VUnit Hands-On

Tutor: Olof Kraigher olof.kraigher@gmail.com

Hand-On

- The goal is to give you a feel for a test-driven workflow using VUnit
- You will be given:
    1. A simple task
    2. A skeleton Test Bench and DUT
    3. A set of incremental steps to perform
- The steps will lead to a working & verified DUT
- I hope everyone will have time to complete the basic steps
- After the basic steps there are a few stretch goals for the faster participants

The Task

- Implement a Tag-Length-Value (**TLV**) stream extractor (`tlv_extractor`)
- A **TLV** stream consists of variable sized packets with single byte and fields followed by a of (0-255) bytes
- The `tlv_extractor` should consume a **TLV**-stream one byte at a time
- The `tlv_extractor` should output a new **TLV**-stream for a specific only and ignore other
- The tlv_extractor has a generic defining the to extract

TLV Extractor

```
    tlv_extractor
       (
  -- The tag to extract
  tag : std_logic_vector(7        0)
);
    (
  clk :   std_logic;

  -- Input stream of bytes
  in_tvalid :   std_logic;
  in_tdata :   std_logic_vector(7        0);

  -- Output stream of extracted bytes
  out_tvalid :    std_logic;
  out_tdata :    std_logic_vector(7        0)
);
      ;
```

Some advice

- Follow the steps, it is the point of the lab
- Do **not** focus on synthesis aspects of the DUT
  - Foremost since this is simple lab with limited time
  - Also it is often a good workflow to first create a complete test bench for a non-synthesizeable DUT and handle synthesis aspects while leaning on a good test bench

Step 1 - Stimuli

- In the main process of the test bench implement a for loop which streams random data into the DUT.
  - The data has to be complete legal **TLV** packets
- Use OSVVM `RandomPkg` to create random data.
- You are done when the test bench is **passing**, the DUT is not touched in this step

```
       osvvm;
    osvvm.RandomPkg.  ;
...
main :
          rnd : RandomPType;

  ...
  rnd.InitSeed(rnd'instance_name);
  byte := rnd.RandSlv(0, 255, 8);
  ...
          ;
```

Step 2 - Data checking

- Create a data checker process that consumes the output stream
  - Check that the DUT extracts **all** data through to the output for now
- Use a `queue_t` to pass data from main process to the checker process
- Use `check_equal` to check the values
- Add wait for empty queue to the main process
- Add `test_runner_watchdog` to fail the test bench with timeout due to no output from the DUT
- You are done when the test bench is **failing**, the DUT is not touched in this step

```
    vunit_lib;
    vunit_lib.data_types_context;

        tb    tb_tlv_extractor
        checker_queue : queue_t := new_queue;
```

Step 3 - Make it pass

- Implement enough code in the DUT to **pass** the test bench from the previous step and **no** more

Step 4 - Refactor

- Create a dedicated stimuli process
- Use another `queue_t` to pass data from the main process to the stimuli process
- Synchronize with the stimuli and checker queues being empty in the main process
- Test bench should still **pass**

Step 5 - Refactor some more

- Refactor data generation code into a procedure in the main process
- Test bench should still **pass**

```
        create_tlv_data(tag : std_logic_vector(7        0);
                        length : natural);
```

Step 6 - Make a named test case

- Add a `should_extract` Boolean argument to the `create_tlv_data` procedure
  - Data is only pushed to the checker queue if `should_extract` is true
- Create a new test case called
- Use `create_tlv_data` to create a few packets **all** containing the
- Test bench should still **pass** since all data is extracted by the DUT

```
run("Test all packets are extracted")
-- Add a few packets, all with the extraction tag
create_tlv_data(tag => extraction_tag, length => 11,
                should_extract => true);
...
   ;
```

Step 7 - Add a failing test

- Create a new test case called
- Use `create_tlv_data` to create a few packets where some do not contain the

```
   run("Test not all packets are extracted")
-- Add a few packets, some with the extraction tag and some without
create_tlv_data(tag =>     extraction_tag, length => 11,
                should_extract => false);
...
   ;
```

- You are done when the test bench is **failing** sincethe DUT will extract everything

Step 8 - Make it pass

- Implement enough code in the DUT to **pass** the test bench from the previous step

Are you done?

- Will it work for zero length values?
- Will it work with `in_tvalid` low during random gaps?
- Will it work for the maximum length value?

Stretch goals

- Add named directed tests for corner cases
  - Zero length
  - Maximum length
  - Sequence of zero length non-extracted and extracted tags
- Add test named which will push a large number of random packets into the DUT
- Add a test named which will have random gaps with in_tvalid low

Did you find any defects with the additional tests?

If you are done

- Optimize for synthesis
- Would you have introduced defects when optimizing for synthesis without having the test bench to lean on?
    - This is a simple DUT, what would happen for a complex DUT?

We are done!