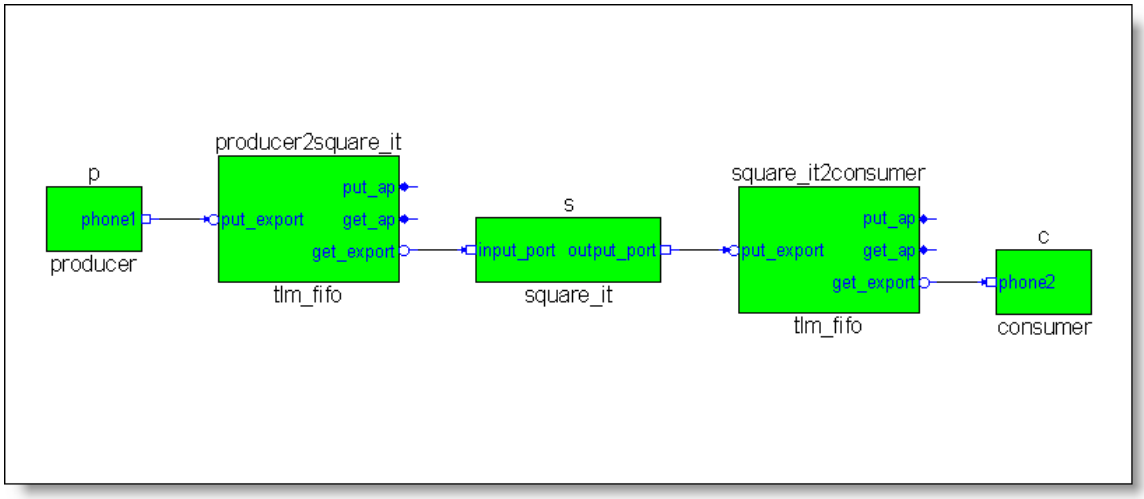
Lab 7: Communicating Between Objects

In this lab you will create an `ovm_agent` that fits into the `producer_consumer` example from the lecture. You will add an object that goes between the `producer` and the `consumer` and squares the number before it's printed. Here's what the completed environment looks like in Certe:



You can see that the `tlm_fifo` named `satellite` has been replaced by two `tlm_fifos`. There is also a new object called `square_it`. This object takes the numbers from the producer, squares them, and passes them on to the consumer where they are printed to the screen. When you run the solution using `run.do`, the output looks like this:

```
# ----------------------------------------------------------------
# OVM-2.0.1
# (C) 2007-2009 Mentor Graphics Corporation
# (C) 2007-2008 Cadence Design Systems, Inc.
# ----------------------------------------------------------------
# OVM_INFO @ 0: reporter [RNTST] Running test producer_consumer_test...
# OVM_INFO @ 0: ovm_test_top.env.p [run] put 1
# OVM_INFO @ 0: ovm_test_top.env.p [run] put 2
# OVM_INFO @ 0: ovm_test_top.env.p [run] put 3
# OVM_INFO @ 0: ovm_test_top.env.c [run] get 1
# OVM_INFO @ 0: ovm_test_top.env.p [run] put 4
# OVM_INFO @ 0: ovm_test_top.env.c [run] get 4
# OVM_INFO @ 0: ovm_test_top.env.p [run] put 5
# OVM_INFO @ 0: ovm_test_top.env.c [run] get 9
# OVM_INFO @ 0: ovm_test_top.env.c [run] get 16
# OVM_INFO @ 0: ovm_test_top.env.c [run] get 25
#
```

Extra Credit: Notice that the producer puts three numbers into its tlm_fifo before any numbers come out of the consumer. This is the expected behavior. Why does this happen?

Step 1: Create the `square_it` object

The file `square_it.svh` defines the `square_it` object. There are some parts missing, so you need to do the following:

- Declare an `ovm_get_port #(int)` so that `square_it` can get numbers from the producer.
- Declare an `ovm_put_port #(int)` so that `square_it` can give numbers to the consumer.
- Modify the `run()` task to get a number from the producer, square it, and give it to the consumer.
    - The square operation in SystemVerilog is `**`.
- Modify the `build()` function to create a new `ovm_get_port` and `ovm_put_port`. Use these ports in the `run()` task.

Step 2: Modify the `producer_consumer_env` object.

The `producer_consumer_env` object needs to create a `square_it` object, and connect the producer, `square_it`, and `consumer` objects. Please do the following:

- Declare a variable to hold the `square_it` object.
- Declare a `tlm_fifo #(int)` that will go from the `square_it` object to the consumer.
- Modify the `build()` function to create a new `square_it` object using the factory.
- Modify the `build()` function to create a new `tlm_fifo` to go from the `square_it` object to the consumer.
- Modify the `connect()` method to connect the `square_it` object to the producer.
- Modify the `connect()` method to connect the `square_it` object to the consumer.

Step 3: Run the Simulation

You can run the simulation with the run.do script:

```
% vsim –c –do run.do
```