A Generic Verification Methodology for Chip to Chip Interrupt Handling in a Multi-Chip SoC (3DIC)

Vignesh Adiththan (vignesh.a1@samsung.com)
Padma Vutukuru (padma.v@samsung.com)
Lalithraj Mailappa (lalithraj.km@samsung.com)
Sekhar Dangudubiyyam (sekhar.d@samsung.com)
Samsung Semiconductors India Research, Bangalore, India

Abstract- **Complexity of System on Chip is increasing day by day due to advancements in Machine Learning, Artificial Intelligence and IoT technologies. Along with these advancements in technology, every device is desired to be handheld and portable. In order to achieve these, designers are striving to minimise the size of the devices by following various methodologies and techniques possible. One common technique is by reducing the transistor size and now a stage is reached where the transistor size can no longer be reduced further and there is a need for a different technique to make devices more portable. Three dimensional Integrated circuit (3D-IC) is one of the promising candidate to overcome the limitations of Moore's law due to its advantages like low power consumption, smaller form factor, higher performance and higher function density. A 3D-IC is a three dimensional integrated circuit built by vertically stacking different chips together in a single package which are connected together through silicon via (TSV). Our SoC consists of two chips stacked together in a single package and they are connected through TSV. Each chip has a processor cluster with four cores and an interrupt controller. Since each chip has an interrupt controller there rises a complexity when all the processor cores in a chip are busy or are in low power state and interrupt occurs to both the chips at the same time. This is where inter chip interrupt routing comes in handy. Using the inter chip interrupt controller the interrupts could be routed between chips and the interrupt controller could be used to send wake request to cores that are in low power as and when needed from one chip to another. Inter chip interrupt routing will be helpful in many low power applications and in applications where the traffic to a particular chip is very high**.

I. INTRODUCTION

Traditionally when there are multiple chips with multiple processor clusters in an SoC, each cluster will be associated with an interrupt controller and the interrupt associated with that cluster will be handled by that interrupt controller. This will be an issue when all the cores in a chip are busy servicing interrupts and another interrupt occurs in that chip. In this case the interrupt has to wait until a core becomes free to handle the interrupt. This is a major limitation when the interrupt controllers from different chips in the SoC are not connected so that they could handle interrupts from other chips in the system as and when they occur. In the SoC shown in Figure 1, there are two instances of GIC 600 that are connected to the processor clusters one at the top cluster and one at the bottom. Each GIC 600 supports 256 SPIs, SGIs and 36 PPIs. The communication between the cores and the GIC 600 happens through an axi stream interface (iri and icc) and the communication between the two GIC 600s happen through axi stream interface (icdr and icrd) as shown in the figure. Once the connection between the two GIC 600s is established then they transfer data through the axi stream interface signals to one another. A GIC monitor is integrated in order to monitor the icc and iri signals continuously. The data is shared in the form of packets in a stream interface and the packet contains data like interrupt id, priority, interrupt type etc. The data packets are decoded to get information regarding the interrupt being handled or the ones handled earlier.

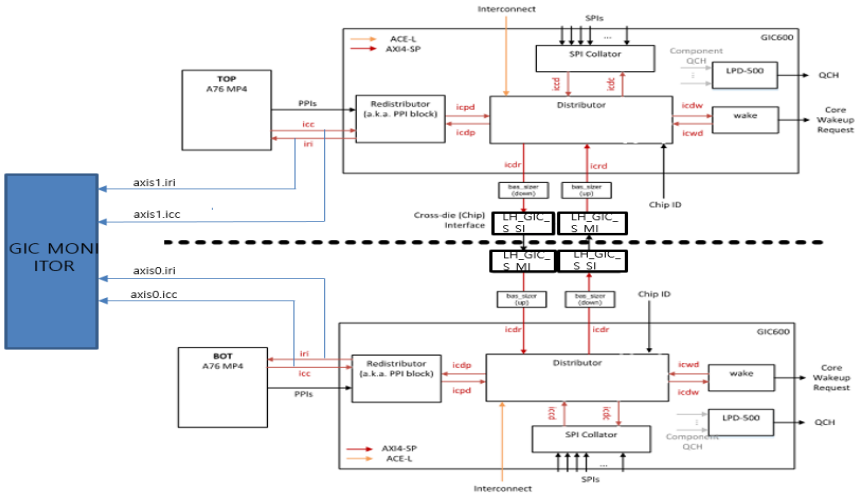

Figure 1. Overview of Gic600 and its integration in SoC

In multi-chip interrupt routing, interrupts are routed from one chip to another through the cross die interface signals and are handled by the cores in the chip which is not the source of the interrupt. The chip that sends the interrupts to be handled by the other chip is called the Routing Table owner (RT owner). The RT owner owns a set of interrupts which can be programmed according to user's requirement and the RT owner can route those specified set of SPI interrupts to the other chip once the connection is established. Then SPI_MIN and SPI_BLOCKS register fields are programmed to specify the interrupt IDs owned by a particular GIC. In order to generate interrupts with unique interrupt id, different set of values are given in SPI_MIN and SPI_BLOCKS field so that the interrupt numbers are different from the core point of view. In order to establish connection between GICs, the routing table owner is decided first. Then SPI_MIN, SPI_BLOCKS and socket state fields of the registers corresponding to both the GICs are programmed with desired values. The connection is established once the Routing Table Set (RTS) bit is set to consistent. The RT owner can be changed at any point of time just that there should not be any pending interrupts in the existing connection.

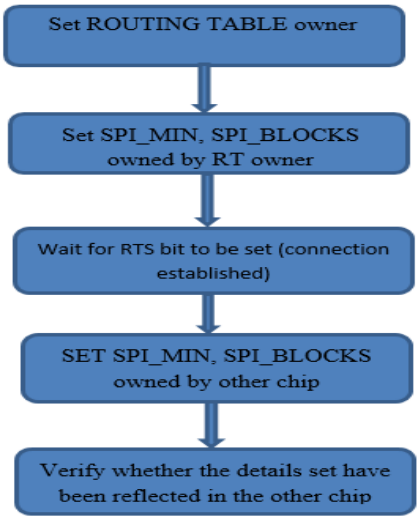

Figure 2. Flowchart of inter chip interrupt routing

II. APPLICATIONS

A. Routing Interrupts from one chip to another chip

In case of routing interrupts from Top chip to Bottom chip with unique interrupt IDs, the top chip is programmed to be the routing table owner and it is set with the blocks of SPI pins that it owns and the connection is established. Then the chip that is going to receive the interrupts from the top chip is set with the blocks of SPI it owns. The setting of spi_min and spi_blocks should be in such a manner that if spi_min and spi_block of the top chip is set as 0 and 8 respectively then bottom chip is set as 8 and 8, so that the top chip owns SPI pins 0-255 and the bottom chip owns SPI pins 256-511.

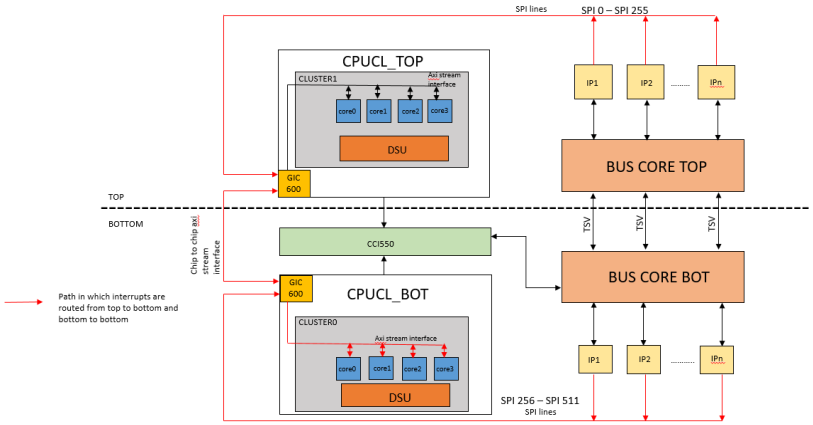

Figure 3. Routing of interrupts from top chip to bottom chip

As shown in the Figure 3, once the connection between the two GICs is established, data transfers will be visible in the chip to chip axi stream interface. Now all the interrupts from the top chip can be routed to the bottom chip using the router configuration registers available in the bottom chip. In this case, when an interrupt is generated from the top chip they are not seen on the axi stream interface between GIC and the cores in the top chip as indicated by the black line instead they are routed and handled by the bottom cores as indicated by the red line. Figure 7 shows the waveform of routing interrupts from top to bottom chip with unique interrupt ID. The interrupts being handled could be seen from the axi stream interface signals icc and iri and tdest shows the core in which the interrupt is being processed.

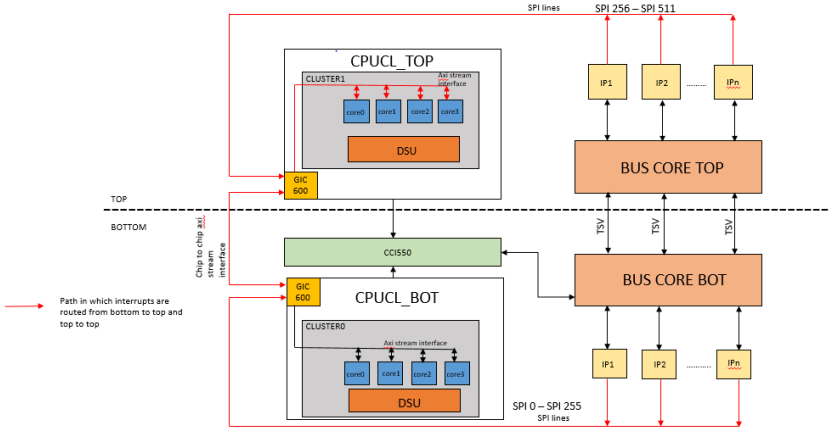

Figure 4. Routing of interrupts from bottom chip to top chip

Figure 4 Shows routing interrupts from bottom chip to to chip. In case of routing interrupts from Bottom chip to Top chip with unique interrupt IDs, the bottom chip is programmed to be the routing table owner and it is set with the blocks of SPI pins that it owns and the connection is established. Then the chip that is going to receive the interrupts from the bottom chip is set with the blocks of SPI it owns. The setting of spi_min and spi_blocks should be in such a manner that if spi_min and spi_block of the bottom chip is set as 0 and 8 respectively then top chip is set as 8 and 8, so that the bottom chip owns SPI pins 0-255 and the top chip owns SPI pins 256-511. Once the connection between the two GICs is established, data transfers will be visible in the chip to chip axi stream interface. Now all the interrupts from the bottom chip can be routed to the top chip using the router configuration registers available in the top chip. In this case, when an interrupt is generated from the bottom chip they are not seen on the axi stream interface between GIC and the cores in the bottom chip as indicated by the black line instead they are routed and handled by the top cores as indicated by the red line. Figure 8 shows the waveform of routing interrupts from bottom to top chip with unique interrupt ID. The interrupts being handled could be seen from the axi stream interface signals icc and iri and tdest shows the core in which the interrupt is being processed.

B. Inter chip low power wake request generation

Another important feature of Inter chip interrupt routing is generating wake request to cores that are in low power state to turn them ON. Applications which are power critical, in order to reduce power consumption, put the cores that are not functional at that moment in low power state and when a need arises for a core in low power to turn on and handle a certain task the GIC generates a wake request when a particular interrupt is targeted towards the core in low power state. Since the inter chip GIC communication is enabled, a core in a chip could be woken by the interrupt from the other chip.

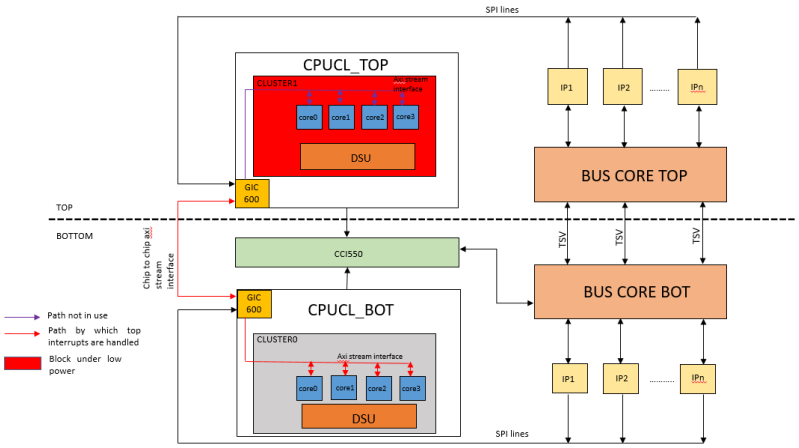

Figure 5. Low power wake request generation to top die

As shown in the Figure 5, the cluster at the top is in low power state. When an interrupt is generated from the top chip the normal path by which it will be handled is indicated in purple line. But since the cores are in low power and since both the GICs are interconnected it is routed to the bottom cores as per the router program for handling it. And in case where the bottom cores are busy and are not able to handle the interrupt then GIC can route to the top and generate a wake request to the low power core and wake it in order to handle the interrupt. Once the wake request is generated, it changes the current power state of the particular core and wakes up the core through the central Power Management Unit (PMU). Once the GIC generates wake request to a particular core in low power state then there is handshake mechanism that happens between the central power management unit and the cores by which the wake request for a particular core is processed the PMU. Once the PMU receives the request for powering ON a core the M0+ processor which is a part of the always ON block and which has access to all the blocks of the design is used to program the P channel registers required to power ON the core. After successful powering ON the interrupt that

raised the wake request is handled by that core. Figure 9 shows the waveform of wake request being generated to the low power top core for handling the interrupt. The wake request is generated once the interrupt is routed to that core as seen in the waveform. The interrupts being handled could be seen from the axi stream interface signals icc and iri and tdest shows the core in which the interrupt is being processed.

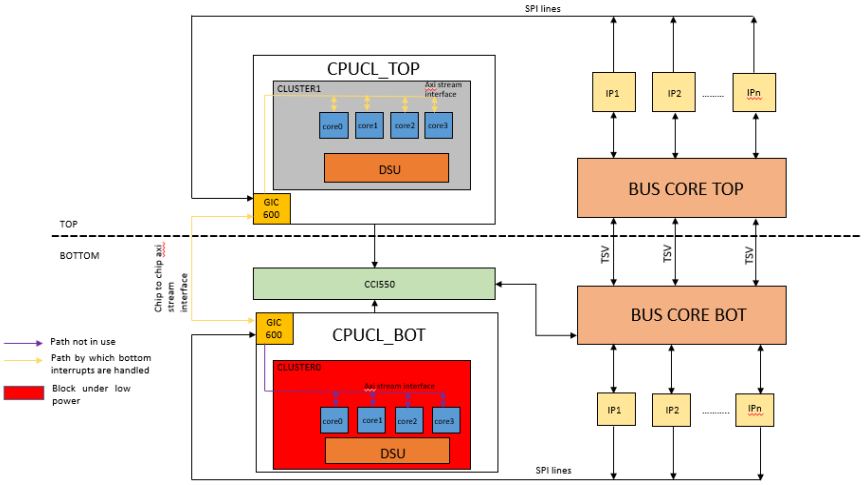

Figure 6. Low power wake request generation to bottom die

Figure 6 shows that the cluster at the bottom is in low power state. When an interrupt is generated from the bottom chip the normal path by which it will be handled is indicated in purple line. But since the cores are in low power and since both the GICs are interconnected it is routed to the top cores as per the router program for handling it. And in case where the top cores are busy and are not able to handle the interrupt then GIC can route to the bottom and generate a wake request to the low power core and wake it in order to handle the interrupt. As mentioned above, once the wake request is generated to switch ON a particular core handshake happens between PMU and the cluster and the required P channel programming is done by M0+ and the core is powered ON. Figure 10 shows the waveform of wake request being generated to the low power bottom core for handling the interrupt. The wake request is generated once the interrupt is routed to that core as seen in the waveform. The interrupts being handled could be seen from the axi stream interface signals icc and iri and tdest shows the core in which the interrupt is being processed.

C. Dynamic Interrupt off loading

GIC has the intelligence to dynamically route the interrupt to cores of its choice depending on workload and the current power state of the cores. This can be utilized by programming the Interrupt Routing Mode register after establishing the connection between the chips. Once we set the Interrupt Routing Mode register, the GIC by itself routes the interrupts to the cluster of its choice depending on the workload of the cores. In this mode the user need not specify the cluster and core which has to process the interrupt as the Interrupt Routing Mode setting overrides the Router register settings.

Using the Interrupt Routing Mode setting, the interrupt is handled by GIC in the following manner in these cases mentioned below:

1) When all cores except one is in low power mode in a cluster and the core which is ON is free then the GIC routes the interrupt to that core for it to be handled. Figure 11 shows the waveform of this case in which cores 0, 1, 2 are in low power mode and the GIC has routed the interrupt to core 3 which is ON and free to accept the interrupt for processing.
2) When all cores except one is in low power mode in a cluster and the core which is ON is busy and not able to accept the interrupt for processing, the GIC has the intelligence to send wake request to the cores in low power state and wakes the core for handling the interrupt through PMU handshake mechanism mentioned

earlier. This is achieved by programming register field that enables GIC to send wake request to cores in low power state along with the Interrupt Routing Mode setting. Figure 12 shows the waveform of this case in which cores 0, 1, 2 are in low power mode and core 3 which is ON is disabled from processing the interrupt through register programming so that it is not available for routing the interrupt. So in this case we find that the wake request is generated to core 0 for powering it ON and hence handling the interrupt.

III. RESULTS

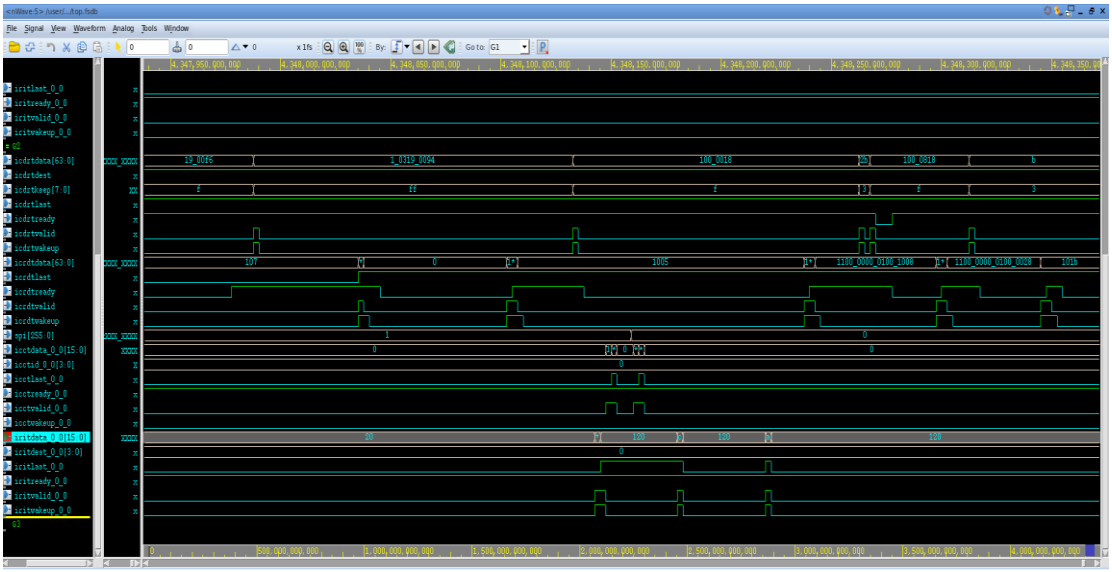

Figure 7. Waveform of inter chip interrupt routing from top chip to bottom chip with unique id

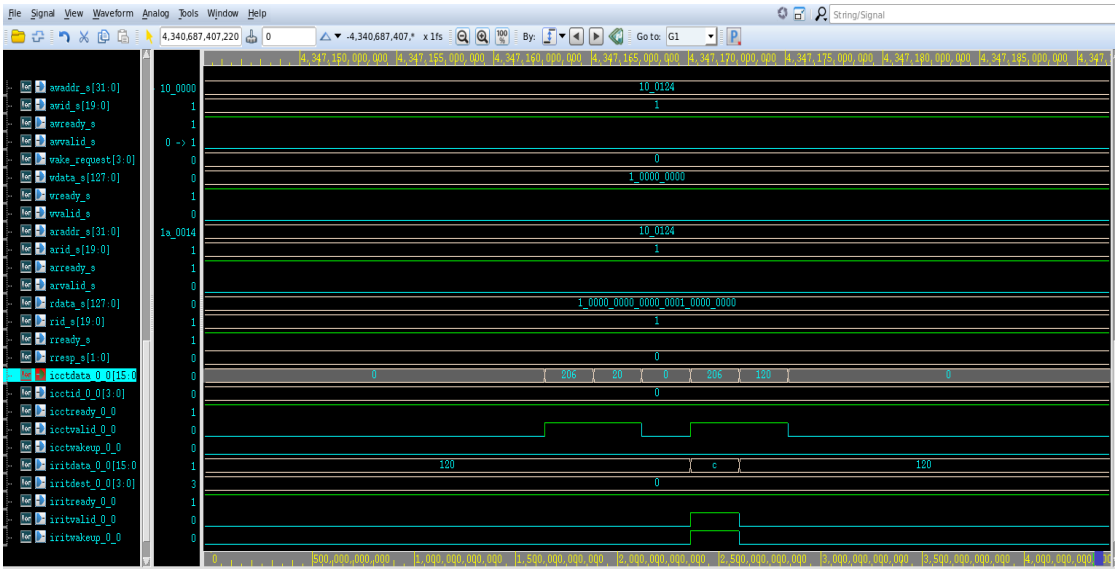

Figure 8. Waveform of inter chip interrupt routing from bottom chip to top chip with unique id

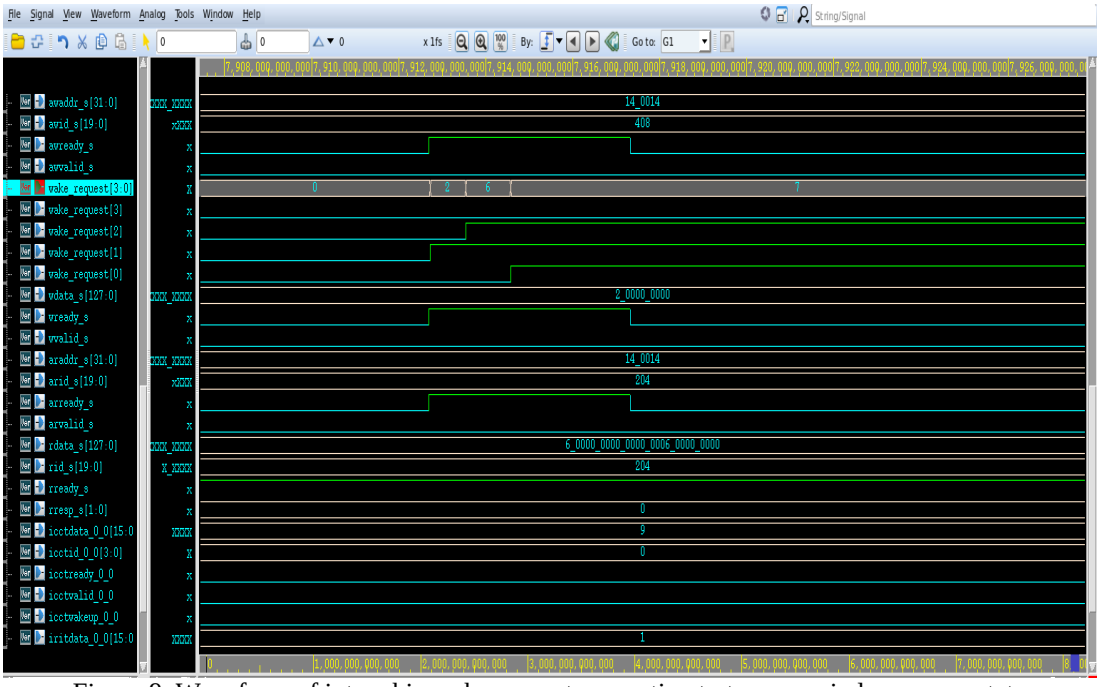

Figure 9. Waveform of inter chip wake request generation to top cores in low power state

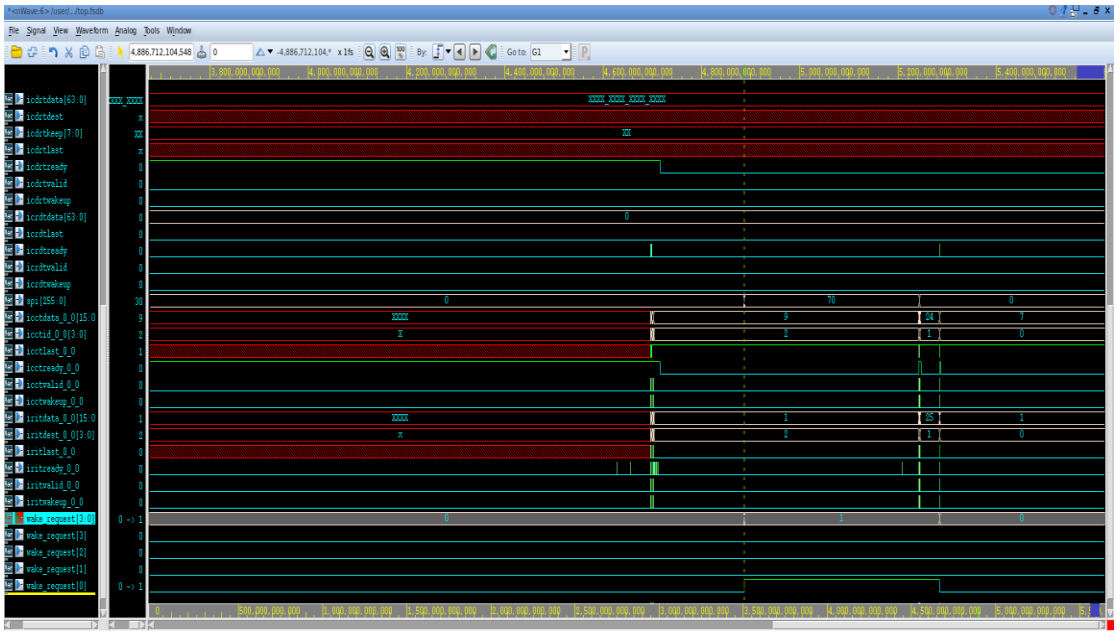

Figure 10. Waveform of inter chip wake request generation to bottom cores in low power state

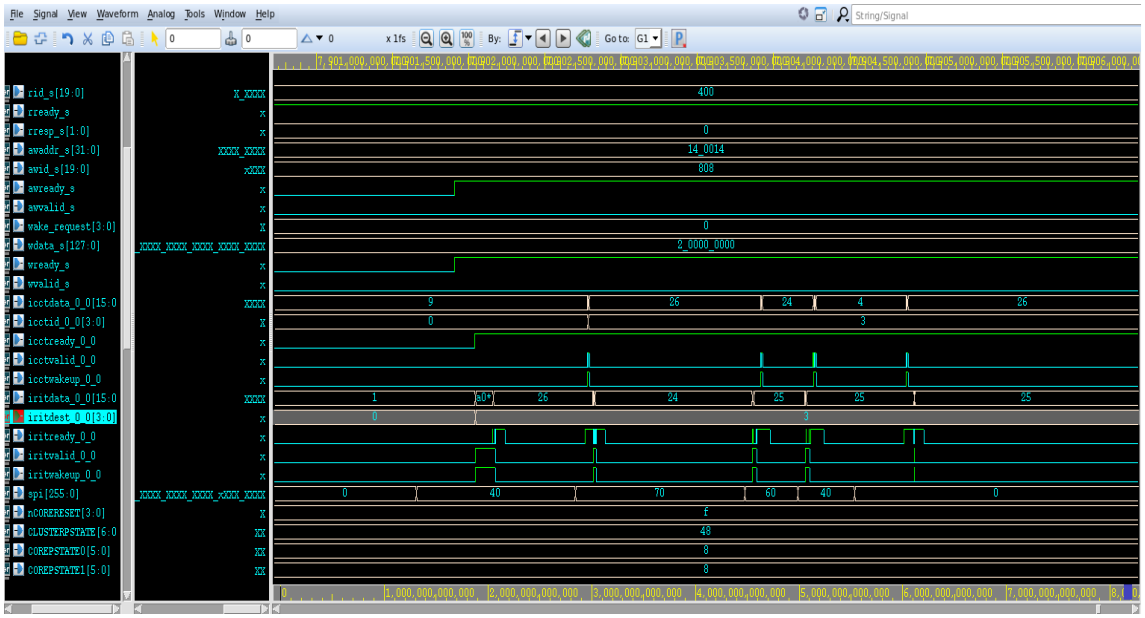

Figure 11. Waveform of dynamic interrupt offloading to ON core

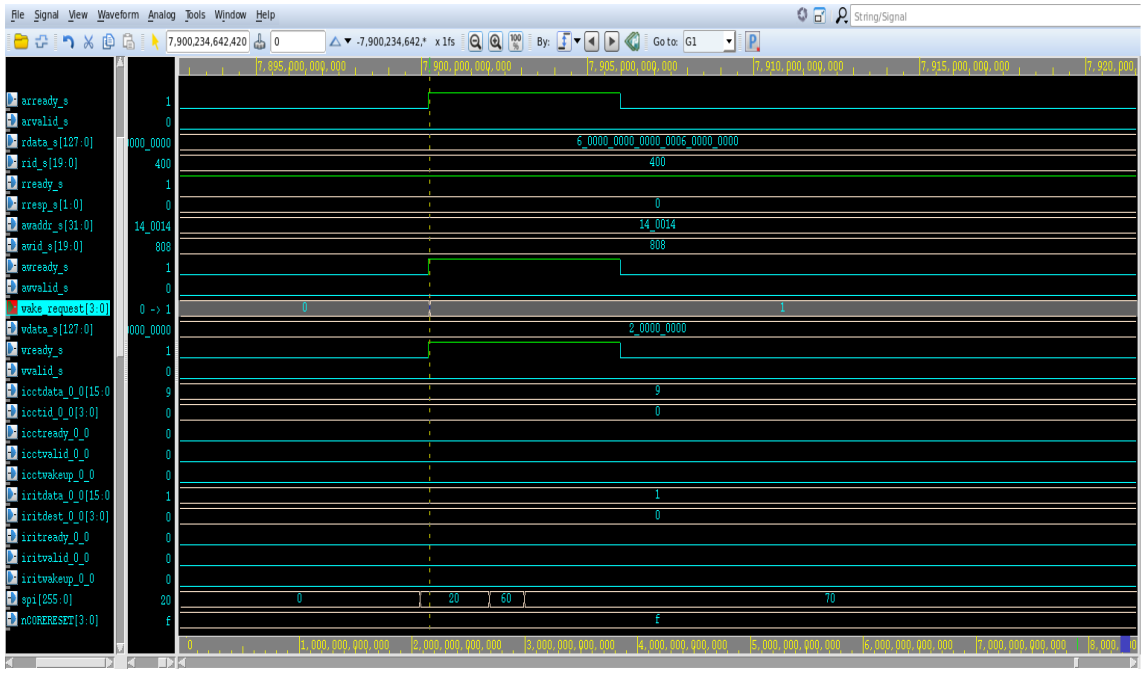

Figure 12. Waveform of dynamic interrupt offloading by wake request generation to OFF core

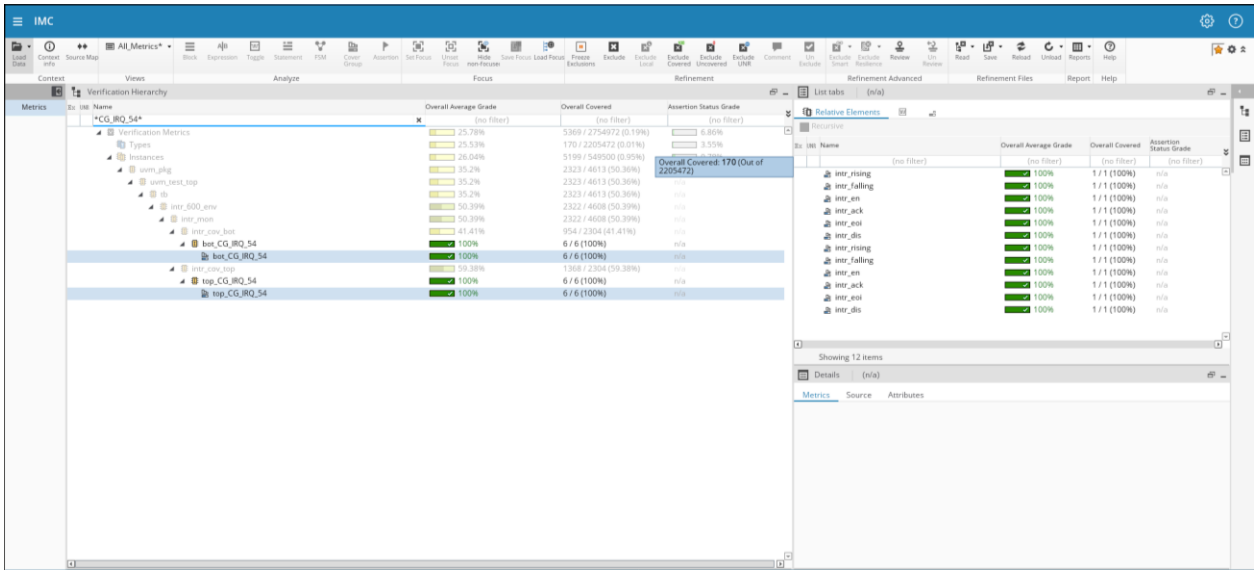

Figure 13. Functional coverage of interrupts from top and bottom chips

1) Co Simulation based interrupt verification Environment development

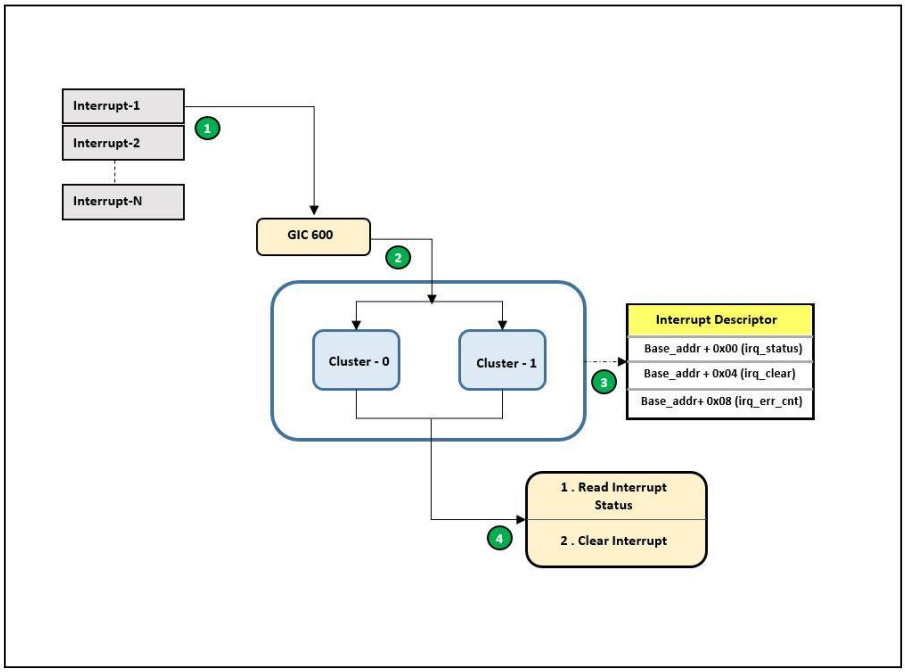

Figure 14. ISR handling flow in a co simulation environment

In the Co-Simulation environment ISR handling is executed by the cores and the interrupts are generated functionally in the SV environment through AHB transactor. GIC will monitor these interrupts continuously and maps it to cores present in the Clusters. Upon receiving the interrupt, core fetches the ISR descriptor from DRAM (ISR_DESCRPTORBASE_ADDR points to DRAM) based on the INTR ID and service it. These descriptors can be

copied to stack allowing the processor to fetch and service interrupts at shortest possible time. SPIs, PPIs are functionally generated in the System Verilog Environment by the IP owners. GIC 600 continuously monitors the interrupt generated from the SV environment, which has the information related to Interrupt ID, Priority and Type. If single/multiple interrupts are generated in both top and bottom chip, then each GIC will monitor and route it to the corresponding cluster-cores based on the GIC configuration. As soon as the GIC routes the interrupt to corresponding core now it is the responsibility of the core to service and clear it. A C-Code is implemented in such way that it reads the interrupt ID which is being currently routed to that particular core and fetches the interrupt status/clear register information and also error counter information based on the interrupt type. Descriptor information can be added/updated through backdoor method which doesn't require C-programming. Based on descriptor information processor can read status and clear the interrupts functionally.

2) This framework helped to build a robust system level power down scenarios. A wide variety of use cases and all possible scenarios with respect low power state of the system were covered. From these scenarios, a significant amount of power saving was evident with respect to both the chips and it was achieved with low latency due to the inter chip interrupt routing technique.
3) This generic framework can be ported to emulation/software directly and used at their end for interrupt verification.
4) This framework can be easily ported to projects that have multiple chips irrespective of the number.
5) This framework can be extended to multiple GICs without any hassle if there arises a need in the future.
6) Using a single ISR image all the functional interrupts were verified from both the chips by assigning unique interrupt ids for the interrupts from top and bottom chips. This helped in reducing the time taken to functionally verify all the interrupts in system associated with both the chips.