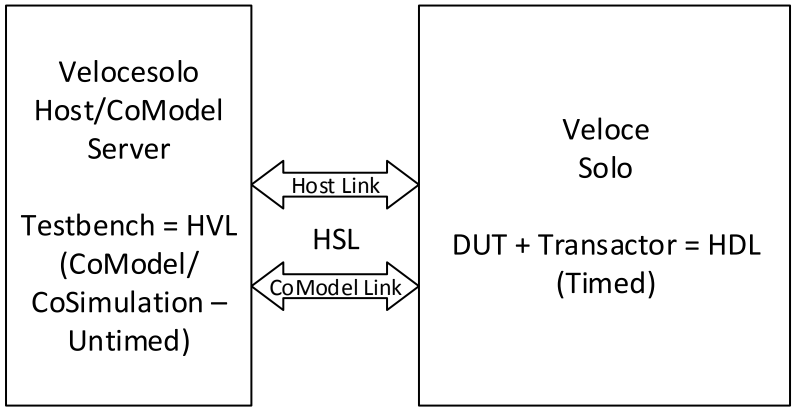
Veloce Emulation TBX Mode Introduction

Updated for VeloceOS3 by- Sameer Ghewari, May 2015

This document portrays project flow for TBX Mode.



Prerequisites –

1. Thoroughly read and complete setup as explained in “Setup and Usage Instructions for Mentor Graphics Veloce Solo at PSU”, April 2015
2. Understand Veloce usage mode explained in “Veloce Solo – Usage Modes”, April 2015
3. Preferred – Read, understand and try out Veloce Standalone mode explained in “Veloce_Standalone_Mode_Flow.pdf”
4. You are connected to Velocesolo with Putty/SSH access

Recommended Reading -

Veloce Languages and Communication Channels User Guide, Software Version 3.0.0, September 2014

Veloce User Guide, Software Version 3.0.0, September 2014

Review : Booth’s Multiplier DUT

- For all examples of TBX mode, a 32 bit Booth’s Multiplier is chosen as the DUT.
- This is implemented as a Finite State Machine.
- It follows Booth’s Algorithm and supports signed as well as unsigned multiplication.
- As this is a 32 bit multiplier, it takes 32 clock cycles to produce a result.

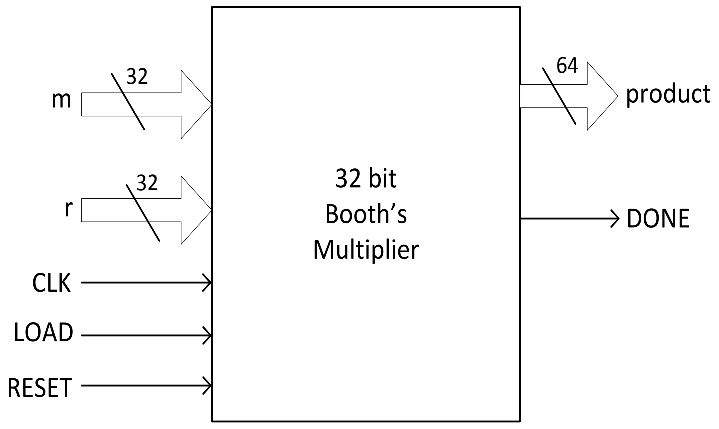


Fig.a Block Diagram of 32 Bit Booth’s Multiplier

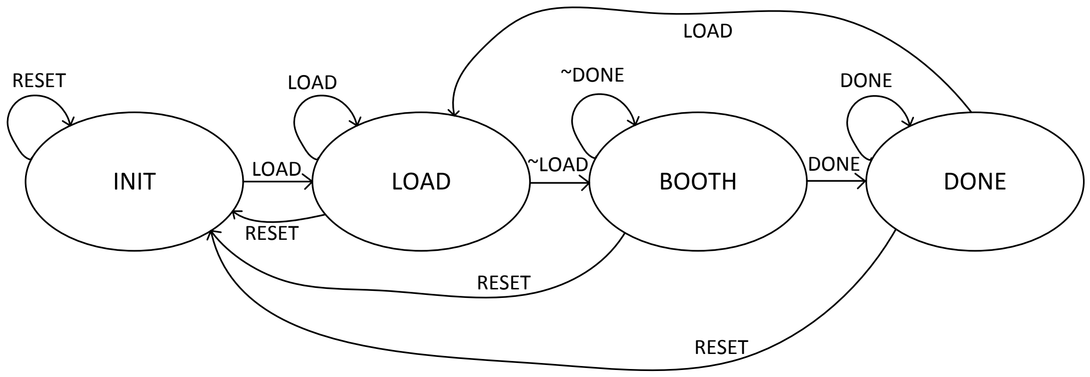
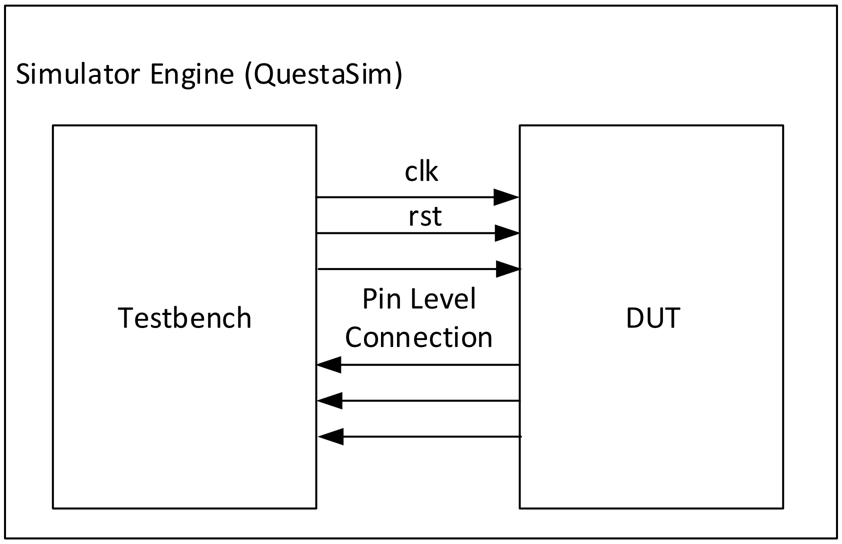


Fig.b State Transition Diagram for the FSM

Review : Traditional Testbench



The above figure shows a typical traditional testbench environment. A simulator engine, such as QuestaSim runs the environment. Testbench has pin level access to the DUT.

Key Features of Traditional Testbench

1. Has pin level access to the DUT
2. Generates clock, reset. Hence has a knowledge of timing.
3. Follows protocol to issue test case and observe result. Hence has knowledge of protocol. (*unless you are using Interfaces and Tasks inside Interface (BFM))

It is important to note above key features, as these no longer stay in testbench for TBX mode Veloce environment. The testbench is essentially partitioned.

Example – $PSU_EXAMPLES/TBX/Examples/Booth_TraditionalTB.tar.gz

Review : Transaction Level Modeling (TLM)

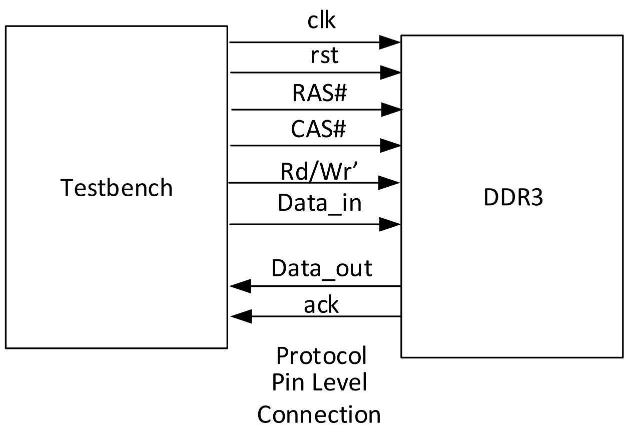


Fig. 1 Traditional Pin Level Connection (Non-TLM)

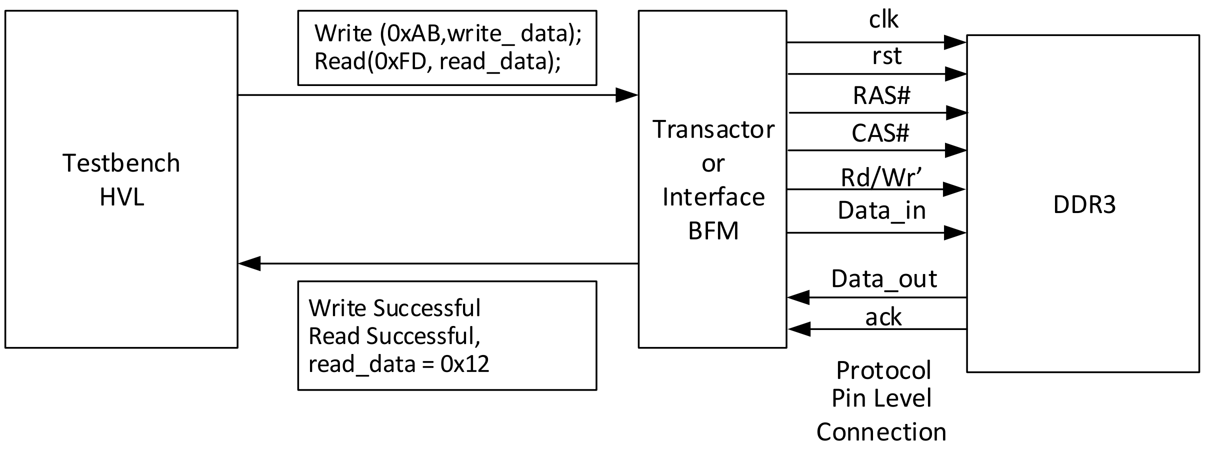


Fig.2 TLM Level Connection

- TLM elevates abstraction level of the communication
- Communicate using tasks / methods / functions / pipes / FIFO rather than at pin level
- Encapsulate data into packets
- The requestor doesn't have to worry about pin level protocol details
- The Transactor converts a transaction into pin-level protocol, and vice-versa
- The emphasis is on the data transfer, not on the protocol
- Convenient, reusable way of connection to high-level entities (HVL) to inject stimulus / observe results.
- The above example shows a DDR3 design. With traditional approach, the testbench drivers protocol, clock and other signals. With TLM, testbench makes calls to methods/tasks. Transactor converts TLM <-> Pin Level Protocol.

Introduction – TestBench Xpress (TBX)

- TBX is methodology to describe CoModel/CoSimulation verification environment for Veloce emulator. Accelerates verification environment.
- Uses TLM communication between HVL and HDL
- Uses Extended RTL (XRTL) to support synthesis of certain non-synthesizable constructs
- Extended RTL (XRTL)
  Allow synthesis of certain non-synthesizable constructs by using pragmas
  e.g. Clock generation

  ```
  //tbx clkgen
  initial
  begin
          clk = 0;
          forever #5 clk = ~clk;
  end
  ```

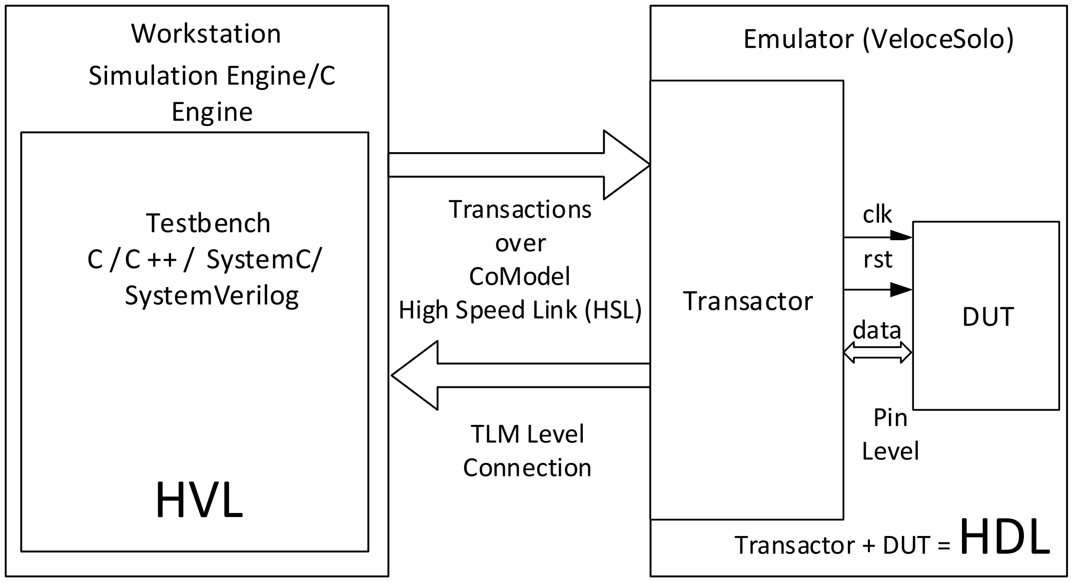
  Initial blocks are usually omitted for synthesis by ASIC synthesis tools. By using //tbx clkgen pragma, the synthesis tool understands that a clock generator is to be generated for this initial block

  e.g. Synthesize Tasks/BFM inside Interface

  ```
  interface my_if (input clk, rst);
  //pragma attribute my_if partition_interface_xif

  //The above pragma specifies that this interface should be made externally accessible (for HVL)

  //port_list
  //modports

  task do_bus_transaction (inputs, outputs);      //pragma tbx xtf
  //The above pragma declares task as externally accessible (for HVL). This is synthesized.
  @(posedge clk);                          //clocked task is required for synthesis
  //Protocol
  endtask
  ```

Guidelines : Develop TBX Test Environment
(or Modify Traditional Testbench)

- Recall the features of traditional testbench. These need to be partitioned for a TBX compatible environment



What is HVL ?

- HVL – HVL is purely software part of test/stimulus generation, checker, scoreboard, etc. HVL cannot have any timing control or clocked logic. HVL is untimed in terms of emulation time, that is executes in zero time.
- Since HVL is purely software code, it can leverage all constructs of SystemVerilog/C. In other words this is not synthesized so feel free to use any construct like queues, associative arrays, randomization, classes.

What is HDL ?

The HDL comprises of two parts

1. The Transactor
    - Transactor has pin level access to the DUT
    - It knows the protocol. Converts a transaction from HVL into pin level protocol, and the other way around
    - Also generates clock, reset signals. This is in essence partitioned part of former traditional testbench.
    - Transactor is synthesized, hence must follow XRTL guidelines (in next section).

2. The DUT
    - The DUT should be purely RTL for being synthesizable. Cannot use non-synthesizable constructs.

It is worth noting that a TBX mode test environment has **two tops** – the HVL and the HDL. HVL is often referred as CoModel. We launch both tops to begin emulation.

It is worth deciding which mode you would like to use in your project well ahead of time. It is often difficult to modify a traditional testbench if the testbench heavily depends on clock/timing. If you develop environment targeting a specific usage mode and method, it is easier to port the same environment for emulation.

Transactor Guidelines

- Preferred way of writing Transactor is FSM (implicit or explicit)
- Must use clock generated by //tbx clkgen
- Cannot have asynchronous inputs in any block

  e.g.

```
//tbx clkgen
initial
begin
clk = 0;
forver #5 clk = ~clk;
end

my_design DUT (.inputA(inputA), .inputB(inputB), .done(done), .output(output) );

//Transactor
always @(done)          // Not allowed, asynchronous

always @(posedge clk) //Allowed
begin
   if(done)
  //Send result to HVL
end
```

- If your design uses interface, and if the access is provided through Interface Tasks (BFM), you don’t have to write a transactor. Instead you can utilize BFM for synthesis.

TBX Mode Environment Guidelines

- The purpose of TBX mode is to accelerate/speedup the environment
- Hardware Emulator does not ensure advertised speedup, unless environment helps emulator by cutting down unnecessary communication between HVL and HDL
- Partitioning testbench essentially moves sequential(software) part on HVL and all timing/sequential/parallelism part on HDL.
- It is essential that your environment has some type of automatic testing mechanism in HVL. Without this, the bugs won't be identified.

There are several ways to achieve TLM between HVL and HDL. The examples include

1. DPI-C (CoModel)
2. SCIMI Pipes
3. Synthesizable BFM
4. Virtual Interface Methodology
5. UVM/OVM

Please continue reading "Veloce_TBX_Mode_Flow.pdf". The document explains project flow for above examples.