

Integrating TLM with RTL!

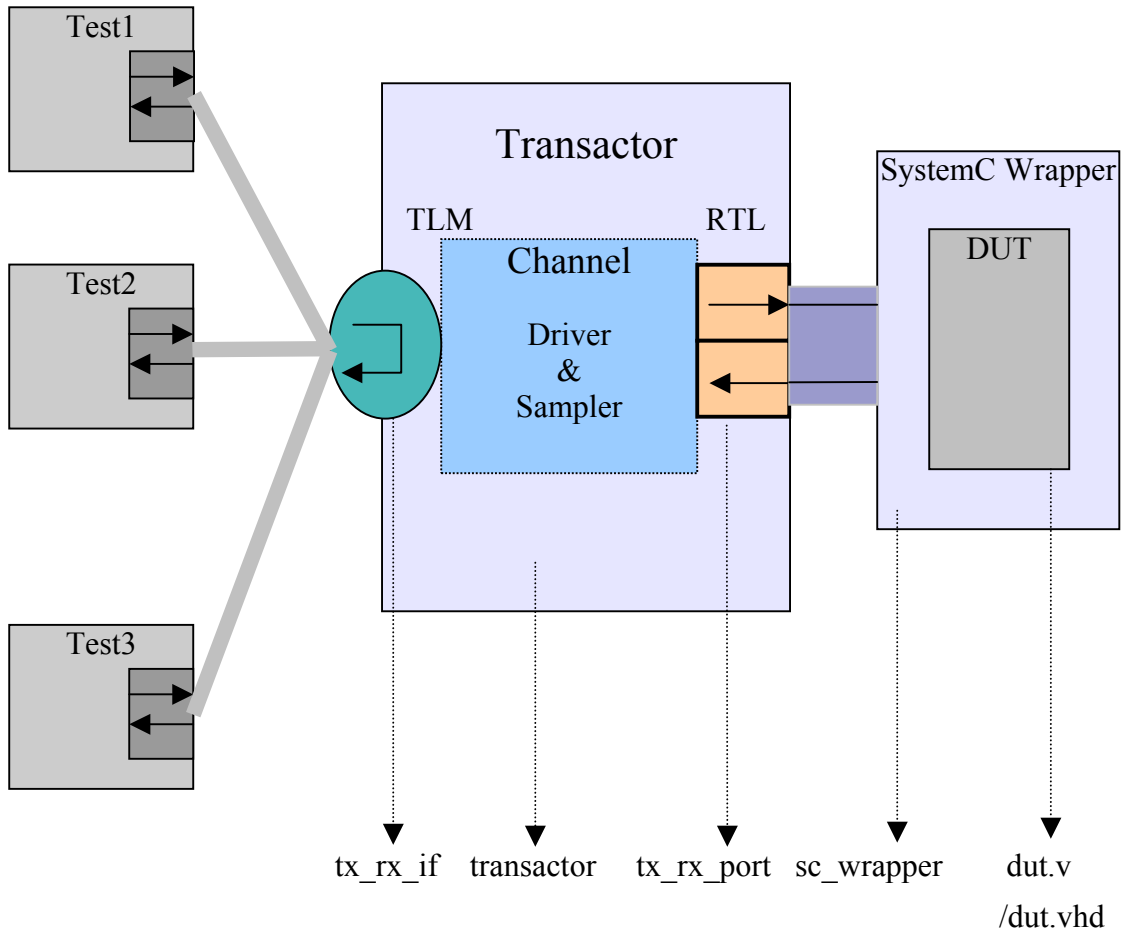
Let's come out from the traditional way of verification and use today's efficient test benches i.e. TLM Test benches offered by SystemC. TLM is Transaction Level Modeling. It is nothing but a set of information that represents some bounded activity within the execution of a design or test benches. It is high level of modeling than RTL.


TLM has its own significant advantages to be chosen for Verification Component Designing. Few of them are mentioned below.

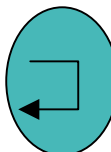
- Object Oriented Architecture leads to reusability so it turns into saving of time in development.
- Easily upgradeable and use for multi purpose
- Provides optimized architectural models for use as software development platforms, months prior to the availability of the RTL implementation.
- Enables the very fast design space exploration and performance evaluation that are necessary to develop and verify a SoC architecture optimized for its intended application.
- Easy configurability
- Verifies the architectural design at upwards of 1,000 times faster than at RTL level, and provides a functional test bench to verify the RTL implementation.
- Enables HW/SW co-verification upwards of 1,000 times faster than at C/RTL, enabling the processing of much larger volumes of software code.
- With the use of SystemC Verification Library (SCV), Randomization, Transaction Recording and Monitoring can also be added to get the effective results
- Speed the evaluation, integration and re-use of intellectual property (IP).

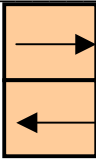
But the problem is how to interconnect this TLM based test bench with a DUT that implements RTL interface. One of the best solutions is to use Transactor. Transactor is nothing but an adaptor between a TLM based model and a module at a different level (e.g. RTL).

Let's see how TLM Test bench looks like with RTL DUT and a Transactor.



 It is an "sc_port" defined in Test case class.

 It is a "tx_rx_if" interface class that inherits sc_interface class and defines all virtual methods.



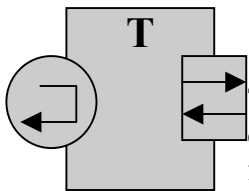
It is “tx_rx_port” class that inherits sc_module and contains all RTL ports.



It represents port binding of “tx_rx_port” and “sc_wrapper” classes.

sc_wrapper class is nothing but a wrapper around a DUT that binds a Transactor and a DUT together. Generally sc_wrapper is the property of a simulator. So there is no standard way to design this wrapper. It may differ for different simulators. e.g. NC-SystemC has its own way of designing it and Modelsim has its own.

Following symbol can represent transactor.



Transactor implements all the virtual methods declared in tx_rx_if class. So it works as channel between TLM and RTL interfaces. Its function is to sample TLM signal and drive RTL signal with it and vice-versa.

Let's go into the code level details of each one. For that here example is taken for transmitting and receiving 8-bit data between Test case class (e.g. Test1) and a DUT (dut.v/dut.vhd).

```
/* DUT that has two 8bit tx and rx data ports
```

```
    dut.v
*/
module dut(txdata, rxdata)
input [7:0] rxdata
output[7:0] txdata
    .....
    .....
    .....
endmodule
```

```
/* tx_rx_if is the interface class that implements all the virtual methods used the
transaction
```

```
*/
class tx_rx_if : public sc_interface
{
```

```
public:
virtual sc_uint<8> get_rxddata();
virtual void set_txdata(sc_uint<8> );
};
```

```
/* SystemC module that implements RTL interface at the other side of a transactor that
communicates with a DUT.
```

```
*/
class tx_rx_port : public sc_module
{
public:
sc_out<sc_uint<8> > txdata;
sc_in<sc_uint<8> > rxdata;
};
```

```
/* Transactor class that inherits both an interface class tx_rx_if and a systemc module that
implements RTL interface i.e. tx_rx_port.
```

```
*/
class transactor : public tx_rx_if,
public tx_rx_port
{
public:
virtual sc_uint<8> get_rxddata()
{
return(rxdata.read());
}
virtual void set_txdata(sc_uint<8> data)
{
txdata.write(data);
}
};
```

```
/* Testcase class that defines sc_port for tx_rx_if interface. So user can access those
interface methods using tx_rx_scport.
```

```
*/
class Test1 : public sc_module
{
public:
sc_port<tx_rx_if> tx_rx_scport;
void do_transaction();
SC_CTOR(Test1)
{
SC_THREAD(do_transaction);
//sensitivity list
}
}
```

```
void do_transaction()  
{  
    sc_uint<8> test_txdata = 55;  
    sc_uint<8> test_rxdata;  
  
    tx_rx_scport->set_txdata(test_txdata);  
    test_rxdata = tx_rx_scport->get_rxdata();  
}  
};
```

/* sc_wrapper class has an instance of a DUT. There may be many other methods to do it but here we have assume that SystemC is the top level module that instantiates dut inside it. And it is simulator specific so here it not metioned in detail.

```
*/  
class sc_wrapper : public sc_module,  
                  public tx_rx_port  
{  
    protected:  
    dut dut_instance;  
    //simulator specific logic for binding ports of dut and  
    //tx_rx_port systemc module  
};
```

Few simulator does not support `sc_main()`. They may have their own defined macro that replaces `sc_main()`. (e.g. In NC-SystemC does not support `sc_main` and in that we have to pass the name of the wrapper class into its macro defined as `NC_MODULE_EXPORT(...)`) So here we have not mentioned anything about `sc_main()`. But in the top-level systemc module we have to bind transactor class in which interface methods are defined with the `tx_rx_scport` defined in Test1 class.

```
Test1->tx_rx_scport = transactor;
```

At the end you might have come to know that TLM is the best example for “Dynamic Binding”.