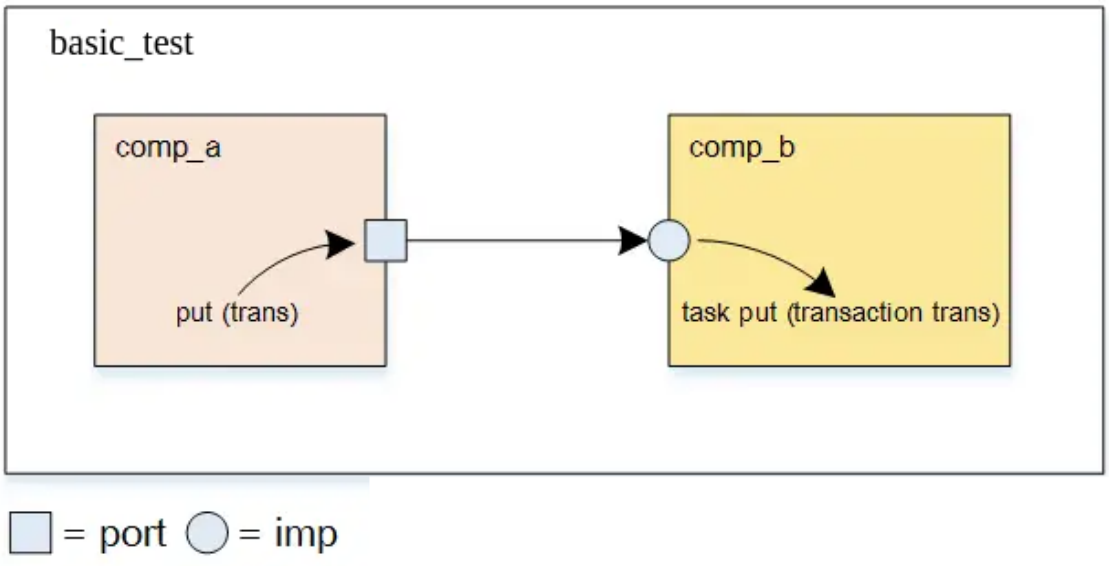
Connecting TLM Port and Imp Port



Let's consider an example consisting of two components component_a and component_b, and a transaction class.

- component_a and component_b objects are created in the test with the name comp_a and comp_b respectively
- transaction class is randomized in comp_a and sent to the comp_b through TLM Communication mechanism

Below are the steps to implement a TLM Communication mechanism between comp_a and comp_b.

1. Declare and Create TLM Port in comp_a
2. Declare and Create TLM Imp Port in comp_b
3. **Connect** TLM Port and Imp Port in basic_test
4. **Call** interface method in comp_a to send the transaction
5. **Implement** an interface method in comp_b to receive the transaction

Declare and Create TLM Port in comp_a

1. Declaring blocking put port with the name trans_out. As the comp_a sends the packet out of the component, so named as trans_out

```
uvm_blocking_put_port #(transaction) trans_out;
```

2. Creating the port

```
trans_out = new("trans_out", this);
```

3. Verify the above mentioned code in the component_a.sv file.

Declare and Create TLM Port in comp_b

1. Declaring blocking put imp port with the name trans_in. As the comp_b receives the packet from another component, so named as trans_in

```
uvm_blocking_put_imp #(transaction,component_b) trans_in;
```

2. Creating the port

```
trans_in = new("trans_in", this);
```

3. Verify the above mentioned code in the component_b.sv file.

Connect TLM Port and Imp Port in basic_test

1. Declare and Create the componet_a and component_b in basic_test class

```
component_a comp_a;

component_b comp_b;

comp_a = component_a::type_id::create("comp_a", this);

comp_b = component_b::type_id::create("comp_b", this);
```

2. Connect comp_a port and comp_b imp port

```
comp_a.trans_out.connect(comp_b.trans_in);
```

3. Place the connection code in the connect phase

```
function void connect_phase(uvm_phase phase);

  comp_a.trans_out.connect(comp_b.trans_in);

endfunction : connect_phase
```

4.Verify the above mentioned code in the basic_test.sv file.

Call interface method in comp_a to send the transaction

1. Randomize the transaction packet and send to comp_b by calling put() method

```
void'(trans.randomize());

trans_out.put(trans);
```

2. Place the code in run_phase

```
virtual task run_phase(uvm_phase phase);

  phase.raise_objection(this);


  trans = transaction::type_id::create("trans", this);

  void'(trans.randomize());

  trans_out.put(trans);


  phase.drop_objection(this);

endtask : run_phase
```

3. Verify the above mentioned code in the component_a.sv file.

Implement an interface method in comp_b to receive the transaction

1. In order to receive the transaction packet from imp port, explicit method has to be implemented
Implement put method with the input argument of transaction type

```
virtual task put(transaction trans);

  `uvm_info(get_type_name(),$sformatf(" Received trans on IMP
Port"),UVM_LOW)

  `uvm_info(get_type_name(),$sformatf(" Printing trans, \n
%s",trans.sprint()),UVM_LOW)

endtask
```

2. Verify the above mentioned code in the component_b.sv file.

Run the *basic_test.sv* file and observe the result. The transaction should be sent from comp_a to comp_b.

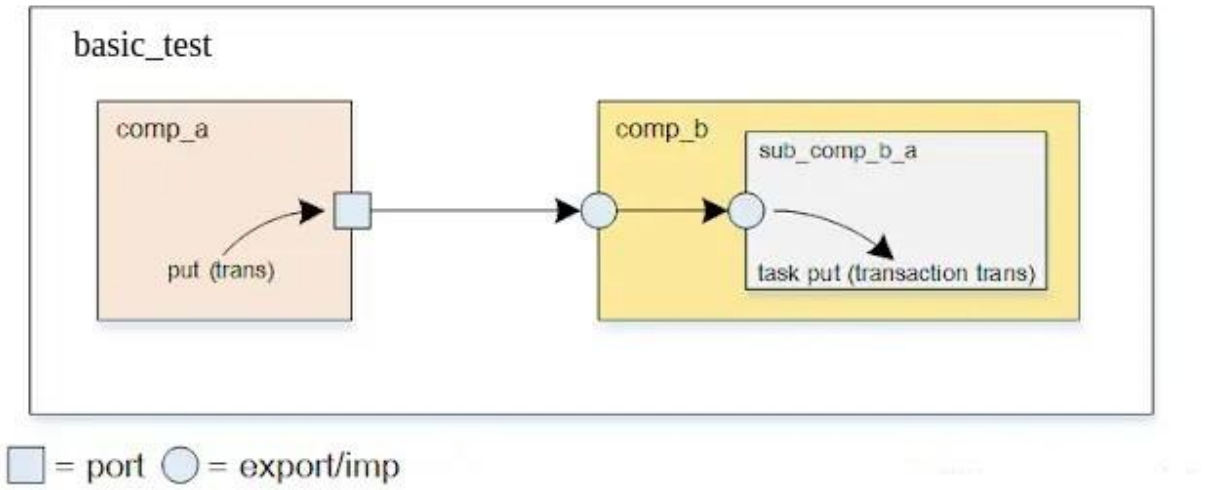
Task 1
As, we are using blocking put and imp ports. So, in order to verify the blocking nature.
Add some delay in the *put task* in comp_b and verify that the comp_a has to wait until the *put task* is returned back.

Submit the screenshot of the output.

For next tasks Submission
Make a directory with the name "Task2/Task3" and move all the code files in them and then start working (some files will remain the same). (This whole folder along with the snapshot are the submission for the respective task)

Task 2

Connecting TLM Port export imp port



Previously, we have seen connecting the port to the imp port.This task requires connecting TLM Port -> Export -> Imp_port.

Comp_a
This component doesn't need any change for this task.

Sub_comp_b_a (extended from comp_b)
This component involves the below steps

1. Declare the uvm_blocking_put_imp
2. Create the imp port (either in build phase or constructor)
3. Implement the put() method to receive the transaction

Comp_b

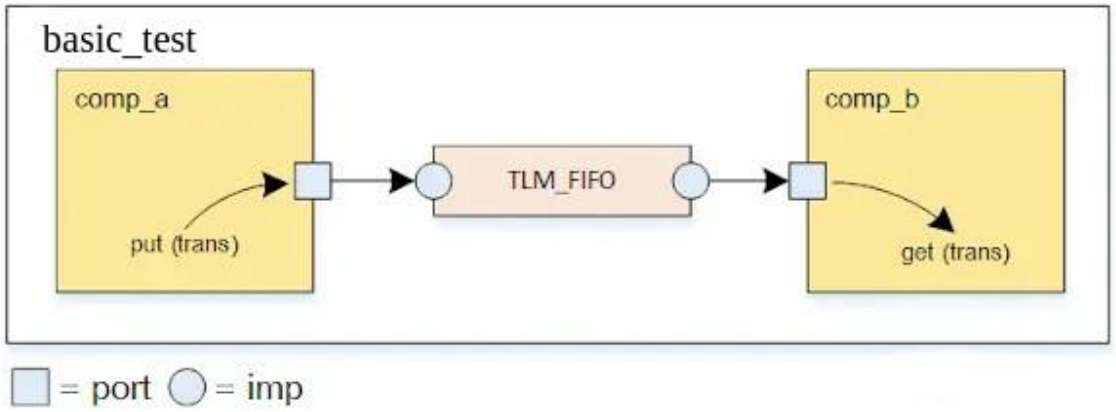
This component involves the below steps

1. Declare the uvm_blocking_put_export and `sub_comp_b_a`.
2. Create the export and `sub_comp_b_a` (in build phase)
3. Connect export to the imp port in connect_phase

Note: Use `uvm_info instead of $display for printing.

Task 3

TLM FIFO



The TLM FIFO provides storage for the transactions between the two independently running processes.

- FIFO can be used as a buffer between the producer and consumer
- TLM FIFO consists of put and get methods
- Producer port is connected to the put_export of the FIFO
- Consumer port is connected to the get_export of the FIFO

Comp_a
This component doesn't need any change for this task.

Comp_b

This component involves the below steps

1. Declare the `uvm_blocking_get_port.`
2. Create the get port (either in build phase or constructor)
3. Create the run_phase virtual task (don't forget to raise and drop the objection)
4. Use the get method of the get port to receive the transaction from the fifo.

Basic_test

This component involves the below steps

1. Declare the TLM FIFO (e.g, `uvm_tlm_fifo #(type) fifo_name;`)
2. Create the FIFO (in build phase)
3. Connect the TLM FIFO put_export with comp_a port
4. Connect the TLM FIFO get_export with comp_b port
   For example *comp.port.connect(fifo.put_export/get_export)* in the connect_phase