International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

AUTOMATED SEQUENCE ITEM GENERATION FOR DIFFERENT SERIAL COMMUNICATION PROTOCOLS

Thirumavalavasethurayar P[1], Dr. Ravi T.[2]

[1] Research Scholar, Faculty of electronics and communication,

Sathyabama institute of science and technology,

thiru_vlsi@zoho.com

Orcid ID: 0000-0002-2392-5415"

[2] Professor,Faculty of Electronics and Communication,

Sathyabama Institute of Science and Technology, Chennai, Tamil Nadu, India

ravivlsi123@gmail.com"

Abstract

Due to the complexity of Application Specific Integrated Circuits , the efforts required for designing and verifying are increasing at higher rate. Researches are mostly conducted to reduce the design time. Verification environment generates and drives the signals to design under test based on the respective frame format for every specific protocol. The coding process is done manually, which consumes more time. To overcome this, an automated script has been proposed to reduce the verification time. The frame format of a protocol which needs to be verified is fed into the spreadsheet, results in generation of sequence item. Thus, resulting in 75% reduction of time while coding the sequence item.

Keywords: Application Specific Integrated Circuits, Universal Verification Methodology, Testbench architecture, Universal Verification Components, Frame formats.

1.Introduction

Nowadays the ASIC designs are more and more complex. The design and verification teams are starting in parallel because the verification effort takes almost 70% of the design cycle time [2]. Due to the complex ASIC chips the direct verification is not possible [11]. The verification environment is developed in Verilog language at the earlier time [6]. With the inclusion of OOP concept, the system Verilog language is used for verification [13]. Industry uses various methodologies of system Verilog recently. They are ERM, VMM, OVM and UVM. UVM is the most commonly used verification methodology in ASIC industry [7]. The UVM is released by ACCELLERA on 2011[5]. The base class of the UVM can be inherited to code various components and objects. The components can be reused or overridden by registering the components or objects in the factory. By the modular structure of UVM the

Received: July 29, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

complex verification is more reliable and reusable [10]. Single system on chip uses various protocol or interconnects [9]. The major difference in these protocols are the frame format and the way the signals are driven into the Design Under Test DUT. The above mentioned two different things are taken care in the sequence item and the driver respectively. If the pack/unpack function is used in sequence item to address the different frame format and different driving mechanism. Some different protocol frame structures are UART frame structure [2], MAC or ethernet frame structure [4] and CAN frame format [14][8]. Even though the pack and unpack build-in functions of UVM used, the frame formats need to be coded individually in the sequence item for various protocols [1]. The proposed Automated script is used to convert the spreadsheet table into the sequence item for various protocols which saves more time in the verification process. Automated script is coded in Python scripting language. Some novel table architecture is used for coverage purpose only, but that did not reduce the verification time effort. That novel table ensures that the verification is done based on the verification plan [17]. These verification components are used in both block level testbench as well as the chip level test bench [12].

2. Basic UVM Testbench:

The basic UVM testbench architecture is shown in Figure 1. This test bench has been used to verify the basic functions of CAN protocol [3] or the Ethernet protocol. Testbench has the top module, test, environment, and the sequence files [15]. The details of the above-mentioned components are as follows.

Top:

The 16MHz clock is generated in top module. The reset generation is also done in the top module based on the specification whether the reset is active high reset or active low reset. The Design Under Verification (DUV) can be instantiated in the top. The interface is set in the configuration database and can be accessed anywhere in the test bench or the UVC by calling the get function from the configuration database. The get function fetch the interface and assigns it locally from the database. The run_test () built-in function starts the uvm_phases used in the components.

Basic Test:

Basic_test is extended from the uvm_test which is the built-in UVM class. UVM treats the uvm_test as component, so the component is registered in the factory using the macro `uvm_component_utils. Env, agent_cfg and sequence gets created using the create () function. Here the mode is set to ‘0’ to select the CAN UVC and the mode is set to ‘1’ to select the ethernet UVC. The mode variable is available in the agent configuration. The agent configuration is set in the configuration database after the specific configuration is set for the UVC. The object creation and setting in the database are done in the build_phases. The run_phases start and drop the objections. The run_phase of the test starts the specific sequence in the appropriate sequencer.

Received: July 29, 2025

International Journal of Applied Mathematics
Volume 38 No. 2s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Basic Env:

Basic_env is extended from the uvm_env which is the built-in UVM class. UVM treats the uvm_env as component, so the component is registered in the factory using the macro `uvm_component_utils. Basic_agent of the UVC gets created using the create () function.

Basic Sequence:

Basic_sequence is extended from the uvm_sequence which is the built-in parameterized UVM class. UVM treats the uvm_sequence as object, so the object is registered in the factory using the macro `uvm_object_utils. Basic_txn of the UVC gets created using the create () function. The transaction got randomized and sent to the driver via sequencer. This operation is repeated for five times which can be automated by getting the values through the valueplusargs in basic_test_cfg. Basic_test_cfg is the configuration file which configures the variables of the UVM testbench.

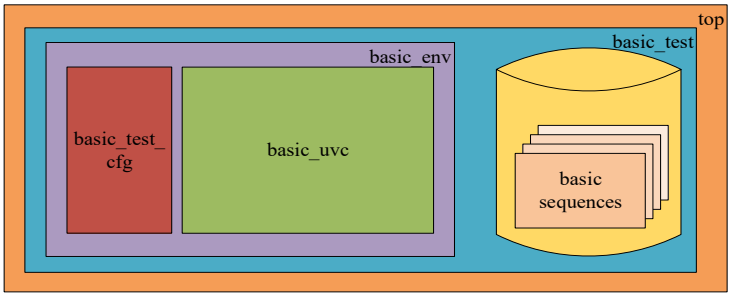

Figure 1: Basic UVM Testbench Architecture

3. Basic UVC:

The architecture for the basic UVC is shown in Figure 2. The UVC is basically an IP based on how the driver and monitor is coded. UVC has UVM components as well as UVM objects. UVC has agent configuration, agent, sequencer, driver, monitor, sequence_item and interface [16]. The detailed view of the above are explained below.

Basic Agent Config:

Basic agent configuration configures the specific UVC. Basic_agent_cfg is extended from the uvm_object which is the built-in UVM class, so the basic_agent_cfg is registered in the factory using the macro `uvm_object_utils. In this scenario the UVC is used as CAN node by configuring the mode variable as '0'.

Basic Transaction:

Basic txn is the transaction which extends from the uvm_sequence_item. Uvm_sequence_item is child class of uvm_object, so basic_txn is registered in factory using the macro 'uvm_object_utils. Here the fields of specific IP frame or the variables are randomized or packed based on the requirements. All the CAN frame fields are randomized using the rand keyword in system Verilog. The variables are sof-start of frame, b_id-base identifier, e_id-extended identifier, rtr-remote transmission request, ide-identifier extension,

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

rsvd-reserved, dlc-data length code, data, crc-cyclic redundancy check, crc_delimiter, ack-acknowledgment, ack_delimiter, eof-end of frame and inter_frame_gap. Constraints are used in the sequence item to handle randomization properly. Say sop of the CAN protocol is always '0', then we can use the constraints as "constraint c_sof {sof == 1'b0;}" similarly all the sequence items may or may not use the constraints based on the protocol specification. Pack/Unpack build-in UVM functions can be used to pack the variable into frames and vice-versa.

Basic Interface:

Interface is the bunch of signals. The single bit data signal used in the interface. Normally interface may have clocking blocks and modports. This UVC supports only serial interface protocol. Interface has virtual and physical interfaces to separate the dynamic and static worlds in a testbench. Objects created in the testbench environments are dynamic world and the module and DUT are considered as static world.

Basic Driver:

Basic driver class extends from uvm_driver. Uvm_driver is a component, so the same is registered in factory through the UVM macro `uvm_component_utils. Basically, the transaction variables are converted into signals in driver. Agent cfg and the basic interface are got from the configuration database in build phase via the uvm_config_db method. After reset the sequence item is put from the sequencer in the run phase. Once the randomize values are get from the sequencer then the variables are packed and formed as frame in driver. The frame is split into serial data and get transferred for every clock. The inter_frame_gap i.e. the idle cycles are transferred after sending the entire frame serially.

Basic Monitor:

Basic Monitor class extends from uvm_monitor. uvm_monitor is a component, so the same is registered in factory through the UVM macro `uvm_component_utils. Basically, the signals from interface gets converted into transaction variables in monitor. Agent cfg and the basic interface gets from the configuration database in build phase via the uvm_config_db method. After reset the monitor monitors signals and gets collected in the run phase. The collected data are packed to form the frame in monitor. The frame gets unpacked and assigned it to the specific sequence items in transactions. Finally, the transactions may transfer to the scoreboard for data checking or for coverage via the analysis port.

Basic Agent:

Basic agent class extends from uvm_agent. uvm_agent is a component, so the same is registered in factory through the UVM macro `uvm_component_utils. Agent creates the UVC architecture. The sequencer, driver and monitor get created in the agent through the create function of UVM in build phase. The connection between the sequencer and driver through sequence_item_port and the connection between the monitor and scoreboard/coverage collector via analysis port are done in the connect phase.

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

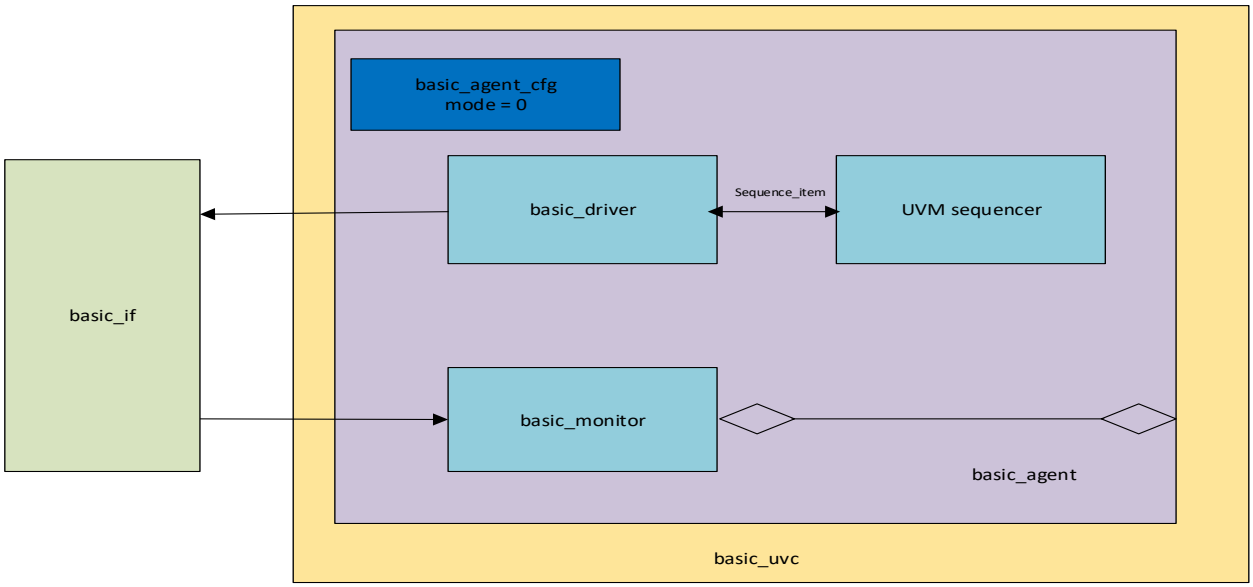

Figure 2: Basic UVC Architecture

The testbench is compiled and simulated using the QUESTA simulator. The results of this basic UVC used as CAN in Figure 3,4 and 5.

The fragments of log file which shows the CAN frame:

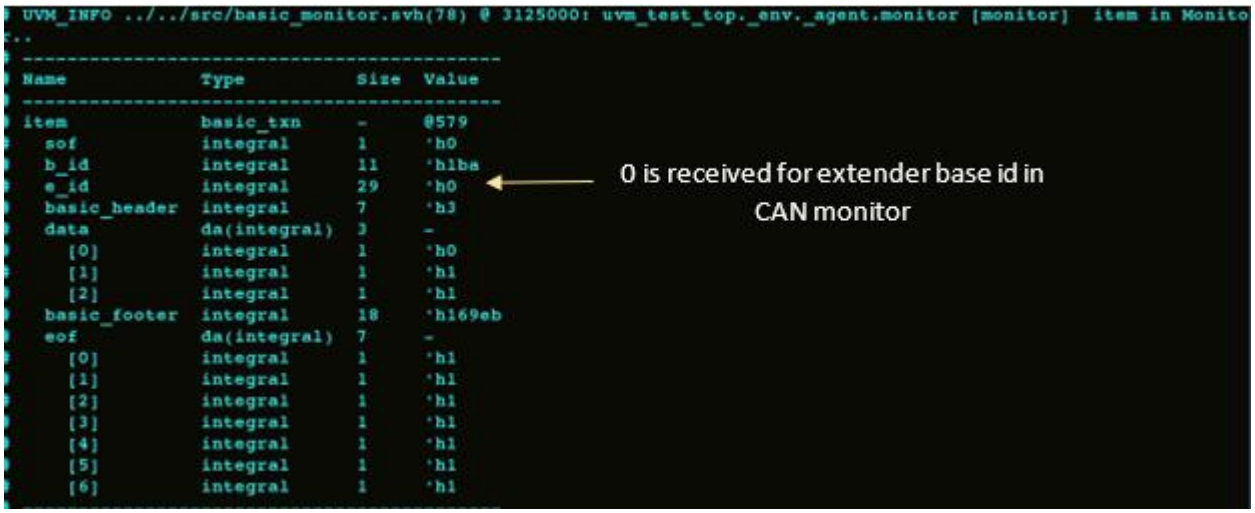

Figure 3: CAN transaction from Monitor

CAN frame format is captured in Figure 3. The transactions which are transmitted from the CAN bus is sampled in the monitor and logged the same in the simulation log file. CAN frame fields monitored are SOF, base or extended identifier, header and footer part. Finally, the completion of the CAN frame is identified by receiving the 7 sequential 1's.

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

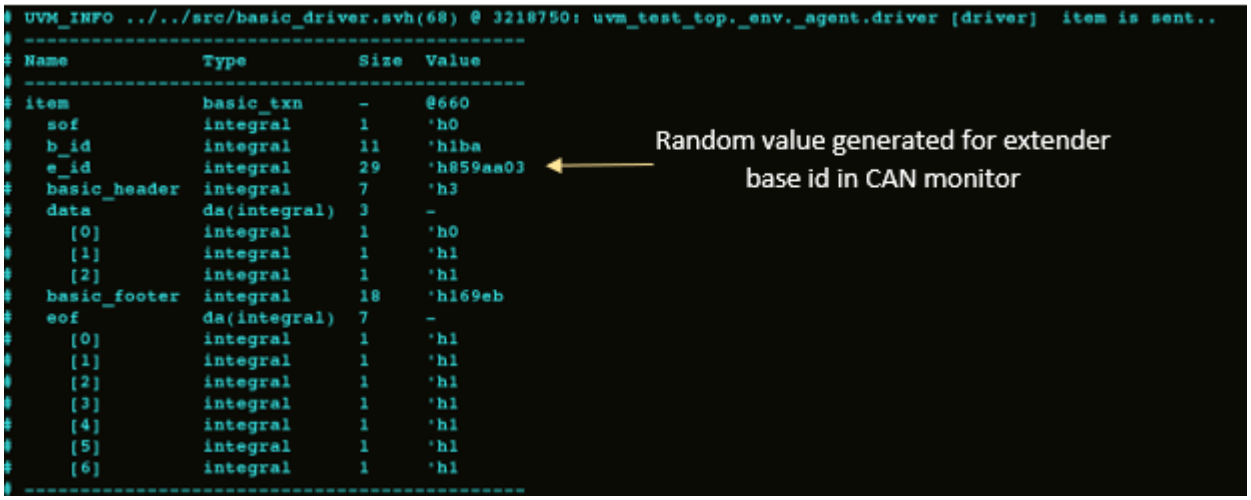

Figure 4: CAN transaction from Driver

CAN frame format transmitted in the CAN bus is shown in Figure 4. The transactions are already explained for Figure 3, but there is one main difference between the driver and monitor transactions. Even though the extended base identified is generated randomly in the driver, CAN DUT does not transmit the same in the CAN bus. So, CAN monitor samples zero for the e_id frame field.

Wave for the basic UVC used as CAN protocol:

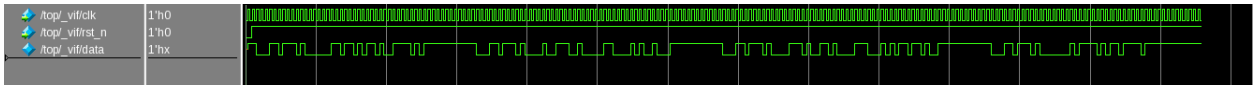

Figure 5: CAN Waveform

Figure 5 shows the clock, reset and data values sent over the CAN bus. All the frame fields are transmitted serially in the CAN bus.

4.UVC Supports for Multiple Protocols:

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

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

in the Ethernet protocol, because the mode is selected as '0'. The results are shown in figure 6 and 7.

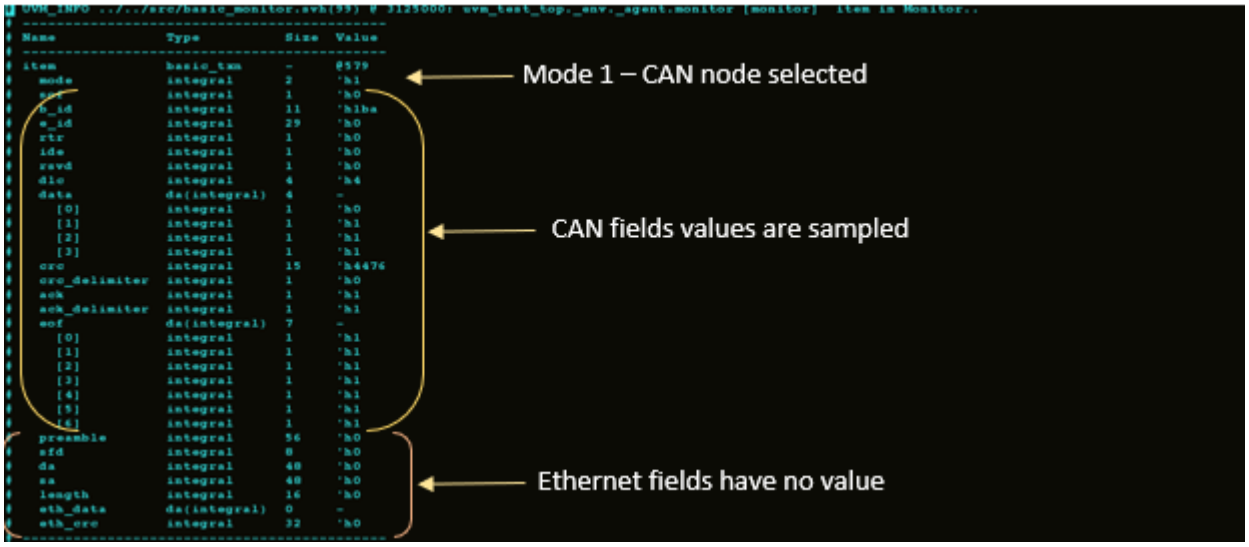

Figure 6: valid CAN and invalid Ethernet transaction from monitor

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.

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

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

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

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 of the protocol 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.

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

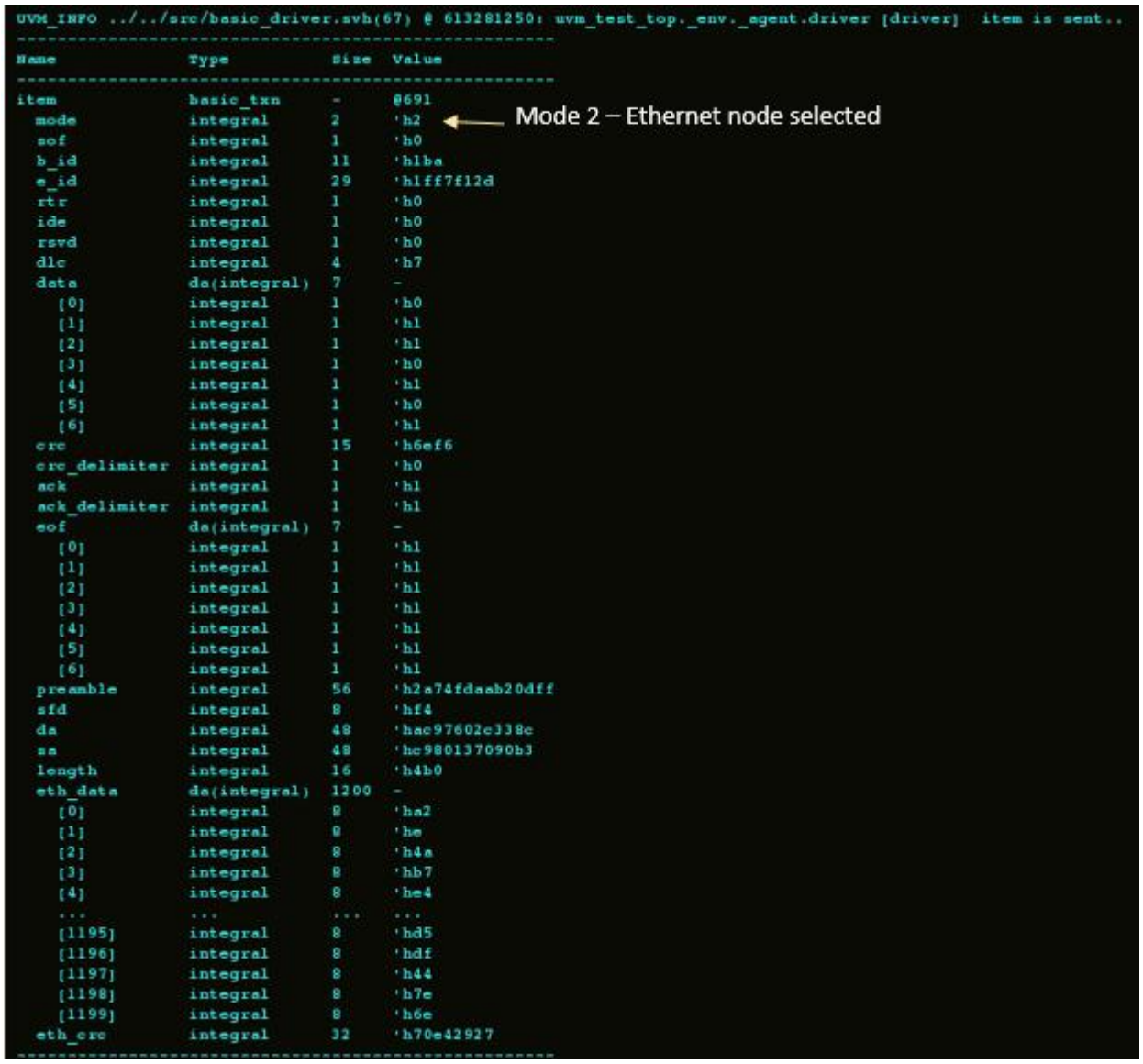

Figure 9: Valid CAN and Ethernet transaction from driver.

5. Override UVC:

The code is bit messy in variable UVC approach, if the UVC supports a greater number of protocols as all the protocols needs to be coded in a single file. It makes the process more complicated, if there occurs an error that needs to be debugged and then it becomes difficult to decide on which protocol the error is. There is a method in UVM called overrides which is useful in overriding the class only the class is registered in factory. The overriding is done in basic test file. Two transaction needs to be created: one for CAN and other one for ethernet. Both the transactions need to be extended from the basic transaction. The basic transaction has some common variables for the protocol used. Mode and the inter_frame_gap are the common variables, so they are declared in the basic transaction. The can variables are declared in the CAN transaction, similarly for ethernet. If the mode is '0' then the basic transaction is

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

overridden by CAN transaction by set_type_override_by_type method. The base class is the first argument and the child class object is the last argument for that method. If there is an error in the CAN pack/unpack methods, then the debug is much simpler compared to the multiple UVC. Since, the file contains only one protocol logic. Observe the log files in Figure 10 and 11, now the transaction has only CAN variable because the mode value is '0'.

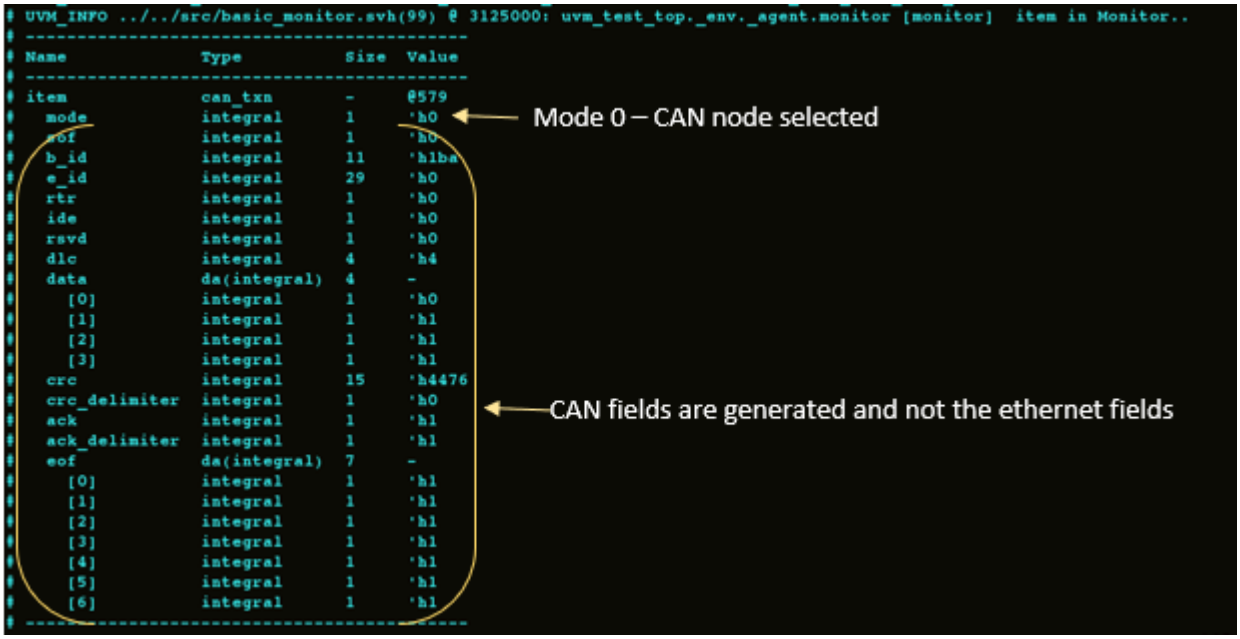

Figure 10: CAN transaction from monitor with mode 0

Mode 0 is selected from the test component. Variable UVC's sequence item has both the CAN members and the ethernet members, but only basic sequence item is present for override UVC approach. The appropriate transaction items override the basic sequence item based on the mode value. CAN transaction item file is selected, because mode variable carries the value 0. So, CAN members only present in both the interface and the sequence item. Now the monitor captures only the CAN member values and it does not bother about the ethernet members which is shown in Figure 10.

Unlike variable UVC there are no overhead present in driver to segregate the signals based on the mode value. Driver gets the randomized variables from the sequence and directly transmits the same in to the bus. Randomized CAN members are received in the driver because of the mode value 0 and there are no ethernet members. Now the CAN transactions are transmitted serially in the CAN bus which is shown in Figure 11. Driver logic is independent of mode variable. Thus, UVC will not get disturbed if another protocol got added. Only change required is that the new protocol needs to override the basic transaction item. Variable UVC changes mostly all the components of UVC which consumes more time, but the override UVC method adds only a single line to the existing code which is an efficient and a faster way.

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

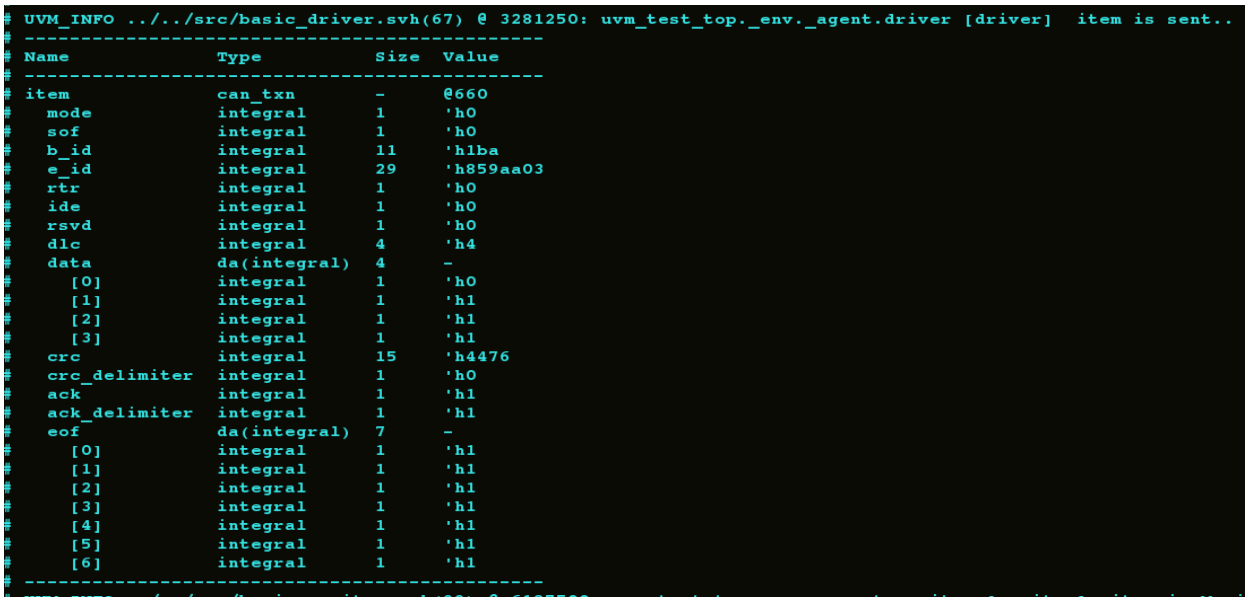

Figure 11: CAN transaction from driver with mode 0

After changing the mode variable to '1' then the UVC acts as an ethernet node and the results are shown in Figure 12 and 13. Here, only the ethernet members are present in the transaction of the override UVC which is shown in Figure 12. The driver converts the UVC's transaction into ethernet signals and transmits it over the bus.

Monitor does an exact opposite function of driver. The transmitted values are broadcasted into the UVC bus. The signals got sampled from the interface once the proper control signals are received. Then the collected samples are converted into the transaction of override UVC type. Ethernet frames are decoded into the respective ethernet members via the unpack system function present in the uvm_transaction base class. The monitor transaction is shown in Figure 13.

Figure 12: Ethernet transaction from monitor with mode 1

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

```
# UVM_INFO ../../src/basic_monitor.svh(99) @ 613125000: uvm_test_top._env._agent.monitor [monitor]  item in Monitor..
# ------------------------------------------------
# Name        Type          Size  Value
# ------------------------------------------------
# item        eth_txn       -     @661
#   preamble  integral      56    'h2a74fdaab20dff
#   sfd       integral      8     'hf4
#   da        integral      48    'hac97602c338c
#   sa        integral      48    'hc980137090b3
#   length    integral      16    'h4b0
#   eth_data  da(integral)  1200  -
#     [0]     integral      8     'ha2
#     [1]     integral      8     'he
#     [2]     integral      8     'h4a
#     [3]     integral      8     'hb7
#     [4]     integral      8     'he4
#     ...     ...           ...   ...
#     [1195]  integral      8     'hd5
#     [1196]  integral      8     'hdf
#     [1197]  integral      8     'h44
#     [1198]  integral      8     'h7e
#     [1199]  integral      8     'h6e
#   eth_crc   integral      32    'h70e42927
# ------------------------------------------------
```

Figure 13: Ethernet transaction from driver with mode 1

6. Automated UVC:

Even though the overrides are used in the variable declaration, pack and unpack needs to be coded manually. There is huge amount of human time and effort is needed while creating the frames from the variables. Python scripting is used to create the script to convert the spreadsheet into the transaction class. The spreadsheet should fill in the following way. Column A: name of the variable, Column B: bit size, Column C: Data type, Column D: pack (Whether variable needs to pack or not) Column E: unpack (Whether variable needs to unpack or not) Column F: Compare (Whether variable needs to compared in scoreboard or not), Column G: rand (Whether variable needs to randomize or not), Column H: FA (Fixed Array) or DA (Dynamic Array), Column I: Factory (Whether variable needs to be registered in factory to use the pack/print/unpack/compare or not) , Column J: This needs to filled only when the Column H is DA. Size of the Dynamic array. Possible values are Variable name or fixed value. Column K and L: Constraint details. L is for constraints of dynamic array and for the rest column K is used. If there is more than one constraint is used for a variable, then the constraints are separated by ';'. The example of CAN and ethernet spreadsheet is shown in Figure 14 and 15 respectively.

	A	B	C	D	E	F	G	H	I	J	K	L
1	sof	1	bit	pack	unpack	compare	rand	FA	Factory		1'b0	
2	b_id	11	bit	pack	unpack	compare	rand	FA	Factory		11'h1BA	
3	e_id	29	bit			compare	rand	FA	Factory			
4	rtr	1	bit	pack	unpack	compare	rand	FA	Factory		1'b0	
5	ide	1	bit	pack	unpack	compare	rand	FA	Factory		1'b0	
6	rsvd	1	bit	pack	unpack	compare	rand	FA	Factory		1'b0	
7	dlc	4	bit	pack	unpack	compare	rand	FA	Factory			
8	data	1	bit	pack	unpack	compare	rand	DA	Factory	dlc		
9	crc	15	bit	pack	unpack	compare	rand	FA	Factory			
10	crc_delimiter	1	bit	pack	unpack	compare	rand	FA	Factory			
11	ack	1	bit	pack	unpack	compare	rand	FA	Factory		1'b1	
12	ack_delimiter	1	bit	pack	unpack	compare	rand	FA	Factory		1'b1	
13	eof	1	bit	pack	unpack	compare	rand	DA	Factory	7		1'b1
14	inter_frame_gap	2	bit				rand	FA			2'h3	

Figure 14: CAN Frame in Spreadsheet

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The signal size of the CAN frames are filled in the Column B. CAN frames does not have e_id and inter_framr_gap, so the cell D3, E3, D14 and E14 in the Figure 14 are left blank. Only data and eof are mentioned as dynamic array and rest are fixed array. So, the column J8 and J13 has dlc and 7 respectively, which are the size of the dynamic array. The directed values are entered in the column K with integer values and for random values the respective cells in column K are left blank. The directed dynamic data values are entered in column L.

	A	B	C	D	E	F	G	H	I	J	K
1	preamble	56	bit	pack	unpack	compare	rand	FA	Factory		55:55 => 0
2	sfd	1	byte	pack	unpack	compare	rand	FA	Factory		
3	da	48	bit	pack	unpack	compare	rand	FA	Factory		
4	sa	48	bit	pack	unpack	compare	rand	FA	Factory		
5	length	16	bit	pack	unpack	compare	rand	FA	Factory		46:1500
6	eth_data	8	bit	pack	unpack	compare	rand	DA	Factory	length	
7	eth_crc	32	bit	pack	unpack		rand	FA	Factory		
8	inter_frame_gap	2	bit				rand	FA			3;<=3

Figure 15: Ethernet Frame in Spreadsheet

Ethernet frames does not have inter_framr_gap. So, the cell D8 and E8 in the Figure 15 are left blank. Only eth_data is mentioned as dynamic array and rest are fixed array. So, the column J6 has length, which are the size of the dynamic array. The directed dynamic data values and their respective constraints are entered in column L.

This spreadsheet is given as an input for python script and the appropriate transaction file can_txn.sv and eth_txn.sv is created same as the files in override UVC. To run the python script the command needs to be “python <script_name.py> <spreadsheet_name>”. transaction class name is created only from the name of the spreadsheet. So, do not select the same name for both the protocols. The bit size for single bit sized variable and the vector typed variable got created, if the variable is dynamic array, then the symbol needs to be appended. To register in factory Column D, E and F values are used. Different macro is selected for dynamic array while registering in factory. Constraint got coded based on the column J and K. Pack method is coded by the column D and H. Unpack method is done based on the columns E, H and J. The generated basic_txn is overridden by can_txn or eth_txn same as the case of overriding UVC.

7. Results and Conclusions

For testing, UART frame format is demonstrated here. UART Spreadsheet is created in Figure 16. On purpose, a wrong data has been entered in the spreadsheet to check whether the script works as expected.

1	start	1	bit	pack	unpack	compare	rand	FA	Factory		1'b1
2	data	9	bit	pack	unpack	compare	rand	DA	Factory		
3	parity	1	bit	pack	unpack	compare	rand	FA	Factory		
4	stop	2	bit	pack	unpack	compare	rand	FA	Factory		1'b1

Figure 16: UART Frame in Spreadsheet

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

UART has start, data, parity and stop signal whose sizes are 1, 9, 1 and 2 bits as shown in Figure 16. All the fields are present in the UART protocol so the column D and E are mentioned as pack and unpack respectively. In the above excel sheet, data is declared as “DA” but the length is not given. Then the python script throws an error as Figure 17.

```
Traceback (most recent call last):
  File "excel.py", line 57, in <module>
    if str(sheet.cell_value(i, 11)) != "" :
  File "/smsc/tech/tools/python/Linux/x86_64/lib/python2.5/site-packages/xlrd/sheet.py", line 412, in cell_value
    return self._cell_values[rowx][colx]
IndexError: list index out of range
```

Figure 17: Error message for missing field in Spreadsheet

After changing the dynamic array to fixed array for data the uart_txn.svh file is created as expected in Figure 18.

```
class uart_txn extends basic_txn;

  rand  bit           start;
  rand  bit [8:0]         data;
  rand  bit           parity;
  rand  bit [1:0]         stop;

  `uvm_object_utils_begin(uart_txn)
    `uvm_field_int(start,UVM_ALL_ON|UVM_NOPACK)
    `uvm_field_int(data,UVM_ALL_ON|UVM_NOPACK)
    `uvm_field_int(parity,UVM_ALL_ON|UVM_NOPACK)
    `uvm_field_int(stop,UVM_ALL_ON|UVM_NOPACK)
  `uvm_object_utils_end

   constraint c_start_0 {start == 1'b1;}
   constraint c_stop_0 {stop == 1'b1;}

  function new (string name = "uart_txn");
    super.new(name);
  endfunction:new

  function void do_pack (uvm_packer packer);
    super.do_pack(packer);
    packer.pack_field_int(start,$bits(start));
    packer.pack_field_int(data,$bits(data));
    packer.pack_field_int(parity,$bits(parity));
    packer.pack_field_int(stop,$bits(stop));
  endfunction : do_pack

  function void do_unpack (uvm_packer packer);
    super.do_unpack(packer);
    start = packer.unpack_field_int($bits(start));
    data = packer.unpack_field_int($bits(data));
    parity = packer.unpack_field_int($bits(parity));
    stop = packer.unpack_field_int($bits(stop));
  endfunction : do_unpack

endclass : uart_txn
```

Figure 18: Generated Sequence item code from spreadsheet

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Time taken to write the transaction class is reduced by creating the spreadsheet using the python script. It saves enormous amount of time and human effort. The manual or human efforts are reduced by 75% as per Table 1, Figure 19 and 20. If the specification of protocol is changing as per the R&D inputs, then the changes need to be updated manually in the spreadsheet. Human error is reduced by automating the coding process. This automated UVC works only for the serial interface protocols. The total time span of the work done has been explained in the below table. The time taken by variable UVC increased exponentially by the number of protocol present in the design. But the override UVC takes only half compared to the variable UVC. The sequence generation and integration time for the automated UVC is almost nil. There is no latency in time even though the number of protocols used in the design grows in large number.

	CAN	Ethernet	UART	ENV	Total man power in weeks
Variable UVC	2	2	2	2	8
Override UVC	1	1	1	2	5
Automated UVC	0	0	0	2	2

Table 1: Timeline for the generation of UVC's using various methods

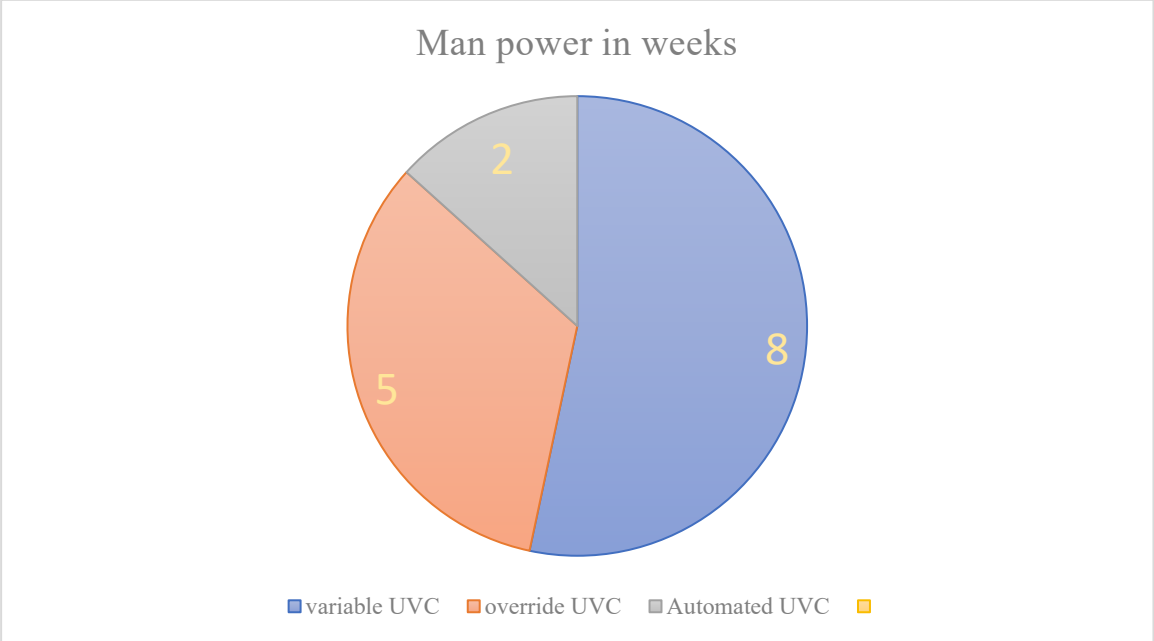

Figure 19 : Pie Chart for man power in weeks

Received: July 29, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

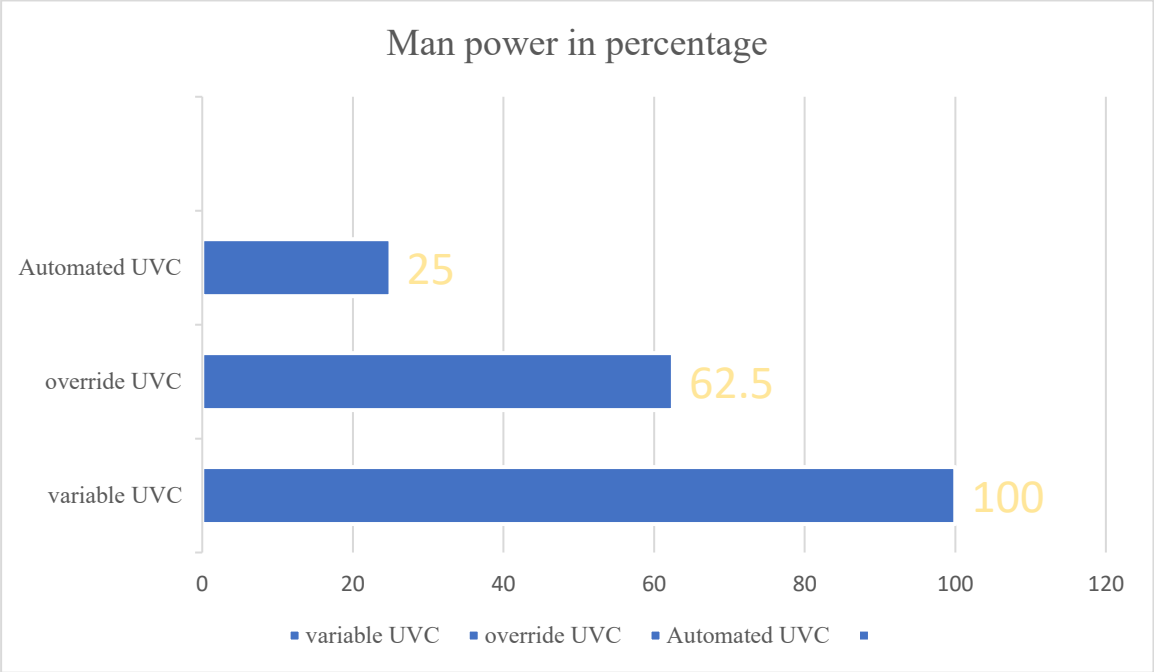

Figure 20 : Bar Chart for man power in percentage

Future work: This script needs to be updated if the same fields gets repeated. The script can be updated to create the driver and monitor based on the spreadsheet inputs; this helps in the usage of the UVC for all types of protocols.

References

[1] Ahmad, H., Gulzar, M. M., Aziz, S., Habib, S., & Ahmed, I. (2024). AI-based anomaly identification techniques for vehicles communication protocol systems: Comprehensive investigation, research opportunities and challenges. *Internet of Things*, Volume 27.

[2] Biswajit, D., Md, H., & Foisal, A. (2015). Design and Implementation of UART Serial Communication Module Based on FPGA. *International Conference on Materials, Electronics & Information Engineering, ICMEIE*, 5-6.

[3] Brodie, S., & Oksanen, T. (2025). Securing CAN-Based ISO 11783 communications in agricultural vehicles using OPC UA. *Computers and Electronics in Agriculture*, Volume 231.

[4] Chitti, S., Chandrasekhar, P., & Rani, M. A. (2015). Gigabit Ethernet Verification using Efficient Verification Methodology. *International Conference on Industrial Instruments and Control (ICIC)*, 1231-1235.

[5] Elakkiya, C., N, S. M., Babu, C., & Jalan, G. (2017). Functional Coverage-Driven UVM Based JTAG Verification. *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 1-7.

[6] Gollapudi, A., Kumar, A. K., & Charyul, M. L. (2024). Implementation of an Optimized AXI using Verilog. *International Conference on Circuit Power and Computing Technologies (ICCPCT)*, 684-690.

Received: July 29, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[7] Khalifa, K. (2017). Extendable generic base verification architecture for flash memory controllers based on UVM. *IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD), Wellington, New Zealand*, 584-589.

[8] Larry, L. P., & Bruce, S. D. (2022). 2 - Direct Links. In M. Kaufmann, *Computer Networks (Sixth Edition)* (pp. 66-160). In The Morgan Kaufmann Series in Networking.

[9] Plasencia, F., Mitacc, E., Raffo, M., & Silva, C. (2018). Robust Functional Verification Framework Based in UVM Applied to an AES Encryption Module. *IEEE 2nd New Generation of Circuits and Systems (NGCAS)*, 194-197.

[10] Plasencia, F., Mitacc, E., Raffo, M., & Silva, C. (2019). Flexible UVM-Based Verification Framework Re usable with Avalon, AHB, AXI and Wishbone Bus Interfaces for an AES Encryption Module. *IEEE Latin American Test Symposium (LATS)*, 1-4.

[11] Saji, S. A., & Sivasankaran, K. (2017). Test suite for SoC interconnect verification. *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS), Vellore, India*, 1-6.

[12] Salah, K. (2017). A Unified UVM Architecture for Flash-Based Memory. *IEEE 18th Microprocessor and SOC Test and Verification (MTV)*, 1-4.

[13] Sharma, G., Bhargava, L., & Kumar, V. (2017). Self-Assertive Generic UVM Testbench for Advanced Verification of Bridge IPs. *14th IEEE India Council International Conference (INDICON)*, 1-6.

[14] Siddiqui, A. S., Gu, Y., J, P., & F, S. (2017). Secure communication over CANBus. *IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA,*, 1264-1267.

[15] Srinivas, M., & Musala, S. (2016). Verification of AHB_LITE protocol for waited transfer responses using re-usable verification methodology. *International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India*, 1-3.

[16] Thirumavalavasethurayar,, P., & Ravi, T. (2021). Implementation of Replay Attack in Controller Area Network Bus using Universal Verification Methodology. *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 1142-1146.

[17] Wang, p. (2017). A unique centralized-management methodology block/architecture and a novel random input stimulus controlled variable table implementation for the latest marvell ethernet PHY UVM verification platform. *IEEE International Conference on Integrated Circuits and Microsystems (ICICM), Nanjing, China*, 299-303.

Received: July 29, 2025