin
         force `AXIAHBG2DEC_HIER.ARID_g2m=      g2_axi_m_IF.arid;
         force `AXIAHBG2DEC_HIER.ARADDR_g2m=    g2_axi_m_IF.araddr;
         ....
         force `AXIAHBG2DEC_HIER.WVALID_g2m=    g2_axi_m_IF.wvalid;
         force `AXIAHBG2DEC_HIER.RREADY_g2m=    g2_axi_m_IF.rready;
         force `AXIAHBG2DEC_HIER.BREADY_g2m=    g2_axi_m_IF.bready;
      end
   end

   assign #(IF_TIME) g2_axi_m_IF.arready=  `AXIAHBG2DEC_HIER.ARREADY_g2m;
   assign #(IF_TIME) g2_axi_m_IF.awready=  `AXIAHBG2DEC_HIER.AWREADY_g2m;
   assign #(IF_TIME) g2_axi_m_IF.wready=   `AXIAHBG2DEC_HIER.WREADY_g2m;
   assign #(IF_TIME) g2_axi_m_IF.rid=    `AXIAHBG2DEC_HIER.RID_g2m;
   assign #(IF_TIME) g2_axi_m_IF.rdata=    `AXIAHBG2DEC_HIER.RDATA_g2m;
   assign #(IF_TIME) g2_axi_m_IF.rresp=    `AXIAHBG2DEC_HIER.RRESP_g2m;
   assign #(IF_TIME) g2_axi_m_IF.rlast=    `AXIAHBG2DEC_HIER.RLAST_g2m;
   assign #(IF_TIME) g2_axi_m_IF.rvalid=   `AXIAHBG2DEC_HIER.RVALID_g2m;
   assign #(IF_TIME) g2_axi_m_IF.bid=    `AXIAHBG2DEC_HIER.BID_g2m;
   assign #(IF_TIME) g2_axi_m_IF.bresp=    `AXIAHBG2DEC_HIER.BRESP_g2m;
   assign #(IF_TIME) g2_axi_m_IF.bvalid=   `AXIAHBG2DEC_HIER.BVALID_g2m;

endmodule

// Local Variables:
// verilog-library-directories:("../../../../../../prj/ip/axiahbg2dec/stub/")
// End:
```

Bus VIP or Master VIP?

- AHB/APB Bus VIP
  - More reasonable
  - Easy to maintain
- AXI Master VIP
- Unify the register read/write operation

```
initial begin
   force `DUT_HIER.g2_ahb_haddr          =g2_ahb_s_IF.haddr;
   force `DUT_HIER.g2_ahb_hready         =g2_ahb_s_IF.hready_in;
   force `DUT_HIER.g2_ahb_hsel           =g2_ahb_s_IF.hsel;
   force `DUT_HIER.g2_ahb_hsize          =g2_ahb_s_IF.hsize;
   force `DUT_HIER.g2_ahb_htrans         =g2_ahb_s_IF.htrans;
   force `DUT_HIER.g2_ahb_hwdata         =g2_ahb_s_IF.hwdata;
   force `DUT_HIER.g2_ahb_hwrite         =g2_ahb_s_IF.hwrite;
end

assign #(IF_TIME) g2_ahb_s_IF.hrdata            =`DUT_HIER.g2_ahb_hrdata;
assign #(IF_TIME) g2_ahb_s_IF.hready            =`DUT_HIER.g2_ahb_hready_resp;
assign #(IF_TIME) g2_ahb_s_IF.hresp             =`DUT_HIER.g2_ahb_hresp;
```

```
virtual task WRITE(input bit[31:0] DesAddr, input bit[31:0] Value);
   if(DesAddr[31:0]>=proj_configuration::DDR_START && DesAddr[31:0]<=proj_configuration::PCIE_END) begin
      axi.write(DesAddr,Value);
   end
   else if(DesAddr[31:0]>=proj_configuration::G2_AHB_START && DesAddr[31:0]<=proj_configuration::H2_AHB_END)begin
      ahb.write(DesAddr,Value);
   end
   else if(DesAddr[31:0]>=proj_configuration::DDRC_APB_START && DesAddr[31:0]<=proj_configuration::A5CH2CH_APB_END) begin
      apb.write(DesAddr,Value);
   end
   else begin
      `uvm_fatal("WRITE_MACRO",$psprintf("Invalid Write Address:addr=%h,data=%h",DesAddr,Value));
      return;
   end
   `uvm_info($psprintf("%s.WRITE_MACRO",get_name()),$psprintf("Address=%h,Write Data=%h",DesAddr,Value),UVM_MEDIUM);
endtask
```

Agenda

- Project Introduction
- **Hantro Stimulus File Translation**
- DDR Initialization and Backdoor Access

Hantro(G2) stimulus

Test Data: simulation_ctrl_unmapped.tr

```
C -------------------------------------------------------
C -- Decoding picture 0, case 0
C --   Size 320 x 256, buffer 0
C -------------------------------------------------------
C Write external memory tables
WRITE_DEC_TILES BASE_DEC_TILES 16
C Write stream data
WRITE_DEC_INDATA BASE_DEC_INDATA 8128
C SW sets up hardware
W swreg1/00000000
...
W swreg64/00000000
W swreg65/BASE_DEC_DPB+0
...
W swreg77/BASE_DEC_DPB+132400
...
C Enabling decoder hardware
W swreg1/00000001
TIMER_START
C Waiting for PIC_READY interrupt
B swreg1/00001100 POLL_CYCLE TB_TIMEOUT
TIMER_END 320
C Reading stream end address
R swreg169/BASE_DEC_INDATA+00001FBA
C Checking output picture
CHECK_DEC_PIC_FRM_Y BASE_DEC_DPB+0 81920
CHECK_DEC_PIC_FRM_C BASE_DEC_DPB+163840 40960
C Checking direct mode motion vectors
CHECK_DEC_DIRMV BASE_DEC_DPB+245760 5120
```

Expected Hantro Sequence

- Hantro sequence
  - simple
  - reusable
  - extendable

```
`ifndef GUARD_G2_320X256_SEQUENCE_SV
`define GUARD_G2_320X256_SEQUENCE_SV

`include "proj_hantro_base_sequence.sv"
class g2_320x256_sequence extends proj_hantro_base_sequence;

   `uvm_object_utils(g2_320x256_sequence)

   function new(string name="g2_320x256_sequence");
      super.new(name);
   endfunction

   `include "320x256/g2_map.vh"
   parameter POLL_CYCLE=1000;
   parameter TB_TIMEOUT=0;
   virtual task body();

      `include "320x256/g2_map_ctrl.sv"
   endtask // body


endclass // g2_320x256_sequence

`endif
```

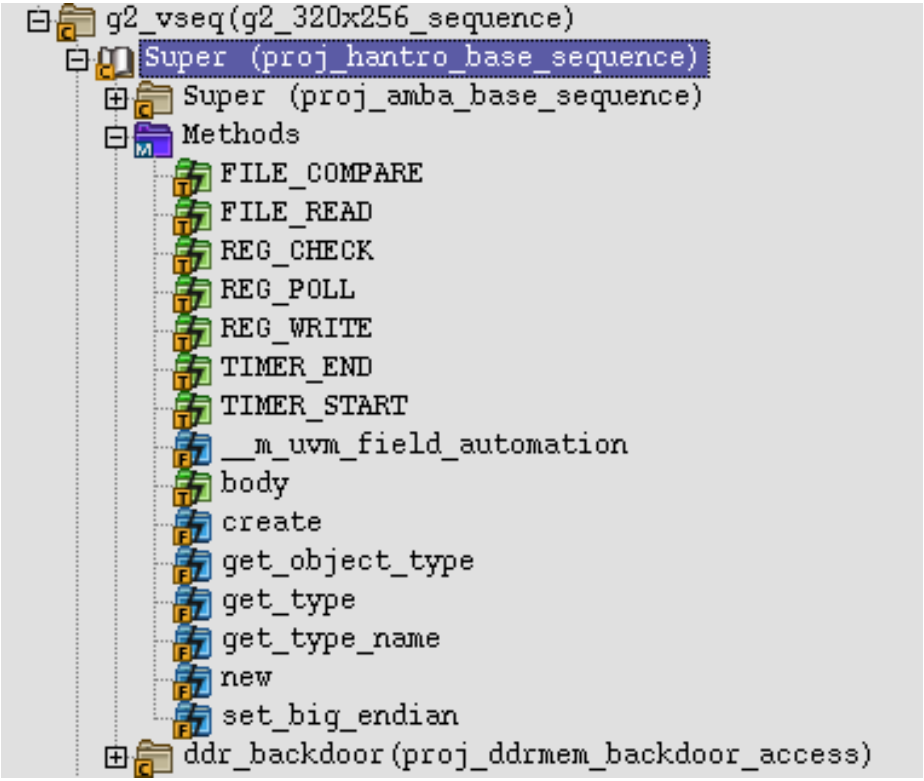
Included File in Hantro Sequence

- Perl script translated test data into

```
$display("------------------------------------------------------");
$display("-- Decoding picture 0, case 0");
$display("--   Size 320 x 256, buffer 0");
$display("------------------------------------------------------");
$display("Write external memory tables");
FILE_READ(BASE_DEC_TILES, 32'h00000010, "g2_input_tiles.bin");
$display("Write stream data");
FILE_READ(BASE_DEC_INDATA, 32'h00001fc0, "g2_input_strm.bin");
$display("SW sets up hardware");
REG_WRITE(swreg1, 32'h00000000);
$display("Enabling decoder hardware");
REG_WRITE(swreg1, 32'h00000001);
TIMER_START;
$display("Waiting for PIC_READY interrupt");
REG_POLL(swreg1, 32'h00001100, POLL_CYCLE, TB_TIMEOUT);
TIMER_END(320);
$display("Reading stream end address");
REG_CHECK(swreg169, BASE_DEC_INDATA+32'h00001FBA);
$display("Checking output picture");
FILE_COMPARE(BASE_DEC_DPB+32'h00000000, 32'h00014000, "g2_output_dec.bin");
FILE_COMPARE(BASE_DEC_DPB+32'h00014000, 32'h0000a000, "g2_output_dec.bin");
$display("Checking direct mode motion vectors");
FILE_COMPARE(BASE_DEC_DPB+32'h0001e000, 32'h00001400, "g2_output_dir_mode_mvs.bin");
```

Hantro Base Sequence

- proj_hantro_base_sequence
  - Implements tasks in the translated file
    - FILE_COMPARE
    - FILE_READ
    - REG_CHECK
    - REG_POLL
    - TIMER_END
    - TIMER_START
  - DDR backdoor
    - READ8/32/64
    - WRITE8/32/64



Hantro Base Sequence Code Snippet

- FILE_READ
  - file seek
  - backdoor access

```
virtual task FILE_READ(input bit[31:0] addr, input bit[31:0] length,  input string filename);
   bit[7:0] data=0;
   bit[31:0] addr_t=0;
   integer filehandler;
   int i;
   if(filehandler_registry.exists(filename)) begin
      filehandler=filehandler_registry[filename];
   end
   else begin
      filehandler=$fopen(filename, "r");
      if(filehandler==0) begin
         `uvm_fatal("FILE_READ","open ERROR")
      end
      filehandler_registry[filename]=filehandler;
   end
   for (i=0;i<length;i=i+1) begin
      data[7:0] = $fgetc(filehandler);
      addr_t=addr+i;
      if(big_endian) addr_t[3:0]=~addr_t[3:0];
      ddr_backdoor.write8(addr_t,data);
   end
   `uvm_info("FILE_READ",$psprintf("Done:%s",filename),UVM_MEDIUM)
endtask
```

Agenda

- Project Introduction
- Hantro Stimulus File Translation
- **DDR Initialization and Backdoor Access**

DDR initialization asm

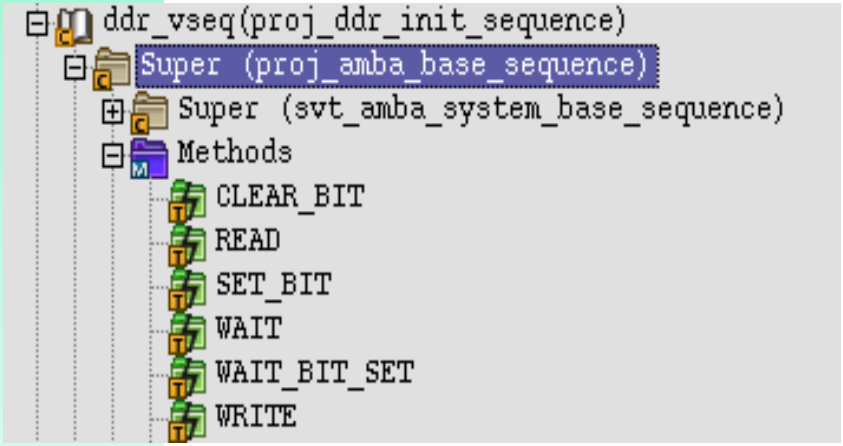
ARM assembly code

```
@Release PHY DLL reset
     SET_BIT        RSTG_BASE_ADDR, 0xc
@Release PHY Ctrl reset
     SET_BIT        RSTG_BASE_ADDR, 0xf
@Release PHY APB reset
     SET_BIT        RSTG_BASE_ADDR, 0xe

@ ---------- 1 ----------
@  Configure uMCTL2 registers
@-----------------------------------

     CLEAR_BIT        SWCTL, 0x0

@ Set DFIMISC.dfi_init_complete_en to 0
      CLEAR_BIT    DFIMISC, 0x0

      WRITE        DBG1, 0x00000001

      WRITE        MSTR, 0x03040004  @active ranks: 2 ranks
                                     @burst length: BL8
                                     @LPDDR2 DRAM type

@.......

     SET_BIT       RSTG_BASE_ADDR , 0x11

      @Wait uMCTL2 STAT[2:0] to be 3'b001 (normal mode)
      WAIT_BIT_SET 6, STAT, 0x0
```

Expected DDR Initial Sequence

- proj_ddr_init_sequence
  - asm alike code in body()
  - extended from proj_amba_base_sequence

```
`ifndef GUARD_PROJ_DDR_INIT_SEQUENCE_SV
`define GUARD_PROJ_DDR_INIT_SEQUENCE_SV
`include "top_defines.vh"
`include "proj_amba_base_sequence.sv"
class proj_ddr_init_sequence extends proj_amba_base_sequence;
  `uvm_object_utils(proj_ddr_init_sequence)
  `include "proj_mmap_params.svi"

  function new (string name = "proj_ddr_init_sequence");
    super.new(name);
  endfunction : new

  virtual task body();
       //  asm alike code translated by script, see next slide
  endtask

endclass
`endif
```



ASM Alike Code in Above Sequence

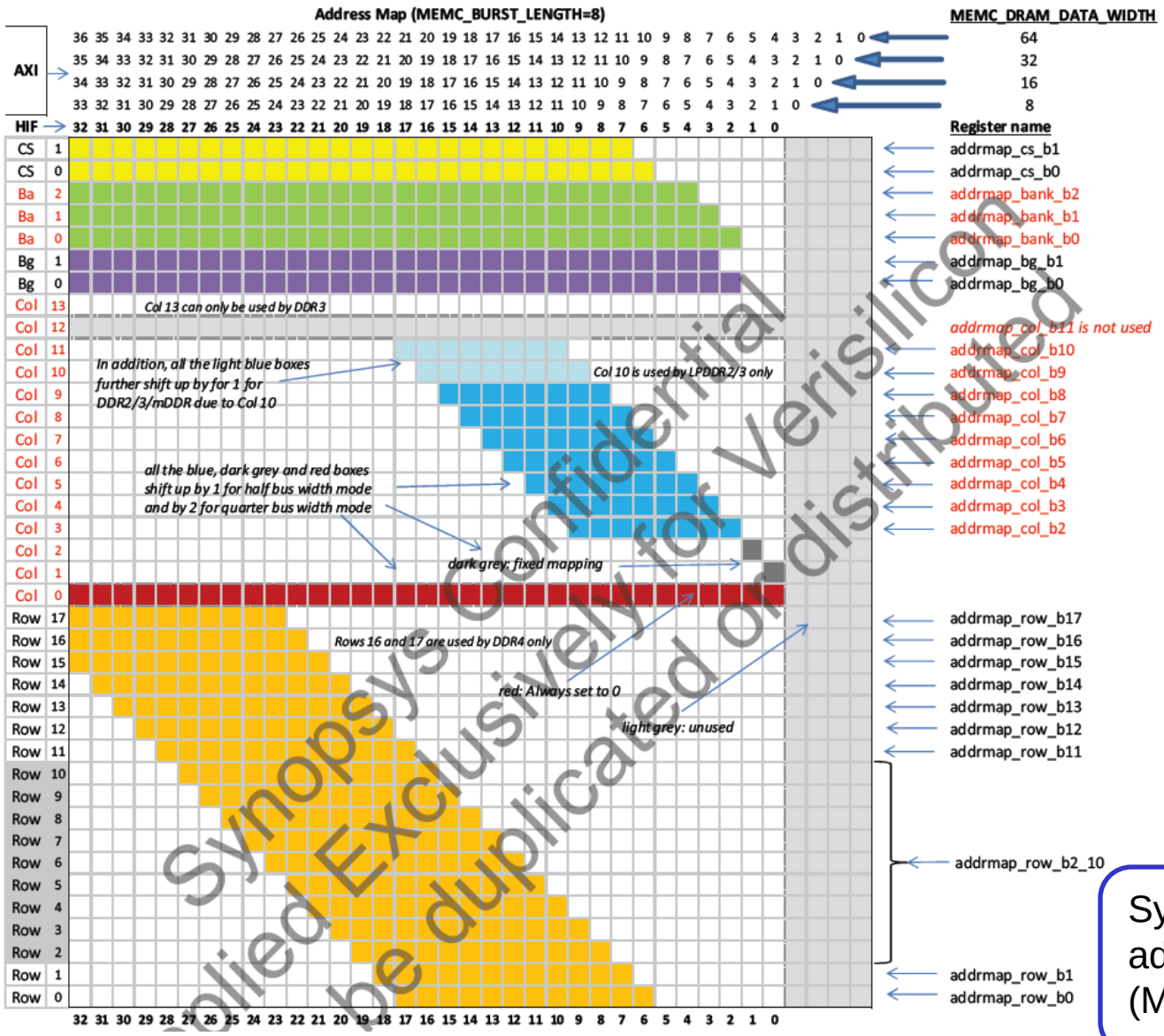
- Perl script translates the ASM code into

```
//Release PHY DLL reset
      SET_BIT(RSTG_BASE_ADDR, 32'hc);
//Release PHY Ctrl reset
      SET_BIT( RSTG_BASE_ADDR, 32'hf);
//Release PHY APB reset
      SET_BIT(RSTG_BASE_ADDR, 32'he);

// ---------- 1 ----------
//   Configure uMCTL2 registers
// ----------------------------------

      CLEAR_BIT ( SWCTL, 32'h0);

// Set DFIMISC.dfi_init_complete_en to 0
       CLEAR_BIT (DFIMISC, 32'h0);

       WRITE(DBG1, 32'h00000001);

       WRITE(MSTR, 0x03040004);  //active ranks: 2 ranks
                                 //burst length: BL8
                                 //LPDDR2 DRAM type

//.....

      SET_BIT(RSTG_BASE_ADDR , 32'h11);

       //Wait uMCTL2 STAT[2:0] to be 3'b001 (normal mode)
       WAIT_BIT_SET( 6, STAT, 32'h0);
```

Expected DDR Backdoor Access

- DDR backdoor access class
  - Store or load data without delay
    - should drain off the outstanding transaction before data load
  - Mapping AXI address into micron DDR memory cell
    - mapping function in backdoor access should change if controller changes the setting for the mapping between AXI and HIF.
    - You can prefer to reference RAL model in backdoor access class with penalty of universality

AXI and HIF Address Mapping



System address to physical address mapping (MEMC_BURST_LENGTH = 8)

AXI and DDR model Address Mapping

```
// My test mapping in this test case, for 1Gb x32 LPDDR2 (col addr:9 bits  row addr: 13 bits  bank addr: 3bits  2 ranks)
// MEMC_BURST_LENGTH = 8  {bank,row, col}
//-------------------------------------------------------------------------------------------------------------------
// AXI add: 29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13   12  11  10  9   8   7   6   5   4   3   2   1   0
//-------------------------------------------------------------------------------------------------------------------
// HIF add: 26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0   -   -   -
//-------------------------------------------------------------------------------------------------------------------
//                cs  b2   b1  b0 r12 r11 r10 r9  r8  r7  r6  r5  r4  r3  r2   r1  r0  c8  c7  c6  c5  c4  c3  c2  c1
//Int base         6  4    3   2  18  17  16  15  14  13  12  11   10  9   8   7   6   7   6    5   4   3   2   -   -
//p Value         18  19   19 19  2   2   2   2   2   2    2   2   2   2   2   2   2   0   0    0   0   0   0   ?   -
//-------------------------------------------------------------------------------------------------------------------
```

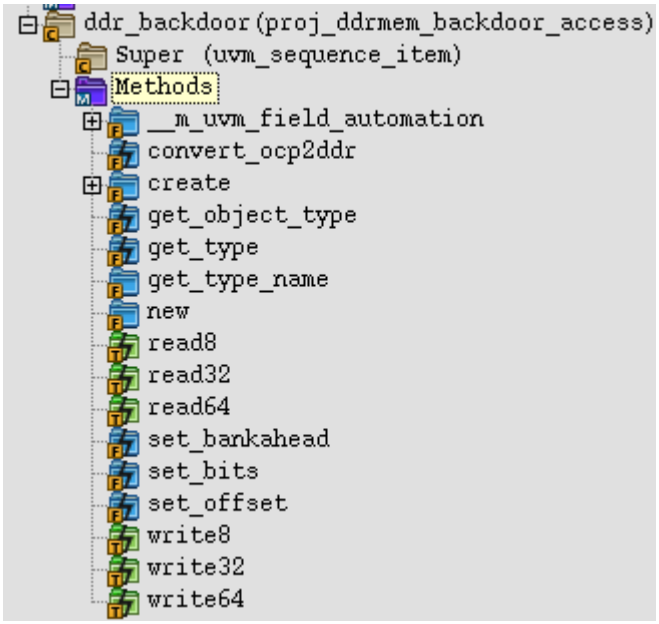
Synopsys AXI/HIF Address Map

Micron DDR2 Verilog model

```
    task memory_read;
        input  [BA_BITS-1:0]  bank;
        input  [ROW_BITS-1:0] row;
        input  [COL_BITS-1:0] col;
        output [2*DQ_BITS-1:0] data;
        reg    [`MAX_BITS-1:0] addr;
        begin
            // chop off the lowest address bits
            addr = {bank, row, col}/2;
`ifdef MAX_MEM
            data = read_from_file( memfd[bank], {row, col}/2 );
`else
            if (get_index(addr)) begin
                data = memory[memory_index];
            end else begin
                data = {2*DQ_BITS{1'bx}};
            end
`endif
        end
    endtask
```

DDR Backdoor Access Examplary Code

- convert_ocp2ddr function
- read and write task




```
virtual task read64(input bit[31:0] addr, output bit[63:0] data);
   convert_ocp2ddr(addr);
   `DDRMODEL_HIER.memory_read(ba,row,col,data);
endtask

virtual task write64(input bit[31:0] addr, input bit[63:0] data);
   convert_ocp2ddr(addr);
   `DDRMODEL_HIER.memory_write(ba,row,col,data);
endtask
```

Performance

▼Performance in IP level and Chip Level

▲Master connect to AXI SRAM directly in IP env

G2 case 1500000(1920*1080)	cycles			
(axi3 bus)	ip level	chip level	add cycles	add percentage
picutre 0	3591767	3591843	76	0%
picutre 1	1952628	2015856	63228	3.24%
picutre 2	1524088	1619805	95717	6.28%
picutre 3	1521596	1620459	98863	6.50%
picutre 4	1507443	1568802	61359	4.01%
picutre 5	1504485	1557786	53301	3.54%
picutre 6	1506811	1578612	71801	4.77%
picutre 7	1537454	1592094	54640	3.56%
picutre 8	1632880	1730877	97997	6%
picutre 9	1604841	1646787	41946	2.61%
G2 case 13002(320*256)	cycles			
(axi3 bus)	ip level	chip level	add cycles	add percentage
picutre 0	94203	94149	54	6%%
picutre 1	93854	93802	52	6%%
picutre 2	89133	89082	51	6%%
picutre 3	91926	91879	47	6%%
picutre 4	91743	91695	48	6%%
picutre 5	92893	92847	46	6%%
picutre 6	89063	89008	55	6%%
picutre 7	88752	88706	46	6%%
picutre 8	95334	95287	47	6%%
picutre 9	39287	93241	46	6%%

QA

Thanks!