RESEARCH ARTICLE OPEN ACCESS

Design and Verification of AHB Lite to CAN Bus Bridge

Naga Subrahmanyam Tirumala[1]

[1]Student, Department of VLSI Engineering, The Veda Educational Society, MRP Towers, Vidyanagar 1st Line, Guntur, Andhra Pradesh, India.

Subbu99gate@gmail.com

----------**********----------

Abstract:

Sophisticated Automotive vehicles have brought a new challenge of obtaining data from sensors, process them and control various sub systems of the vehicle. All the sensors are connected to can bus which has decreased pin count and wiring harness, and communicate with the processor through AHB-Lite. This can be achieved through a CAN to AHB-Lite Bridge. This Bridge builds the communication path between sensors and the processor.

Keywords — **CAN – Controlled Area Network, AHB – Advanced High Performance Bus, SoC – System On Chip, ASIC – Application Specific Integrated Circuit.**

----------**********----------

I. INTRODUCTION

This project aims to build a CAN bus to AHB bus interface module. This is an essential part in the electronics of any car. It facilitates communication between the sensors and the microprocessor and the transfer of commands from the microprocessor to the engine modules.

Essentially, this is a way for an SoC design to converse with different electronics. This interface module makes communications between different standards easier and allows the microprocessor to be implemented in more designs. There are not many processors that are capable of connecting to a CAN bus.

This project will make the connection between an SoC and CAN protocol possible. This design is ideally suited for ASIC implementations, because this module is only required to perform one task, translation between an AHB bus and a CAN bus interface. There is no need to reprogram this for any other purposes so there is no reason to use a non-ASIC design. Another major factor that justifies the design of this ASIC module is cost effectiveness.

Since ASIC circuits only need to perform one action, they tend to require less components which makes them cheaper. Also, less components means less surface area which is another advantage of using ASIC. The final and perhaps the most important advantage is speed.

This process needs to be as fast as possible to reduce the amount time it takes for the sensor inputs to be interpreted by the microprocessor and for the modules to receive the commands of the microprocessor. Using an ASIC will best satisfy these requirements.

II. DESIGN ARCHITECTURE

The design will consist of two modules, one each for the AHB Lite interface and one for the CAN bus interface. These modules are used to connect the design's internal data and status locations with the external SoC environment of the processor as well as the external sensors connected to the CAN bus module.

The intended architecture for an AHB bus Slave interface to a CAN bus interface and also shows the intended architecture for a CAN bus interface to an AHB bus Slave interface. Each bus will have a

value storage module, one to house the commands coming in from the AHB bus and another to store the sensor input data coming in from the CAN bus. They will both be implemented using a stream register design.

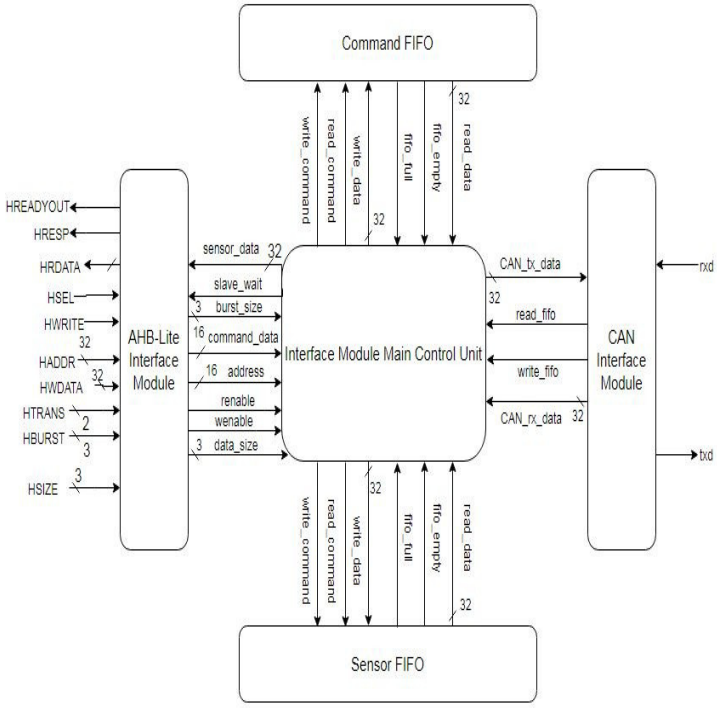

Fig.1 Architecture Diagram for AHB Lite to CAN bus interface module

The main functioning of the module will be split up into two control units, with each one handling the transfer of data in one direction. This will allow for simultaneous transfer of data between both the bus modules thus increasing the efficiency of the design. These two control units, namely the Processor-to-Sensor and the Sensor-to-Processor control unit will be implemented using Finite State Machines (FSMs) with included flags to check for the progression of data between the two buses. Status signals are sent between the two buses to acknowledge the completion of a certain byte transfer and to signal a new transfer from the storage modules. Furthermore, these storage modules will have priority bits assigned to each data byte stored in order to promptly complete time sensitive tasks. thus increasing the efficiency of the design. These two control units, namely the Processor-to-Sensor and the Sensor-to-Processor control unit will be implemented using Finite State Machines (FSMs) with included flags to check for the progression of data between the two buses. Status signals are sent between the two buses to acknowledge the completion of a certain byte transfer and to signal a new transfer from the storage modules. Furthermore, these storage modules will have priority bits assigned to each data byte stored in order to promptly complete time sensitive tasks.

This cycle show first how data gets saved into the command fifo and send out the CAN Module, second how data gets received through the CAN module and send to the sensor fifo and third how the AHB master picks up the data in the sensor fifo. The variable CAN Module show what type of transmission the CAN bus is currently in. At the beginning, we can see how the master makes writing request by keeping the HWRITE high and raising the HSEL. This makes the command fifo read the data from the slave.

Then after some time, the CAN Module sends the read command to the fifo. Later on in the write cycle, the tx register shifts out the data read by the can module. If this data is a read request. Then the CAN module goes into Read mode. During this stage the CAN module receives the data via the rx line. Then if the data is correct it will assert the write enable for the sensor fifo and give it this data. After this, the master asserts HSEL again, but this time it lowers the HWRITE signal. This signals that it will dequeue the value in the sensor fifo with a read enable fifo and the HWDATA will take the value previously in the fifo. It is important to know that CAN keeps running a writing cycle during this.

III. DESIGN VERIFICATION

This project was a success. We were capable of achieving all of our design specific success criteria for both mapped version and source version of the

project. The following section is a specific breakdown of how we tested the design.

AMBA AHB-Lite protocol interfacing

Main verification/test steps:

1. Send in data for a particular type of bus transaction in the AHB-Lite protocol (Ex. burst WRITE transaction, single WRITE transaction, READ transaction)
2. Compare design's output data with expected response values present in the response test vectors
3. If correct, repeat the above steps for all the following transaction types

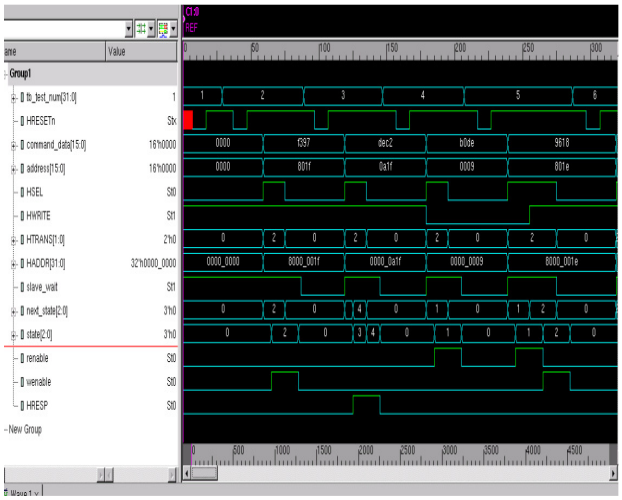

Fig. 2 Test cases 1 to 6 for the AHB-Lite testbench

The AMBA AHB-Lite protocol was tested on its own. We created a separate test bench that would check all the different transitions this module has to make. In the case of our project there was no master module present. We only had a slave module. In regular applications, the master is the microcontroller itself and in our case this was represented by the test bench.

The timing diagrams above illustrate the functionality of the AHB-Lite bus module. As can be seen, there are only 5 states required to implement the design, namely IDLE, READ, WRITE, FIRST_ERROR and SECOND_ERROR. The two error states are simply to allow two error cycles to occur in case any condition is violated. As can be seen in the first figure in test case 2, a simple write function is taking place. The HSEL and HWRITE signals are asserted indicating that the master has selected the slave and wants to write data to it. Hence the state machine moves into the WRITE state and wenable is asserted so the data on the command_data bus gets stored into the corresponding FIFO. In test case 3, an invalid address (max allowed address is 32) is sent in on the HADDR bus, so that module goes into the first of the two error cycles from the IDLE state. The HRESP signal also gets asserted when the module is in the error states telling the master that an error has occurred. In test cases 5 and 6, simple read and writes occur in successive cycles. Notice that the slave_wait signal is asserted whenever the module is in a state that is not IDLE indicating that it is busy and the master has to wait. The HTRANS signal (that indicates transfer type) also changes to 2'b01 which indicates a busy transfer (2'b00 indicates idle). Again in the corresponding states the renable and wenable signals are asserted to allow reading and writing. Also the HWRITE signal must be pulled low and HSEL must be pulled high for the renable signal to be asserted. This means that the master has selected the slave and wants to read data from it, meaning the data on the HRDATA bus gets read by the master when the module is in the READ state.

Correct enqueuing and dequeuing of data in the Command FIFO

Main Verification Test Steps:

1. Start with an empty FIFO
2. Check status of the simulated AHB Lite Master to determine when the control module must assert write enable.
3. Send multiple 16 bit data strings
4. Check if the FIFO is storing them in the correct order.
5. Repeat steps 1 to 4 until the fifo is full.
6. Wait for the simulated CAN Bus to say that

it is ready
7. Main control Unit sends read enable signal
8. Check if the output of the FIFO is correct
9. Repeat steps 6 to 8 until FIFO is empty
10. Repeat the entire process until correct operation is guaranteed.

The correct enqueuing and dequeuing of values from the command fifo was first tested on its own. By creating a separate testbench we assured that the fifo was working as expected. This meant that the pointers inside the fifo where increasing and decreasing correctly, the fifo full and fifo empty flags where both asserted at the right time and most importantly the data read and the data written where correct. . After that we tested it in along the entire design. By having our master send commands whenever it wanted the fifo would have to enqueue the values given to it. This was covered during the writing phase of the protocol. Since the CAN bus is slow, the fifo can get full rather quickly, so had to ensure that the HREADY flag was being de-asserted when the fifo was full. The next part of the test included using the CAN bus.

Our read enable signal comes out of the can bus when it wins arbitration. This part of the test had to deal with the fifo empty. In case the fifo were empty, the can bus should not attempt to read data from it. Finally, by using both the waveforms generated and the $display() function in our test bench we were able to determine that the data was correct.

This was very helpful while testing the mapped version of the code, since we could compare the correct results of the source version to the outputs of the mapped version. This method was successful and the mapped version behaved as expected.

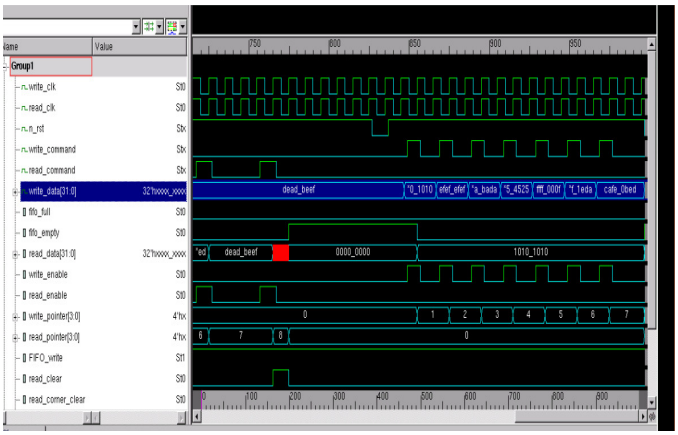

Fig. 3 Data being enqueued in the command

In this image we can see how after the the AHB master makes several writing requests by asserting HSEL and HWRITE from the AHB master, which leads to the write_command signal being asserted. When this signal is asserted, the fifo stores the data in the write_data port, adds 1 to the write pointer and initializes the new value that the write pointer is pointing to. During this test the first read_command also enters the fifo, which leads to the fifo queueing the first value enqueued, lowers the write pointer and re-asserts the signal fifo_empty.

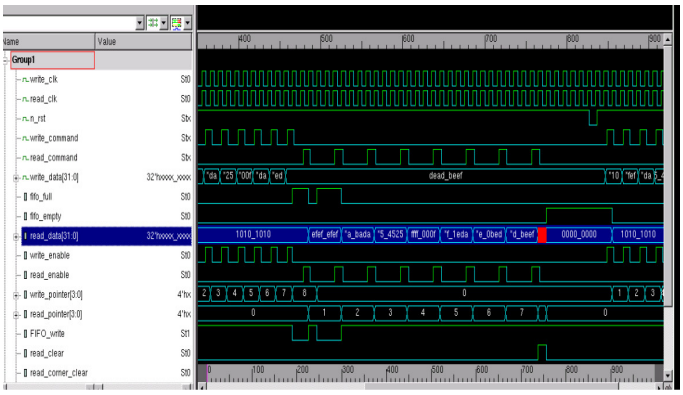

Fig. 4 Data being dequeued from the command FIFO

This image shows two cycles where the can bus wins arbitration and as a result it sends a read command to the command fifo. This read signal causes the read_pointer to move and point to a different value. We can see that the pointer starts of

as being 0. On the first read command it increases to 1 and the read_data value changes to be the value in the address 1. The values don't get reset until a the AHB master attempts to write a new command on it. We can see the same behaviour on the second write command, where the read_data value changes alongside the read_pointer. In this waveform we can also see the win signal asserted by the can module, which goes along the read read enable signal for the fifo.

Correct queueing and dequeuing of data in the Sensor FIFO

Main Verification Test Steps:

1. Start with and empty FIFO
2. Check status of the simulated CAN Bus Module to determine when the control module must assert write enable.
3. Send multiple 16 bit data strings from the CAN Bus Module.
4. Check if the FIFO is storing them in the correct order.
5. Repeat steps 1 to 4 until the fifo is full.
6. Wait for the simulated AHB Bus to say that it is ready to receive data
7. Main control Unit sends read enable signal
8. Check if the output of the FIFO is correct
9. Repeat steps 6 to 8 until FIFO is empty
10. Repeat the entire process until correct operation is guaranteed.

The initial stages for this test were very similar to the previous test. We started off by assuring correct functionality of the fifo. Then we tested the sensor fifo inside of the top level design. For this to work we had to assure that the command fifo, control module and CAN bus module were all working as intended. The tests started of by sending the can bus many read requests, this lead to it attempting to write data on the command fifo. This signal is asserted when the crc value received by the can. It is the inverse of the ACK bit. After assuring that the values sent by the CAN were being correctly enqueued and following the restrictions regarding the fifo full. We had to to tell the sensor fifo to now dequeue those values. We accomplished this by sending reading requests to the AHB slave. We also had to make sure that the requests did not come through when the fifo was empty. After each request we would check if the values enqueued were coming out in the order of first in first out. We did this by checking at the waveforms and the outputs of the $display function in our test bench. Using the waveforms were were able to determine that the outputs from the $display functions were correct. This was very helpful while testing the mapped version of the code, since we could compare the correct results of the source version to the outputs of the mapped version. This method was successful and the mapped version behaved as expected.

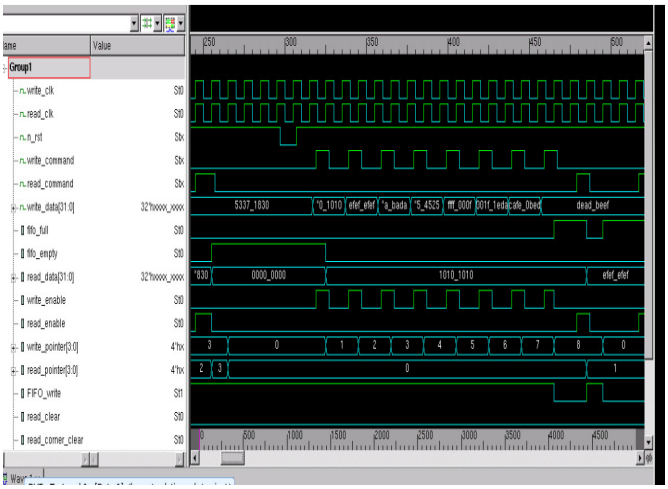

Fig. 5 Data being enqueued in the sensor FIFO

This FIFO works just like the one previously discussed. The main difference between them is the context of their operation. The sensor fifo relies on a fully correct can protocol. The write command in this image is asserted during the EOF stage of the CAN bus protocol, but the value gets determined by the ACK value. If the can bus acks, then the write enable will go high. Just like with the previous fifo, the write pointer goes up and the value gets stored.

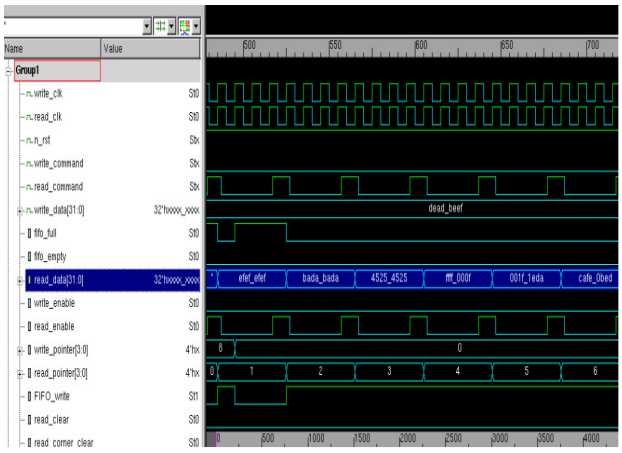

Fig. 6 Data being dequeued from the sensor FIFO

This image shows how the sensor fifo de-queues data. This happens when the AHB-Lite Master keeps the HWRITE signal low and asserts the HSEL signal. This causes the read pointer to increment and the read_data changes its value to the one the read_pointer is pointing to. If we look at the last 2 images we can see that the values are correct and follow the order of first in first out.

IV. CONCLUSIONS

This project aims to build a CAN bus to AHB bus interface module. This is an essential part in the electronics of any car. It facilitates communication between the sensors and the processor and the transfer of commands from the processor to the engine modules. Essentially, this is a way for an SoC design to converse with different electronics. This interface module makes communications between different standards easier and allows the processor to be implemented in more designs. There are not many processors that are capable of connecting to a CAN bus. This project will make the connection between an SoC and CAN protocol possible. The final and perhaps the most important advantage is speed. This process needs to be as fast as possible to reduce the amount time it takes for the sensor inputs to be interpreted by the processor and for the modules to receive the commands of the processor.

ACKNOWLEDGMENT

I express my sincere thanks to Industry guide Mr. Murali Mohan Kilari, SMTS, Invecas Pvt. Ltd, and VEDA IIT guide Mr. Sivaji Satya Ganesh Nandigama, Assistant Professor, VEDA IIT, Amaravati carry out the Project Work.

I convey my deep sense of gratitude to Prof. K.Malakondaiah, Director, VEDAIIT Amaravati, and Mrs. Madhuri Nallapaneni, Director, Invecas, Guntur, for providing me with an opportunity for carrying out this Project Work.

REFERENCES

[1] 2006 ARM "AMBA 3 AHB-Lite Protocol".
[2] ARM, "AMBA_3_AHB-Lite", available at http://www.arm.com/.
[3] ARM, "AMBA Specification (Rev 2.0)", available at http://www.arm.com.
[4] ARM, "AMBA AXI Protocol Specification", available at http://www.arm.com.
[5] Samir Palnitkar, "Verilog HDL: A Guide to Digital Design and Synthesis", Second Edition.
[6] Chris Spear, SystemVerilog for Verification, New York : Springer, 2006.
[7] http://www.testbench.co.in.
[8] http://www.doulos.com/knowhow/sysverilog/ovm/tutorial_0.
[9] http://www.inno-logic.com/resourcesVMM.html.
[10] Akhilesh Kumar and RichaSinha, "Design and Verification Analysis of APB3 Protocol with Coverage"IJAET Journal, Vol. 1, Issue 5, pp. 310-317, Nov 2011.
[11] Verilog HDL Synthesis by J.Baskar.
[12] Springer - SystemVerilog for Verification by CHRIS SPEAR Synopsys, Inc.
[13] C. Miller, C. Valasek, in Black Hat USA, 2014. A survey of remote automotive attack surfaces (2014), p. 94.
[14] M. Wolf, A. Weimerskirch, T. Wollinger, State of the art: Embedding security in vehicles. EURASIP J. Embed. Syst. 2007(1), 074706 (2007).
[15] T. Nohet, H. Hanssont, L.L. Bello, in IEEE Symposium on Emerging Technologies and Factory Automation. Automotive Communications-past, Current and Future (Catania, 2005).
[16] C. Miller, C. Valasek, in Black Hat USA, 2015. Remote exploitation of an unaltered passenger vehicle (2015).
[17] S. Al-Sultan, M.M. Al-Doori, A.H. Al-Bayatti, H. Zedan, A comprehensive survey on vehicular ad hoc network. J. Netw. Comput. Appl. 37, 380–392 (2014).
[18] A. Humayed, J. Lin, F. Li, B. Luo, Cyber-Physical Systems Security—A Survey. IEEE Internet Things J. 4(6), 1802–1831 (2017).
[19] F. Sakiz, S. Sen, A survey of attacks and detection mechanisms on intelligent transportation systems: VANETs and IoV. Ad Hoc Netw. 61, 33–50 (2017).

