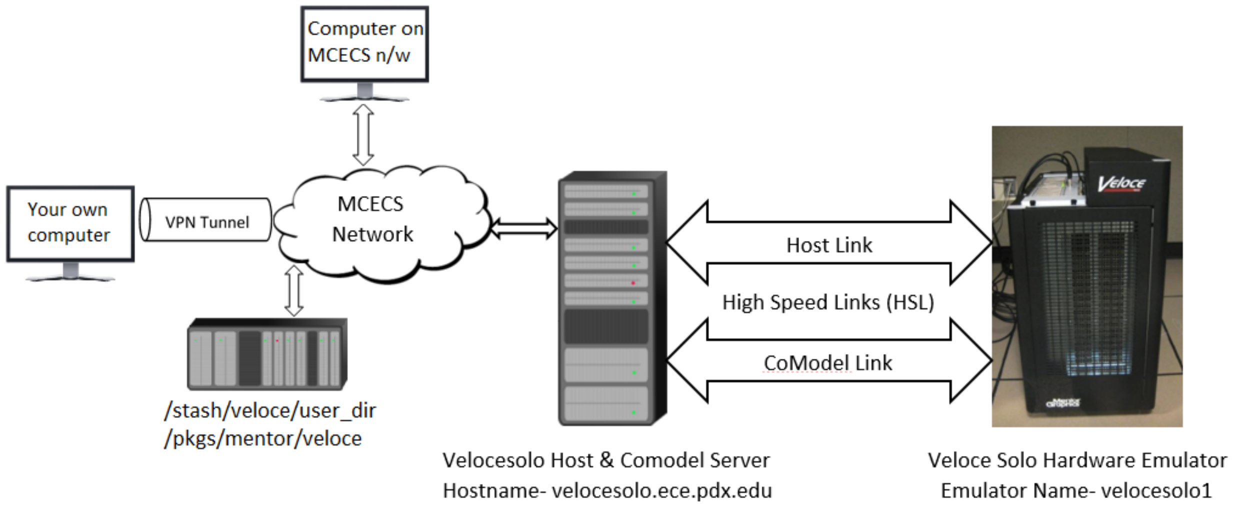
Veloce Solo – Usage Modes
Updated for VeloceOS3
April 2015

Veloce Connection Details

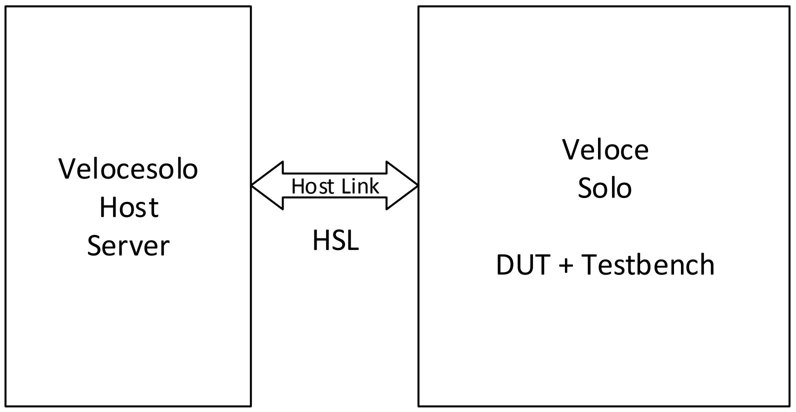


Host Link – This is a High Speed Link (HSL) cable connecting emulator to the host server. This connection is the primary mode of connecting to the emulator. It carries control signals, commands and is also used to download trace/emulation statistics from the emulator.

CoModel Link – If a CoModel / CoSimulation mode is used, your testbench runs on the Comodel Server. The testbench generates transaction that are carried over the CoModel link. The usage modes are explained in detail in the following section –

Veloce Usage Modes

1. Standalone Emulation



In this mode, the DUT and the test bench are in the emulator. Once the DUT + Testbench is configured or downloaded on the Veloce, there are no external interfaces. Hence there is no CoModel or CoSimulation. As a result, this mode uses only the Host Link connection.

This mode can be achieved by –

i. DUT + A Synthesizable Testbench (Example – Standalone_Tutorial_1.tar.gz)
   If your testbench and test generation logic is made synthesizable, it can be compiled and synthesized along with the DUT.
ii. DUT + Test Cases loaded in Memory (Example – Standalone_Tutorial_2.tar.gz)
   In this alternative approach, the test cases are generated using a C code or even Excel and exported to a text file. A memory is defined and loaded with $readmemh. A small piece of logic, more like an FSM reads from the memory and provides test cases to the DUT.
iii. Only DUT (Example – Standalone_Tutorial_3.tar.gz)
   You can also have just the DUT synthesized and then drive its inputs manually. In other words, use Veloce as an ultra big FPGA and drive its inputs from commands.

Once the emulation is over, you can download the waveform trace data and observe the signals.

Advantages –

i. An obvious advantage of this mode is that the maximum speedup can be achieved. As there are no external interfaces, the emulation speedup is not dependent on CoModel/CoSimulation.
ii. This mode can be used to quickly check if your design is synthesizable

Disadvantages –

i. Making synthesizable testbench/testcases is not always possible. Complex test cases are easier to generate through CoModel/CoSimulation.
ii. There is no interactive control or visibility once emulation starts, unlike CoModel/CoSimulation modes.

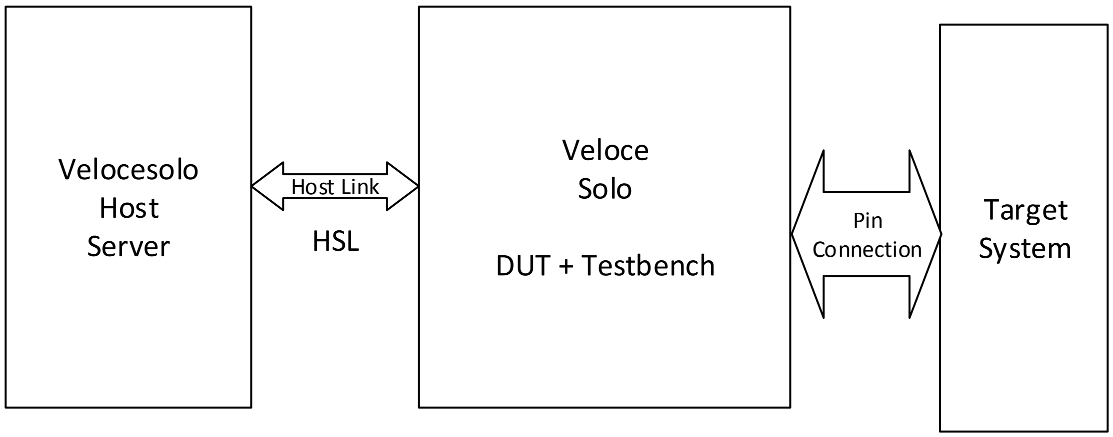
Usage –

This mode is usually isn’t used a lot by itself. It can be used along with CoModel/CoSimulation. However some applications like Graphics Chip Verification require huge test vectors be driven in the DUT. In such applications, DUT + Testcases in Memory mode is beneficial.

More Reading –

Please read “Standalone_Mode_Project_Flow.pdf” document and try out examples.

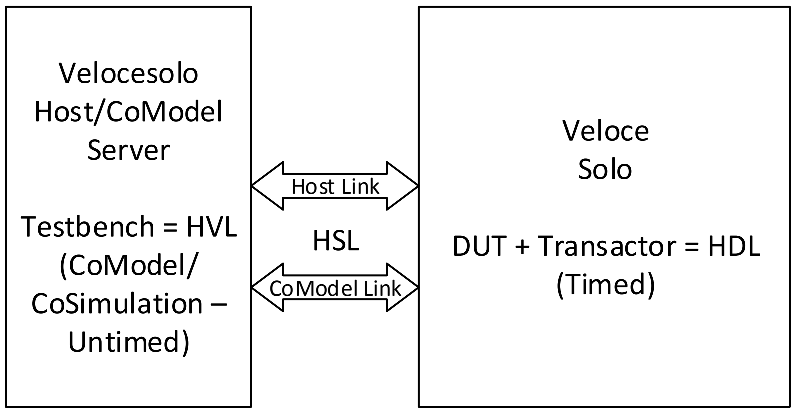
2. Classical In-Circuit Emulation (ICE)



This mode appears very similar to the Standalone Emulation mode, except the emulator connects to a Target System at pin level interface. The emulator has standard IO connectors for pin level connection. The target system is a real world hardware component.
An example application of this mode is – Your DUT is a PCI Bus Arbiter/Controller and Target System is an array of PCI cards.
We don't utilize this mode at PSU.

3. CoSimulation and CoModeling

This is the most interesting usage mode and is widely used. The idea behind this mode is to make best of both worlds – speedup from Veloce Emulator and testbench features from SystemVerilog or C/SystemC. Note that only DUT is on Veloce but the Testbench is on the CoModel Server. The DUT on emulator is the only "timed" portion, in that it uses clock. Testbench on the other hand, is completely "untimed", in that it has no timing control and runs as a program in zero emulation time.



We refer to Testbench, the untimed portion as HVL. The DUT and Transactor, the timed portion is referred to as HDL.

In this mode, Host Link and CoModel Link, both are used. The host link is used to connect, configure, download emulation stats (similar to Standalone/ICE mode). CoModel Link is used to transfer the data between the testbench and the design running on the emulator. The data transfer occurs at a higher abstraction level, called "Transaction Level Modeling". The timing protocol itself is at the pin level and a top level entity called "Transactor" converts a transaction to pin level protocol and vice-versa.

It Is important to note that every time a testbench component (untimed) is called, the emulation clock is paused. Hence in order to achieve maximum speed up you need to reduce and avoid unnecessary communication between the two entities.

The methodology is called TestBench Xpress (TBX). CoSimulation is the mode when your testbench is in SystemVerilog/Verilog and runs on a simulator. CoModel is the mode when your testbench is C/SystemC and runs on a C engine.

When developing projects in this mode, it is important to note that you are partitioning your project based on what is timed and what is untimed. Your testbench can no longer have any timing control.

Before you begin building your own project in this usage mode, please review document – TBX_Mode_Project_Flow.pdf and pay particular attention to Puresim and Veloce run modes.

The transactions between the testbench and the DUT can be achieved by several ways –

1. DPI-C Calls (CoModel) (Example – hello_world example in $VMW_HOME/examples)
   Your testbench is in C or SystemC. It leverages SystemVerilog Programming Language Interface (PLI) features to achieve Direct Programming Interface (DPI). The functions in C testbench can be imported in the Transactor (that runs on emulator). Similarly tasks/functions inside the Transactor can be exported to C code.

2. SCEMI Pipes – (Example – BoothTBX.tar.gz)
   SCEMI stands for Standard CoEmulation Modeling Interface (pronounced SKI-MI) SCEMI is a standard developed by Accelera. The SCEMI pipes work similar to a FIFO queue and buffers data between the HVL and HDL sides.

3. Bus Functional Model (BFM) Hierarchical Access (Example – BFMTBX.tar.gz)
   BFM is industry standard way or providing protocol level access through SystemVerilog interfaces. An interfaces replaces the Transactor. BFM task in the interface can be accessed with hierarchical reference from the HVL.

4. BFM Virtual Interface Access (Example – $VMW_HOME/vif_methodology)
   This mode uses a BFM as previous mode, however the access to BFM is made through Virtual Interface handle instead of hierarchical. There are certain advantages to virtual interface handle in terms of testbench reusability and scalability. This also forms the basis for support to UVM and OVM methodologies.

5. UVM Environment
   UVM Environment for emulation follows split-top as like any CoSimulation mode. The BFM in interface is accessed through virtual handles.