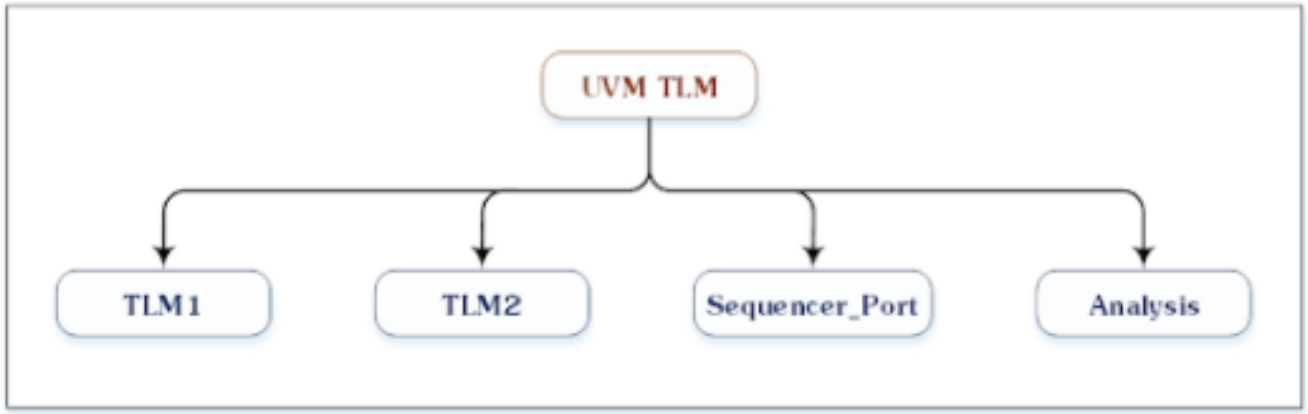
TLM Communications

Transaction Level Modeling (TLM)

- Used for communication among modules
- TLM interfaces consist of methods for sending and receiving the transaction
- All different types of TLM ports are used like PIPES to connect between the components

Introduction



- TLM1 ports provide blocking and non-blocking pass-by-value transaction-level interfaces
- TLM2 sockets provide blocking and non-blocking transaction-level interfaces with well-defined completion semantics
- Sequencer Port – A push or pull port, with well-defined completion semantics
- Analysis interface is used to perform non-blocking broadcasts of transactions to connected components

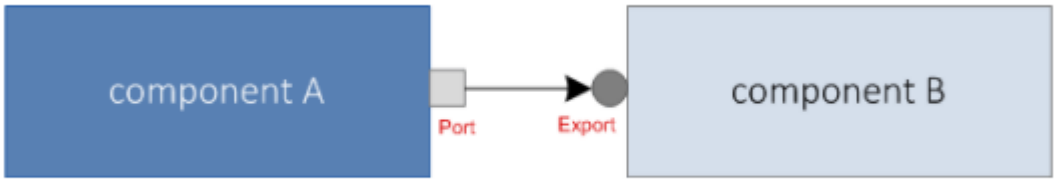
Introduction

- Each TLM interface is either blocking, non-blocking, or a combination of these two
- Blocking – Blocking TLM call methods will not return until the transaction has been successfully sent or retrieved
- Non-blocking – This attempts to convey a transaction without consuming simulation time
- Combination – A combination interface contains both the blocking and non-blocking variants

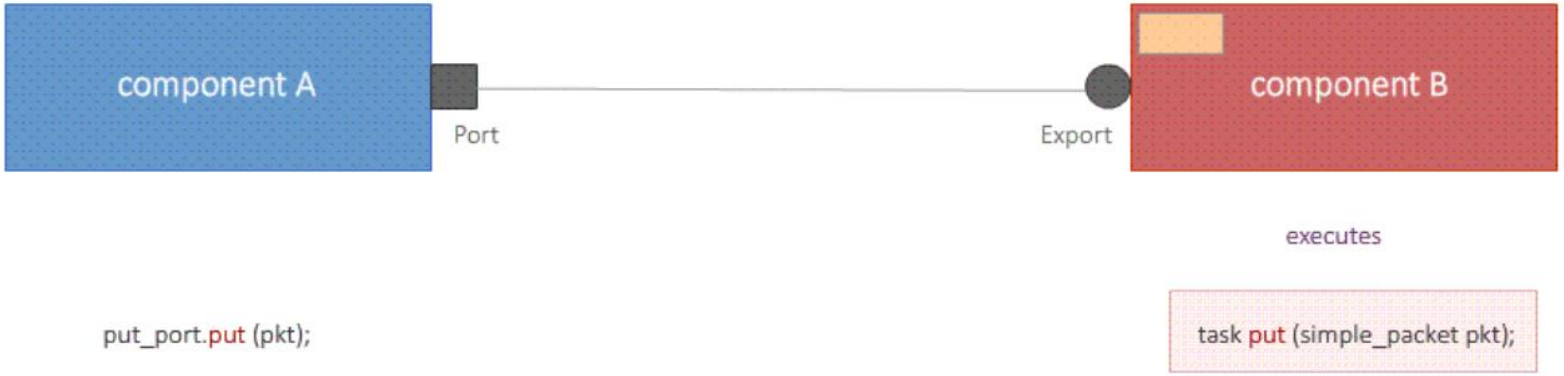
Introduction

- TLM Blocking put_port method
- TLM Non-Blocking put_port method
- TLM can_put method
- TLM Blocking get_port method
- TLM Non-Blocking get_port method
- TLM can_get method

```
class simple_packet extends uvm_object;
    `uvm_object_utils (simple_packet)

    rand bit [7:0] addr;
    rand bit [7:0] data;
         bit       rwb;

    constraint c_addr { addr > 8'h2a; };
    constraint c_data { data inside {[8'h14:8'he9]};

endclass
```



UVM TLM Blocking Put Port



- Any component can send a transaction to another component through a TLM put port.
- The receiving component should define an implementation of the put port.
- The implementation gives receiver a chance to define what has to be done with the incoming packet.
- The port can either be blocking or nonblocking in nature, that will decide whether the put method will block execution in the sender until the receiver accepts the object.

UVM TLM Port Example

```
class packet extends uvm_object;
  rand bit [7:0] addr;
  rand bit [7:0] data;

  `uvm_object_utils_begin(packet)
  `uvm_field_int(addr, UVM_ALL_ON)
  `uvm_field_int(data, UVM_ALL_ON)
  `uvm_object_utils_end

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

endclass
```

- A class called packet is defined to act as the data item that will be transferred from one component to another.
  - Has two random variables that can be randomized before sending.

1. Create sender class with a port type uvm_blocking_put_port

```
`include "packet.svh"

class componentA extends uvm_component;
  `uvm_component_utils(componentA)
  uvm_blocking_put_port #(packet) m_put_port;
  int m_num_tx;

  function new (string name="componentA", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase (uvm_phase phase);
    super.build_phase (phase);
    m_put_port=new("m_put_port", this);
  endfunction

  virtual task run_phase (uvm_phase phase);
    phase.raise_objection(this);
    repeat (m_num_tx) begin
      packet pkt = packet::type_id::create("pkt");
      assert (pkt.randomize());
      `uvm_info("compA", "Packet sent to compB", UVM_LOW);
      pkt.print(uvm_default_line_printer);
      m_put_port.put(pkt);
    end
    phase.drop_objection(this);
  endtask
endclass
```

- componentA is created
- Port has to be instantiated with the new() method preferably in the build_phase
- Ex: Packet is created, randomized and sent via the put_port handle by calling the put() method.
- Many such packets can be sent using a simple loop controlled by a configuration variable.

2. Create receiver class that implements the put method

```
class componentB extends uvm_component;
  `uvm_component_utils(componentB)
  uvm_blocking_put_imp #(packet, componentB) m_put_imp;
  function new (string name="componentB", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase (uvm_phase phase);
    super.build_phase (phase);
    m_put_imp=new("m_put_imp", this);
  endfunction

  virtual task put(packet pkt);
    `uvm_info("compB", "Packet received from compA", UVM_LOW);
          pkt.print(uvm_default_line_printer);

    endtask
endclass
```

- receiver class needs to define an implementation using uvm_blocking_put_imp.
- Since the port is blocking in nature, the put() implementation is a task which has to be defined by this component.

3. Connect port and its implementation at a higher level

```
`include "componentA.svh"
`include "componentB.svh"

class my_test extends uvm_test;
    `uvm_component_utils(my_test)
  componentA compA;
  componentB compB;
  function new (string name="my_test", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase (uvm_phase phase);
    super.build_phase (phase);
    compA=componentA::type_id::create("compA", this);
    compB=componentB::type_id::create("compB", this);
    compA.m_num_tx = 2;
  endfunction

  virtual function void connect_phase (uvm_phase phase);
    compA.m_put_port.connect(compB.m_put_imp);
    endfunction
  endclass
```

- The connection between a port and its implementation has to be done at a higher hierarchical level.
- Both components are instantiated directly within the test class in this example, the connection between them can be done during the connect_phase of the test.

3. Connect port and its implementation at a higher level

- Results

```
UVM_INFO componentA.svh(22) @ 0: uvm_test_top.compA [compA] Packet sent to compB
pkt: (packet@395) { addr: 'h85  data: 'hf1  }
UVM_INFO componentB.svh(16) @ 0: uvm_test_top.compB [compB] Packet received from compA
pkt: (packet@395) { addr: 'h85  data: 'hf1  }
UVM_INFO componentA.svh(22) @ 0: uvm_test_top.compA [compA] Packet sent to compB
pkt: (packet@400) { addr: 'h72  data: 'hf4  }
UVM_INFO componentB.svh(16) @ 0: uvm_test_top.compB [compB] Packet received from compA
pkt: (packet@400) { addr: 'h72  data: 'hf4  }
```

Put Port Blocking Behavior

```
// Implementation of the 'put()' method in this case simply prints it.
  virtual task put (Packet pkt);

  // Lets assume the receiver takes some time to process the packet after
  // which this task will return. The "put" method in the sender should be
  // stalled there until this task returns.
  `uvm_info("COMPB", $sformatf("Processing packet"), UVM_LOW)
  #20;
  `uvm_info("COMPB", $sformatf("Processing packet finished ..."), UVM_LOW)

  `uvm_info ("COMPB", "Packet received from CompA", UVM_LOW)
  pkt.print(uvm_default_line_printer);
endtask
```

- A blocking put port declared of type uvm_blocking_put_port which would stall the sender from resuming until the put() task returns .
- This blocking behavior can be observed if there is a delay inside the implementation of the put method within componentB.
- The time consumed by componentB inside put task that blocks the sender from sending the next packet.
- At 20ns, componentB finishes the task and componentA proceeds to send the next packet.

Put Port Blocking Behavior

```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(579) @ 0: reporter [UVMTOP] UVM testbench top
ology:
----------------------------------------------------
Name              Type                       Size  Value
----------------------------------------------------
uvm_test_top     my_test                     -     @1837
  compA           componentA                 -     @1906
    m_put_port   uvm_blocking_put_port      -     @1972
  compB           componentB                 -     @1937
    m_put_imp    uvm_blocking_put_imp       -     @2011
----------------------------------------------------

UVM_INFO testbench.sv(63) @ 0: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2049) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(97) @ 0: uvm_test_top.compB [COMPB] Processing packet
UVM_INFO testbench.sv(99) @ 20: uvm_test_top.compB [COMPB] Processing packet finished ...
UVM_INFO testbench.sv(102) @ 20: uvm_test_top.compB [COMPB] Packet received from CompA
pkt: (Packet@2049) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(63) @ 20: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2059) { addr: 'hc1  data: 'hb9  }
UVM_INFO testbench.sv(97) @ 20: uvm_test_top.compB [COMPB] Processing packet
UVM_INFO testbench.sv(99) @ 40: uvm_test_top.compB [COMPB] Processing packet finished ...
UVM_INFO testbench.sv(102) @ 40: uvm_test_top.compB [COMPB] Packet received from CompA
pkt: (Packet@2059) { addr: 'hc1  data: 'hb9  }
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 40: reporter [TEST_DONE] 'run' p
hase is ready to proceed to the 'extract' phase
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 40: reporter [UVM/REPORT/SERV
ER]
--- UVM Report Summary ---
```

UVM TLM Nonblocking Put Port

```
// Create a class data object that can be sent from one
// component to another
class Packet extends uvm_object;
  rand bit[7:0] addr;
  rand bit[7:0] data;

  `uvm_object_utils_begin(Packet)
    `uvm_field_int(addr, UVM_ALL_ON)
    `uvm_field_int(data, UVM_ALL_ON)
  `uvm_object_utils_end

  function new(string name = "Packet");
    super.new(name);
  endfunction
endclass
```

1. Create sender class with port type uvm_nonblocking_put_port

```
class componentA extends uvm_component;
  `uvm_component_utils (componentA)

 // Create a nonblocking TLM put port which can send an object
 // of type 'Packet'
 uvm_nonblocking_put_port #(Packet) m_put_port;
 int m_num_tx;

  function new (string name = "componentA", uvm_component parent= null);
     super.new (name, parent);
  endfunction

  // Remember that TLM put_port is a class object and it will have to be
  // created with new ()
  virtual function void build_phase (uvm_phase phase);
     super.build_phase (phase);
    m_put_port = new ("m_put_port", this);
  endfunction

 // Create a packet, randomize it and send it through the port
 // Note that put() is a method defined by the receiving component
 // Repeat these steps N times to send N packets
  virtual task run_phase (uvm_phase phase);
    phase.raise_objection(this);
    repeat (m_num_tx) begin
       bit success;
        Packet pkt = Packet::type_id::create ("pkt");
        assert(pkt.randomize ());

       // Print the packet to be displayed in log
        `uvm_info ("COMPA", "Packet sent to CompB", UVM_LOW)
        pkt.print (uvm_default_line_printer);

       // Try to put the packet through the put port
       success = m_put_port.try_put(pkt);
       if (success)
          `uvm_info("COMPA", $sformatf("COMPB was ready to accept and transfer is successful"), UVM_MEDI), UVM_MEDIUM)
        else
          `uvm_info("COMPA", $sformatf("COMPB was NOT ready to accept and transfer failed"), UVM_MEDIUM)UVM_MEDIUM)
     end
    phase.drop_objection(this);
  endtask
endclass
```

2. Create receiver class that implements the put method

```
class componentB extends uvm_component;
   `uvm_component_utils (componentB)

   // Mention type of transaction, and type of class that implements the put ()
  uvm_nonblocking_put_imp #(Packet, componentB) m_put_imp;

   function new (string name = "componentB", uvm_component parent = null);
      super.new (name, parent);
   endfunction

   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      m_put_imp = new ("m_put_imp", this);
   endfunction

  // 'try_put' method definition accepts the packet and prints it.
  // Note that it should return 1 if successful so that componentA
  // knows how to handle the transfer return code
  virtual function bit try_put (Packet pkt);
    `uvm_info ("COMPB", "Packet received", UVM_LOW)
      pkt.print(uvm_default_line_printer);
      return 1;
  endfunction

  virtual function bit can_put();
  endfunction

endclass
```

3. Connect port and its implementation at a higher level

```
class my_test extends uvm_test;
  `uvm_component_utils (my_test)

   componentA compA;
   componentB compB;

  function new (string name = "my_test", uvm_component parent = null);
      super.new (name, parent);
   endfunction

     // Create objects of both components, set number of transfers
   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      compA = componentA::type_id::create ("compA", this);
      compB = componentB::type_id::create ("compB", this);
      compA.m_num_tx = 2;
   endfunction

   // Connection between componentA and componentB is done here
   // Note that the "put_port" is connected to its implementation "put_imp"
   virtual function void connect_phase (uvm_phase phase);
     compA.m_put_port.connect (compB.m_put_imp);
   endfunction

  virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);
    uvm_top.print_topology();
  endfunction
endclass
```

3. Connect port and its implementation at a higher level

```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(579) @ 0: reporter [UVMTOP] UVM testbench top
ology:
----------------------------------------------------
Name              Type                          Size  Value
----------------------------------------------------
uvm_test_top     my_test                       -     @1836
  compA           componentA                   -     @1905
    m_put_port   uvm_nonblocking_put_port     -     @1971
  compB           componentB                   -     @1936
    m_put_imp    uvm_nonblocking_put_imp      -     @2010
----------------------------------------------------

UVM_INFO testbench.sv(60) @ 0: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2048) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(96) @ 0: uvm_test_top.compB [COMPB] Packet received
pkt: (Packet@2048) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(66) @ 0: uvm_test_top.compA [COMPA] COMPB was ready to accept and transfer is
successful
UVM_INFO testbench.sv(60) @ 0: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2091) { addr: 'hc1  data: 'hb9  }
UVM_INFO testbench.sv(96) @ 0: uvm_test_top.compB [COMPB] Packet received
pkt: (Packet@2091) { addr: 'hc1  data: 'hb9  }
UVM_INFO testbench.sv(66) @ 0: uvm_test_top.compA [COMPA] COMPB was ready to accept and transfer is
successful
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 0: reporter [TEST_DONE] 'run' ph
ase is ready to proceed to the 'extract' phase
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 0: reporter [UVM/REPORT/SERVE
R]
--- UVM Report Summary ---
```

UVM TLM can_put example

```
class componentA extends uvm_component;
   `uvm_component_utils (componentA)

   // Rest of the code remains same

   virtual task run_phase (uvm_phase phase);
     phase.raise_objection(this);
     repeat (m_num_tx) begin
        bit success;
         Packet pkt = Packet::type_id::create ("pkt");
         assert(pkt.randomize ());

        // Print the packet to be displayed in log
         `uvm_info ("COMPA", "Packet sent to CompB", UVM_LOW)
         pkt.print (uvm_default_line_printer);

        // Another way to do the same is to loop until can_put returns a 1. In this case, its is
        // not even attempted to send a transaction using put, until the sender knows for sure
        // that the receiver is ready to accept it
        `uvm_info("COMPA", $sformatf("Waiting for receiver to be ready ..."), UVM_MEDIUM)
        do begin
            success = m_put_port.can_put();
        end while (!success);
        `uvm_info("COMPA", $sformatf("Receiver is now ready to accept transfers"), UVM_MEDIUM)
        m_put_port.try_put(pkt);
      end
     phase.drop_objection(this);
   endtask
endclass
```

```
class componentB extends uvm_component;
   `uvm_component_utils (componentB)

  // Rest of the code remains same

  virtual function bit try_put (Packet pkt);
    `uvm_info ("COMPB", "Packet received", UVM_LOW)
    pkt.print(uvm_default_line_printer);
    return 1;
  endfunction

  // Return a random value to model readiness
  virtual function bit can_put();
    return $urandom_range(0,1);
  endfunction
endclass
```

```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(579) @ 0: reporter [UVMTOP] UVM testbench top
ology:
----------------------------------------------------
Name              Type                          Size  Value
----------------------------------------------------
uvm_test_top     my_test                       -     @1837
  compA           componentA                   -     @1906
    m_put_port   uvm_nonblocking_put_port     -     @1972
  compB           componentB                   -     @1937
    m_put_imp    uvm_nonblocking_put_imp      -     @2011
----------------------------------------------------

UVM_INFO testbench.sv(60) @ 0: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2049) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(81) @ 0: uvm_test_top.compA [COMPA] Waiting for receiver to be ready ...
UVM_INFO testbench.sv(85) @ 0: uvm_test_top.compA [COMPA] Receiver is now ready to accept transfers
UVM_INFO testbench.sv(125) @ 0: uvm_test_top.compB [COMPB] Packet received
pkt: (Packet@2049) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(60) @ 0: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2059) { addr: 'h24  data: 'hba  }
UVM_INFO testbench.sv(81) @ 0: uvm_test_top.compA [COMPA] Waiting for receiver to be ready ...
UVM_INFO testbench.sv(85) @ 0: uvm_test_top.compA [COMPA] Receiver is now ready to accept transfers
UVM_INFO testbench.sv(125) @ 0: uvm_test_top.compB [COMPB] Packet received
pkt: (Packet@2059) { addr: 'h24  data: 'hba  }
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 0: reporter [TEST_DONE] 'run' ph
ase is ready to proceed to the 'extract' phase
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 0: reporter [UVM/REPORT/SERVE
R]
--- UVM Report Summary ---
```

UVM TLM Blocking Get Port


```
// Create a class data object that can be sent from one
// component to another
class Packet extends uvm_object;
  rand bit[7:0] addr;
  rand bit[7:0] data;

  `uvm_object_utils_begin(Packet)
    `uvm_field_int(addr, UVM_ALL_ON)
    `uvm_field_int(data, UVM_ALL_ON)
  `uvm_object_utils_end

  function new(string name = "Packet");
    super.new(name);
  endfunction
endclass
```

1. Create a sender class with a port of type uvm_blocking_get_imp

```
class componentA extends uvm_component;
   `uvm_component_utils (componentA)

   // Create an export to send data to componentB
   uvm_blocking_get_imp #(Packet, componentA) m_get_imp;
   Packet  pkt;

  function new (string name, uvm_component parent);
      super.new (name, parent);
   endfunction

   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      // Remember that m_get_imp is a class object and it will have to be
      // created with new ()
      m_get_imp = new ("m_get_imp", this);
   endfunction

   // This task will output a new packet
   virtual task get (output Packet pkt);
      // Create a new packet
      pkt = new();
      assert (pkt.randomize());
      `uvm_info ("COMPA", "ComponentB has requested for a packet", UVM_LOW)
      pkt.print (uvm_default_line_printer);
   endtask
endclass
```

2. Create receiver class that waits on the get method

```
class componentB extends uvm_component;
   `uvm_component_utils (componentB)

   // Create a get_port to request for data from componentA
   uvm_blocking_get_port #(Packet) m_get_port;
   int m_num_tx = 2;

  function new (string name, uvm_component parent);
      super.new (name, parent);
   endfunction

   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      m_get_port = new ("m_get_port", this);
   endfunction

   virtual task run_phase (uvm_phase phase);
      Packet pkt;
     phase.raise_objection(this);
     repeat (m_num_tx) begin
         m_get_port.get (pkt);
         `uvm_info ("COMPB", "ComponentA just gave me the packet", UVM_LOW)
        pkt.print (uvm_default_line_printer);
      end
     phase.drop_objection(this);
   endtask
endclass
```

3. Connect port and its implementation at a higher level

```
class my_test extends uvm_test;
  `uvm_component_utils (my_test)

   componentA compA;
   componentB compB;

  function new (string name = "my_test", uvm_component parent = null);
      super.new (name, parent);
   endfunction

   // Create objects of both components, set number of transfers
   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      compA = componentA::type_id::create ("compA", this);
      compB = componentB::type_id::create ("compB", this);
   endfunction

   // Connection between componentA and componentB is done here
   virtual function void connect_phase (uvm_phase phase);
     compB.m_get_port.connect (compA.m_get_imp);
   endfunction

   virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);
    uvm_top.print_topology();
  endfunction
endclass
```

3. Connect port and its implementation at a higher level

```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(579) @ 0: reporter [UVMTOP] UVM testbench top
ology:
----------------------------------------------------
Name              Type                       Size  Value
----------------------------------------------------
uvm_test_top     my_test                     -     @1836
  compA           componentA                 -     @1905
    m_get_imp    uvm_blocking_get_imp       -     @1971
  compB           componentB                 -     @1936
    put_export   uvm_blocking_get_port      -     @2010
----------------------------------------------------

UVM_INFO testbench.sv(86) @ 0: uvm_test_top.compA [COMPA] ComponentB has requested for a packet, giv
e the following packet to componentB
Packet: (Packet@1877) { addr: 'hdd  data: 'hce  }
UVM_INFO testbench.sv(54) @ 0: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
Packet: (Packet@1877) { addr: 'hdd  data: 'hce  }
UVM_INFO testbench.sv(86) @ 0: uvm_test_top.compA [COMPA] ComponentB has requested for a packet, giv
e the following packet to componentB
Packet: (Packet@2058) { addr: 'h4b  data: 'hf5  }
UVM_INFO testbench.sv(54) @ 0: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
Packet: (Packet@2058) { addr: 'h4b  data: 'hf5  }
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 0: reporter [UVM/REPORT/SERVE
R]
--- UVM Report Summary ---
```

Get Port Blocking Behavior

```
virtual task get (output Packet pkt);
   // Create a new packet
   pkt = new();
   assert (pkt.randomize());

   // Lets assume componentA takes some time to prepare packet
  `uvm_info("COMPA", "Preparing packet ...", UVM_LOW)
  #20;
  `uvm_info("COMPA", "Preparing packet over ...", UVM_LOW)
   pkt.print (uvm_default_line_printer);
endtask
```

Get Port Blocking Behavior

```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(579) @ 0: reporter [UVMTOP] UVM testbench top
ology:
----------------------------------------------------
Name              Type                       Size  Value
----------------------------------------------------
uvm_test_top     my_test                    -     @1837
  compA           componentA                -     @1906
    m_get_imp    uvm_blocking_get_imp       -     @1972
  compB           componentB                -     @1937
    m_get_port   uvm_blocking_get_port      -     @2011
----------------------------------------------------

UVM_INFO testbench.sv(89) @ 0: uvm_test_top.compA [COMPA] Preparing packet ...
UVM_INFO testbench.sv(91) @ 20: uvm_test_top.compA [COMPA] Preparing packet over ...
Packet: (Packet@1904) { addr: 'he8  data: 'hc5  }
UVM_INFO testbench.sv(55) @ 20: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
Packet: (Packet@1904) { addr: 'he8  data: 'hc5  }
UVM_INFO testbench.sv(89) @ 20: uvm_test_top.compA [COMPA] Preparing packet ...
UVM_INFO testbench.sv(91) @ 40: uvm_test_top.compA [COMPA] Preparing packet over ...
Packet: (Packet@2071) { addr: 'hd6  data: 'hd  }
UVM_INFO testbench.sv(55) @ 40: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
Packet: (Packet@2071) { addr: 'hd6  data: 'hd  }
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 40: reporter [TEST_DONE] 'run' p
hase is ready to proceed to the 'extract' phase
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 40: reporter [UVM/REPORT/SERV
ER]
--- UVM Report Summary ---
```

TLM Non-blocking Get Port

```
// Create a class data object that can be sent from one
// component to another
class Packet extends uvm_object;
  rand bit[7:0] addr;
  rand bit[7:0] data;

  `uvm_object_utils_begin(Packet)
    `uvm_field_int(addr, UVM_ALL_ON)
    `uvm_field_int(data, UVM_ALL_ON)
  `uvm_object_utils_end

  function new(string name = "Packet");
    super.new(name);
  endfunction
endclass
```

1. Create receiver class with a port type uvm_nonblocking_get_port

```
class componentB extends uvm_component;
  `uvm_component_utils (componentB)

  // Create a get_port to request for data from componentA
  uvm_nonblocking_get_port #(Packet) m_get_port;
  int m_num_tx = 2;

 function new (string name, uvm_component parent);
     super.new (name, parent);
  endfunction

  virtual function void build_phase (uvm_phase phase);
     super.build_phase (phase);
    m_get_port = new ("m_get_port", this);
  endfunction

  virtual task run_phase (uvm_phase phase);
     Packet pkt;
    phase.raise_objection(this);

    // Try to get a transaction which does not consume simulation time
    // as try_get() is a function
    repeat (m_num_tx) begin
      if (m_get_port.try_get(pkt))
       `uvm_info ("COMPB", "ComponentA just gave me the packet", UVM_LOW)
      else
        `uvm_info ("COMPB", "ComponentA did not give packet", UVM_LOW)
       pkt.print (uvm_default_line_printer);
     end
    phase.drop_objection(this);
  endtask
endclass
```

2. Create sender class that implements the get method

```
class componentA extends uvm_component;
   `uvm_component_utils (componentA)

   uvm_nonblocking_get_imp #(Packet, componentA) m_get_imp;

  function new (string name, uvm_component parent);
      super.new (name, parent);
   endfunction

   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      m_get_imp = new ("m_get_imp", this);
   endfunction

   virtual function bit try_get (output Packet pkt);
      pkt = new();
      assert (pkt.randomize());
      `uvm_info ("COMPA", "ComponentB has requested for a packet", UVM_LOW)
      pkt.print (uvm_default_line_printer);
      return 1;
   endfunction

     virtual function bit can_get();
     endfunction
endclass
```

3. Connect port and its implementation at a higher level

```
class my_test extends uvm_test;
  `uvm_component_utils (my_test)

   componentA compA;
   componentB compB;

  function new (string name = "my_test", uvm_component parent = null);
      super.new (name, parent);
   endfunction

   // Create objects of both components, set number of transfers
   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      compA = componentA::type_id::create ("compA", this);
      compB = componentB::type_id::create ("compB", this);
   endfunction

   // Connection between componentA and componentB is done here
   virtual function void connect_phase (uvm_phase phase);
     compB.m_get_port.connect (compA.m_get_imp);
   endfunction

   virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);
    uvm_top.print_topology();
  endfunction
endclass
```

3. Connect port and its implementation at a higher level

```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(579) @ 0: reporter [UVMTOP] UVM testbench top
ology:
----------------------------------------------------
Name              Type                          Size  Value
----------------------------------------------------
uvm_test_top     my_test                        -     @1836
  compA           componentA                    -     @1905
    m_get_imp    uvm_nonblocking_get_imp        -     @1971
  compB           componentB                    -     @1936
    m_get_port   uvm_nonblocking_get_port       -     @2010
----------------------------------------------------

UVM_INFO testbench.sv(97) @ 0: uvm_test_top.compA [COMPA] ComponentB has requested for a packet
Packet: (Packet@1903) { addr: 'he8  data: 'hc5  }
UVM_INFO testbench.sv(67) @ 0: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
Packet: (Packet@1903) { addr: 'he8  data: 'hc5  }
UVM_INFO testbench.sv(97) @ 0: uvm_test_top.compA [COMPA] ComponentB has requested for a packet
Packet: (Packet@2058) { addr: 'hd6  data: 'hd  }
UVM_INFO testbench.sv(67) @ 0: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
Packet: (Packet@2058) { addr: 'hd6  data: 'hd  }
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 0: reporter [TEST_DONE] 'run' ph
ase is ready to proceed to the 'extract' phase
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 0: reporter [UVM/REPORT/SERVE
R]
--- UVM Report Summary ---
```

UVM TLM can_get example

```
class componentB extends uvm_component;
  `uvm_component_utils (componentB)

  // Create a get_port to request for data from componentA
  uvm_nonblocking_get_port #(Packet) m_get_port;
  int m_num_tx = 2;

 function new (string name, uvm_component parent);
     super.new (name, parent);
  endfunction

  virtual function void build_phase (uvm_phase phase);
     super.build_phase (phase);
    m_get_port = new ("m_get_port", this);
  endfunction

  virtual task run_phase (uvm_phase phase);
     Packet pkt;
    phase.raise_objection(this);

    // Try to get a transaction which does not consume simulation time
    // as try_get() is a function
    repeat (m_num_tx) begin
      while (!m_get_port.can_get()) begin
        #10 `uvm_info("COMPB", $sformatf("See if can_get() is ready"), UVM_LOW)
      end

      `uvm_info("COMPB", $sformatf("COMPA ready, get packet now"), UVM_LOW)
      m_get_port.try_get(pkt);
      pkt.print (uvm_default_line_printer);
     end
    phase.drop_objection(this);
  endtask
endclass
```

UVM TLM can_get example

```
class componentA extends uvm_component;
   `uvm_component_utils (componentA)

   uvm_nonblocking_get_imp #(Packet, componentA) m_get_imp;

  // Rest of the code remains same

   virtual function bit try_get (output Packet pkt);
      pkt = new();
      assert (pkt.randomize());
      `uvm_info ("COMPA", "ComponentB has requested for a packet", UVM_LOW)
      pkt.print (uvm_default_line_printer);
      return 1;
   endfunction

     virtual function bit can_get();
       bit ready;
       std::randomize(ready) with { ready dist {0:/70, 1:/30}; };
       return ready;
     endfunction
endclass
```

UVM TLM can_get example

```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_root.svh(579) @ 0: reporter [UVMTOP] UVM testbench top
ology:
----------------------------------------------------
Name              Type                          Size  Value
----------------------------------------------------
uvm_test_top     my_test                       -     @1837
  compA           componentA                   -     @1906
    m_get_imp    uvm_nonblocking_get_imp       -     @1972
  compB           componentB                   -     @1937
    m_get_port   uvm_nonblocking_get_port      -     @2011
----------------------------------------------------

UVM_INFO testbench.sv(60) @ 10: uvm_test_top.compB [COMPB] See if can_get() is ready
UVM_INFO testbench.sv(60) @ 20: uvm_test_top.compB [COMPB] See if can_get() is ready
UVM_INFO testbench.sv(60) @ 30: uvm_test_top.compB [COMPB] See if can_get() is ready
UVM_INFO testbench.sv(62) @ 30: uvm_test_top.compB [COMPB] COMPA ready, get packet now
UVM_INFO testbench.sv(97) @ 30: uvm_test_top.compA [COMPA] ComponentB has requested for a packet
Packet: (Packet@2065) { addr: 'h8c  data: 'h99  }
Packet: (Packet@2065) { addr: 'h8c  data: 'h99  }
UVM_INFO testbench.sv(62) @ 30: uvm_test_top.compB [COMPB] COMPA ready, get packet now
UVM_INFO testbench.sv(97) @ 30: uvm_test_top.compA [COMPA] ComponentB has requested for a packet
Packet: (Packet@2095) { addr: 'h97  data: 'hb8  }
Packet: (Packet@2095) { addr: 'h97  data: 'hb8  }
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 30: reporter [TEST_DONE] 'run' p
hase is ready to proceed to the 'extract' phase
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 30: reporter [UVM/REPORT/SERV
ER]
--- UVM Report Summary ---
```

TLM FIFO

- Data rate of sender is much faster than the rate at which the receiver can get packets.

- A FIFO element is required in between to store packets so that it allows both sender and receiver to operate independently

UVM TLM FIFO Example

```
class Packet extends uvm_object;
  rand bit[7:0] addr;
  rand bit[7:0] data;

  `uvm_object_utils_begin(Packet)
    `uvm_field_int(addr, UVM_ALL_ON)
    `uvm_field_int(data, UVM_ALL_ON)
  `uvm_object_utils_end

  function new(string name = "Packet");
    super.new(name);
  endfunction
endclass
```

1. Create sender class with a port of type uvm_blocking_put_port

```
class componentA extends uvm_component;
   `uvm_component_utils (componentA)

  // Create a blocking TLM put port which can send an object
  // of type 'Packet'
  uvm_blocking_put_port #(Packet) m_put_port;
  int m_num_tx = 2;

   function new (string name = "componentA", uvm_component parent= null);
      super.new (name, parent);
   endfunction

   // Remember that TLM put_port is a class object and it will have to be
   // created with new ()
   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
     m_put_port = new ("m_put_port", this);
   endfunction

  // Create a packet, randomize it and send it through the port
  // Note that put() is a method defined by the receiving component
  // Repeat these steps N times to send N packets
   virtual task run_phase (uvm_phase phase);
     phase.raise_objection(this);
     repeat (m_num_tx) begin
         Packet pkt = Packet::type_id::create ("pkt");
         assert(pkt.randomize ());
        #50;
          // Print the packet to be displayed in log
         `uvm_info ("COMPA", "Packet sent to CompB", UVM_LOW)
         pkt.print (uvm_default_line_printer);

         // Call the TLM put() method of put_port class and pass packet as argument
         m_put_port.put (pkt);
      end
      phase.drop_objection(this);
   endtask
endclass
```

2. Create receiver class that receives using the get method

```
class componentB extends uvm_component;
   `uvm_component_utils (componentB)

   // Create a get_port to request for data from componentA
   uvm_blocking_get_port #(Packet) m_get_port;
   int m_num_tx = 2;

  function new (string name, uvm_component parent);
      super.new (name, parent);
   endfunction

   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      m_get_port = new ("m_get_port", this);
   endfunction

   virtual task run_phase (uvm_phase phase);
      Packet pkt;
     phase.raise_objection(this);
     repeat (m_num_tx) begin
       #100;
         m_get_port.get (pkt);
         `uvm_info ("COMPB", "ComponentA just gave me the packet", UVM_LOW)
        pkt.print (uvm_default_line_printer);
      end
     phase.drop_objection(this);
   endtask
endclass
```

3. Connect the two components via a TLM FIFO at a higher level

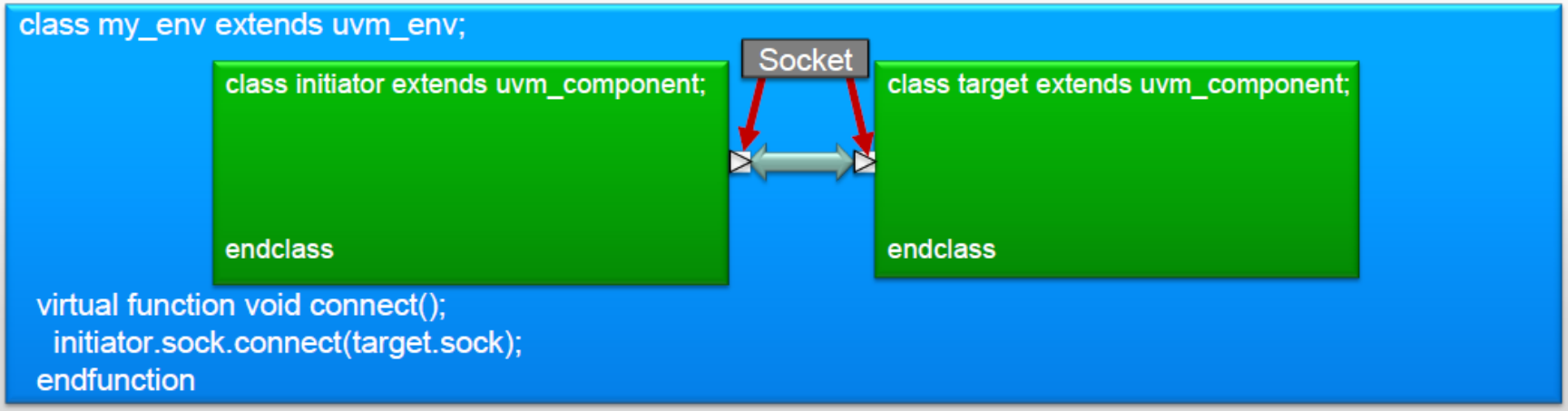
```
class my_test extends uvm_env;
  `uvm_component_utils (my_test)

   componentA compA;
   componentB compB;

   int m_num_tx;

   // Create the UVM TLM Fifo that can accept simple_packet
   uvm_tlm_fifo #(Packet)    m_tlm_fifo;

  function new (string name = "my_test", uvm_component parent = null);
      super.new (name, parent);
   endfunction

   virtual function void build_phase (uvm_phase phase);
      super.build_phase (phase);
      // Create an object of both components
      compA = componentA::type_id::create ("compA", this);
      compB = componentB::type_id::create ("compB", this);
     std::randomize(m_num_tx) with { m_num_tx inside {[4:10]}; };
     compA.m_num_tx = m_num_tx;
     compB.m_num_tx = m_num_tx;

      // Create a FIFO with depth 2
      m_tlm_fifo = new ("uvm_tlm_fifo", this, 2);
   endfunction

   // Connect the ports to the export of FIFO.
   virtual function void connect_phase (uvm_phase phase);
     compA.m_put_port.connect(m_tlm_fifo.put_export);
     compB.m_get_port.connect(m_tlm_fifo.get_export);
   endfunction

   // Display a message when the FIFO is full
   virtual task run_phase (uvm_phase phase);
      forever begin
        #10;
        if (m_tlm_fifo.is_full ())
          `uvm_info ("UVM_TLM_FIFO", "Fifo is now FULL !", UVM_MEDIUM)
      end
   endtask
endclass
```


```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_INFO testbench.sv(49) @ 50: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2234) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(91) @ 100: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
pkt: (Packet@2234) { addr: 'ha1  data: 'h64  }
UVM_INFO testbench.sv(49) @ 100: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2256) { addr: 'hc1  data: 'hb9  }
UVM_INFO testbench.sv(49) @ 150: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@2270) { addr: 'hf0  data: 'hae  }
UVM_INFO testbench.sv(137) @ 150: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 160: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 170: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 180: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 190: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(91) @ 200: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
pkt: (Packet@2256) { addr: 'hc1  data: 'hb9  }
UVM_INFO testbench.sv(49) @ 200: uvm_test_top.compA [COMPA] Packet sent to CompB
pkt: (Packet@1879) { addr: 'h68  data: 'h7d  }
UVM_INFO testbench.sv(137) @ 200: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 210: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 220: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 230: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 240: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 250: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 260: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 270: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 280: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(137) @ 290: uvm_test_top [UVM_TLM_FIFO] Fifo is now FULL !
UVM_INFO testbench.sv(91) @ 300: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
pkt: (Packet@2270) { addr: 'hf0  data: 'hae  }
UVM_INFO testbench.sv(91) @ 400: uvm_test_top.compB [COMPB] ComponentA just gave me the packet
pkt: (Packet@1879) { addr: 'h68  data: 'h7d  }
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 400: reporter [TEST_DONE] 'run'
phase is ready to proceed to the 'extract' phase
UVM_INFO /playground_lib/uvm-1.2/src/base/uvm_report_server.svh(847) @ 400: reporter [UVM/REPORT/SER
VER]
--- UVM Report Summary ---
```

TLM2 in UVM

- **TLM2 focuses on bus-based communication**
  - TLM2 uses *sockets*, which contain both a port and an export
  - Pass-by-reference
  - Supports *generic payload* to model typical bus transactions

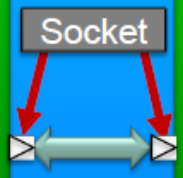
- **Connections are between sockets, just like ports/exports**
  - Initiator socket connects to target socket



TLM2 in UVM – Blocking Transport

- **Completes the entire transaction within a single method call**
  - Uses the b_transport() task

```
class my_env extends uvm_env;

  class initiator extends uvm_component;
    uvm_tlm_b_initiator_socket#(apb_rw) sock;

    virtual task run_phase(uvm_phase phase);
      …
      sock.b_transport(rw, delay)
      …
      endtask
  endclass

  virtual function void connect();
    initiator.sock.connect(target.sock);
  endfunction
```

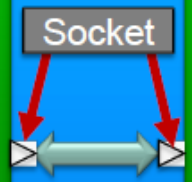



```
  class target extends uvm_component;
    uvm_tlm_b_target_socket #(target, apb_rw) sock;

    task b_transport(apb_rw rw, uvm_tlm_time delay);
      …
    endtask


  endclass
```

TLM2 in UVM – Nonblocking Transport

- **Bidirectional communication**
  - Uses the nb_transport_fw() and nb_transport_bw() functions
  - Initiator implements nb_transport_bw(); Target implements nb_transport_fw()

```
class my_env extends uvm_env;
  class initiator extends uvm_component;
    uvm_tlm_nb_initiator_socket#(initiator, apb_rw
                                 tlm_phase) sock;

    function uvm_tlm_sync_e b_transport_bw(
                  apb_rw rw,  ref tlm_phase ph,
                  uvm_tlm_time delay);
      ...
    endfunction

  endclass
```




```
  class target extends uvm_component;
    uvm_tlm_nb_target_socket #(target, apb_rw,
                               tlm_phase) sock;

    function uvm_tlm_sync_e b_transport_fw(
                  apb_rw rw,  ref tlm_phase ph,
                  uvm_tlm_time delay);
      ...
    endfunction

  endclass

  virtual function void connect();
    initiator.sock.connect(target.sock);
  endfunction
```