Holistic Verification of Bus Health Monitor in Automotive SoC using BHMVC and ParaHunter

Gaurav Kumar Yadav, Abhisek Hota, Prashantkumar Shukranath Sonavane, Garima Srivastava
Samsung Semiconductor India research (SSIR)
Bagmane Goldstone Building, Mahadevapura
Bengaluru 560037

***Abstract*-Nowadays, in order to ensure Functional Safety in automotive systems like ADAS/AD, various safety mechanisms are incorporated with the intent to detect faults in the safety critical functions of the design.**

BUS (Interconnect) is very important infrastructure for any SoC. Hence in this paper, we focus on a holistic verification of Bus Health Monitor (BHM) IP which is an ASIL oriented hardware framework for BUS's safety in an automotive SoC. BHM can detect and report failures in any bus component which can lead to Safety Goal violation. This paper showcases the novel "Self Contained, Plug & Play Verification Component" based verification approach for BHM instances and their interactions at SoC. This Verification component is named as a "Bus Health Monitor Verification Component" (BHMVC). BHMVC supports the exhaustive feature-wise configuration, auto functional checks for each instances of the BHM IP. BHMVC also focuses on the SoC context scenarios where other Masters and Slaves within SoC are also running the parallel traffic along with BHM, hence helps in stress testing the Bus functionality.

This paper also describes the need to check the RTL parameters associated with each BHM IP's instance at SoC and does discuss the solution named "ParaHunter", which turns out to be critical factor impacting the overall SoC die size.

Keywords—Advanced Driver Assistant Systems (ADAS), Safety Mechanism (SM), Bus Health Monitor (BHM), Automotive Safety Integrity Level (ASIL), System on Chips (SoCs)

I. INTRODUCTION

Automotive SoCs require stringent safety measures. Fault occurring at one component can result in failure at another component which is called dependent failure and fault in bus can lead to potential dependent failures. To detect these faults, Parameter and Register based configurable "Bus Health Monitor (BHM)" IP is introduced to run predefined periodic test patterns to monitor bus health. BHM is capable of detecting more than 90% permanent faults in the bus which is required for functional safety ASIL-B certification of Automotive SoC.
BHM has 3 variants namely BHM AXI Master, BHM AXI Slave, BHM APB Slave and which are connected in-parallel to respective Master and slaves.
In this paper we showcase how we verified BHM IP at SoC level in an efficient way amid all the challenges imposed by parameterized design and system constraints. We not only checked SoC level scenarios based on safety use cases but also checked BHM IP integrity across SoC with IP-IP communications, including the support to check Self-test feature, negative scenarios which resulted into the comprehensive end to end check. In system-level Scenarios depending upon the bus architecture, the bus latency will come into consideration, So a complete break the design approach is used while randomizing and stretching the test cases to catch the corner cases which ultimately led in finding IP bugs in BHM SoC verification. In order to verify BHM IP a UVM based verification component is developed named "BHMVC"
Since BHM is parameterized design DV engineer has to go manually and compare each parameters across blocks with specification which leads lot of manual efforts and has probable of human error. These issues were addressed by script-ware solutions named Para-Hunter which compare each BHM instances parameters between IPXACT JSON file which is used for RTL generation and specification excel sheet Para-Hunter has been developed.

II. BHMVC USAGE IN SOC VERIFICATION

To implement a holistic approach to verify BHM for all Blocks, UVM based Bus Health Monitor Verification Component (BHMVC) has been developed and divided into two parts

A. Integration Sanity Checks

To check BHM interrupt output integrity from all instances (within each block) to Fault Aggregator and GIC which helps in making BHM up quickly in SOC TB along with clock and reset integrity

B. Functional Checks

To verify all BHM Masters (~15) to Slaves (~40) combinations at SoC which includes passing simple argument by end-user and easing out burden of configuring both masters and slave's register for respective combinations

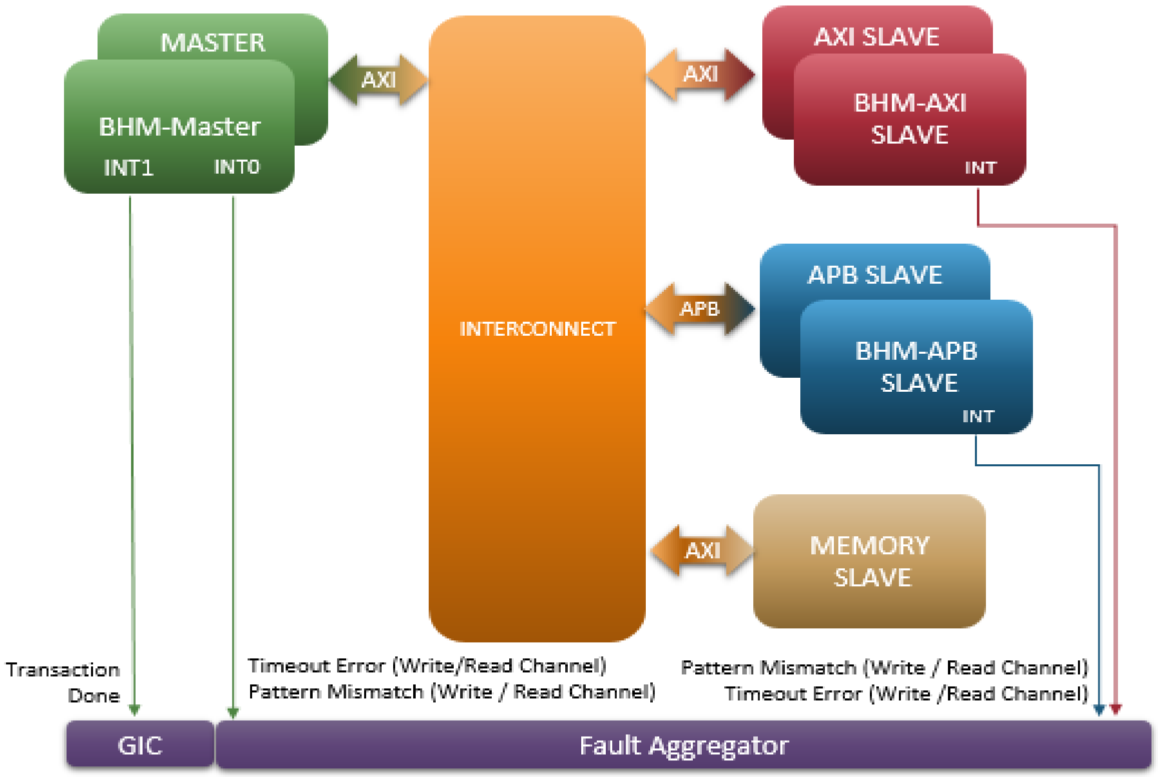

MASTER
BHM-Master
INT1
INT0
AXI
INTERCONNECT
AXI
AXI SLAVE
BHM-AXI SLAVE
INT
APB
APB SLAVE
BHM-APB SLAVE
INT
AXI
MEMORY SLAVE
Transaction Done
Timeout Error (Write/Read Channel)
Pattern Mismatch (Write / Read Channel)
Pattern Mismatch (Write / Read Channel)
Timeout Error (Write /Read Channel)
GIC
Fault Aggregator

Figure 1 BHM integration in SoC

III. BHM AXI MASTER TO SLAVE COMBINATION

Usually we have around 40-50 instances of BHM across all blocks in SoC, so to make the verification efficient and for the ease of use by end user of each blocks we categorized functional check in 3 major parts, name BHM AXI Master to BHM AXI Slave, BHM Master to BHM APB Slave and BHM Master to Memory Slave

Based on safety use cases there are multiple Master-Slave combinations, hence different types of target types have been thought through so that user can select the target type in which that Blocks falls under and simply call that target type in its sequence to check whether all corner cases are covered or not. Below Table shows all SoC level target scenarios supported to cover all possible combinations. Same target types have been used to create parallel traffic scenario as well to ensure bus is stressed to its full capacity to explore corner cases.

TABLE I
BHM AXI MASTER AND ITS SLAVE COMBINATIONS

S.No	**SoC Target Types Supported**	
	Target Type	**Comment**
1	AXI_MASTER_SINGLE_APB_SLAVE	(1-1) BHM Master to BHM_APB Slave
2	AXI_MASTER_SINGLE_AXI_SLAVE	(1-1) BHM Master to BHM_AXI Slave
3	AXI_MASTER_MULTIPLE_AXI_SLAVE	(1-Many) BHM Master to BHM_AXI Slave
4	AXI_MASTER_MULTIPLE_APB_SLAVE	(1-Many) BHM Master to BHM_APB Slave
5	MULTIPLE_AXI_MASTER_SINGLE_APB_SLAVE	(Many-1) BHM Master to BHM_APB Slave
6	AXI_MASTER_ALL_SLAVE	BHM Master to all possible slaves
7	AXI_MASTER_AXI_SLAVE_SAFE_DRAM	(1-Many) BHM Master to BHM_AXI Slave & DRAM
8	AXI_MASTER_SAFE_DRAM	(1-1) BHM Master to Safe DRAM

A. BHM AXI MASTER TO BHM APB SLAVE

BHM Master interacts with BHM APB slave to test REGISTER bus integrity from safety CPU and Main CPU to all safe APB slaves. Here both masters are connected to APB Slaved through AXI-APB Bridges.
To check bus integrity BHM AXI Master and APB Slave interact with each other with pre-defined test patterns, which will be repeated periodically as per the configured intervals. Below are procedure to follow

To check this feature below steps are used: -

1. After power on Reset configure BHM Master and BHM Slave REGISTER as follows.
2. First BHM APB slave Register need to be configured such as control register which has address alignment information based on the data width of the master selected.
3. After that, BHM AXI Master Register is configured with slave addresses, numbers, slave types et.al (TYPE = APB) and similarly same randomized slave alignment is programmed for APB slave.
4. Then timer and transaction interval register are configured for BHM Master to ensure test patterns are repeated over certain interval of time while keeping SoC bus latency in consideration.
5. Once BHM Master configuration done, control bit is programmed to initialize Master and wait for initialization completion.
6. Once initialization is done BHM Master will initiate Write and Read transactions to all configured slaves and Polling for Transfer done status to 1 to make sure all transactions are successfully completed.
7. At the same time status Register is also checked at both master and slave end to make sure error is reported to Fault Aggregator and GIC if transfer is not completed successfully.
8. With Transfer done status = 1 and error status for each channel at both master and slave end being 0 test was considered PASS else FAIL.
9. Support for number of test pattern iterations is also added so that user can select how many times test patterns can be repeated while actual transfer is going on parallel so that end to end bus integrity is checked.

B. BHM AXI MASTER TO BHM AXI SLAVE

BHM Master sends test patterns to BHM AXI Slaves in order to test BUS integrity from Safety CPU and Main CPU to other AXI Slaves which is sharing AXI path with the respective BHM Slave. To test the bus integrity BHM AXI master and BHM AXI Slave are integrated in parallel to safety master and safety slave respectively using XIU. BHM Master and slave interact with each other with pre-define test patterns. If any corrupted/out-of-order patterns is received, the respective BHM master/Slave is expected to generate interrupt which will be sent to FA and capture the cause of the error in STATUS REGISTER.
BHM AXI Master and BHM AXI Slave interacts with each other with pre-defined test patterns, which will be repeated periodically as per the configured intervals.

To check this feature below steps are used: -

1. After power on Reset configure BHM Master and BHM Slave REGISTER as follows.
2. First BHM AXI Slave Register need to be configured such as sideband signal and Data register with the value of AXI side band signals AxLEN, AxSIZE, AxBURST, AxCACHE, AxPROT along with data patterns respectively.
3. Using ADDR REGISTER slave address is configured which will be initiated by BHM Master through AxADDR.
4. Then BHM AXI Slave is initialized by configuring control register and wait for initialization done status for BHM Slave, which will keep slave ready to receive patterns from Master.
5. After slave configuration done, BHM AXI Master’s REGISTER is configure such as side band and data register same as slave register value.
6. After that, BHM AXI Master Register is configured with slave addresses, numbers, slave types et.al (TYPE = AXI)
7. Then timer and transaction interval REGISTER are also configured for BHM Master to ensure test patterns are repeated over certain interval of time while keeping SoC bus latency in consideration.
8. Once BHM Master configuration done, control bit is programmed to initialize Master and wait for initialization completion.
9. Once initialization is done BHM Master will initiate Write and Read transactions to all configured slaves and Polling for Transfer done status to 1 to make sure all transactions are successfully completed.
10. At the same time status Register is also checked at both master and slave end to make sure error is reported to Fault Aggregator and GIC if transfer is not completed .
11. With Transfer done status = 1 and error status for each channel at both master and slave end being 0 test was considered PASS else FAIL.

B. BHM AXI MASTER TO MEMORY SLAVE

BHM AXI Master performs write and read back operation to DRAM which is memory slave. Here DRAM role is just to store the test patterns. BHM Master performs Write-Read operation to configured DRAM locations with predefined data patterns. BHM AXI Master checks whether read data matches with write data for same location of memory or not, if mismatched, it will be reported as fault to Fault aggregator

To check this feature below steps are used: -

1. After power on Reset configure BHM Master as follows.
2. After that, BHM AXI Master Register is configured with slave addresses, numbers, slave types et.al (TYPE = MEMORY)
3. Then timer and transaction interval REGISTER are also configured for BHM Master to ensure test patterns are repeated over certain interval of time while keeping SoC bus latency in consideration.
4. Once BHM Master configuration done, control bit is programmed to initialize Master and wait for initialization completion.
5. Once initialization is done BHM Master will initiate Write and Read transactions to all configured slaves and Polling for Transfer done status to 1 to make sure all transactions are successfully completed.

6. At the same time all status register is checked at master end to make sure error is reported to Fault Aggregator and to GIC if transfer is not completed properly.
7. With Transfer done status = 1 and error status for each channel at both master and slave end being 0 test was considered PASS else FAIL.

IV. Script-ware parahunter

BHM design is completely parameterized, any issue in BHM parameter in each instance will result in BHM operation issues. In some corner cases issue won't be caught as many parameters are defined based on safety use case of each block.

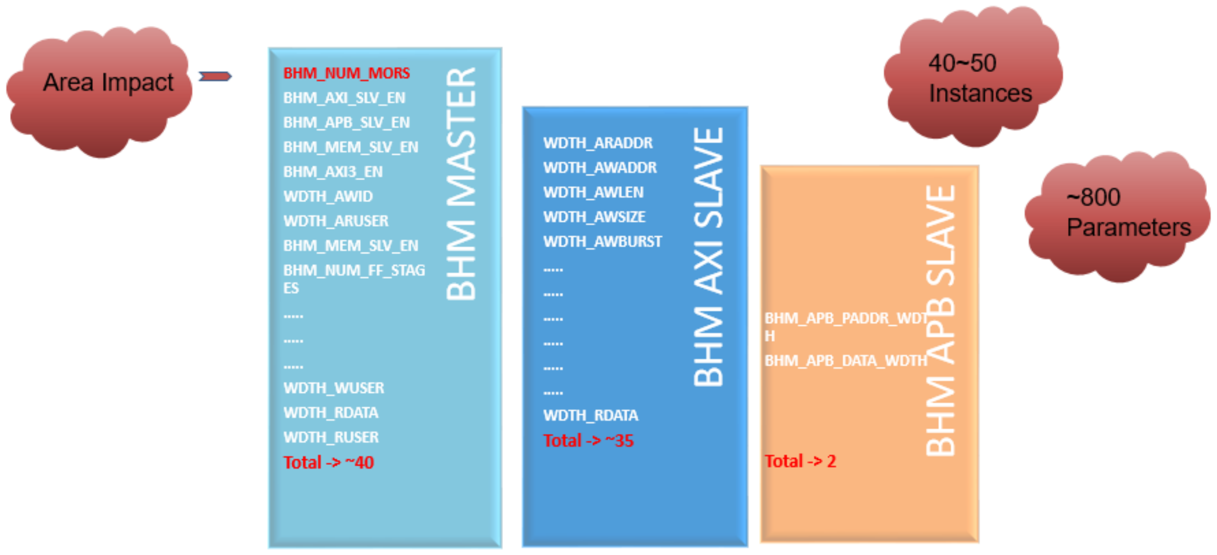
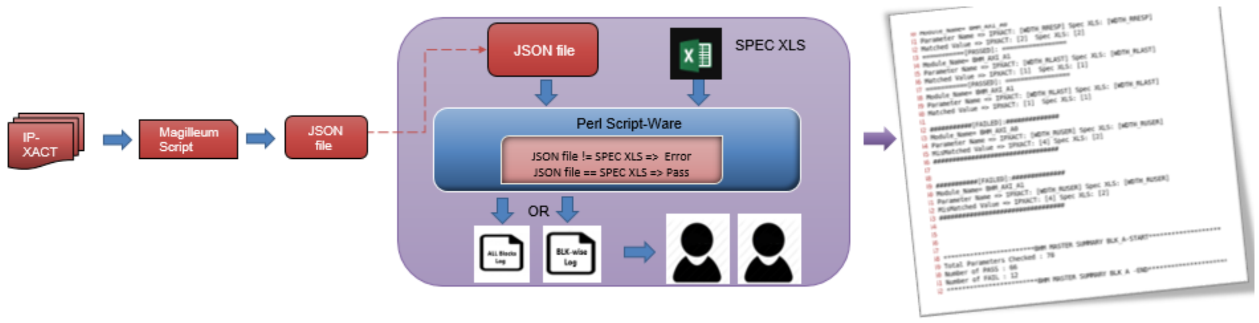


Figure 2 BHM parameter for all Instances

For example, a parameter namely BHM_SLV_NUM represents total number of slaves supported for a particular master which is defined by safety use case. Based on this parameter value, those number of slave will be instantiated for each master, so if this parameter is wrong it will result in defining wrong number of register. If this number happens to be more than the expected number of slaves it will impact SoC area number.
Similarly, in case of multiple parameters for each BHM Master, BHM AXI Slave and BHM APB Slave instances, each block designer needs to configure these parameters based on safety use case which may result into several parameter bugs across SoC. These bug otherwise would have been caught by DV engineers only by manually checking parameter values in RTL after comparing safety targets for each master and slave in specification excel sheet. This manual effort would have been error prone and would have been required for every RTL release hence there was a need to automate these checks. In SoC we have around approx. 40-50 of BHM instances usually which results in approximately nearly 800 parameters in total to be reviewed.
To overcome this challenge and reduce manual efforts BHM parameter check script-ware was created named Para-Hunter. It will take particular RTL label's JSON file generated from IPXACT and extract each instance wise parameters and compare it with specification excel sheet provided for BHM configuration. To use this script-ware user just need to pass block name to check parameters and it will display the result in customized block-wise logs with comparison PASS and FAIL message as shown in Figure 3.



```
Module_Name= BHM_AXI_A0
Parameter Name => IPXACT: [WDTH_RRESP] Spec XLS: [WDTH_RRESP]
Matched Value => IPXACT: [2]  Spec XLS: [2]
=============[PASSED]: ==================
Module_Name= BHM_AXI_A1
Parameter Name => IPXACT: [WDTH_RLAST] Spec XLS: [WDTH_RLAST]
Matched Value => IPXACT: [1]  Spec XLS: [1]
=============[PASSED]: ==================
Module_Name= BHM_AXI_A1
Parameter Name => IPXACT: [WDTH_RLAST] Spec XLS: [WDTH_RLAST]
Matched Value => IPXACT: [1]  Spec XLS: [1]

###########[FAILED]:###############
Module_Name= BHM_AXI_A0
Parameter Name => IPXACT: [WDTH_RUSER] Spec XLS: [WDTH_RUSER]
MisMatched Value => IPXACT: [4] Spec XLS: [2]
##################################


###########[FAILED]:###############
Module_Name= BHM_AXI_A1
Parameter Name => IPXACT: [WDTH_RUSER] Spec XLS: [WDTH_RUSER]
MisMatched Value => IPXACT: [4] Spec XLS: [2]
##################################




**********************BHM MASTER SUMMARY BLK_A-START********************
Total Parameters Checked : 78
Number of PASS : 66
Number of FAIL : 12
**********************BHM MASTER SUMMARY BLK A -END********************
```

Figure 3 Para-Hunter using Perl script

V. APPLICATION

Since verification of all BHM Masters (~15) to Slaves (~40) combinations is a cumbersome task at SoC level and test case development with proper use-case understanding for each combination which includes probable human-error, BHMVC plays critical role in easing and speeding up the SoC verification of BHM IP's as shown in Figure 4.

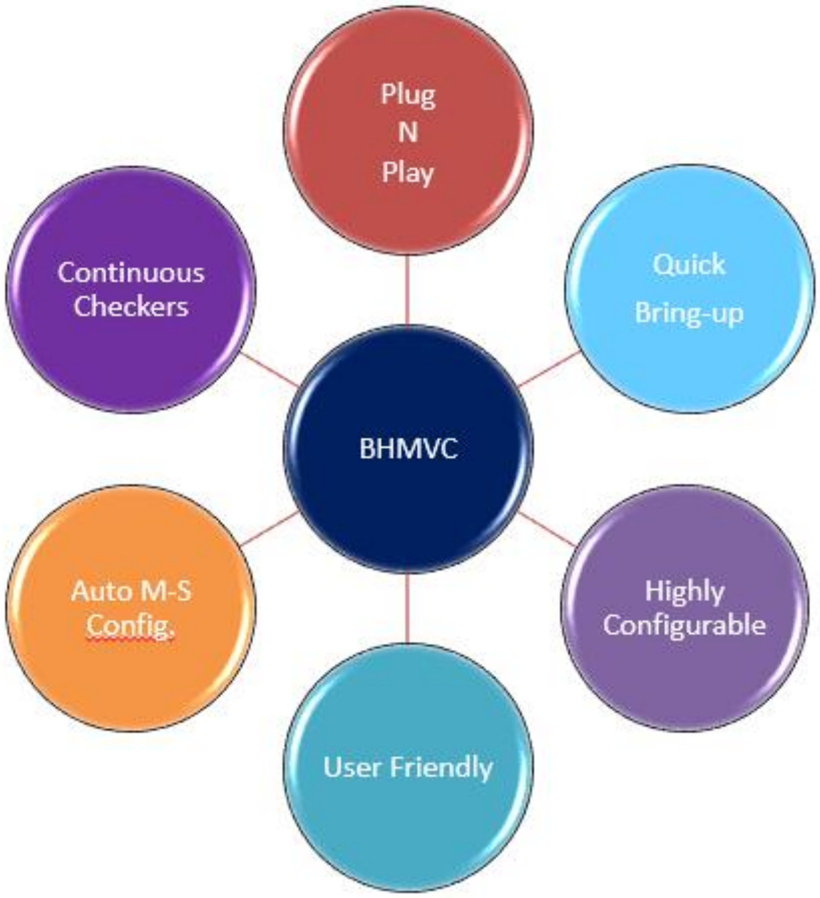


Figure 4 BHMVC Applications

Since for all BHM instances total ~800+ Design parameters need to be reviewed/verified and its highly time taking job with probable human-error "ParaHunter" helps in easing this effort. It will detect wrong parameterization between Specification and Implementation (IP-XACT) and dump out Block wise or SoC level reports to review it.

VI. PRELIMINARY RESULT

BHMVC and Para-Hunter resulted in Design Architecture Impact as it helped out in finding Critical bugs as stated below: -

A. ARCHITECTURE BUG FINDING

1. **Higher Throughput: -** Interconnects Outstanding Transaction handling capacity increased
2. **Area: -** **$\sim 500\mu m^2$** area saved due to wrong sub-module instances
3. **Power: -** Clock Gating support added to request clock for all master

B. IP BUG FINDING

1. **Bus Latency: -** Timeout Counter width increased for masters to cover worst case delay
2. Slave address alignment bug fixed for all masters **(>10 Blocks)**
3. Default value of periodic transaction interval updated for all masters **(>10 Blocks)**

VII. CONCLUSION

Major outcomes from BHMVC and ParaHunter solution includes: -

1. More than **300** tests added to verify all BHM combinations using this plug and play solution, with Turnaround time of **2-3 weeks** instead of **4 man-months**
2. Parameters **(~800)** verification reduced from **2 man-weeks** to **5-10 minutes**
3. Direct architecture impact with power, area saving and critical bug findings with ease of verification at SoC level