Ultra Fast RTL Simulation and Verification Methodology using LGE CPU-Level AXI Virtual Master

2010-07-26

Kim, Si-Hyun

Design Technology (DT) Part
DIS Group
System IC Business Team
LG Electronics

Please send comments via email to: *sihyun1.kim@lge.com*

This documentation contain confidential and proprietary information that is the property of LG.

LGE CONFIDENTIAL

SYSTEM IC BUSINESS TEAM

Document Revision History

- This table describes the changes to Document-Based applications overview.

Version	Date	Notes
1.0	2010-07-23	Added the following new features and updates * *raxi_war*, *swDbgVm* in VM tasks * automatic register R/W test flow File name: LGE_VM_검증환경_2010_0726.ppt
0.9	2010-07-19	This is the first version of AXI VM Guide. File name: LGE_VM_검증환경_2010_0719.ppt

Agenda

- **Introduction to the LGE AXI Virtual Master (VM)**
  - VM feature
  - Customer Feed-Back
- **What's CPU level VM?**
  - Adaptive verification S/W in VM
  - Exclusive ISR execution in VM
- **Case Study**
  - VM-based UART Interrupt Test using L8 CPU platform Verification S/W
- **User Guide**

VM-based Simulation and Verification

❑ VM-based IP/RTOP/CTOP level simulation environment

- Before verifying the entire H/W-S/W system, we verify several of the RTL components with virtual bus master (virtual CPU) model in advance.
  - ✓ simplifies setup the verification environment
  - ✓ exercising corner cases using random traffic generation
  - ✓ fast simulation, ARM core is replaced by a VM
  - ✓ **easy test/debug verification S/W** NEW!

❑ LGE AHB/AXI VM is supported,

- bus transaction-level testbench
- automatic register access and data R/W check
- bus transfer-level functional coverage monitor
- **adaptive verification S/W and ISR execution** NEW!

Bus transaction-level testbench

❑ AHB/AXI transaction-based Verilog (SystemVerilog) tesetbench

- Transaction R/W task call method
  - ✓ ex. Master write data to slave via AXI and/or APB3 bus
    - – axi_w (addr, data, control, …);
    - – raxi_w(addr, data)
    - – rapb_w(addr, data)

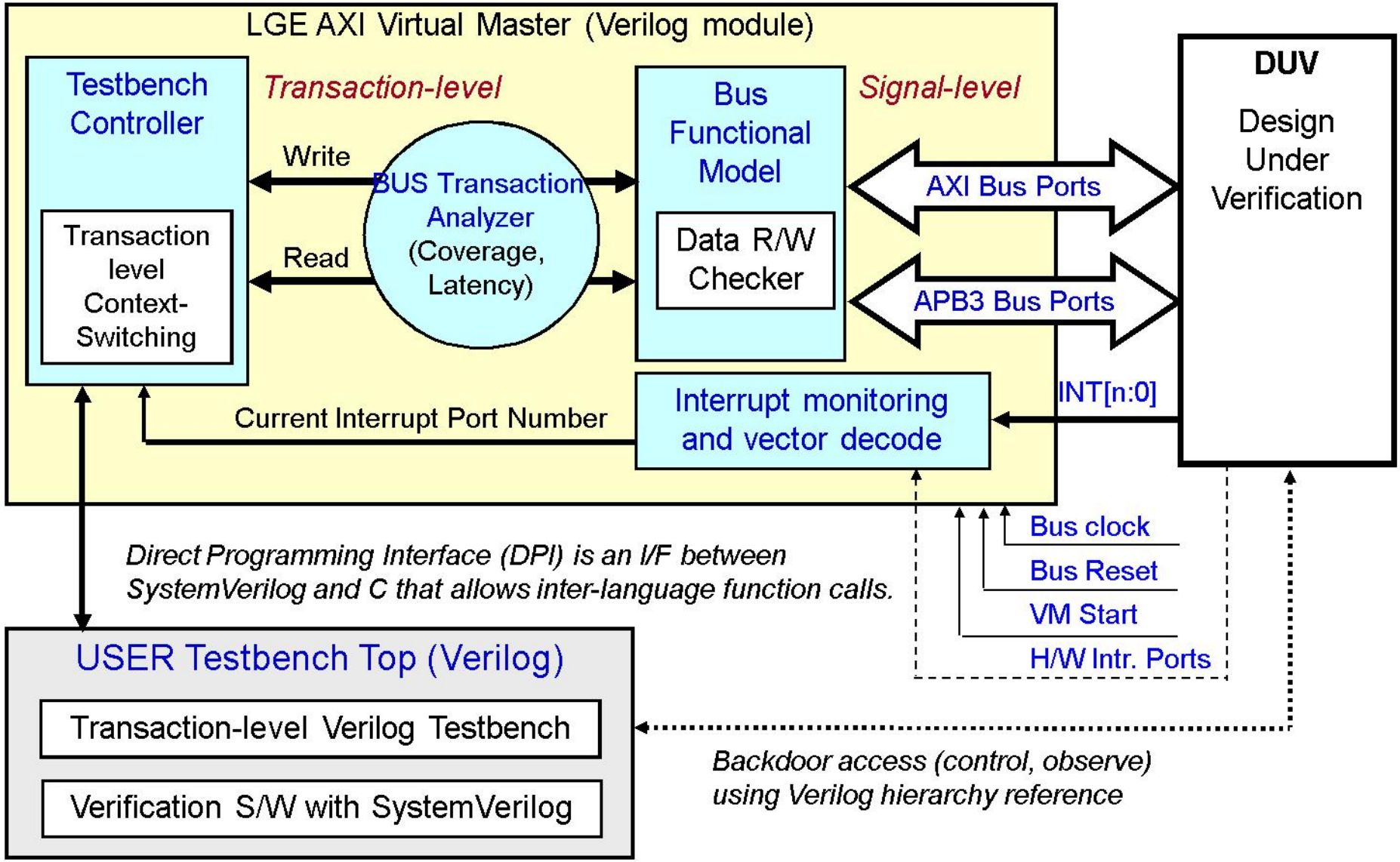
❑ Scenario-based testbench

- Configuration of transaction R/W task argument
  - ✓ Direct
  - ✓ semi-random (constraint, distribution)
  - ✓ (full)-random methods

❑ Code coverage-driven verification

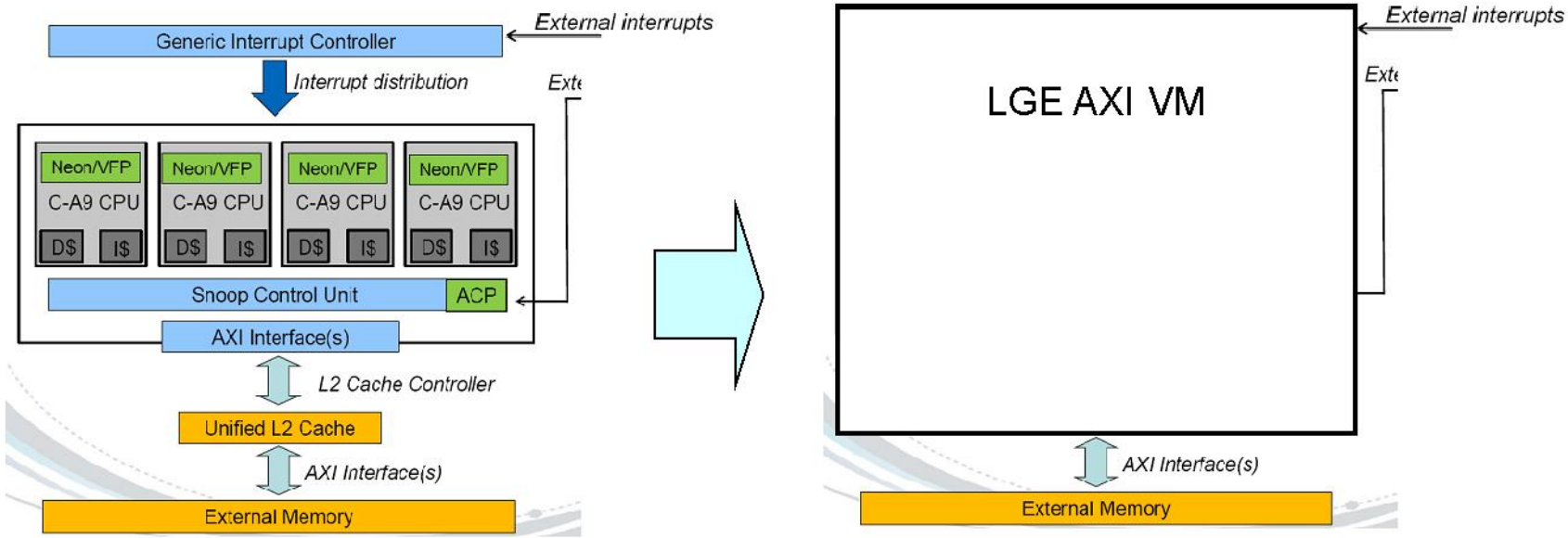
- random-constraint AHB/AXI transaction tasks

Virtual master function diagram



VM Main Transaction Tasks

Access Mode	Register *r*	Burst	Random *rand*	Exclusive *ex*	Multiple Outstanding *mot*
Read	raxi(apb)_r	axi_r	rand_axi_r	axi_rex	axi_r_mot
Write	raxi(apb)_w	axi_w	rand_axi_w	axi_wex	axi_w_mot
RAW	raxi(apb)_rw	axi_rw	rand_axi_rw	axi_rwex	N/A
WAR	raxi_war	N/A			
Bulk data transfer	*mem* memset_vm memcpy_vm memcmp_vm				
S/W code debug	swDbgVm (task for C code variable debugging)				



Ex. VM-based Testbench (1 of 2)

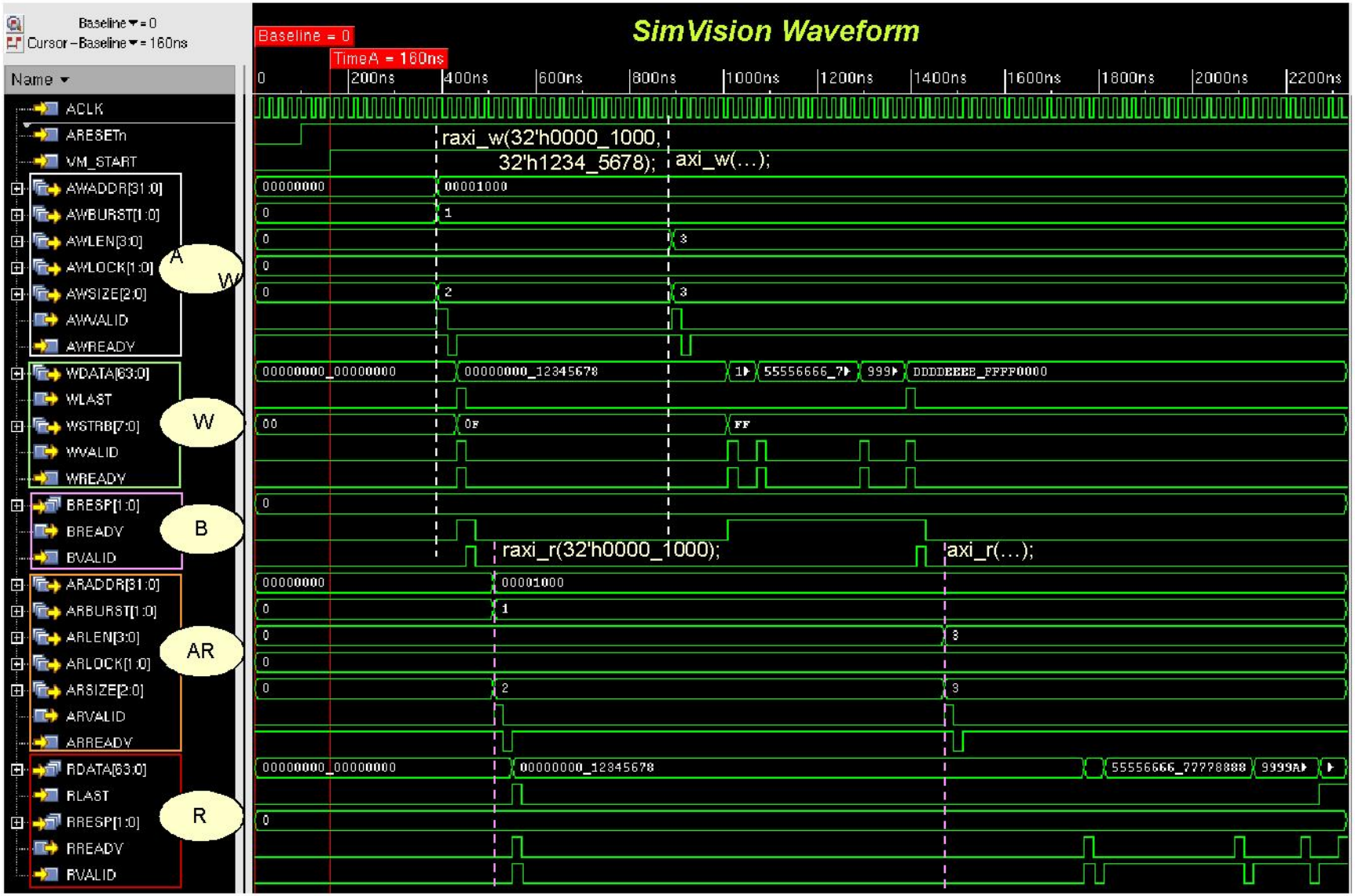
Verilog Testbench Top

```
task run_axi_vm_armd;

    $display("%0s at %0t", VM_NAME, $realtime);

    raxi_w(32'h0000_1000, 32'h1234_5678);
    raxi_r(32'h0000_1000);

    repeat(3)@(posedge ACLK);

    A_AWDATA[0] =  {64'h11112222_33334444};
    A_AWDATA[1] =  {64'h55556666_77778888};
    A_AWDATA[2] =  {64'h9999aaaa_bbbbcccc};
    A_AWDATA[3] =  {64'hdddddeeee_ffff0000};

    axi_w(
        32'h0000_1000,        // address
        `AXI_ALEN_4,          // burst length
        `AXI_ASIZE_64,        // burst size
        `AXI_ABURST_INCR,     // burst type
        `AXI_ALOCK_NOLOCK,    // atomic access
        `AXI_ACACHE_NON,      // cache
        3'd0,                 // protect
        3                     // next wait cycle
    );

    axi_r(
        32'h0000_1000,
        `AXI_ALEN_4,
        `AXI_ASIZE_64,
        `AXI_ABURST_INCR,
        `AXI_ALOCK_NOLOCK,
        `AXI_ACACHE_NON,
        3'd0,
        0
    );
endtask
```

NC-simulation log

```
ncsim> run
[LGE VM #1] START at 350 ns
[LGE VM #1] AXI VM,  WRITE Transaction at 370 ns
    AWADDR  = 00001000
    AWID    = 0
    AWLEN   = Burst 1
    AWSIZE  = WORD
    AWBURST = INCR
    AWLOCK  = 00
    AWCACHE = 0000
    AWPROR  = 001
    AWDATA  = 00000000_12345678
[LGE VM #1] AXI VM, READ Transaction at 490 ns
    ARADDR  = 00001000
    ARID    = 0
    ARLEN   = Burst 1
    ARSIZE  = WORD
    ARBURST = INCR
    ARLOCK  = 00
    ARCACHE = 0000
    ARPROT  = 001
    ARDATA  = 00000000_12345678
[LGE VM #1] AXI VM, READ Transaction at 870 ns
    AWADDR  = 00001000
    AWID    = 0
    AWLEN   = Burst 4
    AWSIZE  = 2 WORD
    AWBURST = INCR
    AWLOCK  = 00
    AWCACHE = 0000
    AWPROR  = 000
    AWDATA  =           1111222233334444
                        5555666677778888
                        9999aaaabbbbcccc
                        ddddeeeeffff0000
[LGE VM #1] AXI VM, READ Transaction at 1450 ns
    ARADDR  = 00001000
    ARID    = 0
    ARLEN   = Burst 4
    ARSIZE  = 2 WORD
    ARBURST = INCR
    ARLOCK  = 00
    ARCACHE = 0000
    ARPROT  = 000
    ARDATA  =           1111222233334444
                        5555666677778888
                        9999aaaabbbbcccc
                        ddddeeeeffff0000
Simulation complete via $finish(2) at time 2330 NS
ncsim> exit
```

Ex. VM-based Testbench (2 of 2)



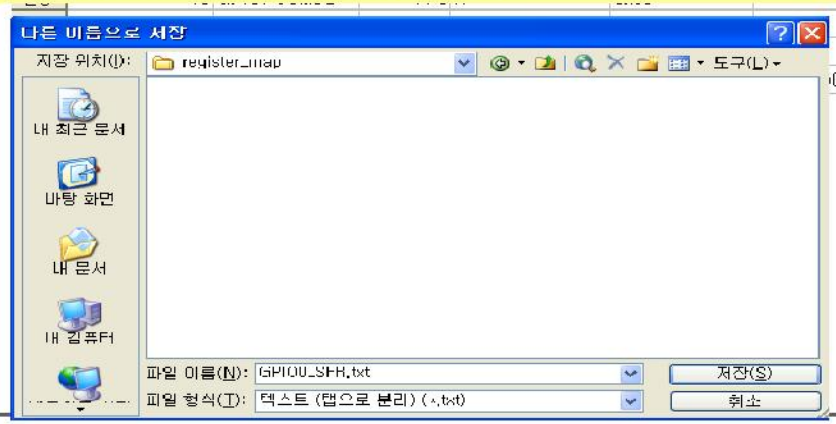
Spec-based Automatic Register Test (1 of 2)

❑ LGE AXI VM support automatic register R/W test and report.

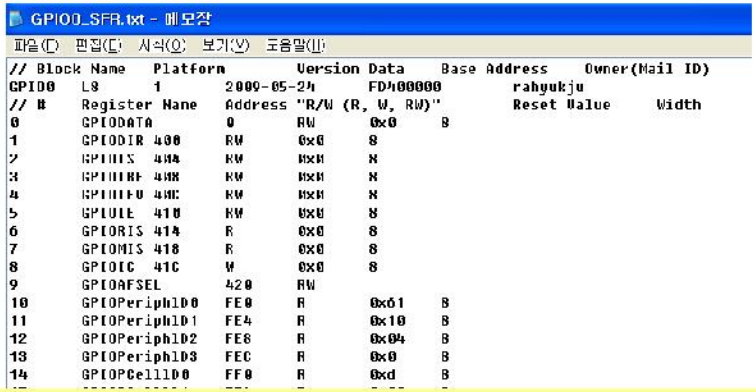
Step.1 Write IP level register map to MS Excel-file

	A	B	C	D	E	F
1	// Block Name	Platform	Version	Data	Base Address	Owner(Mail ID)
2	GPIO0	L8	1	2009-05-24	FD400000	rahyukju
3	// #	Register Name	Address	R/W (R, W, RW)	Reset Value	Width
4	0	GPIODATA	0	RW	0x0	8
5	1	GPIODIR	400	RW	0x0	8
6	2	GPIOIS	404	RW	0x0	8
7	3	GPIOIBE	408	RW	0x0	8
8	4	GPIOIEV	40C	RW	0x0	8
9	5	GPIOIE	410	RW	0x0	8
10	6	GPIORIS	414	R	0x0	8
11	7	GPIOMIS	418	R	0x0	8
12	8	GPIOIC	41C	W	0x0	8
13	9	GPIOAFSEL	420	RW		
14	10	GPIOPeriphID0	FE0	R	0x61	8
15	11	GPIOPeriphID1	FE4	R	0x10	8
16	12	GPIOPeriphID2	FE8	R	0x04	8
17	13	GPIOPeriphID3	FEC	R	0x0	8

Step.2 Save to text file format



Step.3 Transfer to the W/S machine




```
// Block Name   Platform        Version Data     Base Address    Owner(Mail ID)
GPIO0   L8      1       2009-05-24      FD400000        rahyukju
// #    Register Name   Address "R/W (R, W, RW)"        Reset Value     Width
0       GPIODATA        0       RW      0x0     8
1       GPIODIR 400     RW      0x0     8
2       GPIOIS  404     RW      0x0     8
3       GPIOIBE 408     RW      0x0     8
4       GPIOIEV 40C     RW      0x0     8
5       GPIOIE  410     RW      0x0     8
6       GPIORIS 414     R       0x0     8
7       GPIOMIS 418     R       0x0     8
8       GPIOIC  41C     W       0x0     8
9       GPIOAFSEL       420     RW
10      GPIOPeriphID0   FE0     R       0x61    8
11      GPIOPeriphID1   FE4     R       0x10    8
12      GPIOPeriphID2   FE8     R       0x04    8
13      GPIOPeriphID3   FEC     R       0x0     8
14      GPIOPCellID0    FF0     R       0xd     8
```

Step.4 Generate register ROM data, etc. using LGE PERL utility

```
% ./excel2vlogc.pl GPIO0_SFR.txt

Generate ROM, Verilog, C models finished!

Input SFR file = GPIO0_SFR.txt

Output files
  - Regster test file = GPIO0_SFR.rom
  - Verilog head file = GPIO0_SFR.v
  - C        head file = GPIO0_SFR.h


% ./chk_addr_complict.pl
Result: # conflict address = 0
```

Spec-based Automatic Register Test (2 of 2)

ROM data

```
fd400000 2 0 8
fd400400 2 0 8
fd400404 2 0 8
fd400408 2 0 8
fd40040c 2 0 8
fd400410 2 0 8
fd400414 1 0 8
fd400418 1 0 8
fd40041c 0 0 8
```

Verilog register head file

```
`define GPIOO_BASE_ADDR  32'hFD400000

`define rGPIOO_GPIODATA     (`GPIOO_BASE_ADDR + 32'h0)
`define rGPIOO_GPIODIR      (`GPIOO_BASE_ADDR + 32'h400)
`define rGPIOO_GPIOIS       (`GPIOO_BASE_ADDR + 32'h404)
`define rGPIOO_GPIOIBE      (`GPIOO_BASE_ADDR + 32'h408)
`define rGPIOO_GPIOIEV      (`GPIOO_BASE_ADDR + 32'h40C)
`define rGPIOO_GPIOIE       (`GPIOO_BASE_ADDR + 32'h410)
`define rGPIOO_GPIORIS      (`GPIOO_BASE_ADDR + 32'h414)
`define rGPIOO_GPIOMIS      (`GPIOO_BASE_ADDR + 32'h418)
`define rGPIOO_GPIOIC       (`GPIOO_BASE_ADDR + 32'h41C)
```

C register head File

```
#define GPIOO_BASE_ADDR  0xFD400000

#define rGPIOO_GPIODATA (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x0))
#define rGPIOO_GPIODIR  (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x400))
#define rGPIOO_GPIOIS   (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x404))
#define rGPIOO_GPIOIBE  (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x408))
#define rGPIOO_GPIOIEV  (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x40C))
#define rGPIOO_GPIOIE   (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x410))
#define rGPIOO_GPIORIS  (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x414))
#define rGPIOO_GPIOMIS  (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x418))
#define rGPIOO_GPIOIC   (*(volatile unsigned *)(GPIOO_BASE_ADDR + 0x41C))
```

Test bench

Step.5 Set AXI VM parameters and run RTL simulation

6. Check the result log file

SFR R/W test result

#	result	address	write	read	
1	PASS	fd400000	f1243734	00000034	– R/W
2	PASS	fd400400	184369ab	000000ab	– R/W
7	PASS	fd400414	78af5431		– W
9	PASS	fd40041c		000000ff	– R

Testbench Top

AXI VM

i_axibfm_0

RTL Design

```
% ./run_sim.csh vm UART1 a a

if($2 == r || $2 == reg) then
    #---------------------------------------
    # (2-1) only register I/F test
    #---------------------------------------
    set AXI_VM_SFR_TEST = 1
    ${NCELAB} -MESSAGES -NOWARN BNDWRN \
        -f option.f \
        worklib.${TOP} \
        -DEFPARAM ${HIE}.SEVERITY_LEVEL=2 \
        -DEFPARAM ${HIE}.SFR_ROM_DATA=\"$BENCH_TOP/VM_TEST/TBENCH/SFR/MSOC_$3_SFR.rom\" \
        -DEFPARAM ${HIE}.SFR_RESULT_FILE=\"$BENCH_TOP/VM_TEST/TBENCH/LOG/MSOC_$3_SFR.rpt\" \
        -DEFPARAM ${HIE}.TEST_TYPE=${AXI_VM_SFR_TEST}
```

32 page 참고

The Role of the CPU

- **Controls the transfer of data and instructions.**

- **All the CPU does in run programs**
  - Known as the FETCH-DECODE-EXECUTE-CYCLE
    - ✓ by fetching instruction from RAM, evaluation them,
      - – Computation part (C objects by gcc compiler)
    - ✓ and executing them in sequence
      - – Communication part (VM Verilog)
    - ✓ Responsible for detecting exception.
      - – H/W ISR (VM Verilog)

Adaptive verification S/W in VM

❑ The SystemVerilog Direct Programming Interface (DPI)

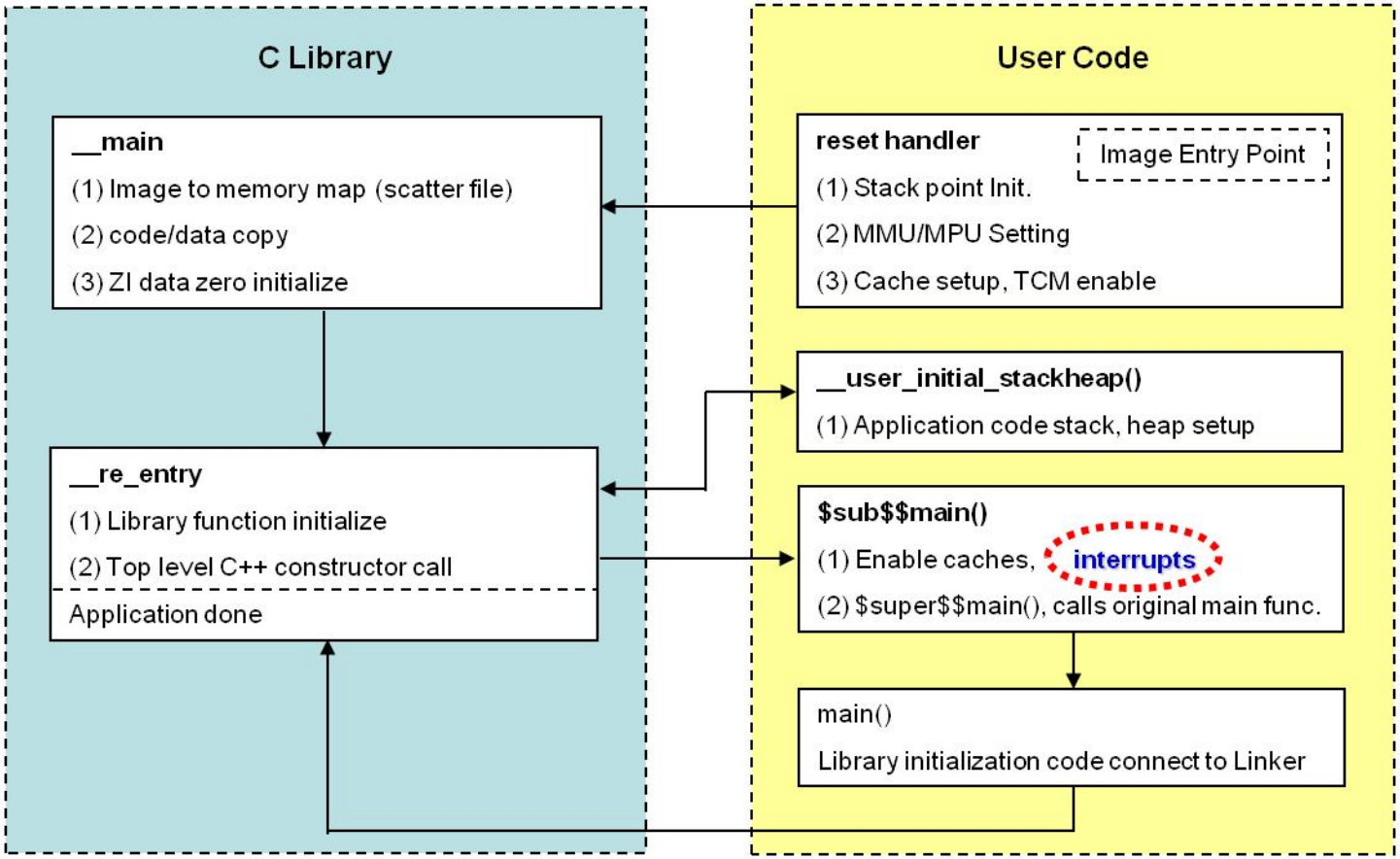
- interface between SystemVerilog and the C (and C++, SystemC) language
- It allows the designer to easily,
  - ✓ call C functions from SystemVerilog : computation part
  - ✓ call SystemVerilog tasks/functions from C : communication part

❑ The DPI has great advantages

- it allows the user to reuse existing C code and
- does not require the knowledge of Verilog Programming Language Interface (PLI) or Verilog Procedural Interface (VPI) interfaces

```
% g++ -shared -o L8_VM.so ./SW/isr/isr.c ./SW/uart/uart_tst_VM.c –DLGE_VM_FW
% ncelab -dpiheader dpi.h …
% ncsim -sv_lib ./SW/L8_VM.so …
```

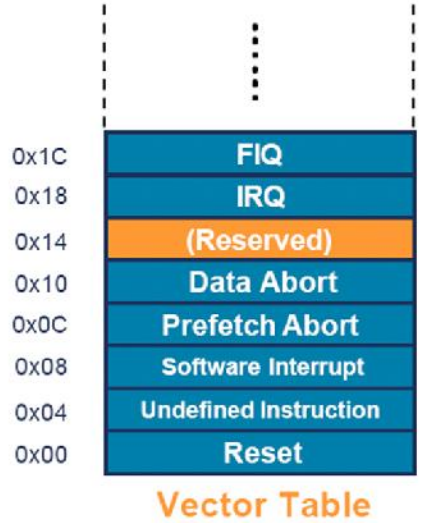
Ex. System Initialization Sequence



ARM Processor Modes & Exception Handing

❑ Exception Handler Basic Behavior

- Copies CPSR into SPSR_<mode>
- Sets appropriate CPSR bits
- Change to ARM state
- Change to exception mode
- Disable interrupts (if appropriate)
- Stores the return address in LR_<mode>
- Sets PC to vector address
- To return, exception handler needs to:
- Restore CPSR from SPSR_<mode>
- Restore PC from LR_<mode>



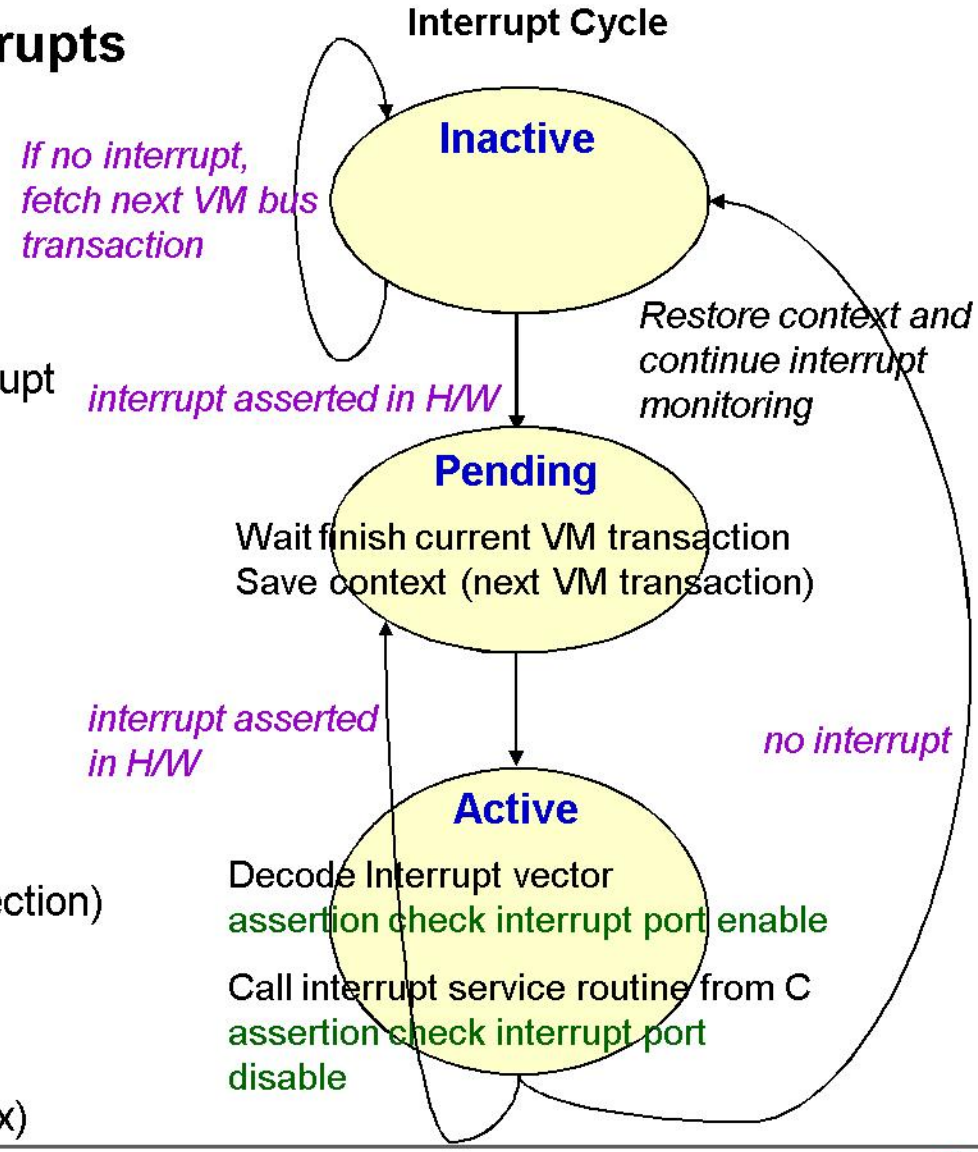
Exception modes

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ VM Operation	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

Source: "ARM Architecture Overview", ARM

Exclusive ISR execution in VM

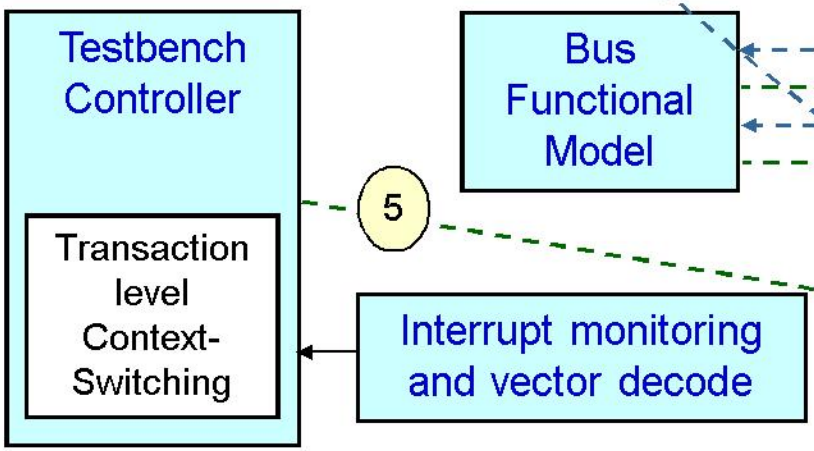
- **VM features for handling interrupts**
  - H/W port INT[63:0] based level-sensitive monitoring
  - 1-N model
    - ✓ Only one VM handles this interrupt
  - support nested interrupt
  - does not support interrupt prioritization and distribution
    - ✓ Soft type RTOS kernel
  - multiprocessing environments
    - ✓ Bus transaction level (critical section) virtual context-switching
    - ✓ it secures exclusive-bus access to the H/W block. (Mutex)



Ex. VM-based UART Verification S/W

Verilog Testbench Pop

```
1  // Verilog task export to C domain
2
3  export "DPI-C" task raxi_w;
4  export "DPI-C" task raxi_r;
5  export "DPI-C" task raxi_rw;
6  export "DPI-C" task raxi_rret;
7
8  // C function import from C domain
9  import "DPI-C" context task testUARTInt_VMC();
10 import "DPI-C" context task pl_uart1_handler_VMC();
11
12 // Testbench Top
13
14 task run_axi_vm_armd;
15     testUARTInt_VMC();
16
17     repeat(100000)@(posedge ACLK);
18     $finish(2);
19 endtask
```

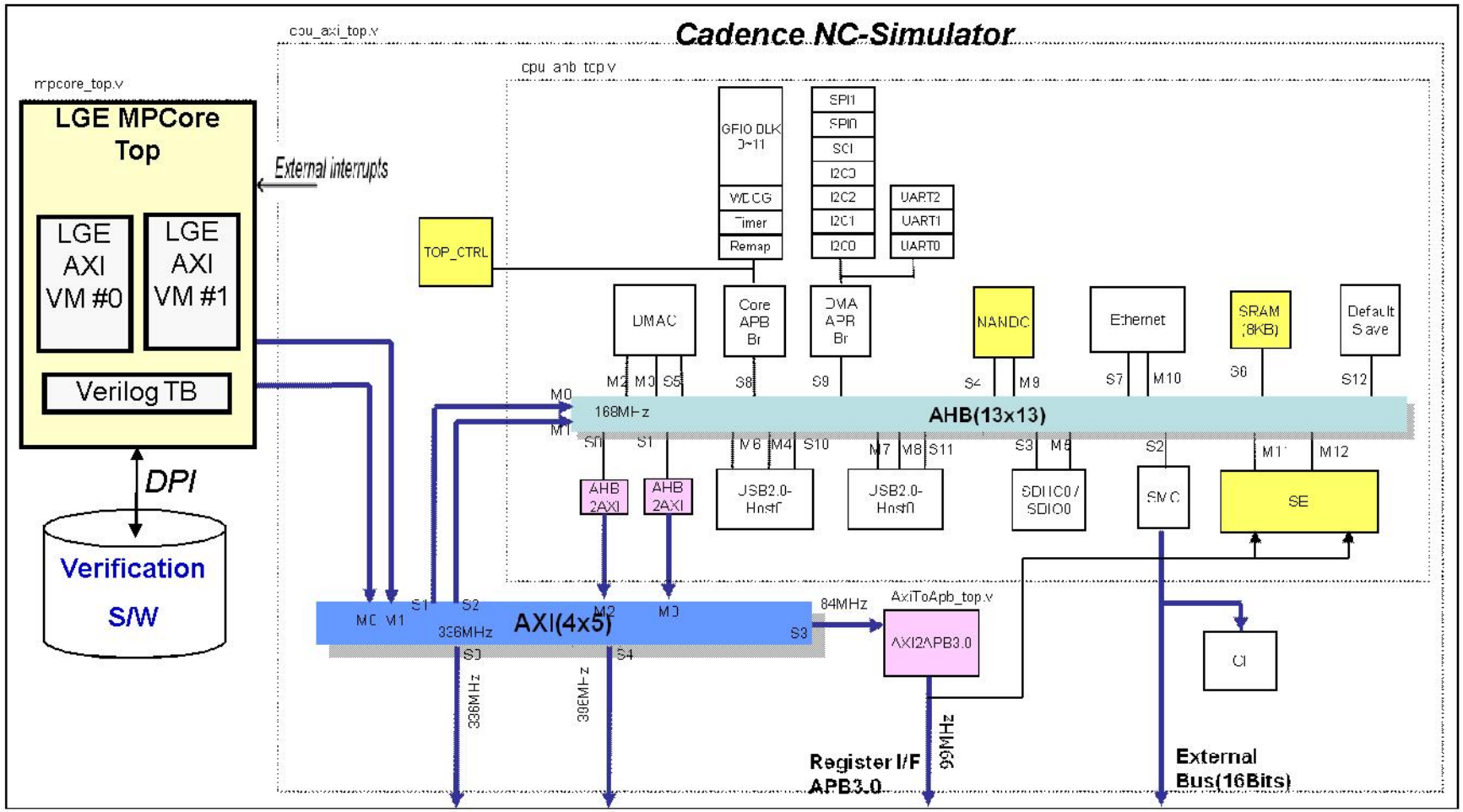


ARM Verification S/W

```
1  #include <svdpi.h>
2
3  /* bus communication using SystemVerilog tasks */
4
5  extern "C" void  raxi_w    (U32 , U32);
6  extern "C" void  raxi_r    (U32);
7  extern "C" void  raxi_rret (U32, U64 *);
8
9  /* UART Interrupt Verification S/W (L8) */
10
11 static TestStatus testUARTInt_VMC(void)
12 {
13     /* computation part */
14
15     Word temp, i;
16     int iBaudRate, iFraction;
17
18
19     TestStatus status     = PASS;
20     uart1IrqFlag          = FALSE;
21
22     iBaudRate = 174000000 /( 16 * 115200);
23     iFraction = 174000000 %( 16 * 115200);
24
25     /* communication part */
26
27     raxi_w(rUART0IBRD,  iBaudRate);
28     raxi_w(rUART0FBRD,  iFraction);
29     raxi_w(rUART0IMSC,  0x50);
30     raxi_w(rUART0LCR_H, 0x60);
31
32     printf("[ARM C    ] uart1 -> uart2 \n");
33     return status;
34 }
35
36 /* UART Interrupt Handler (L8) */
37
38 extern "C" int pl_uart1_handler_VMC()
39 {
40     raxi_w   (rUART1ICR, 0xFF);
41     raxi_rret(rUART1DR,  &rdata);
42
43     printf("[ARM C    ] Uart1 \n");
44     return 0;
45 }
```

Case Study: UART INTR Verification (1/2)

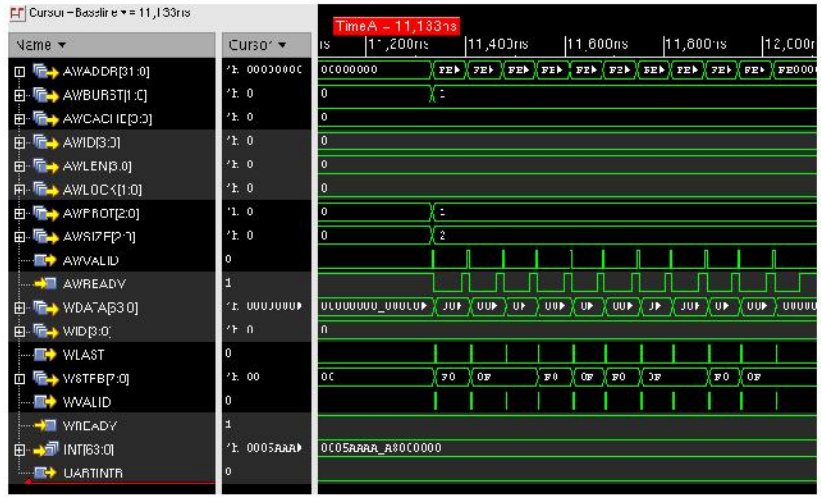
❑ Simulation/Verification Environment

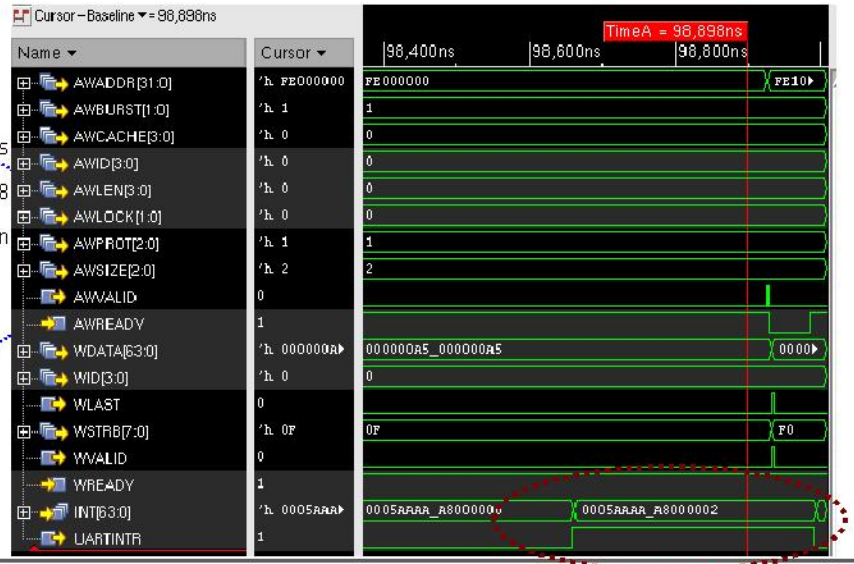


Case Study: UART INTR Verification (2/2)

❑ Simulation Results /SD/DT/sihyun/Project/LG1150/trunk/CPU/sim/ncsim.log@hyperX3

```
ncsim: 08.20-s003: (c) Copyright 1995-2008 Cadence Design Systems, Inc.
ncsim
    worklib.tb_cpu_subsystem
    -sv_lib ../tb/VM_TEST/TBENCH/SW/L8_VM.so
    -input ncsim.run

ncsim> run
[LGE VM #1] ------------------------------------------------------------
[LGE VM #1] Init interrupt vector at 11007.360 ns
[LGE VM #1] HEX = 0005_aaaa_a800_0000
[LGE VM #1] ------------------------------------------------------------
[LGE VM #1] run_vm_armd start at 11327.040 ns !
[LGE VM #1] WR fe000024, 0000005e_0000005e, 01 at 11330.000 ns
[LGE VM #1] WR fe000028, 00000019_00000019, 01 at 11401.040 ns
[LGE VM #1] WR fe000038, 00000050_00000050, 01 at 11472.080 ns
[LGE VM #1] WR fe00002c, 00000060_00000060, 01 at 11537.200 ns
[LGE VM #1] WR fe000030, 00000301_00000301, 01 at 11608.240 ns
[LGE VM #1] WR fe100024, 0000005e_0000005e, 01 at 11673.360 ns
[LGE VM #1] WR fe100028, 00000019_00000019, 01 at 11744.400 ns
[LGE VM #1] WR fe100038, 00000050_00000050, 01 at 11809.520 ns
[LGE VM #1] WR fe10002c, 00000060_00000060, 01 at 11880.560 ns
[LGE VM #1] WR fe100030, 00000301_00000301, 01 at 11945.680 ns
[ARM C    ] uart1 -> uart2
[LGE VM #1] WR fe000000, 000000a5_000000a5, 01 at 12016.720 ns
[LGE VM #1] ------------------------------------------------------------
[LGE VM #1] Init interrupt vector = 0005_aaaa_a800_0000
[LGE VM #1] Curr interrupt vector = 0005_aaaa_a800_0002 at 98917.880 ns
[LGE VM #1] ------------------------------------------------------------
[LGE VM #1] ISR: Start Interrupt Handler: Decode Interrupt Vector at 98917.880 ns
[LGE VM #1]     -> interrupt vector position = 1 bit at 98917.880 ns
[LGE VM #1] ISR: Start Interrupt Handler: Check Interrupt Port Enable at 98917.88
[LGE VM #1]     -> uart1_intr=1 Interrupt port active check PASSED
[LGE VM #1] ISR: Start Interrupt Handler: Stall normal transaction at 98917.880 n
[LGE VM #1] ISR: Start Interrupt Handler: Run uart1_intr ISR at 98917.880 ns
[LGE VM #1] ISR: uart1 routine START at 98919.360 ns !
[LGE VM #1] WR fe100044, 000000ff_000000ff, 01 at 98922.320 ns
[LGE VM #1] ISR: uart1 routine End   at 98993.360 ns !
[LGE VM #1] ISR: Start Interrupt Handler: Check Interrupt Port Clear at 99000.760
[LGE VM #1]     -> uart1_intr=0 Interrupt port clear check PASSED
[LGE VM #1] ISR: test done at 99006.680 ns

Memory Usage - 16.1M program + 804.2M data = 820.3M total
CPU Usage - 0.5s system + 11.6s user = 12.1s total (37.5% cpu)
Simulation complete via $finish(2) at time 99006680 PS + 3
../tb/VM_TEST/LGE_AMBA_VIP/AXI/src/func/lge_axi_vm_bfm.v:3010       $finish(2);
ncsim> exit
```





User Guide

- 본 page 이후부터는, 실제 VM 을 사용하기 위한, simulation 환경 setup 방법과 testbench 추가 방법에 관하여 기술한다.

- **1. LGE VM 환경 SVN check out 과 VM 환경 설정 (C-shell script)**
  - IP_NAME/tb/VM_TEST/lge_vmpcore_axivm_env.csh

- **2. VM testbench 추가**
  - Verilog 로만 testbench 기술
    - ✓ IP_NAME/tb/VM_TEST/TBENCH/run_axi_vm_armd.v
  - ARM Code 사용시
    - ✓ C ISR : IP_NAME/tb/VM_TEST/TBENCH/SW/isr/isr.c
    - ✓ Testbench: IP_NAME/tb/VM_TEST/TBENCH/SW/IP/IP_tst_VM.c

- **3. ARM C ISR I/F module 수정 (Verilog)**
  - C ISR call
    - ✓ IP_NAME/tb/VM_TEST/LGE_AMBA_VIP/AXI/src/func/lge_axi_vm_isr_if.v
  - Interrupt port binding and vector table modeling
    - ✓ IP_NAME/tb/VM_TEST/LGE_AMBA_VIP/AXI/src/func/lge_axi_vm_isr_conf.v

- **4. run script**
  - IP_NAME/sim/run_sim.csh

VM Instantiation

```
initial begin
    w_vm0_start = 1'b0;
    wait(!ARESETn4[0]);
    wait(!ARESETn4[1]);
    repeat(10)@(posege ALKC);
    w_vm0_start = 1'b1;
end

lge_axi_vm #(
    0,                      // VM stall enable(1), disable(0)
    "VM_ARM11_D",           // active VM name
    64,                     // data width, 32, 64, 128 bits
    `AXI_VM_USER_TEST,      // test type: SFR TEST, USER TEST
    `AXI_VM_ERR_STOP        // severity level: ERR WARNING/STOP/FINISH
) lge_axi_vm_D (
    // VM Activation
    .VM_START       ( w_vm0_start   ),
    .VM_STALL       ( 0             ),
    .VM_DONE        (               ),
    // Global Signals
    .ACLK           ( ACLK          ),
    .ARESETn        ( ARESETn4[0]   ),
    // H/W Interrupt Vector
    .INT            ( INT           ),
    // WRITE Address Channel
    .AWID           ( AWID0         ),
    .AWADDR         ( AWADDR0       ),
    .AWLEN          ( AWLEN0        ),
    .AWSIZE         ( w_awsize0     ),
    .AWBURST        ( AWBURST0      ),
    .AWLOCK         ( AWLOCK0       ),
    .AWCACHE        ( AWCACHE0      ),
    .AWPROT         ( AWPROT0       ),
    .AWVALID        ( AWVALID0      ),
    .AWREADY        ( AWREADY0      ),
    ...
    ...
);
```

상세한 것은 VM top file 을 참조 바랍니다.

IP_NAME/tb/VM_TEST/LGE_AMBA_VIP/AXI/src/top/lge_axi_vm.v

1. VM 환경 설정

❑ IP_NAME/tb/ directory 밑에 VM 환경 check-out.

- svn …… VM_TEST
- Check-in 은 관리자만 할 수 있도록 permission 관리 필요.
- file(IP_NAME/tb/VM_TEST/lge_vmpcore_axivm_env.csh) 을 검증 환경에 맞게 select.

```
// Case 1. use AXI bus
23  ${NCVLOG} -work worklib -message \
24  #   -define LGE_VM_APB3 \
25  #   -define LGE_VM_FW \


// Case 2. use AXI/APB3 bus
23  ${NCVLOG} -work worklib -message \
24      -define LGE_VM_APB3 \
25  #   -define LGE_VM_FW \


// Case 3.1 use AXI bus and ARM C code
23  ${NCVLOG} -work worklib -message \
24  #   -define LGE_VM_APB3 \
25      -define LGE_VM_FW \


// Case 3.2 use AXI/APB3 bus and ARM C code
23  ${NCVLOG} -work worklib -message \
24      -define LGE_VM_APB3 \
25      -define LGE_VM_FW \
```

2. VM testbench 추가

- 해당 **file: IP_NAME/tb/VM_TEST/TBENCH/run_axi_vm_armd.v**

- **run_axi_vm_armd.v** 는 **VM testbench** 의 **top file** 입니다.
  본 **file**은, **SystemVerilog file** 이라, **Verilog syntax** 로 아래의 **task** 안에 **testbench** 를 추가하면 됩니다.

```
task run_axi_vm_armd ;



endtask
```

- **Testbench** 를 기술할 때, 아래 **3**가지의 **case** 로 나누어 생각 할 수 있으며, **run_axi_vm_armd.v file** 에 **user** 는 필요에 맞는 기술법을 사용 하시면 됩니다.

  - Case 1. register setting 에 VM 의 AXI bus 를 사용함.
  - Case 2. register setting 에 VM 의 APB3 bus 를 사용함.
  - Case 3. ARM C testbench and/or ARM C ISR code 를 이용함.

Ex. Case 1: Register setting using AXI bus

```
task run_axi_vm_armd ;

    //-------------------------------------------
    // Register Configuration
    //-------------------------------------------

    // 1. register Write
    // raxi_w (32 bit address, 32 bit data);

    raxi_w(32'hC000_0000,32'h110000FF);

    // 2. register read
    // raxi_r (32 bit address);

    raxi_w(32'hC000_0000,32'h110000FF);

    // display read data

    $display("read data = %0h", A_ARDATA[0]);

    // 3. register RAW (read after write) auto-check
    // raxi_rw(32 bit address, 32 bit data);

    raxi_rw(32'hC000_0000,32'h110000FF);

    // simulation finished
endtask
```

Ex. Case 2: Register setting using APB3 bus

```
task run_axi_vm_armd ;

    //-------------------------------------------
    // Register Configuration
    //-------------------------------------------

    // 1. register Write
    // rapb_w (32 bit address, 32 bit data);

    rapb_w(32'hC000_0000,32'h110000FF);

    // 2. register read
    // rapb_r (32 bit address);

    rapb_w(32'hC000_0000,32'h110000FF);

    // display read data

    $display("read data = %0h", A_ARDATA[0]);

    // 3. register RAW (read after write) auto-check
    // rapb_rw(32 bit address, 32 bit data);

    rapb_rw(32'hC000_0000,32'h110000FF);

    // simulation finished
endtask
```

Ex. Case 3.1: ARM C code integration to VM

```
1  `ifdef LGE_VM_FW
2      //----------------------------------------
3      // VM Verilog Tasks
4      //----------------------------------------
5  `ifdef LGE_VM_APB3
6      export "DPI-C" task rapb_w;
7      export "DPI-C" task rapb_r;
8      export "DPI-C" task rapb_rw;
9  `endif
10     export "DPI-C" task raxi_w;
11     export "DPI-C" task raxi_r;
12     export "DPI-C" task raxi_rw;
13     export "DPI-C" task delayCycleVm;
14     export "DPI-C" task swDbgVm;
15     //----------------------------------------
16     // C Tasks
17     //----------------------------------------
18     import "DPI-C" context task testUART_VMC();
19     import "DPI-C" context task pl_uart1_handler_VMC();
20 `endif
21
22 task run_axi_vm_armd;
23
24     // call ARM C testbench
25     testUART_VMC();
26
27     // register set
28     raxi_w(32'hfd30_0000, 32'h001);
29
30     // wait 100,000 cycle for interrupt test
31     repeat(100000)@(posedge ACLK);
32 endtask
```

C Testbench (Page 17 참조)

C ISR (Page 26, 28 참조)

Ex. Case 3.2: ARM C code Compile

1. ISR code : IP_NAME/tb/VM_TEST/TBENCH/SW//isr/isr.c

```
PROTECTED VOID pl_uart1_handler(VOID)
{
    uart2.UART1ICR = 0xFF;
    uart2IrqFlag = TRUE;
}
```

```
extern "C" int pl_uart1_handler_VMC()
{
    raxi_w (rUART1ICR, 0xFF);
  //rapb_w (rUART1ICR, 0xFF);

    printf("Uart1 data = 0x\n", rdata);
    return 0;
}
```

2. Testbench code : IP_NAME/tb/VM_TEST/TBENCH/SW/uart/uart_tst_VM.c

3. Compile

IP_NAME/tb/VM_TEST/TBENCH/SW/Makefile

```
## customize
CFILE   = \
    ./isr/isr.c \
    ./uart/uart_tst_VM.c

##########################################################
OBJ     = L9_VM.so
NCHOME  = ${shell echo `ncroot`}
RM      = \rm -f
GPP     = $(NCHOME)/tools/systemc/gcc/bin/g++
OPT     = -fPIC -g -shared
INC     = -I./include -I$(NCHOME)/tools/include
DEF     = -DLGE_VM_FW

automatic: clean
    $(GPP) $(OPT) -o $(OBJ) $(CFILE) $(INC) $(DEF) -time
    ls -alF $(OBJ)

clean:
    $(RM) $(OBJ)
```

Ex. VM Testbench

```
task run_axi_vm_armd;
    for(int i = 0; i < 10; i++) begin
        if(axi_vm_rand.randomize() != 1) begin
            $stop;
        end
        // random write data generation
        genRandWdata(64,7); // (data width, random seed)

        // axi burst write
        axi_wid(
            addr + (i * 32'h100),
            axi_vm_rand.alen,
            axi_vm_rand.asize,
            `AXI_ABURST_INCR,
            1, // AWID
            1, // WID
            0, 0, 0, 0
        );
        delayCycle(3);

        // axi burst read
        axi_rid(
            addr + (i * 32'h100),
            axi_vm_rand.alen,
            axi_vm_rand.asize,
            `AXI_ABURST_INCR,
            axi_vm_rand.aid, // ARID
            0, 0, 0, 0
        );
    end
endtask
```

3.1 ISR call module 수정 (Verilog)

- 해당 **file: IP_NAME/tb/VM_TEST/LGE_AMBA_VIP/AXI/src/func/lge_axi_vm_isr_if.v**

  에 모든 **interrupt port** 에 관하여, **ARM C ISR code** 를 **call** 하는 **Verilog task** 가 미리 **define** 되어 있습니다.

  본 **task** 안에서, **USER** 는 **isr.c** 에서 **define** 한 **ARM C ISR code** 를 **call** 하시면 됩니다. **ISR** 은, **VM** 이 **INT** 값을 **monitoring** 하면서, 자동으로 **call** 하게 되어 있습니다.

  task 이름은, 가능한 **isr_*interruptPortName*** 으로 정했습니다.

```
// NOP
task isr_na;

endtask

// template task for spi0_intr
task isr_spi0_intr;

endtask

// uart1_intr
task isr_uart1_intr;
   pl_uart1_handler_VMC(); // ARM C code
endtask
```

3.2 ISR I/F module 수정 (Verilog)

- 해당 file: IP_NAME/tb/VM_TEST/LGE_AMBA_VIP/AXI/src/func/lge_axi_vm_isr_conf.v 에 모든 실제 H/W interrupt port 를 INT[n:0] 에 assign 하여, vector table 을 만듭니다.

  CPU sub-system TOP/RTOP/CTOP 은 미리 만들어 져 있습니다.

  그 외 Design 에서는, 실제 사용되는 interrupt port 만 assign 하고, vector table 에서 사용하지 않는 interrupt port 들은, default 값으로 stuck 시키며, 아래의 file 에서 `define ISR_USER_MODE 를 active 시킵니다.

  IP_NAME/tb/VM_TEST/LGE_AMBA_VIP/AXI/src/func/lge_axi_vm_defines.v 은 DUT mode 를 설정합니다.

```
// select DUT mode

`ifdef LGE_VM_FW
    `define RUN_VM_CPUTOP
  //`define RUN_VM_RTOP
  //`define RUN_VM_CTOP
  //`define ISR_USER_MODE
`endif
```

Ex. CPU TOP ISR Configuration

```
`ifdef RUN_VM_CPUTOP
    `define CPU_PERI_INTR    tb_cpu_subsystem.cpu_axi_top.inst_cpu_ahb_top
`endif

`ifdef RUN_VM_RTOP
    `define CPU_PERI_INTR    tb_l9_core.l9_core0.cpu_gpu_top0.cpu_axi_top.inst_cpu_ahb_top
    `define AV_INTR          tb_l9_core.l9_core0.cpu_gpu_top0
`endif

`ifdef RUN_VM_CTOP
    `define CPU_PERI_INTR    tb_l9_ctop.lg1152.l9_core0.cpu_gpu_top0.cpu_axi_top.inst_cpu_ahb_top
    `define AV_INTR          tb_l9_ctop.lg1152.l9_core0.cpu_gpu_top0
    `define EXT_INTR         tb_l9_ctop.lg1152
`endif

    wire usb_bt_ehci_interrupt_o    = `CPU_PERI_INTR.usb_bt_ehci_interrupt_o;// 57 Peri.
    wire usb_bt_ohci_1_irq_o_n      = `CPU_PERI_INTR.usb_bt_ohci_1_irq_o_n;  // 56 Peri.
    wire usb_bt_ohci_0_irq_o_n      = `CPU_PERI_INTR.usb_bt_ohci_0_irq_o_n;  // 55 Peri.
    wire TOUT_INTR                  = `CPU_PERI_INTR.TOUT_INTR;              // 27 Peri.
    wire sc_intr                    = `CPU_PERI_INTR.sc_intr;                // 26 Peri.
    wire sp_intr                    = `CPU_PERI_INTR.sp_intr;                // 25 Peri.
    wire emac_intr                  = `CPU_PERI_INTR.emac_intr;              // 24 Peri.
    wire ci_intr                    = `CPU_PERI_INTR.ci_intr;                // 23 Peri.
```

```
`ifdef RUN_VM_CPUTOP
    wire [63:0] arr_hw_intr = {
        //-----------------------------------------------
        1'b0,                          // 63 L2CCINTR  N/A
        1'b0,                          // 62 PMUIRQ[5] N/A
        1'b0,                          // 61 PMUIRQ[4] N/A
        1'b0,                          // 60 PMUIRQ[0] N/A
        1'b0,                          // 59 RSV       N/A
        1'b0 ,                         // 58 RSV       N/A
        //-----------------------------------------------
        usb_bt_ehci_interrupt_o,       // 57 Peri.
        usb_bt_ohci_1_irq_o_n,         // 56 Peri.
        usb_bt_ohci_0_irq_o_n,         // 55 Peri.
        //-----------------------------------------------
        1'b0, // ext_intr[3],          // 54 External PAD
        1'b0, // ext_intr[2],          // 53 External PAD
        1'b0, // ext_intr[1],          // 52 External PAD
        1'b0, // ext_intr[0],          // 51 External PAD
        //-----------------------------------------------
        1'b0, // intr_slt_cpu,         // 50 AV
        1'b0, // intr_gfx_cpu,         // 49 AV
        1'b0, // intr_aad_cpu,         // 48 AV
        1'b0, // intr_vp_3ch_cpu,      // 47 AV
        1'b0, // intr_vp_hdmi_cpu,     // 46 AV
        1'b0, // intr_vp_iris_cpu,     // 45 AV
        1'b0, // intr_dvr_cpu,         // 44 AV
        1'b0, // intr_sdec_cpu,        // 43 AV
        1'b0, // intr_ipc_bcpu_cpu,    // 42 AV
        1'b0, // intr_de_cpu,          // 41 AV
        1'b0, // intr_audio1_cpu,      // 40 AV
        1'b0, // intr_audio0_cpu,      // 39 AV
        1'b0, // intr_gpu_mmupp0,      // 38 AV
        1'b0, // intr_gpu_pp0,         // 37 AV
        1'b0, // intr_gpu_mmugp,       // 36 AV
        1'b0, // intr_gpu_gp,          // 35 AV
        1'b0, // intr_vdec_cpu,        // 34 AV
        1'b0, // intr_venc1_cpu,       // 33 AV
        1'b0, // intr_venc0_cpu,       // 32 AV
        1'b0, // intr_frc_cpu,         // 31 AV
        1'b0, // intr_silkroad2_cpu,   // 30 AV
        1'b0, // intr_silkroad1_cpu,   // 29 AV
        1'b0, // intr_silkroad0_cpu,   // 28 AV
        //-----------------------------------------------
        TOUT_INTR,             // 27 Peri.
        sc_intr,               // 26 Peri.
        sp_intr,               // 25 Peri.
        emac_intr,             // 24 Peri.
        ci_intr,               // 23 Peri.
```

VM 는, INT[n:0] 의 값을 monitoring 하면서, 값이 변할 때 변한 bit 를 찾아, interrupt vector table 에 맞게 해당 ISR I/F task 를 call 합니다. (page 28 참조)

Ex. USER ISR Configuration

Default

```
`ifdef ISR_USER_MODE

    wire [63:0] arr_hw_intr = {
        1'b0,
        63'd0
    };
`endif
```

user 가 수정

```
`ifdef ISR_USER_MODE

    wire intr_venc1 = hie_path.intr_venc1_cpu;
    wire intr_venc0 = hie_path.intr_venc0_cpu;

    wire [63:0] arr_hw_intr = {
        31'd0,
        intr_venc1,      // 33 AV
        intr_venc0,      // 32 AV
        31'd0
    };
`endif
```

4. run script

- Workding directory: IP_NAME/sim/

- Main run Script 수정: **run_sim.csh**

```
40      set TOP = "tb_cpu_subsystem"
41      set HIE = "${TOP}.mpcore_top.lge_vmpcore_axi_vm.lge_axi_vm_D"
```

Testbench top

VM instantiation full path name

- RUN Simulation

% run_sim.csh vm a a a (VM testbench test)

% run_sim.csh vm r UART1 a (automatic UART1 register R/W test)