Proceedings of the International Conference on Smart Electronics and Communication (ICOSEC 2020)
IEEE Xplore Part Number: CFP20V90-ART; ISBN: 978-1-7281-5461-9

UVM based Controller Area Network Verification IP (VIP)

Abhinav Goel[1]
Dr. B. Bala Tripura Sundari[3]
Department of Electronics and Communication Engineering
Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham, India
abhinavgoel78@gmail.com[1], b_bala@cb.amrita.edu[3]

Sujith Mathew[2]
Ignitarium Technology Solutions Pvt. Ltd.
Bangalore, India
sujith@ignitarium.com[2]

***Abstract*—As the complexity of System on Chip (SOC) designs is increasing day by day, verification is becoming a complex task to attain. A SOC design consists of various intellectual property cores (IP). To verify so many IPs, a complex testbench has to be developed which is not an easy task to achieve. So to make the verification an easy task, Verification Intellectual Property cores (VIP) are developed. In this paper, the design of the Controller Area Network (CAN) VIP is proposed. This VIP is developed using SystemVerilog based universal verification methodology (UVM). The test environment of this VIP is verified by running appropriate test cases. The coverage is collected based on the test cases to verify whether the functional specifications of the CAN protocol are covered or not. This VIP is simulated using Cadence Xcelium tools to check the effectiveness of the proposed approach.**

***Index Terms*—controller area network (CAN), universal verification methodology (UVM), verification intellectual property (VIP)**

I. INTRODUCTION

CAN is used in various domains in the motor vehicle. Domain requirements are different so according to that our needs will differ:

1. In the case of area related to engine management, the information required is much faster.

2. In case of area related to comfort /convenience, information is not required a fast rate and the control systems are also located further apart as such lines are more prone to damage.

As a result of these different requirements, the bus with different data rates has used that offer an optimal cost-benefit ratio for the field of application concerned. CAN is a multi-transmitter protocol. Fig. 1 shows the diagram of a CAN bus. If one node is acting as a transmitter, then other nodes will automatically become the receiver. By this, it is clear that at a particular time only one node will be the sender, and rest all nodes will be the receiver. Each receiving node will check whether to accept data or not. Each CAN node consists of a micro-controller, CAN controller, and a CAN transceiver. Fig. 2 shows the diagram of a CAN node. Micro-controller decides what the received messages mean and what messages it wants to transmit. The CAN controller is responsible for arbitration, framing, and many other protocol functions. CAN transceiver constitutes the physical layer. It takes into account the timing at which the bit is to be placed on the CAN bus and when a bit is to be delivered to the CAN receiver node. It also converts the data from the CAN controller to the CAN bus levels and also converts the data from the bus levels to suitable levels that the CAN controller uses. Nowadays CAN bus is incorporated into many devices. Apart from the automotive devices, it is used in escalators and lifts. In this paper, the design of CAN Verification Intellectual Property (VIP) is proposed. VIP reduces the effort needed for functional verification of a System on chip (SOC) design. SOC chip has various Intellectual properties (IP) embedded in it. To verify a SOC design, it takes nearly 70% of the design time. To reduce that time, VIP is developed. These VIP's are the reusable agents having an interface same as that of actual IP's. VIP consists of all the specifications of the desired IP to be verified. It is used to test the IP embedded in SOC design. VIPs are developed to verify various standard bus protocols. This CAN VIP will verify the CAN protocol. CAN VIP consists of a transmitter and receiver agent. The Transmitter will generate data and drive that to the Receiver. The Receiver will receive that data and send an acknowledgment to the Transmitter on receiving the correct message.

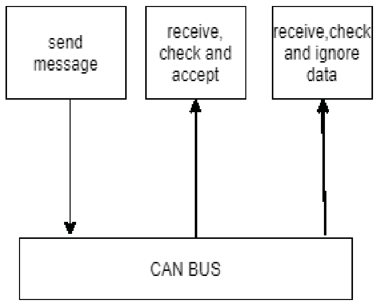

Fig. 1. CAN BUS WITH NODES ATTACHED

II. LITERATURE SURVEY

CAN controller is widely used in automotive applications. Without the CAN network, it is difficult to imagine automotive vehicles. CAN controller is explained in [1]. [12], [13], [17] explain the message format of the CAN network. CAN features like bit timing logic and bit synchronization [10], [11] are used to control how serial transmission will take place.

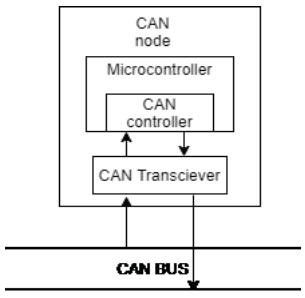

Fig. 2. CAN NETWORK NODES

The receiver will be synchronized to the transmitter by the bit timing logic. The bitstream processor [11] performs the arbitration process, bit stuffing logic, and bit serialization and frame set, etc. Node with the highest priority will access the bus first [3].In CAN, each message has a fixed priority which is a limitation of CAN. To avoid that there is a different approach [14] introduces the concept of multiple priorities to a single message. This introduces various algorithms and concepts for assigning multiple priorities to a single message. Since CAN is a broadcast and id based bus, it is possible that a wrong source is passing the data and to avoid that physical characteristics are added i.e. measuring voltage and convolution characteristics [2]. To avoid sending malicious messages monitoring unit [7] is used which can easily detect malicious messages. The major drawback of the CAN protocol is its inability to detect the actual time for the message to be received at the destination. For this, [4] is proposed which brings the method to detect the worst-case time ability.[9] In this paper, time-triggered communication is used in the transmission of messages. In this, central scheduling along with a planning scheduler is used for achieving flexible time-triggered communication.[5] There are many accidents at night due to glaring effects, high voice of the horn so the method is adopted to reduce the probability of these accidents. [6] CAN controller provides high reliability for any arbitrary number of nodes. So that message can be successfully delivered to the destination. This paper introduces new interface concepts and new reliability techniques for low-cost applications. [15] By this system, bandwidth can be shared among the various devices and a time limit is specified for the devices to receive the message. [16] proposes the concept of detection as well as prevention of unauthorized data transmission. This can be done by slight changes in the electrical control unit structure. [8], [10] indicates how acceptance filtering will take place. Table.I show the CAN frame format content.

III. METHODOLOGY

CAN VIP is developed to reduce the verification complexity as well as time to verify the SOC design which has CAN IP embedded in it. CAN VIP will be integrated to the CAN IP through the interface and verification will be done. In this way, the development of the test environment for the complete SOC design is not required. These VIP blocks are reusable.

TABLE I
CAN FRAME FORMAT

Frame Content	Notation	Size	Comments
Start of frame	SOF	1 bit	This is the first bit sent by any node before transmitting its message.
Identifier field	ID	29 bits	This is the unique id of every CAN node which is transmitting its message.
Remote Transmit Request	RTR	1 bit	distinguishes between data frame and remote frame.
Substitute Remote Request	SRR	1 bit	send as recessive(1).
Identifier Extension	IDE	1 bit	send as recessive.
Reserved bit 1	r1	1 bit	send as dominant(0).
Reserved bit 0	r0	1 bit	send as dominant.
Data length code register	DLCR	4 bits	can send up to a value of 8 only.
Data field	DATA FIELD	0-64 bits	data field length depends on the DLCR field.
CRC field	CRC	15 bits	used for checking whether the received message is correct or not.
CRC delimiter	CRC DE-LIMITER	1 bit	indicates the end of the crc sequence. It is a recessive bit.
Acknowledgment slot	ACK	1 bit	sent as recessive and expected to receive as dominant.
Acknowledgment Delimiter	ACK DE-LIMITER	1 bit	indicates the end of acknowledgment slot.
End Of Frame	EOF	7 bits	indicates the end of frame.

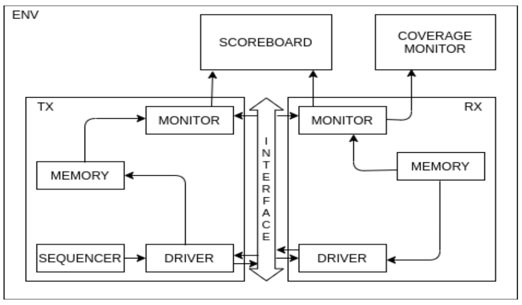

Fig. 3. CAN VIP ARCHITECTURE

This VIP can be used across multiple designs having CAN IP. This VIP block can be inserted directly into the testbench of the SOC design. The methodology used to develop the VIP is SystemVerilog based UVM methodology.

A. CAN VIP ARCHITECTURE

Fig.3 shows the CAN VIP Architecture. In this proposed CAN VIP architecture, there are two agents i.e. the transmitter and the receiver. Here the transmitter is sending the data to the receiver and receiver on getting the correct data will send the acknowledgment to the transmitter indicating that it has correctly received the data. There is an acknowledgment slot present in CAN frame format. The transmitter will send the

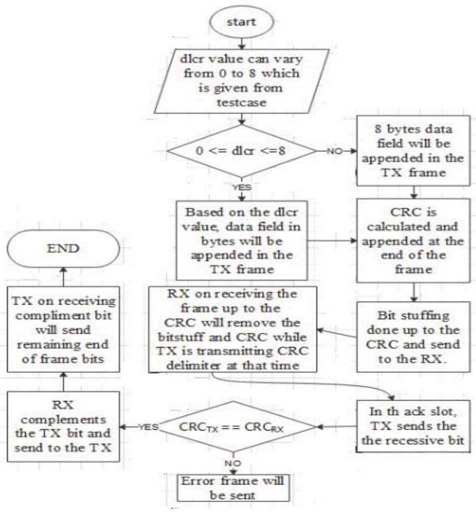
start
dlcr value can vary from 0 to 8 which is given from testcase
0 <= dlcr <=8
NO
8 bytes data field will be appended in the TX frame
YES
Based on the dlcr value, data field in bytes will be appended in the TX frame
CRC is calculated and appended at the end of the frame
END
TX on receiving compliment bit will send remaining end of frame bits
RX on receiving the frame up to the CRC will remove the bitstuff and CRC while TX is transmitting CRC delimiter at that time
Bit stuffing done up to the CRC and send to the RX.
RX complements the TX bit and send to the TX
YES
CRCTX == CRCRX
In th ack slot, TX sends the the recessive bit
NO
Error frame will be sent

Fig. 4. WORKINGOF CAN VIP

recessive bit during the acknowledgment slot and wait for the receiver to send the dominant bit. As soon as the transmitter receive the dominant bit during this slot, it will interpret that receiver has received the data correctly. After that it will continue to transfer the remaining bits of the frame. Forcoding this VIP, FSM based approach is used. This VIP is functionally verified with the help of proper testcases. In CAN protocol, the length of data field is dependent on DLCR which is of four bits. So total combinations can be 16 i.e. it can vary from 0 to 15. But in CAN protocol, it is strictly mentioned that it can vary from 0 to 8 only. Since data field is declared in bytes, so data field can contain maximum up to 64 bits only.

Fig.4 shows the working of this VIP.

1) *CAN VIP OBJECTS:*

a) CAN SEQUENCE ITEM: In this class, all the fields required for sending and storing the stimulus are declared. The fields are data _field, id, ide, srr, rtr, dlcr, r1, r0, sof, acceptance _code, acceptance _mask, frame, crc, frame _crc, frame bit _stuff, received, slave_received, count, count_ before_ adding_crc, receive_fifo, send_slave_data_ to_ scb, slave_ data. Out of this, data_field is a dynamic array of byte type which is declared as randc, id which is a 29 bit field declared as rand and dlcr which is a 4 bit field declared as randc. The ide, srr, rtr, r1, r0, sof are declared as fixed value fields as required by the protocol. The acceptance_code and acceptance_mask are 32 bit value fields used for message filtering. The fields which are required for performing tasks related to protocol includes a frame which is a queue, crc which is a 15 bit field, frame_crc which is queue, frame_ bit_ stuff which is a queue, received which is a queue, slave_ received which is queue, count which is of int data type, count_before_adding_crc which is of int data type, receive_fifo which is a two-dimensional unpacked array, send_slave_data_to_scb which is a queue, and slave _data which is of int data type. The constraints declared in this class includes dlcr varying from 0 to 8 and data_field size equal to dlcr value.

b) CAN SEQUENCE: This sequence sends the stimulus to the driver. In the sequence, two random variables are declared. These random variables are getting the value from the test class. In this way, a sequence is randomized. The variables which are declared as random are s_dlcr which is of 4 bit and s_id which is of 29 bit. They can sequence item_class is passed to this sequence class and instantiated. In this start item method is called for telling the sequencer that sequence is ready for sending the data. Then it wait for get next item call_from_the driver. If it gets that call then it will randomize the sequence with parameterized values whose actual value will be assigned from the test. Randomization is done using assert statement so if it is failed then it will be displayed as an error in the log file. There are some tasks which are performed as required by the protocol are frame develop, crc calc, frame with crc, bit stuff. After completing these tasks sequence is sent to the driver using the finish item method. In the frame develop a task, the data field which is byte type is converted into bit type dynamic array. After this, the transaction fields like sof, id, srr, ide, rtr, r1, r0, dlcr, and data field, all are stored in the frame which is a queue. This is done to develop a CAN data frame. After that frame size is assigned to variable count before adding crc. And data field array size is also stored in variable slave data. In the next task crc calc, crc is calculated and appended at the end of the frame. In the next task, bit stuffing is done on the frame. Bit stuffing means after 5 consecutive bits of the same type, one complimentary bit will be inserted in the frame. This is done to avoid a long stream of 1's or 0's in the frame. It also helps in prevention of data lost during transmission of frame. With this class is finished.

2) *CAN VIP COMPONENTS:*

a) CAN SEQUENCER: This sequencer will allow the sequence to pass to the driver. The driver on receiving the sequence will pass it to the interface pins. It is connected to the TX_driver in the TX agent class.

b) CAN DRIVER: The Driver is the kernel of any UVM testbench. It decides what stimulus should be driven at what time. In this VIP, an FSM approach is used to develop the logic of the driver. There are two drivers in this VIP:-

- TX_ DRIVER:- In this driver, first of all, a virtual interface, sequence item class and memory model are instantiated. The states of FSM are declared through the enum data type. In the build phase, getting the interface using config db which was set in the top module. The memory model which is extended from the uvm component class is get using config db. It is set in the environment class. If getting an interface or memory model is failed, it will display fatal message and simulation will not take place. In the run phase of the driver, first of all, the interface tx pin is set to the default state. The default state in CAN protocol is recessive (1). First of all, reset condition is checked. If reset is active high then interface pins will be set to default state else next state in FSM will start. The next state is IDLE. The transmitter will start to send the frame only when it has detected eleven recessive states after the reset getting active low. For checking that, the interface rx pin is checked. If eleven recessive state is detected, the next state starts i.e. start of frame. Here, first of all, the start of frame bit is sent. This is done after get_next_item method. Through this method, the transaction items are received. Simultaneously the bit which is sent will be monitored for checking whether the bit which is sent to the transmitter is the same as what it is intended. If there is a mismatch, then the error frame will be sent. Now the remaining frame will be sent in the next state. In the frame upto crc state, first of all, reset is checked whether it is active low or not. If it is active low then crc, frame bit stuff size, count before adding crc, slave data are stored in the memory at respective locations. After that remain- ing bits after start of the frame are sent to the receiver according to the bit timing logic. Now the next state is send crc delimiter, here crc delimiter bit is transmitted to the receiver. Now the next_state is ack check, here acknowledgment bit is sent as recessive and waiting for the receiver to send the dominant bit during that slot. If the receiver detects the correct message, it will send the dominant bit. If the transmitter receives the dominant bit then it will assume that there is no acknowledgment error otherwise error frame will be sent. Now it moves to the next state, Ack delimiter, here transmitter will send the recessive bit. Next state is send remaining bits. Here seven recessive bits are sent to indicate the end of the frame. The last state is Interframe space additional bits in which eleven recessive bits are sent consecutively. After that item done method is called. In this way, FSM ended. In the run phase, fork join is used. In the fork join, there are two blocks:- one is FSM, and the other is reset check. For sending the bit, a task is called in which the value is passed as an argument and after that this value is assigned to interface pin. Now, this value is repeated for 160 clock cycles as determined by the bit timing logic. Now this task is ended. If any error occurs while sending the bit i.e. bit error, ack error. Then the error frame will be sent from the next bit onwards by that transmitting station. Error frame consists of two types of flags:- active error flag and passive error flag. If the transmit error counter is greater than 0 but less than or equal to 127 then an active error flag will be sent. If the transmit error counter is greater than 127 but less than or equal to 255 then a passive error flag will be sent. If the transmit error counter achieves value greater than 255 then the bus off will take place. Bus off means the node will be disconnected from the bus. Active error flag involves sending of six dominant bits followed by the eleven recessive bits. With this error frame ends. Passive error flag involves sending of six recessive bits followed by the nineteen recessive bits before recognizing the bus to be idle for sending the data again. If any node attains bus off state, then after connection it has to wait for 128 occurrences of 11 recessive bits before it can take part in bus communication. This is how the error frame works.

- RX_DRIVER:- In the RX driver, first of all, sequence item class, interface, memory handle are instantiated. In the build phase, the object of the sequence item class is created. Then getting the interface and memory model handles from the configuration data base using config db. In the run phase, as long as the reset pin is active high, rx pin of the interface will be in default state i.e. recessive. This state is called the Idle state. Since this driver is coded using FSM so FSM states are declared using enum data type. If the reset is active low, it will move to the next state i.e. start of frame, here, first of all, it will wait for start of frame bit to come. After receiving this bit, it will start to receive the complete bit stuffed frame. After receiving the bit stuffed frame, it has to remove the extra bit to get back the original data frame. For this, a task is called for removing the stuffed bits. After removing the stuffed bits, last fifteen bits of the frame are removed and compared with the crc which was stored in the memory. If they are same, then there is no crc error else error frame will start after receiving the ack delimiter bit. Now the next state is receive crc delimiter. In this state, crc delimiter bit is received and if it is not 1 then form error will be there and error frame will be sent from the next bit onwards. If there is no form error then next state can be reached. The next state is receive_acknowledge. In this, it is checked if there is no crc error as well as ack value received is recessive then dominant bit will be sent by the receiver. Now the next state is receive ack delimiter, here it is checked whether the received bit is recessive or not. If it is not recessive then error frame will be sent. If there is a crc error then also error frame will be sent. If there is no error it can move to the next state i.e. remaining bits here six bits of the end of the frame is received from the transmitter and checked for error. If the all bits are not recessive means form error is there. If the form error is present then error frame will be sent. If the form error is not there, next state will start i.e. interframe space. In this, one bit of the end of frame and three bits of the interframe space will be received. After receiving these bits, it will move to the start of frame state again. For receiving the bits a task is called. In this task, transmit value is received and after waiting for the total number of post edges determined by the bit timing logic, the task will be ended. For sending the acknowledgment bit, a task is called. In this task, acknowledgment bit is transmitted to the transmitter during the acknowledgment slot and it will wait for the 160 clock cycles before exiting this task. Now after receiving the complete frame, fixed bits, as well as crc bits, will be removed from the frame. From the remaining frame, the start of frame bit will be removed. After that message filtering is done. If the acceptance id matches the id that is received then the frame is stored in the received fifo. Otherwise, the frame is rejected. Now run phase is completed.

c) CAN MONITOR: Monitor is responsible for monitoring the pin level activity at the interface. If there is any change in the values of the interface pins, then that change will be sampled by the monitor and transferred to environment components for higher level checks. There are two monitors in this VIP:-

- TX_MONITOR:- In this class, first of all, the sequence item class, interface, memory model and analysis port are instantiated. In the user-defined enum data type declaration, all the states of FSM are included. Analysis port is allocated memory using the new function. In the build phase, the sequence item object is created. Interface as well as memory is get using config db. In the run phase, first state is wait for sof. Here monitor will wait for TX driver to transmit start of frame value. As soon as the monitor sees the tx pin at value zero, it will start executing this state. Then it will store this value inside a queue. Now comes the next state_receive_bit stuffed frame, here sequentially value is received from the tx pin and stored inside the queue. It will continue to do this until the number of values stored becomes equal to the memory value which is stored in the TX driver. Now comes the next state, crc delimiter, here crc delimiter value is received and stored in the queue. Now the next state is acknowledgment, here tx pin value send by the driver during ack slot is stored and and wait is enabled for rx pin to become zero. After the wait is over, next state starts
i.e. remaining bits. In this, values are sequentially stored inside the queue. Now using the analysis port, the stored value is sent to the above levels for higher level tasks. These bits are received from the tx pin using a receive_bit task. In this task after receiving the bit, it will wait for 160 clock cycles. Then it will get out of the task.
- RX_MONITOR:- In this class, first of all, states are declared using FSM approach. Then the sequence item class, virtual interface, and memory model is instantiated. Next, the analysis port to transfer the sample data to the score- board and the coverage monitor is declared. In the run phase, a forever loop is declared. Inside the forever loop, if the posedge of the clock comes or reset is there then the case statements will execute. Ist state is wait for sof, here wait statement is included to wait for the dominant bit to arrive at the receiver. If the dominant bit arrives then only the statements following it will be executed. Next_state is received bit stuffed frame. Here, the data frame which is having redundant bits will be received bit by bit from the transmitter and stored in the queue. After receiving the stuffed frame, the next states are crc delimiter, acknowl- edgment, and remaining bits. These states are executed sequentially and bits received from the transmitter are stored inside the queue. The queue in which data is stored is a part of a sequence item class. After that, a task is called for storing the received data according to the sequence item fields. Making them in the form in which they were initially passed to the TX driver. In this task, first of all, another queue is declared from which all the data present will be copied. Now the fixed form bits are removed from this queue. Now bitstuffing will be removed from this queue data. After stuffing, crc is removed from this queue which is last fifteen bits. After that data field is popped from the queue and stored in the byte type dynamic array. This dynamic array is returned by the task to the actual argument declared in the call of the task. After exiting this task, the dlcr is stored from the queue to the dlcr field in the sequence item. In the same way, remaining data is popped from the queue and stored. The reserved bits r1, r0, id bits, rtr, ide, and all are stored in the individual transaction fields. After that, the sequence item packet is sent to the scoreboard as well as coverage monitor using the analysis port write method.

d) CAN AGENT: There are two agents in this VIP:-

- TX_ AGENT The transmitter agent consists of a sequencer, TX driver and TX monitor. In this, first of all, sequencer, driver and monitor class are instantiated. After that analysis port is declared. In the build phase, agent, driver, monitor and sequencer handles are created. In the connect phase, the driver port is connected to the sequencer export. Monitor port is connected to the agent port.
- RX_AGENT In this agent, both the RX driver and RX monitor components will be instantiated. Analysis part of the agent is also declared. This analysis port is connected to the analysis port of the monitor. In the build phase, driver and monitor components are created. In the connect phase, monitor port is connected to the agent port.

e) MEMORY_ MODEL: This memory model class is extended from uvm component class. In this class, a reg type memory is declared which is of two dimensional. It is of size 16*32. There are two functions declared:- one is function write which is of void data type, other is function read which is of int data type. In the function write, two inputs are declared as formal arguments i.e. addr and data. If addr is less than 16 then data will be stored in the memory else it will not be stored. In function read, input argument is 16 bit addr. If addr is less than 16 then memory will be read otherwise it will display a warning indicating size is exceeded. This memory model is very useful. It is used to store the transactions which are randomized and to store the value of variables which are used for count purpose. While driving, these values are stored.

f) CAN SCOREBOARD: Scoreboard is a very important component of a UVM testbench. In this VIP scoreboard class, first of all, analysis exports are declared. After that analysis fifo's are declared. There will be two analysis exports, one will be for the transmitter and other will be for a receiver. In the same way, there will be two analysis fifo's, one will be for transmitter and other will be for receiver. The sequence item is also instantiated. There will be two handles of the sequence item class, one will be for the transmitter and other will be for a receiver. In the build phase, transmitter export, receiver export, transmitter analysis fifo, and receiver analysis fifo are allocated memory using the new function. In the connect phase, transmitter export is connected to the transmitter analysis fifo and the receiver export is connected to the receiver analysis fifo. In the run phase, the transmitter sequence item handle will get the transaction from the transmitter fifo and receiver sequence item will get the transaction from the receiver fifo. After that, a compare task is called for a comparison of data received from both transmitters as well as receiver fifo. In this task, they are compared bit by bit and the result is displayed using uvm macros.

g) CAN ENVIRONMENT: In the environment class, components like a scoreboard, coverage monitor, TX agent and RX agent components are instantiated. Along with these, memory model, as well as interface, is also instantiated. In the build phase, scoreboard, coverage monitor, TX agent, memory model and RX agent components are created. Here interface is get using config db and memory model is set into config db. In the connect phase, TX agent port is connected to the scoreboard export and RX agent port is connected to another scoreboard export. For collecting coverage informa- tion, RX agent port is connected to the analysis export of the coverage monitor. In the run phase, the assertion is implemented to check whether both the tx as well as rx pin will be set to 1 when reset becomes active high. Now the run phase is ended. With this, environment-class is ended.

h) CAN TEST: Above the environment is the test class. The environment class is instantiated inside the test. More than one environment can be instantiated inside the Test.

i) TB_TOP: From the top, test is run and interface is instantiated.

B. CAN VIP TESTCASES

1) CAN_LENGTH_TEST: From this testcase, CAN data field is generated. Data field depends on DLCR. To generate all 9 values of DLCR i.e. from 0 to 8. Sequence is started from the testcase nine times. Since data field depends on the DLCR, so if the DLCR value is 0 means 0 byte data will be generated. If it is 1, then 1 byte data will be generated. Similarly if it is 8 means 8 byte data will be generated. In this way, with the help of this testcase frames with different length can be generated.

2) ACCEPTANCE_ID_TEST: In this test, case id is given different value and checked whether acceptance filtering is correctly taking place or not. If the acceptance filtering is not correctly taking then it can accept the data from any id. So it might be possible that it can accept wrong data also which can damage the CAN devices. That is the devices will not work. For indicating this error, in the coverage monitor an illegal bin is declared. If that illegal bin is hit, it will display the error in the log file. In the receive fifo also no data will be stored.

3) DLCR_MORE_THA_8_TEST: In the sequence item class, a constraint is set that DLCR can have value from 0 to 8 only. So if the DLCR is randomized from this test case using inline constraints with a value of 9. Then the assertion which is declared in the sequence will get failed. After that, assertion failure message will be displayed in the log file. In this way, it comes to know whether the constraint is working properly or not.

C. COVERAGE PLAN

There are two types of coverage:- code coverage and functional coverage. Code coverage is tool based. The simulator will extract the coverage from the VIP design code automatically. For getting the functional coverage, cover groups are written inside coverage monitor class. It is extended from the uvm subscriber class. It is considered a effective way of getting the coverage. In this class there are two covergroups, one is_can cg for fixed-size variables and one _ is cg 1 for variable size items. In the first covergroup, there are ten coverpoints. First coverpoint is dlcr which is a four bit value but it has been constrained to the value of eight only. To cover values from zero to eight, bins are declared. The next coverpoint is id which is a 29 bit value but it is fixed i.e. it can’t vary so only one value is declared inside the bins. To cover the value which is not inside this bin, an illegal bin is declared. If it is getting hit then an error will come. The next coverpoint is ide which is a one bit value and it is also fixed, so only one value is declared inside the bins. Similarly srr, rtr, r1, r0, sof coverpoints are also fixed value bins, so one value is present inside the bins which have to be hit. Next two coverpoints are acceptance_code and acceptance_mask which are 32 bit variable and their value is fixed. So bins are having fixed values for these coverpoints. Next covergroup is having custom sample function in which data is passed through a sample function call. The coverpoint inside this covergroup is a dynamic array whose value depends on the dlcr value. In these nine bins are declared which cover the value from 0 to
255. If these cover groups are 100% covered which implies that functional coverage is 100%.

IV. RESULTS AND DISCUSSIONS

- Fig.5 indicate that when dlcr value is zero then data field will be absent from the frame. Frame will be transmitted up to the ack bit after that wait is enabled in the transmitter for dominant bit from the receiver side. Receiver will remove the bitstuffing which was done by the transmitter while sending the frame. It will remove the crc bits from the transmitted frame and compare that crc with the crc which is stored in memory. If they match

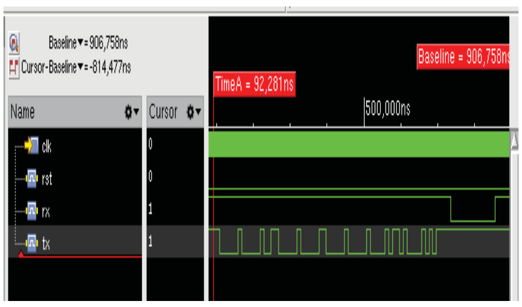

Fig. 5. 0 BYTE FRAME SIMULATION

means that dominant bit will be sent by the receiver to the transmitter. On receiving the dominant bit, ack delimiter will be sent by the transmitter. After that end of frame bits, interframe space and bus idle bits are sent.

- Fig.6 shows simulation when 1 byte data is transferred. Since the data field depends on the dlcr. So if dlcr is 1, means 1 byte data will be transferred along with the frame. So the length of the frame increases.

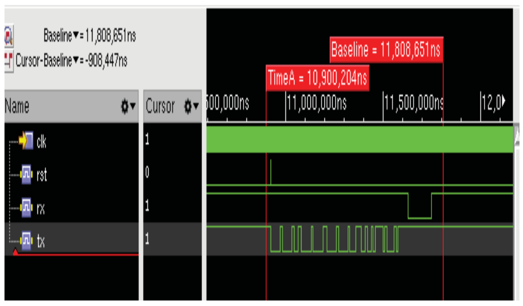

Fig. 6. 1 BYTE FRAME SIMULATION

- Fig.7 shows the simulation when 7 byte data is transferred. Now dlcr value is 7. From the simulation waveform it can be seen that now tx is toggling more. This indicates that the frame length has been increased.
- Fig.8 shows the simulation when 8 byte data is transferred. Now dlcr value is 8. From the simulation wave- form it can be seen that now the data length has reached to the maximum point. The receiver will send the acknowledgement only when the frame up to the acknowledgment bit is received. It will toggle that bit and then send it to the transmitter. The transmitter on receiving the bit will send the remaining fixed bits. As the frame length is increased, it will take more time to receive the acknowledgment.

TABLE II
CAN VIP CODE COVERAGE

Type of Code Coverage	Percentage Covered
Block coverage	100%
Toggle coverage	100%

TABLE III
CAN VIP FUNCTIONAL COVERAGE

Covergroup	Coverpoint	Bins covered	Total bins	Percentage
can_cg	DLCR	9	9	100%
	ID	1	1	100%
	IDE	1	1	100%
	SRR	1	1	100%
	RTR	1	1	100%
	R1	1	1	100%
	R0	1	1	100%
	SOF _	1	1	100%
	ACCEPTANCE CODE	1	1	100%
	ACCEPTANCE MASK	1	1	100%
cg 1	data	9	9	100%

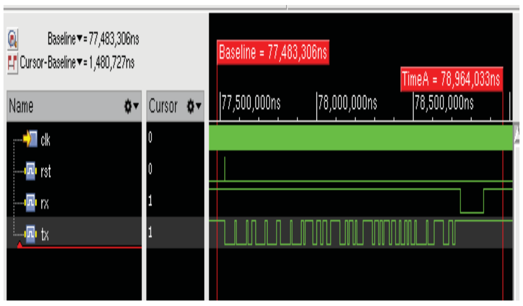

Fig. 7. 7 BYTE FRAME SIMULATION

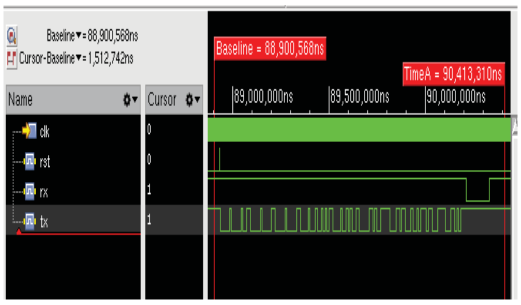

Fig. 8. 8 BYTE FRAME SIMULATION

- Table.II shows the code coverage of this VIP. The parts of the code coverage which are covered are Block coverage and Toggle coverage. They are found using the Xrun tool and loaded on the IMC. From the imc, this coverage can be viewed.
  Block coverage indicates whether all the statements inside the procedural blocks are covered or not. The procedural blocks which are present in the top module are initial and always blocks. Within the always block, the clock is generated. From inside the initial block, a test is run.
  Toggle coverage indicates how many times a signal is undergoing transition. A signal starting from 1 and at the end of the simulation, it should come back to the 1. If it is 0 at the starting, then at the end of the simulation it should come back to 0 only. Then only toggle coverage can be 100%.
- Table.III shows the functional coverage. Functional coverage indicates whether all the design specifications are covered or not. In this figure there are two covergroups can cg and cg_1. In the can cg covergroup, there are ten coverpoints declared. Inside these coverpoints, bins are declared. If all the bins declared inside these coverpoints are hit then only coverage can reach 100%. In this figure, all coverpoints are having coverage of 100%. So all bins inside these coverpoints have been hit. In the next covergroup, there is only one coverpoint. This coverpoint is receiving data from a dynamic array. The index value is sampled and then covered. This coverpoint is having coverage of 100%. In this way, the addition of code coverage and functional coverage gives the total coverage.

V. CONCLUSION

In this paper, VIP of CAN is developed using SystemVerilog based UVM methodology. The test environment of this VIP is verified with the help of appropriate test cases. It achieved 100% functional coverage and code coverage. So the overall coverage is 100%. The simulation waveform of 0, 1, 7 and 8-byte frame shows that there is synchronisation between the transmitter and receiver. It also shows the varied length of the transmitter frame according to the size of the data field. Thus, it can be inferred that this VIP is having the functionalities of CAN protocol so it can be used to verify the CAN IP and it will reduce the complexity of a testbench for a SOC design having CAN IP.

REFERENCES

[1] Vaibhav Bhutada1, Shubhangi Joshi2, Tanuja Zende3, "Design and Implementation of CAN Bus Controller on FPGA".
[2] Pal-Stefan Murvay and Bogdan Groza, "Source Identification Using Signal Characteristics in Controller Area Network".
[3] Qinmu Wu, Yesong Li and Yi Qin, "A Scheduling Method Based on Deadline for CAN-based Networked Control Systems".
[4]K. Tindell, A. Burns and AJ. Wellings, "calculating controller area network(CAN) message response times".
[5]M. Santhosh Kumar and Dr.C.R.Balamurugan, "Self - Propelled Safety System Using CAN Protocol -A Review".
[6]Jens Eltze, "Integrated Controller Area Network (CAN) applications".
[7]Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee, Member, IEEE, "Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks" *IEEE Transactions on vehicular Technology*, VOL. XX, NO. XX, XXX 2015.
[8]William J. Slivkoff, San Jose, CA (US); Neil Edward Birns, Cupertino,CA (US); Hong Bin Hao, San Jose, CA(US); Richard Fabbri, Stamford, CT (US), "Can device featuring advanced can filtering and message acceptance".
[9]L. Almeida, J. A. Fonseca, P. Fonseca, "Flexible Time-Triggered Communication on a Controller Area Network".
[10]CAN Controller, "Lattice Semiconductor" April 2013.
[11]Xilinx LogiCORE IP CAN v3.2, "DS265 April 19, 2010".
[12]Bosch Controller Area Network (CAN) Version 2.0 Protocol Standard.
[13]Nicolas Navet, "Controller Area Network".
[14]K. Anwar and Z.A. Khan, "Dynamic priority based message scheduling on Controller Area Network".
[15] Gianluca Cena and Adriano Valenzano, "An Improved CAN field bus for Industrial Applications" *IEEE transactions on industrial electronics*, Vol. 44, No. 4, August 1997.
[16] Tsutomu Matsumoto, Masato Hata, Masato Tanabe, Katsunari Yoshioka, Kazuomi oishi, "A method of preventing unauthorized data transmission in Controller Area Network".
[17]R. Krishnamoorthy, "Controller Area Network as a serial communication protocol", in *Proceedings of Modelling and Simulation*, MS'2004, Lyon-Villeurbanne, 2004, pp. 13.17-13.19.