Key Gochas in implementing CDC for various Bus Protocols

Nikita Gulliya, Agnisys Technology Pvt. Ltd. Noida, India, nikita[at]agnisys.com
Mukesh Kumar Singh, Agnisys Technology Pvt. Ltd. Noida, India, mukesh[at]agnisys.com
Abhishek Bora, Agnisys Technology Pvt. Ltd. Noida, India, abhishek[at]agnisys.com

***Abstract* – In the industry where the designs are getting complex with huge hierarchy, there is often a requirement of multiple clock domains. But while, multiple clock domains exist various complications occurs, such as setup and hold time differences which further leads to a metastability state. Some of these errors can be caught at an early stage in the design while few occur at a later verification stage. Other issues include violation of the protocols. The 2-D flip flop technique only solves the CDC issues for 1-bit transaction. Automation has been done for the generation of proper code of design and verification from the specification itself for multi-bit transactions eliminating all the CDC related issues. SoC level specification can be used for the generation of correct design code for clock domain crossing, covering all the scenarios and various bus protocols. CDC has been implemented between various bus protocols used in the industry and the custom bus which can be user defined. The paper will talk about the simulation results obtained for the implementation of a low power RTL design and techniques used for interconnection with various bus protocols.**

I. INTRODUCTION

Often the design demands dealing with various clock domains. Specially when the design is implemented with a user defined decode logic working on a custom bus. The interfacing of the custom clock with the widget has always been a challenge. The interfacing can be between custom clock and any other bus protocol, the frequencies of the two clocks may vary in this case leading to various issues such as metastability, loss of data, data incoherence etc. These issues have been resolved and automated, the code for design and verification in terms of clock domain crossing can be generated from the user defined specification itself.

The figure 1 shows the CPU and IP Hardware software interface(HSI). The CPU HSI is the instruction set architecture like ARM, RISC-V etc. The CPU contains bunch of register in it, C/C++ program with the help of compiler is converted in Assembly. The assembly code is burnt on to the CPU. With the help of interconnect fabric, which can be based on any bus protocol such as AMBA- AXI, APB, AHB. Various programmable slaves and memories can be connected, further with the help of bridge which can be based on some other bus protocol, more slaves can be connected. These slaves are programmed by reading/writing embedded registers. As depicted in the figure, the CPU HIS can be divided into two parts. The Software side and the Hardware side. The software side or the bus interface is working on a different clock frequency than the Hardware side or the custom clock domain. The programmable slave side is termed as the custom clock domain. For instance, the clock from the bus interface can work on a fast frequency and the custom clock can work on a slow frequency, due to the power saving requirements, the communication between the fast clock from the bus interface and the slow clock from the custom clock interface can be achieved with the help clock domain crossing.

The clock domain crossing has been implemented with respect to two clocks. The first clock is the custom clock that the user must be working on with its own application logic from the slave side, and the other clock is from the bus domain termed as the master clock. Both these clocks must be working on different clock frequencies, automation has been done for various bus protocols, which includes, AMBA-AHB, AXI, APB, Tilelink etc.

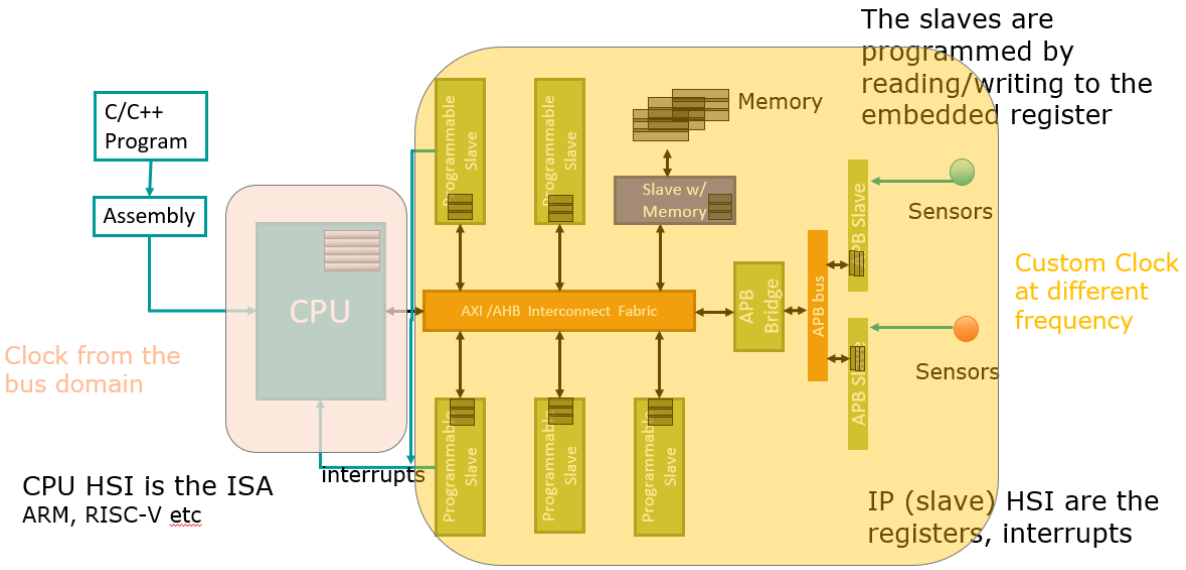

Fig 1: CPU and IP Hardware Software Interface

The clock domain crossing technique has been implemented to handle different clock frequencies for the slave, from both the hardware and software side.

II. CDC from Software Side

The following will show the clock domain crossing technique from the software i.e the bus interface clock to the custom clock slave for various bus protocols.

A. CDC for AMBA-APB Bus Protocol

According to the APB Bus protocol, when the Psel signal from bus domain side is high then the bus is in setup phase. In the next clock cycle when the Penable signal is high then the bus is in the access phase. Pready should be high in the access phase for read or write transaction to take place. But Pready signal is stretched to low in the access phase and no read or write transaction signals are high during this phase till the custom clock is matched. By default, the custom request for transaction is always high it passes through 2-D flip flop method. If APB clock domain gets custom request high and access phase is also present then APB bus domain generates an acknowledge signal.

First the transaction which consists of the address, data and control is ready from the custom clock side, at the same time when the master clock domain is in its access phase, then in the next clock cycle, in address, data and control goes in its corresponding flop, where control signal consists of Pstrb and Pprot for byte enabling and protection respectively. At the same time, acknowledge signal is asserted from the bus domain side to indicate that the transaction has been completed for corresponding custom request. This is done through the 2-D flip flop method.

In the next clock cycle, the request for response is asserted corresponding to the transaction. The response from custom domain indicates whether the read or write transaction has been completed and with or without any error. For all the above transaction the Pready signal will be low.

Therefore, first the the request is high from the custom side for the transaction, then the acknowledge signal is asserted from the bus domain side for the corresponding request and the request for response is asserted at the same clock cycle. After passing through 2-D flip flop method, when request for response reaches custom bus domain, the acknowledge is provided from the custom bus side that the read or write transaction has been completed from its side.

When custom bus domain receive acknowledge from APB bus domain then the request is de-asserted from the custom domain side and acknowledge is also de-asserted. When the acknowledge is low, request will again become high. Therefore, handshaking synchronization takes place.

When the acknowledge is received from the custom bus end then the pready goes from low to high to indicate the transaction has been completed and it is again asserted to low. The response is captured with the help of PSLVERR signal.

In figure 2. The handshake synchronization for AMBA-APB Bus domain can be seen.

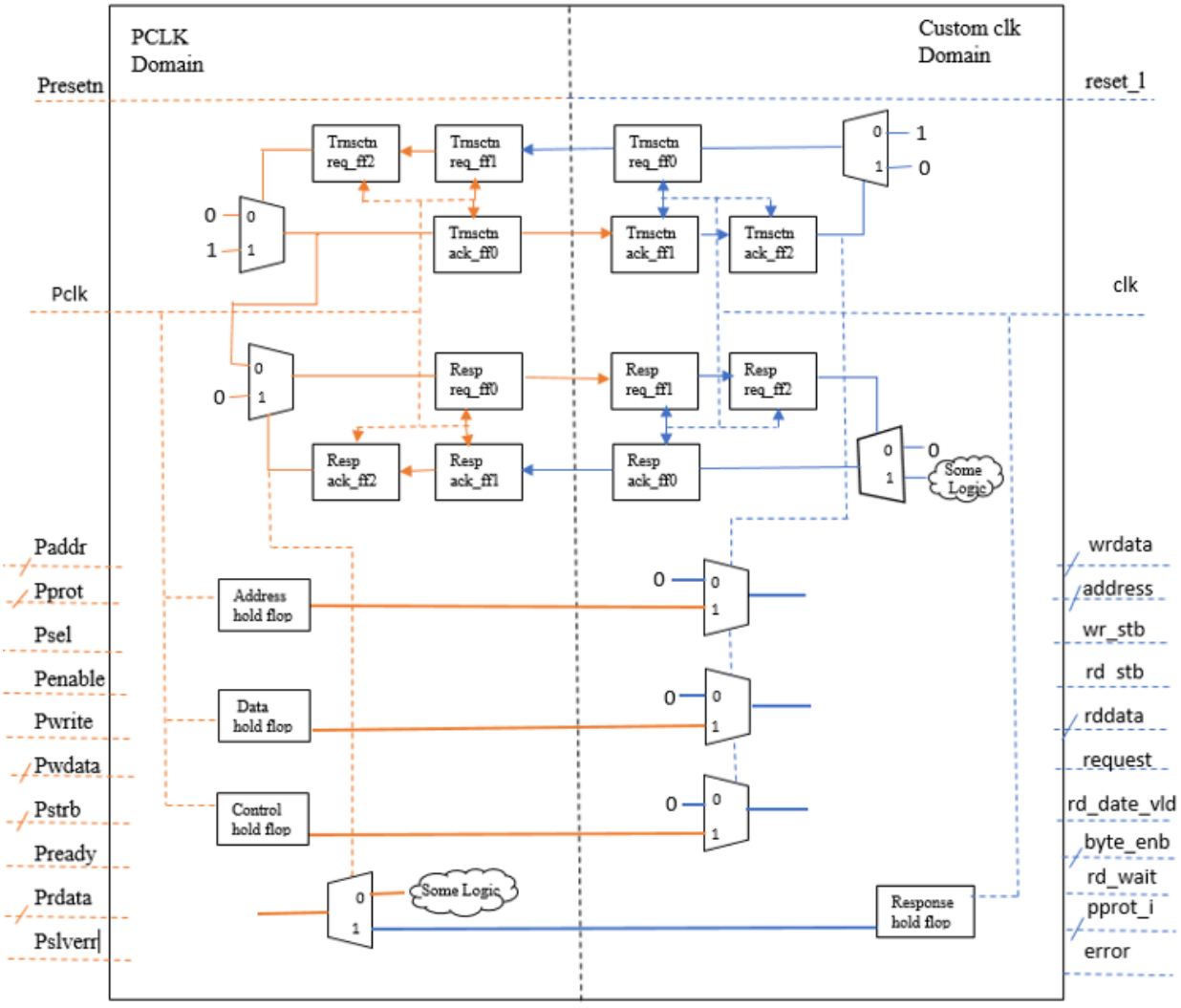

Fig 2: Handshake Synchronization for AMBA-APB Bus domain

The simulation waveforms shown in figure 3 below shows the simple read-write transaction in case of CDC for APB bus domain. It shows when the access phase is present, we keep Pready signal low to indicate that the slave has not completed the transaction. When the slave completes the transaction with the help of handshake synchronization technique, pready is again asserted high for the next transaction.

As shown in the figure 3, at address 0x0, the data 75e51501 is written, then after completing the handshaking process, pready is asserted to indicate that write transaction has been completed. In the next clock cycle, a read transaction takes place on the same address, and when handshake synchronization is completed, pready is again asserted high and valid pdata is obtained.

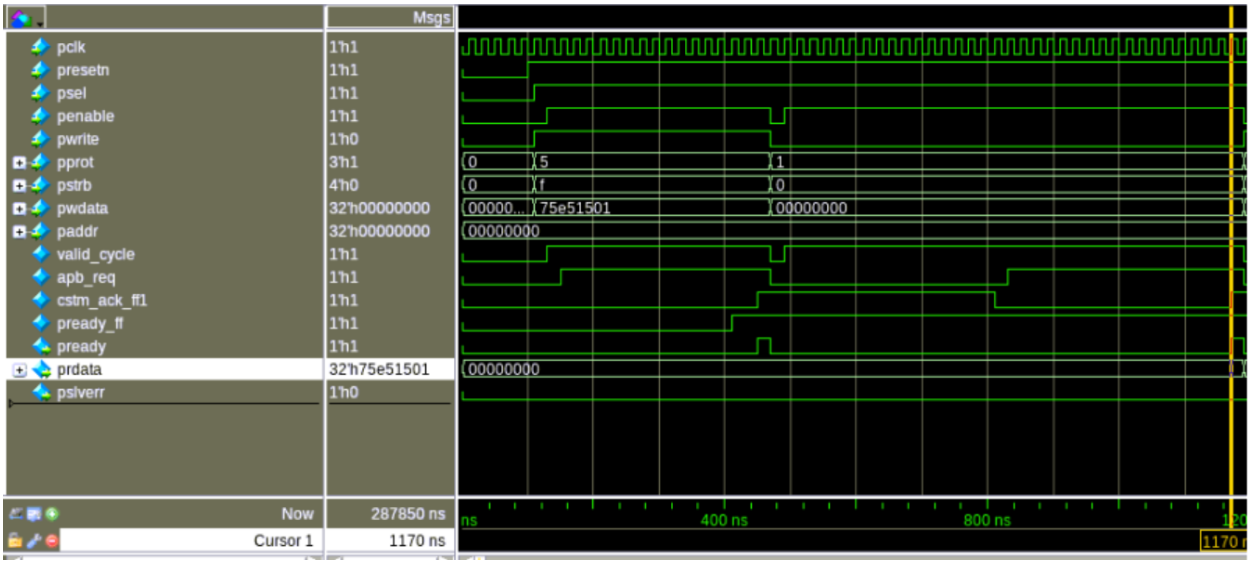

Fig 3: Read/Write in AMBA-APB bus domain

The figure 4 below shows the handshaking taking place taking place between the two clock domains. The request for transaction is initially high from the custom bus domain. Then it passes through 2-D flip flop synchronizer and provide custom request ff1. When there is valid APB transaction and custom request ff1 is also high, APB acknowledge goes high as well as APB request for response goes high, then both of these passes through 2-D ff and asserts APB req ff1 and APB ack ff1. When custom is ready with response and it gets response request from APB, then it asserts custom acknowledge to indicate a valid response is available, then it passes through 2-D ff and complete handshake takes place.

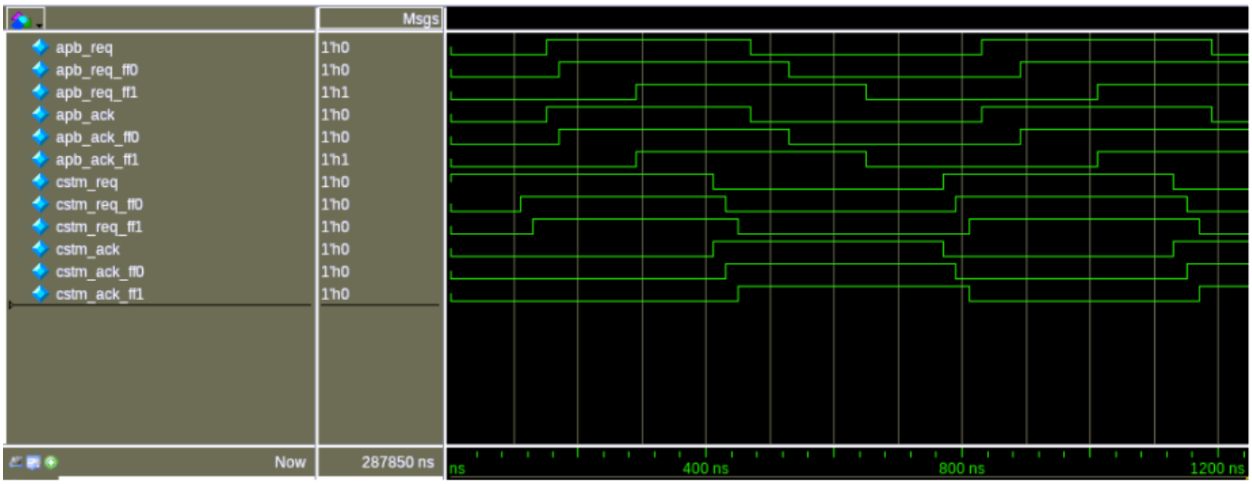

Fig 4: Handshake between AMBA-APB and Custom domain

The figure 5 shown below the CDC at the custom bus domain side, where when APB acknowledge is received, for the write transaction, the data write strobe and address are obtained. After that, a response is generated to indicate whether any error was present or not and whether the transaction is successfully completed or not. Similarly the process takes place for the read transaction.

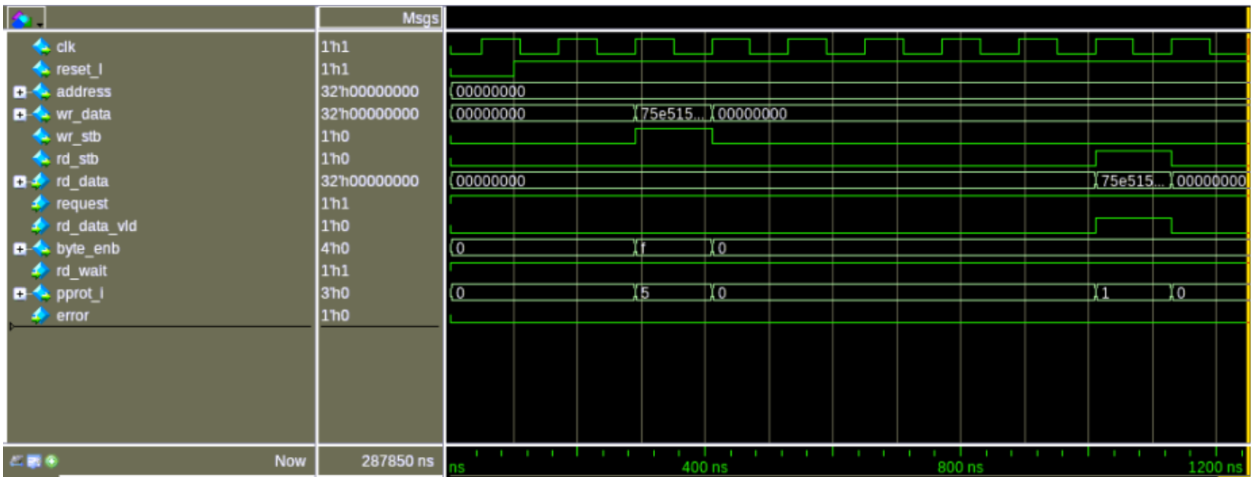

Fig 5: Read/Write in custom bus domain

B. CDC for AMBA-AHB Bus Protocol

If the master bus domain is working on AMBA-AHB bus domain, a similar approach as mentioned in AMBA-APB is followed. Here, hready signal is used to control the flow of transaction.

For normal operation:

In address phase, address is stored in flop and then during data phase we derive hready low and wait for custom request, when custom request occurs ahb_ack signal is asserted to indicate that address, data and control signals are available in flop and at the same time, ahb_req signal is asserted to receive the valid response corresponding to the provided data, address and control. When custom domain receive ahb_ack it will take data, address and control from the flop and derive custom request low then store valid response in response flop and assert custom_ack signal high to indicate that valid response is available in response flop, when AHB domain receive custom_ack it will derive low ahb_ack and ahb_req. Here if hready reads high, response will be taken from response flop, in the next data phase hready will be low again and custom request will be awaited. When custom domain will receive low ahb_ack and ahb_req then, it will again assert custom request for new transaction request. Therefore, one transaction takes place between two clock domains.

For burst/pipeline operation:

Similar approach will be used as mentioned above except that in first transaction(T1)'s data phase there is a need to store second transaction(T2)'s address in next address flop and when a transaction is set complete the next is to move that address from next address flop to address flop.

In the figure 6 shown below, when address phase starts the master asserts haddr, htrans in data phase and hready is pulled down to indicate that clock domain crossing is going on. After handshake synchronization, address and data with wr_stb goes to slave domain and then gives back response, so when a valid response goes hready is high again. It will be in case of write operation, similarly, read operation also takes place. Inverted clocks can be observed be

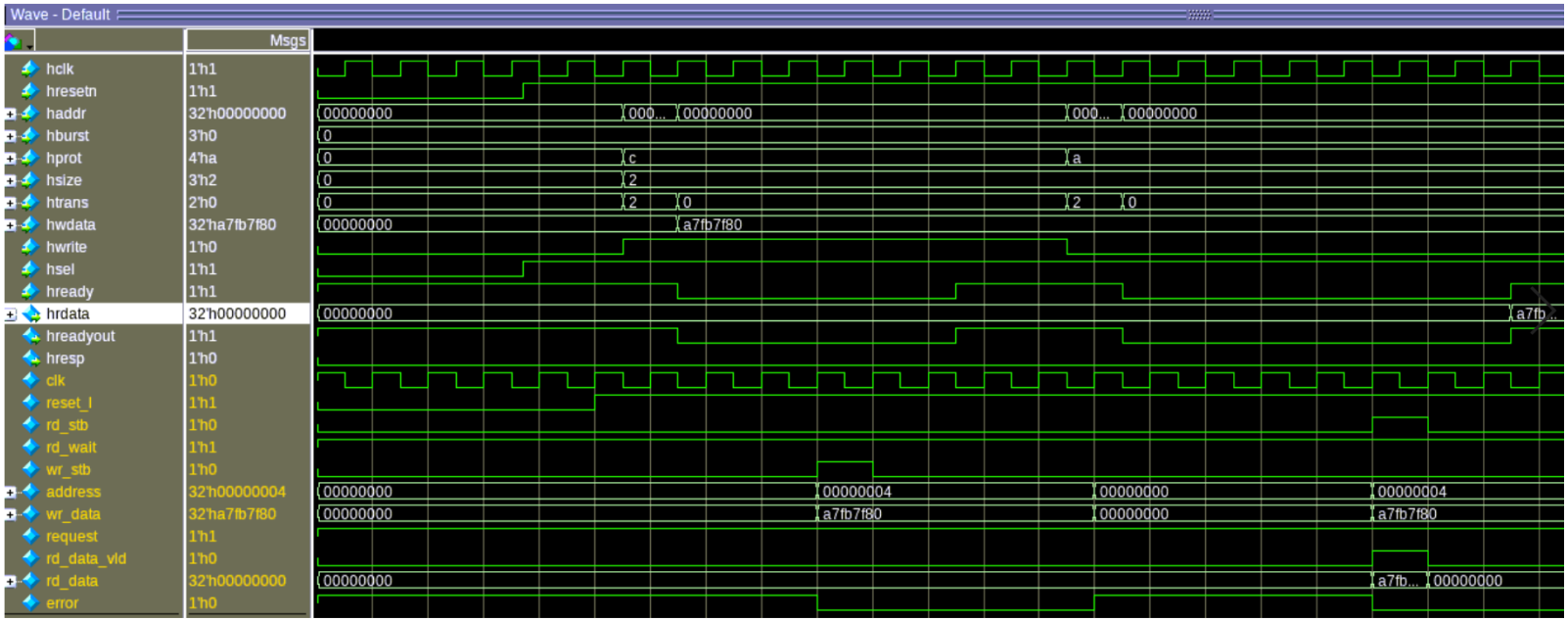

Fig 6: Handshake Synchronization for AMBA-APB Bus domain

III. CDC from Hardware side

In the diagram shown below, two clocks one coming from bus (say APB) on which read write are will take place on register bank from software side and the other one is user define clock(CLK_2) on which hardware side will do the read/write operation on the register bank. The handshake synchronizer is used to translate the transaction coming in from CLK_2 domain to APB_PCLK domain.

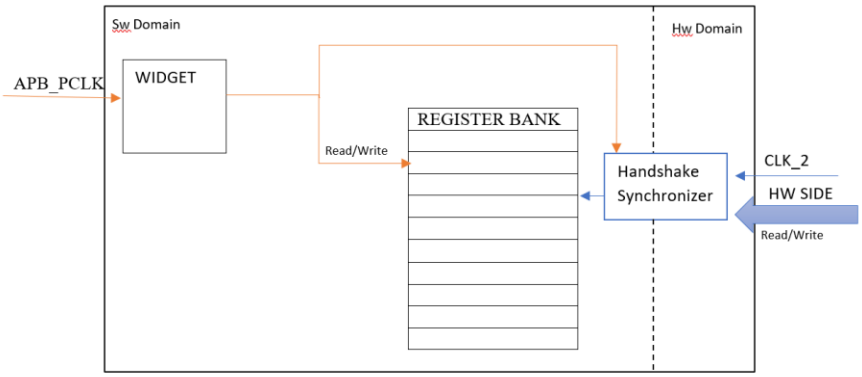

Fig 7 Handshake CDC inside slave from Hardware side

IV. RESULTS

Therefore, various techniques have been used according to the bus protocol for the implementation of clock domain crossing. The implementation and simulation results inferred low power RTL generation. The clock domain crossing was seen from software bus side and hardware end. The disadvantage from hardware side with the current solution is that, there can be multiple requests from the hardware side at the same time to different registers, in this case multiple handshake synchronizers will be created. We are in pursuit to find out the best way to implement the clock domain crossing from hardware side.

V. REFERENCES

[1] https://filebox.ece.vt.edu/~athanas/4514/ledadoc/html/pol_cdc.html
[2] https://zipcpu.com/blog/2017/10/20/cdc.html
[3] https://www.researchgate.net/publication/Clock_Domain_Crossing_CDC_Design_Verification_Techniques_Using_SystemVerilog
[4] http://web.eecs.umich.edu/~prabal/teaching/eecs373-f12/readings/ARM_AMBA3_APB.pdf
[5] http://www.eecs.umich.edu/courses/eecs373/readings/ARM_IHI0033A_AMBA_AHB-Lite_SPEC.pdf
[6] https://static.docs.arm.com/ihi0033/bb/IHI0033B_B_amba_5_ahb_protocol_spec.pdf
[7] Agnisys IDesignSpec™ – https://www.agnisys.com/products/idesignspec-uvm-register-generator/