DFD VERIFICATION IN X86 SoC

[#1]Sireesha Vasipalli, M.Tech in Digital systems and Computer electronics

[#2]Dr. D. Vishnu Vardhan, Assistant professor

[#1,2]Dept of ECE, JNTUA, Ananthapuramu.

ABSTRACT:

SoC will typically integrate a CPU, graphics and memory interfaces, hard disk and USB connectivity, random access and read only memories and secondary storage on a single circuit die. Design-for-Debug (DFD) is usually a non-functional logic within the chip, which helps in debugging a failure on Silicon. Due to lack of in-system controllability and Observability, DFD hardware insertion has become a necessity. DFD logic gives the Observability of needed signals, internal states and registers of SoC, which are needed to find out what went wrong. DFD is different from Design-for-Test (DFT) logic, which is generally used to detect manufacturing defects.

DFD verification at SoC mainly involves the verification of various DFD features. This project aims at dealing with verification of such features by simulation. Debug Master, Debug Bus, Debug Clients, Debug Events & Acks, Debug Filters and Debug Buffers are such Debug Logic in X86 SoCs, which will provide Controllability & Observability on Silicon. Functionality of these Debug Logic needs thorough verification before tape-out, without which, debug on silicon will not be reliable.

Index Terms: programmable bus, debug bus, synchronous and source synchronous debug bus.

I. Introduction:

Today's devices are highly integrated, enabling a single chip to perform many functions such as networking, wireless etc. Each function is done by an Intellectual Property (IP). Grouping all the IPs and allowing them to communicate on one chip is called System on Chip (SoC).

The ever-increasing demand for shorter time to markets has led to the escape of hidden bugs into the silicon. However, debugging in the post-silicon phase is different from the pre-silicon phase from different viewpoints. In pre-silicon debugging, ultimate controllability and Observability are possible through simulation and formal verification tools.

II. Block Diagram Representation of Debug bus:

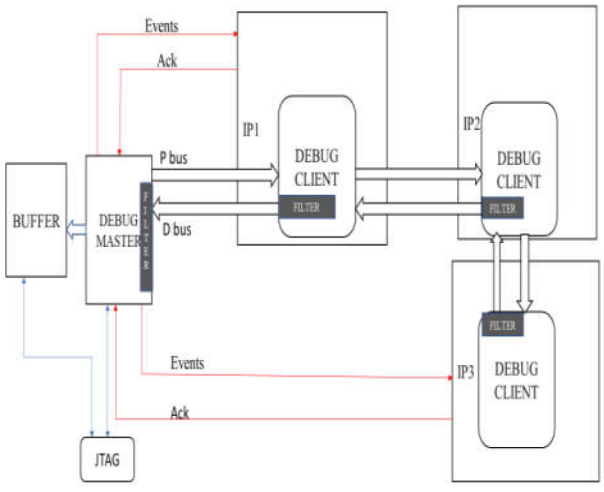


Fig.1 Debug Bus Architecture

SoC is composed of different sub-blocks. Wherever, it is required debug logic is inserted in these sub-blocks and are connected. Connectivity between these debug logics will facilitate - entry to debug mode, route required debug data, collect debug data into buffers, and for hand shaking in some cases.

III. Programmable bus:

Programmable bus is shared between the debug master and different debug clients. Multiple debug clients will be connected to programmable bus, but only one of them can be programmed at a time. Each debug client is assigned with a unique block identification number. All the debug clients connected in a series manner. If we want to program the farthest debug client, then make sure that remaining all debug clients should forward the data. Through programmable bus debug client can be programmed to place needed debug data on debug bus.

IV. Debug bus:

Multiple debug clients connected with debug master in a daisy-chained fashion to create a debug bus. The debug clients can be programmed to capture the data from IPs and pass the data to debug master. Through the debug bus, we can check the functionality of each debug client. If multiple debug clients respond at a time, then the result is based on priority. Nearest debug client to debug master have more priority. Only selected debug client sends the data to debug bus, remaining all debug clients will simply forward the data.

1. Synchronous debug bus

A synchronous debug-bus operates with single clock i.e., the debug master and debug clients are in same clock domain. Debug client does not need to send the clock signal along with debug data to debug master for the purpose of synchronization. Below figure represents the synchronous debug bus and its connections to the debug master and debug clients. Here, IP1, IP2 and IP3 are working with same clock i.e., clka. Through programmable bus debug client is programmed and through debug bus, data is received at debug master from debug client.

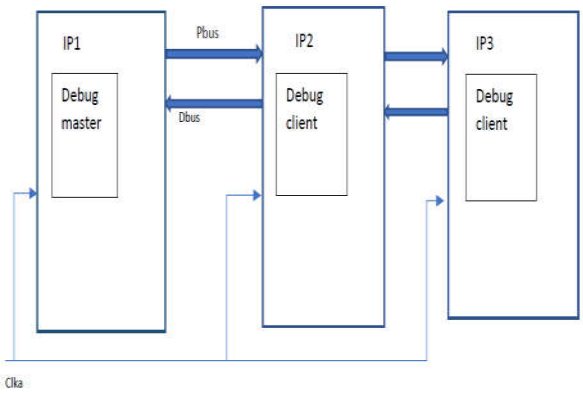


Fig 2. Synchronous debug bus

2. Source Synchronous Debug bus

Source synchronous data transfer refers to the technique of having the transmitting device send a clock signal along with the data signals. Here, the debug client itself contains a clock i.e., in different clock domain. Thus, debug master and debug clients have different clocks. In source synchronous, a synchronizer is present in debug master. Below figure represents a source synchronous debug bus and its connections to the debug master and debug clients. Here, IP1, IP2 and IP3 are working with different clocks i.e., clka, clkb and clkc respectively. Here debug clients receive data signal (P bus) along with cclk signal and in return it sends debug data along with dclk signal as shown in below figure

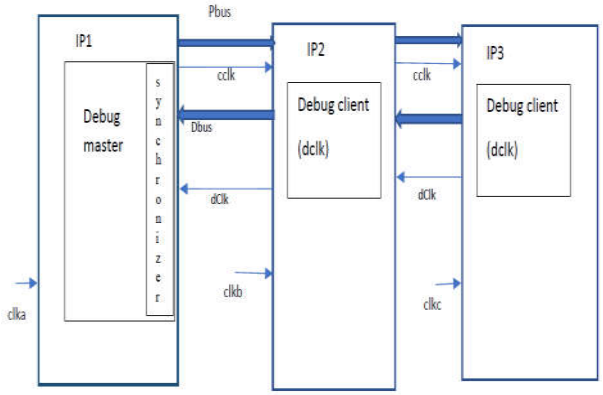


Fig 3. Source Synchronous debug-bus

V. Debug Buffer, Events and Acks

Debug buffer simply acts as a memory which stores the debug data coming from debug unit. Events represent a request from debug master to different IPs to enter debug mode. In return IPs generates Acks to debug master as a response.

VI. Verification of Debug bus and Results

Debug bus verification by considering invalid block Id's:

Each debug client assigned with a unique block id number. This test intends verification of a debug client by driving both invalid and valid block id numbers.

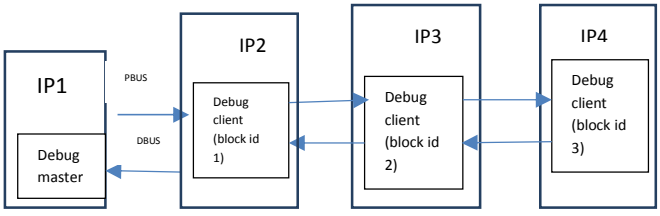


Fig 4: debug bus connections to debug clients.

Following are the steps involved in the verification of debug clients.

1. First step is to declare an array with both valid and invalid block ids. 5 is the invalid block id.

2. Select block ids for programming by using an array.

3. After block id programming, expected data was received at debug bus for valid block ids and zero is received for invalid block ids.

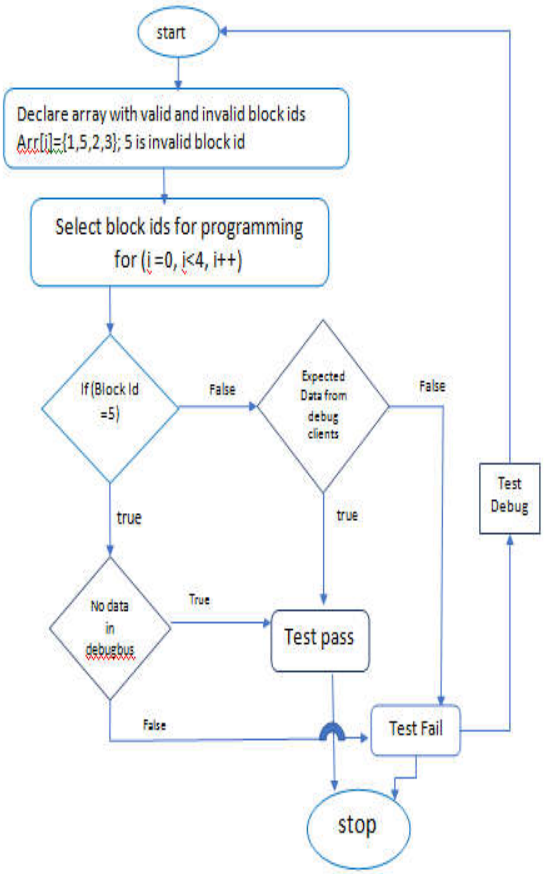
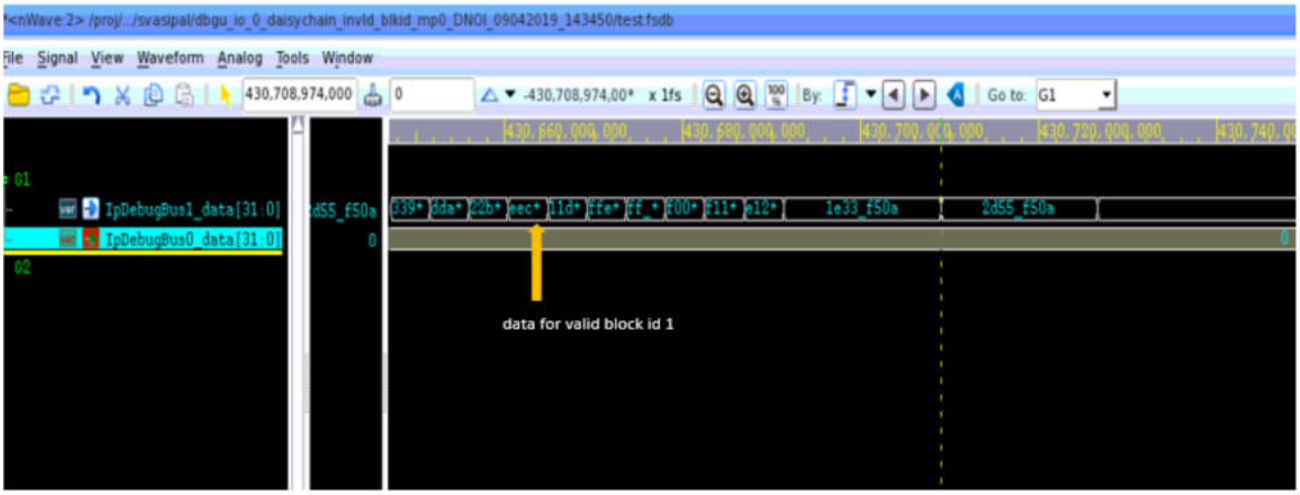
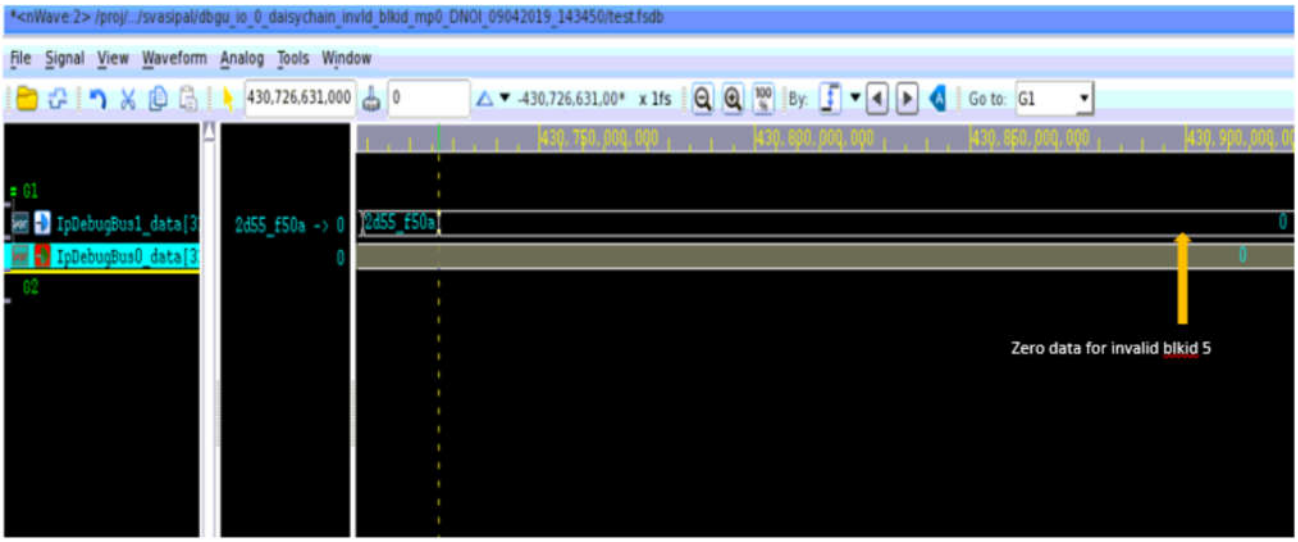
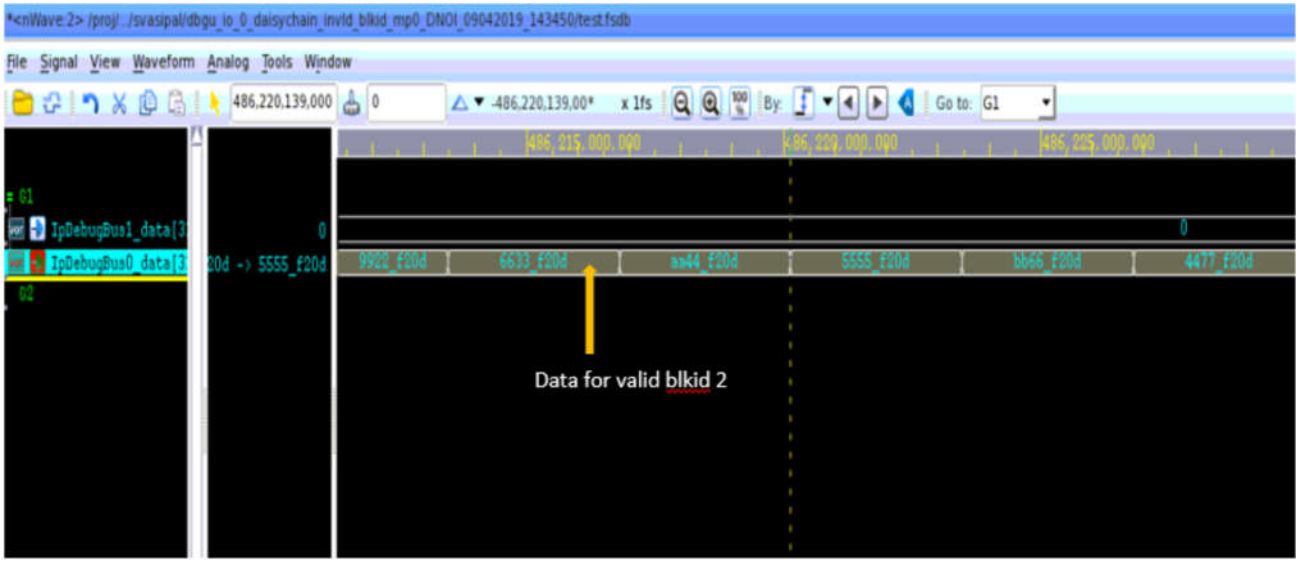


Fig 5: debug bus verification flow.

Following are the waveforms to verify the results, which indicates data for invalid and valid block id's as shown in below figures,


File Signal View Waveform Analog Tools Window
IpDebugBus1_data[31:0]
IpDebugBus0_data[31:0]
data for valid block id 1


File Signal View Waveform Analog Tools Window
Zero data for invalid blkid 5


File Signal View Waveform Analog Tools Window
Data for valid blkid 2

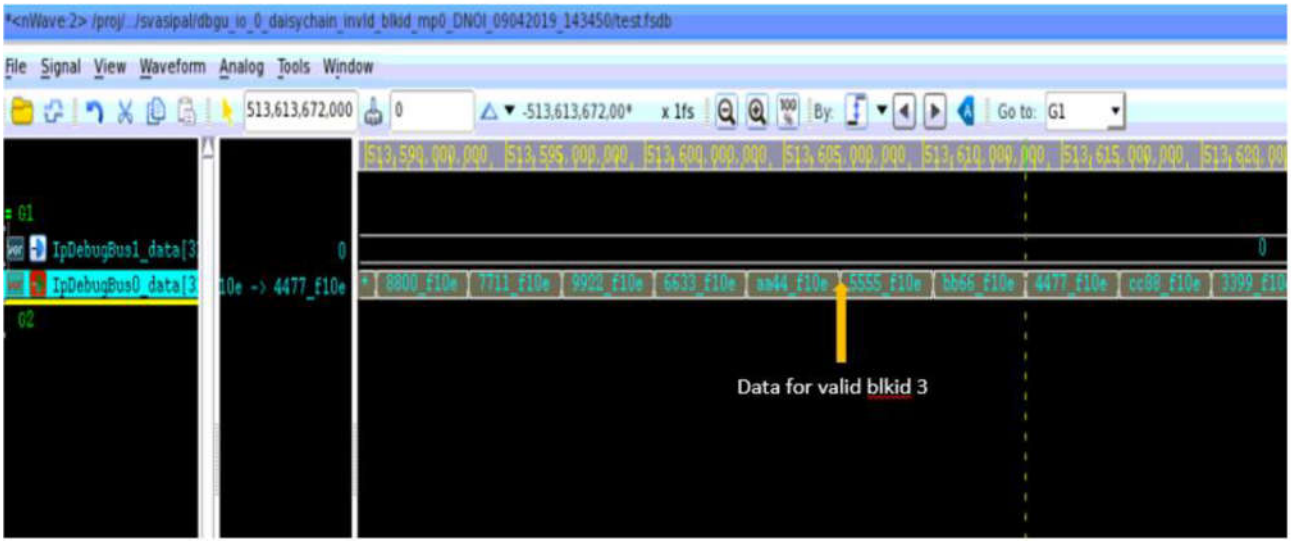


Fig 6: waveforms to verify results

Debug bus verification during power gating of IPs:

Multiple debug clients connected with debug master in a daisy-chained fashion. Each IP contains a debug master or debug client. Here different IPs are in different power domains and the last IP response is verified by power gating the intermediate IPs. There is a pass-through logic in each debug client, which is not power gated. Pass through logic simply forwards the data to further debug clients. Both programmable and debug bus are connected to the pass-through logic, which is not power gated. Here debug master and pass through logics are in same power domain, which is power on, and simply forwards the data even though IPs are power gated. From the figure, IP0, IP1 and IP2 are in different power domains i.e., PD0, PD1 and PD2 but debug master and pass through logics are in same power domain i.e., PD3.

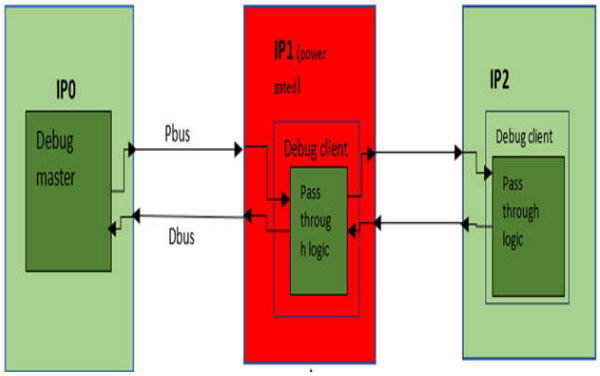


Fig 7: debug master and debug clients with pass through logic.

Following are the steps involved in the verification of debug clients.

1. Here debug clients and debug master are in different power domains. Test intention is to verify the debugbus connections between different power domains. First step is to select different block ids 1 and 2.

2. Here debug master and pass through logics are in same power domain, which are not power gated and simply forwards the data.

3. Program the last debug client i.e., 2 to verify its debugbus data at debug master.

4. Power down the intermediate IP i.e., IP1 to verify, whether it is still passing the data to debug master or not.

5. In the final step, expected data was received at debug master from last debug client i.e.,2 without being blocked at IP1.

Following flowchart represents the verification of debug bus by power gating IPs. The waveforms obtained from above verification is given in below figures,

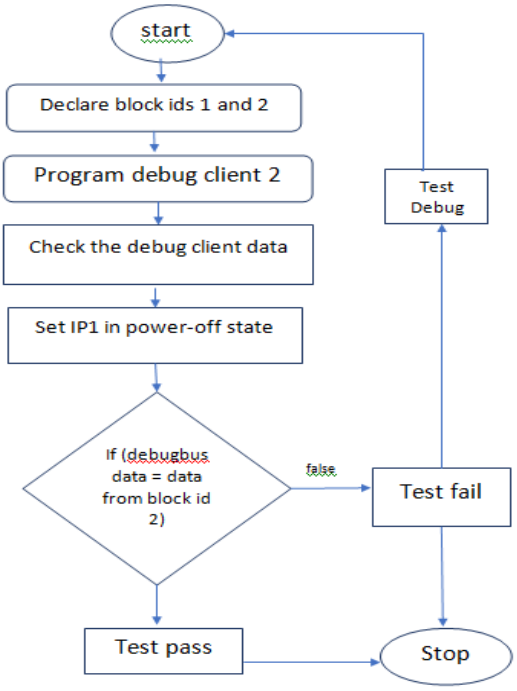


Fig 8: verification flow of debug bus during power gating of IPs.

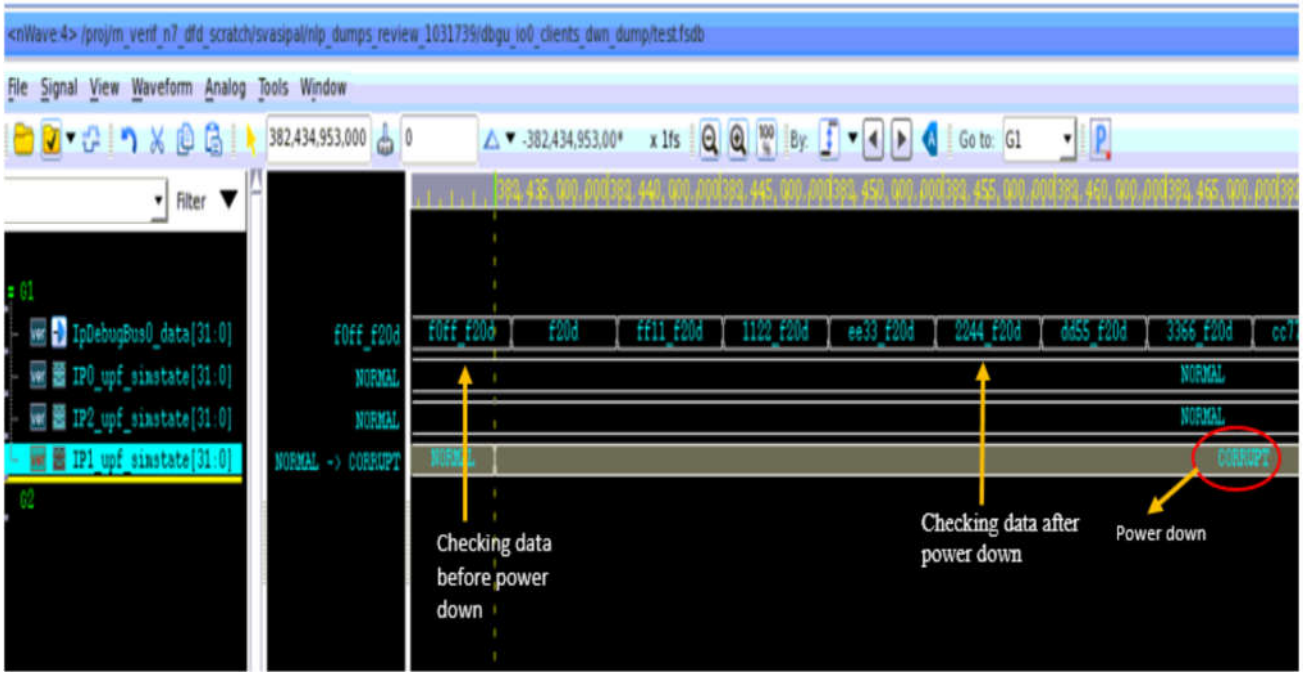


Fig 9: waveforms to verify results

Conclusion:

In this project, I am verifying "design for debug" which is used to find the bugs in post silicon. DFD represents an extra RTL logic which is added to SoC, to achieve controllability and Observability. DFD verification of SoC mainly involves the verification of various DFD features. Here I have verified debug bus only with few scenarios.

References:

- Zhang Peng, Fan Xiaoya, Huang Xiaoping "An on-chip debugging method based on bus access" School of Computer Science, Northwestern Polytechnical University Xi'an, China Xteeq734801864@qq.com
- Masahiro Fujita "Post-silicon verification and debugging with control flow traces and patchable hardware" VLSI Design and Education Center (VDEC) university of Tokyo, CREST/JST Tokyo, Japan.
- Nishit Gupta, Sunil Alag "Unified Approach for Performance Evaluation and Debug of System on Chip at Early Design Phase" R&D in Electronics Group, Department of Electronics & Information Technology.
- Gustavo R. Alves and J. M. Martins Ferreira "From Design-for-Test to Design-for-Debug-and-Test: Analysis of Requirements and Limitations for 1149.1" 1ISEP / DEE Rua de S. Tome 4200 Porto – PORTUGAL