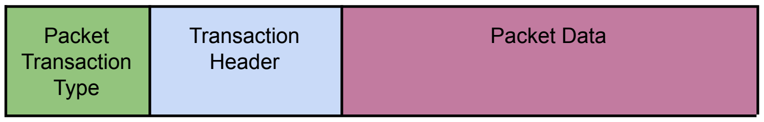
ASIC Verif Engineer Assignment

S/W Data Interface

1. Exchange Packet

Data is sent by S/W to the DUT is in the following application specific protocol



1.1. Packet Transaction Type

Byte 0:
This 8-bit field can take the following values :-
0: Message passthrough
1: Registers update
2-255: Reserved

1.2. Transaction Header

1.2.1. Message passthrough data format

Byte 0-3: Transaction ID
Byte 4: Message Type
- 0: MASS_QUOTE
- 1: HEARTBEAT
- 2-255: Reserved.

Byte 5: Connection ID

Bytes 6-7: Reserved
Byte 8: CheckSum
Bytes 9-20: Padding (to make the data aligned to 32 bytes)

1.2.2. Register Write Data Format

At least 1 register write needs to be packed in such a packet.

Byte 0-1: Transaction ID
Byte 2: Mode
- 0: Normal
- 1: Burst

Byte 3-4: Padding

1.3. Packet Data

1.3.1. Message passthrough data format

Bytes 21-...: Message Data can be represented by random Bytes between 128 and 1017 bytes

1.3.2. Registers write data format

Bytes 5-...: Maximum size of the register write data is 1016 bytes. At the minimum it should write to one register.

The format for two types of register writes are shown below

- ***Normal mode***: addr0 (4 bytes) | data0 (4 bytes) | addr1 | data1 | addr2 | data2 | addr3 | data3 | …

- ***Burst mode***: burst_config (4 bytes) | data0 (4 bytes) | data1 (4 bytes) | …. | burst_config (4 bytes) | data0 (4 bytes) | data1 (4 bytes) | ...

  In burst mode the 4 byte burst_config is defined as follows:
  burst_size (9 bits) | start_addr (23 bits)

  Multiple bursts are possible in a single packet. and would just be strung together like this

burst_config (4 bytes) | data0 (4 bytes) | data1 (4 bytes) | …. | burst_config (4 bytes) | data0 (4 bytes) | data1 (4 bytes) | **burst_config (4 bytes)** | data0 (4 bytes) | data1 (4 bytes) | …. | **burst_config (4 bytes**) | data0 (4 bytes) | data1 (4 bytes) | **…**c

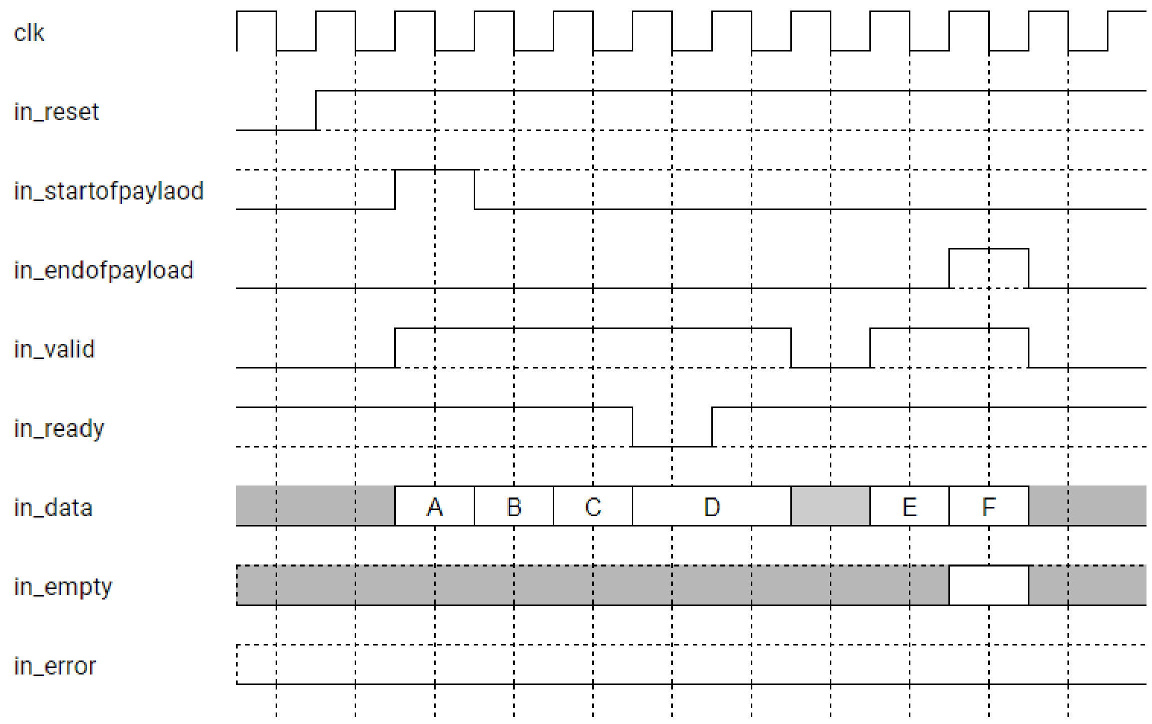
2. Input interface

The verification component to be designed, drives the following interface on the DUT. The description is below is from the point of view of the DUT.

Signal Name	Width (bits)	Direction	Description
clk	1	input	Positive edge triggered clock
reset_n	1	input	Active low reset
in_valid	1	input	High when incoming data is valid, low otherwise
in_startofpayload	1	input	High for 1 cycle, marks the beginning of incoming payload; should be qualified with in_valid
in_endofpayload	1	input	High for 1 cycle, marks the end of incoming payload; should be qualified with in_valid
in_ready	1	output	Asserted by the module to indicate that it is ready to accept data.
in_data	64	input	Data
in_empty	3	input	Always qualified when in_endofpacket is high. Indicates the number of empty bytes in the last

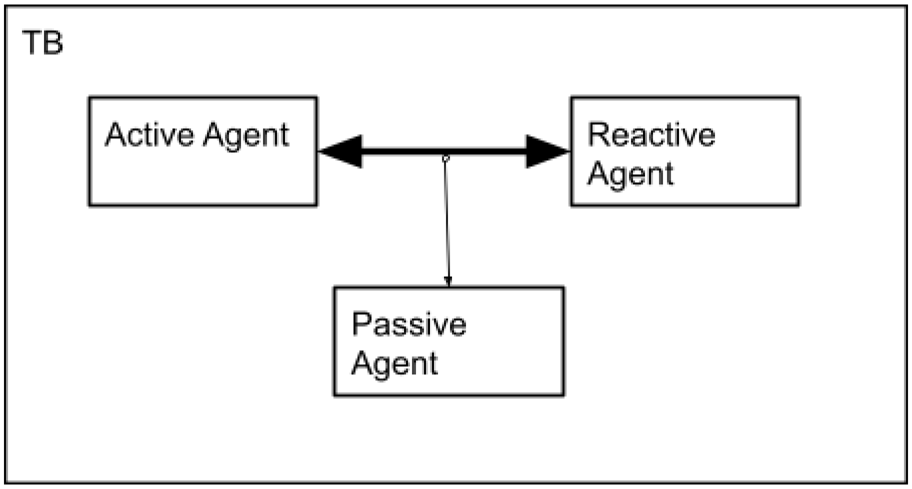
			cycle of the incoming payload
in_error	1	input	Used to indicate an error in the incoming data stream

Timing diagram is given below



3. Questions

1. Please state any assumptions that you are making about the DUT and the verification component
2. Write up a test-plan that will allow you to ensure that this verification component can be thoroughly tested.
3. Construct and cleanly compile a SV-UVM based agent that can drive and monitor the interface described. The agent needs to support being instantiated in passive/active modes, as well as the two types of active modes - proactive/reactive.
4. Construct the following test environment hierarchy. Once constructed you should be able to run a bring-up test that runs just the clock for 10 us (no need for a sequence for this question), and finish gracefully.



5. Write up a test-case that first drives a normal mode packet followed by a burst mode packet. The packet should be accepted successfully by the reactive and the passive agent. Use the reactive agent to stall the active agent.
6. Indicate all the knobs that allow you to randomize the packet so as to be able to get full coverage on the packet structure and satisfy your test plan.
7. **Bonus**: *This is completely optional and not answering will not be considered against you in any way. But if you do have any knowledge about this topic, this where you get to show-off.* Assuming that the reference model needs to be in C. Construct the appropriate structures, functions in SV and C, and the DPI calls that will allow this SV-UVM sequence item to be passed to C. You do not have to consider the rest of the C reference model and how the C-structure would be used by the rest of the model.

Please note for questions 2, 3, 4 - we expect fully working code and instructions for being able to replicate your results. For the others you could either add a section to this document or send in another suitably formatted one.

You will be judged on code quality, comments etc. A functionally accurate code that is hard to read is not preferred.

All the best!

4. Misc

For tools/compilation/elab, please use https://www.edaplayground.com. This site allows free access to most EDA tools without login etc though you are free to create one if needed. Please feel free to use any other tool at your disposal, the only caveat being that the simulator in which we try your results may not be the same as the one which you developed it.