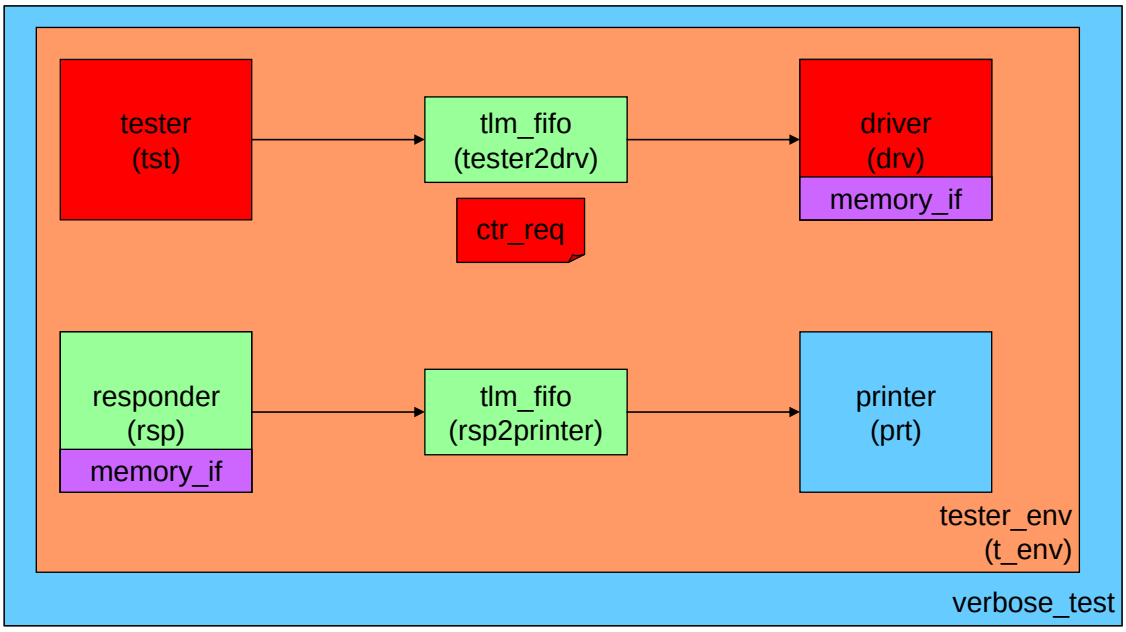
Lab 8: Creating a Transaction Level Test Bench

In this lab, you will create a transaction level test bench to test the counter that we've used for all the labs since the first one. The TLM test bench in this lab looks just like the one in the lecture, and you can use the examples in the lectures as the basis for building this test bench. The test bench looks like this:



- Red Blocks – You will need to implement the red blocks in this test bench to successfully run the simulation in the home work.
- Blue Blocks – These blocks are optional. They allow you to create a quiet_test version of the test that does not write to the screen.

The source files are in the `src` directory. The instructions are stored in the source files as comments, and are repeated here.

Basic Lab

When you implement the steps in this lab you will be able to run the counter's test bench.

`ctr_req.svh`

The `ctr_req` is a transaction that goes from the tester to the driver. It tells the `driver` what kinds of instructions to drive into the interface. The `ctr_req` transaction extends the `ctr_output` transaction.

The `ctr_output` transaction contains an eight-bit data member that holds the counter output and potentially the input. We will do the following in the file:

1. Register `ctr_req` with the factory using the `` `ovm_object_utils() `` macro.

2. Declare a randomizable variable of type `ctr_op`. This is an enumerated type that can take on the values "`load`", "`inc`" and "`nop`".
3. Create a `covert2string()` function that returns a string with the values in the transaction.
4. Create a `do_copy()` function that uses `super.do_copy()` to implement a deep copy.
5. Implement the `load_data` method to load data into the object. You can use `super.load_data()` to load the data member if you wish.

Once you've implemented this transaction, you'll be able to use it in the tester and the driver.

`tester.svh`

The tester creates the new transactions and puts them into the test bench. Do the following to create this object:

1. Use the `` `ovm_component_util() `` macro to register the tester with the factory.
2. Declare an `ovm_put_port` called `req_p` that accepts `ctr_req` objects.
3. `build()` : Create a new `req_p` port in the build method.
4. `run()` : Get the `nloops` integer from the integer configuration memory using `get_config_int()`.
5. `run()` : Create a loop that creates `nloops` transactions and puts them into the test bench using the `put_txn` method.
6. `put_txn()` : Create a task that accepts a `ctr_req` object as an argument. It then clones the object and puts it into the `req_p` port.

`driver.svh`

The driver takes transactions from the `req_p` port and uses the data to drive signals in the interface. The driver extends the `counter_agent` class and that class provides the `build()` method that gets the interface from the test bench.

1. Declare an `ovm_get_port` called `req_p` that works with `ctr_req` objects.
2. `build()` : call `super.build()` to get an interface called `i`.
3. `build()` : Create a new `req_p` object.
4. `run()` : Create code that tries to get a new transaction from `req_p`. If the `try_get()` succeeds the code will look at the operation in the transaction and drive the signals appropriately. The instructions for driving the signals are in the lab source code.

Running the test

You run the test with the

`%vsim –c –do “run.do”`

command:

```
# ----------------------------------------------------------------
# OVM-2.0.1
# (C) 2007-2009 Mentor Graphics Corporation
# (C) 2007-2008 Cadence Design Systems, Inc.
# ----------------------------------------------------------------
# OVM_INFO @ 0: reporter [RNTST] Running test counter_test...
# OVM_INFO @ 0: ovm_test_top.env.tst [run] data: 04 op: load
# OVM_INFO @ 20: ovm_test_top.env.tst [run] data: bf op: load
# OVM_INFO @ 31: ovm_test_top.env.p [run] data: 04
# OVM_INFO @ 40: ovm_test_top.env.tst [run] data: a6 op: nop
# OVM_INFO @ 51: ovm_test_top.env.p [run] data: bf
# OVM_INFO @ 60: ovm_test_top.env.tst [run] data: 89 op: load
# OVM_INFO @ 80: ovm_test_top.env.tst [run] data: e1 op: nop
# OVM_INFO @ 91: ovm_test_top.env.p [run] data: 89
# OVM_INFO @ 100: ovm_test_top.env.tst [run] data: 28 op: inc
# OVM_INFO @ 120: ovm_test_top.env.tst [run] data: a1 op: load
# OVM_INFO @ 131: ovm_test_top.env.p [run] data: 8a
# OVM_INFO @ 140: ovm_test_top.env.tst [run] data: b0 op: load
# OVM_INFO @ 151: ovm_test_top.env.p [run] data: a1
# OVM_INFO @ 160: ovm_test_top.env.tst [run] data: 96 op: inc
# OVM_INFO @ 171: ovm_test_top.env.p [run] data: b0
# OVM_INFO @ 180: ovm_test_top.env.tst [run] data: 6d op: nop
# OVM_INFO @ 191: ovm_test_top.env.p [run] data: b1
#
```

Extra Credit

For extra credit you can create a quiet version of the test. There are two steps to this process:

1. Create a silent version of the `printer` (called `bit_sink`) by extending that class and removing the call to o`vm_report_info`.
2. Override the `printer` object in the factory with the `bit_sink` object.

The environment will ask for a `printer`, but really receive a `bit_sink`.

`bit_sink.svh`

This file is empty. Using `printer.svh` as a model, extend `printer.svh` so that you don't need to copy the `build()` method. Then use a similar run method to `printer.svh` to get transactions off the `out_p` port and throw them away.

`quiet_test.svh`

This test sets the verbosity level to 100 and then overrides the printer with `bit_sink`.

1. `end_of_elaboration()`: Add a call to `env.set_report_verbosity_level_hier` to change the verbosity level in the environment.
2. Add a factory override that replaces the `printer` object with the `bit_sink` object

Running the Test

To run the quiet test, execute the following commands:

```
% vlib work
% vlog –f compile_sv.f
% vsim –c +OVM_TESTNAME=quiet_test top
```

The result is a much quieter test:

```
# ----------------------------------------------------------------
# OVM-2.0.1
# (C) 2007-2009 Mentor Graphics Corporation
# (C) 2007-2008 Cadence Design Systems, Inc.
# ----------------------------------------------------------------
# OVM_INFO @ 0: reporter [RNTST] Running test quiet_test...
#
# --- OVM Report Summary ---
#
# ** Report counts by severity
# OVM_INFO :    1
# OVM_WARNING :    0
# OVM_ERROR :    0
# OVM_FATAL :    0
# ** Report counts by id
# [RNTST]     1
```