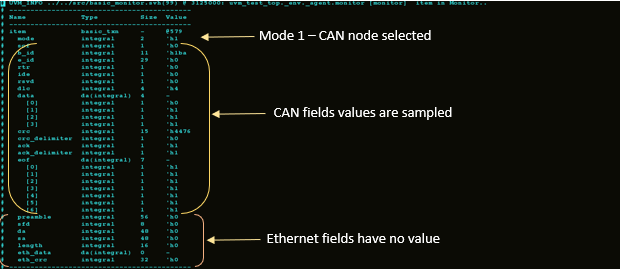
**Variable UVC:**

Normally the UVC can be coded for an IP/protocol, but if the protocol has some similarities in this case the serial interface protocols are taken, then the UVC is coded in a way to support more than one IP. The basic approach is variable controlled UVC. The UVC is used as specific IP, say CAN in the above section. If the UVC needs to be used as different protocol say Ethernet, then the UVC components/objects needs to be updated based on the mode variable in agent configuration file. Transaction should have the Ethernet variable along with the CAN variable. The driver should have the driving logic based on the ethernet protocol along with CAN and same for the monitor. If more than two IP needs to configure as UVC then size of the mode variable should increase. UVC is configured as Ethernet by setting the mode variable to ‘1’. The results are as follows for variable UVC as CAN. Observe the log file that the transaction file has both CAN and Ethernet variables. The Sequencer randomizes all the signals in the transaction class, but the monitor only displays the valid values in the CAN protocol and zeros in the Ethernet protocol, because the mode is selected as ‘0’. The results are shown in figure 6 and 7.

Figure 6: valid CAN and invalid Ethernet transaction from monitor

CAN fields values are sampled

The observation of the Figure 6 is as follows. Variable UVC is configured as CAN node, because the mode variable of the UVC configuration file is set to 1. Both CAN and Ethernet variables are present in the variable UVC transaction class, but only the CAN members of the sequence items present in Figure 6 has the valid values and the rest of the ethernet values are 0 in the UVC bus. We cannot use this method directly to develop an IP, because IP should have the transaction items specific to that particular protocol. In this case, we have both CAN and ethernet members present in the transaction class. An additional care should be given in the driver component to channel the IP members to their respective interface.

Mode 1 – CAN node selected



Figure 7: Valid CAN and Ethernet transaction from Driver

Figure 7 shows the transaction details of the variable UVC from the driver component. Two IP members are present in the transaction as discussed earlier. Driver places a request to the sequencer for the transaction item. Variable UVC’s sequence randomizes all the members of the CAN and ethernet members even though the mode is selected as 0. Thus, all the variables in the Figure 7 carries some random values or constrained random values. Now, the sequencer pushed the transaction item to the driver via the sequence item port. Driver parsed the mode variable from the configuration file and drives the respective members with the generated values, rest of the members are driven with value 0 in the variable UVC’s interface. CAN mode is selected in this scenario, so the CAN members will carry the valid values and the ethernet frames are invalid. The main disadvantage of this method is that it encapsulates other interfaces along with its own interface.

Mode value is changed to two to configure the UVC as an ethernet node. It is evident that only ethernet values are transmitted over the bus and CAN signals holds zero. The configuration of UVC is taken care in the test component. Ethernet hold the random values for preamble, SFD, DA, SA, length, payload and CRC which is shown in the Figure 8. If another protocol added in the specification, then the sequence item, driver and monitors need to be updated to incorporate the new addition.



Figure 8: Invalid CAN and valid Ethernet transaction from monitor

Figure 9 is same as Figure 7. Both the two protocols members carries the random data. But the major difference between figure 7 and 9 is the mode value, which is 0 for the former scenario and 2 for the later one. Driver also overrides the random values generated for CAN protocol from the UVC sequence to zeros. It transmits the CAN frames in the UVC bus serially via the interface.

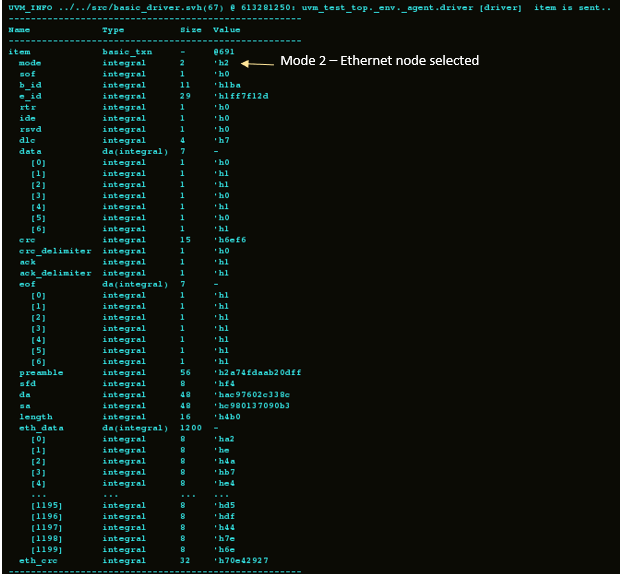


Figure 9: Valid CAN and Ethernet transaction from driver.