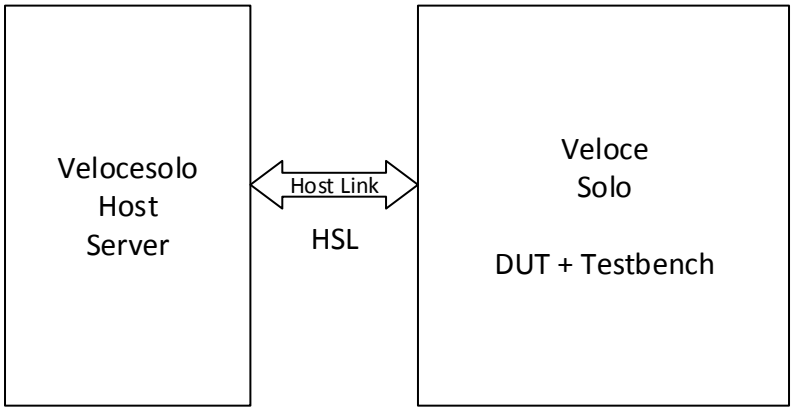
Veloce Standalone Emulation Mode Project Flow
(With Examples)
Examples Author – Haera Chung, September 2010
Updated for VeloceOS3 by- Sameer Ghewari, April 2015

This document portrays project flow for Standalone Emulation 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. You are connected to Velocesolo with VNC GUI Access
   (GUI Access is a essential for Standalone Mode)

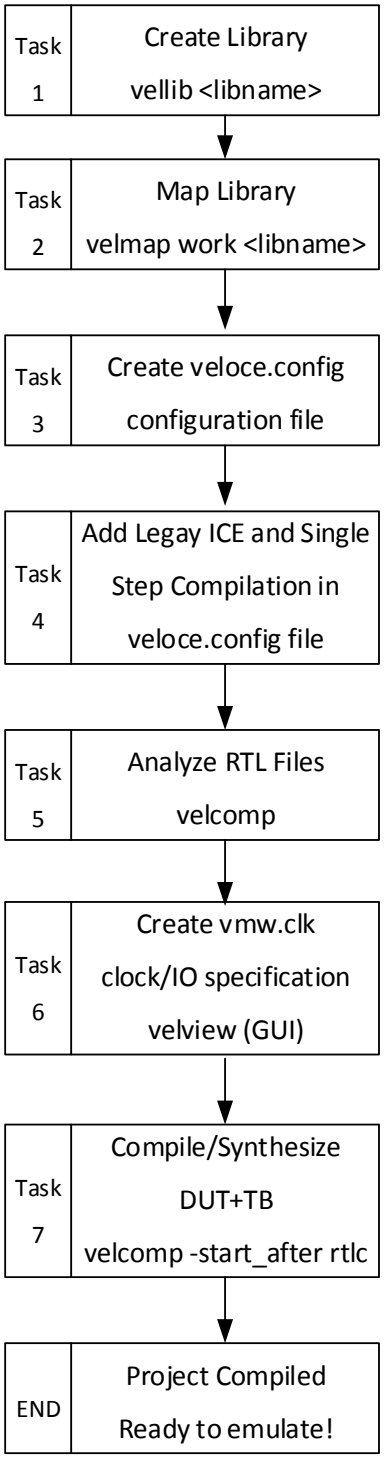
Recommended Reading -
Veloce ICE Flow User Guide, Software Version 3.0.0, September 2014 – Chapter 1 and Chapter 2

Important Note -
All of the flows shown on the following pages follow this characteristics

Left Hand Column	Right Hand Column
The Flow Chart with Generic Commands	Description and the exact command to be executed for – standalone_tutorial_1.tar.gz Note that the command to be typed on the terminal/shell/prompt is highlighted

All three examples of Standalone Mode follow the exact flows. You are encouraged to follow first tutorial step-by-step as per right hand column before making use of Makefile and do file. veloce.config file has been provided.

Flow 1: Compile Flow for Standalone Mode



The compilation flow remains the same for all three methods of achieving standalone emulation. The Makefile in examples follows this flow. Until Step 5, use command line interface (Terminal).

1. Create work library for project
   vellib work
2. Map the library to default work library
   velmap work work

3. veloce.config file specifies compiler options, it is essential before continuing with further flow of compile/synthesis.

4. Standalone mode is achieved by using Legacy ICE support in OS3 compile. This also requires use of Single Step compilation.

5. Analysis step extracts information from the RTL files for compilation.
   velcomp
   Note that it will complain about missing vmw.clk file at this point, this is what we generate next -

6. In standalone mode, clock is generated from clock specification file. More information is on the **next page**. (GUI based steps)

7. Once vmw.clk specification is ready, the project compilation resumes and can be compiled/synthesized. Since we already did rtlc analyze, we continue after that.
   velcomp –start_after rtlc
   You may see many warnings on screen, it is safe to ignore them.

8. If you see- “Info! [velcomp-25215]: : Compilation finished successfully.” Project is now ready to be emulated in standalone mode. Follow Standalone Run Flow.

- *Creating Clock and I/O Specification File (vmw.clk) using velview GUI*

Prerequisites – Complete the compilation flow until Step 5 – Analyze RTL Files.

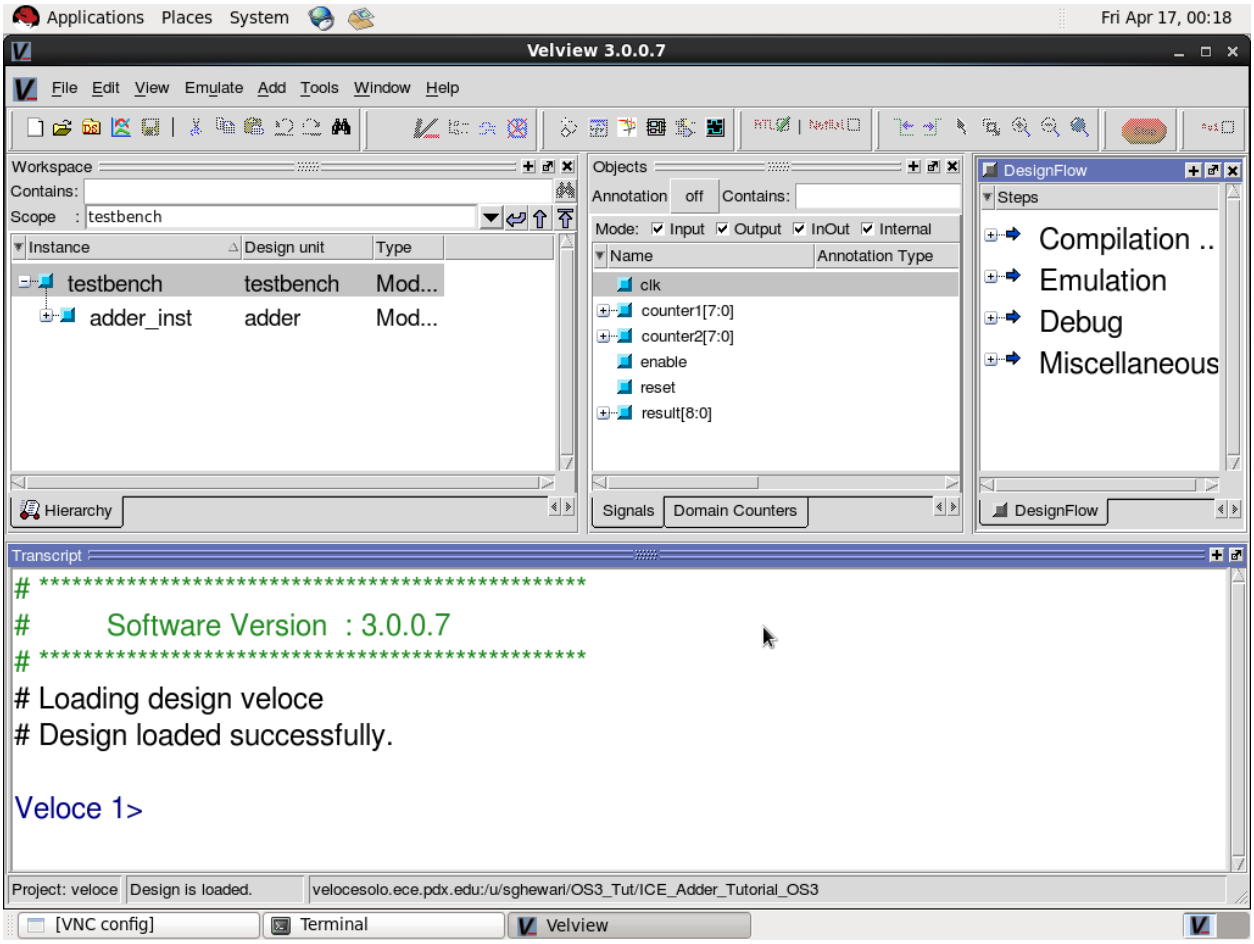
Once the analysis is done, veloce.med directory is created in the project folder. This directory contains essential information of the project to GUI to work. We will use “velview” GUI to view the project and use GUI steps to create clock specification file.

It is also possible to create vmw.clk file by just typing it, but GUI based approach is recommended for easier and correct clock specification file generation.

Step 1: Open velview GUI by typing

velview

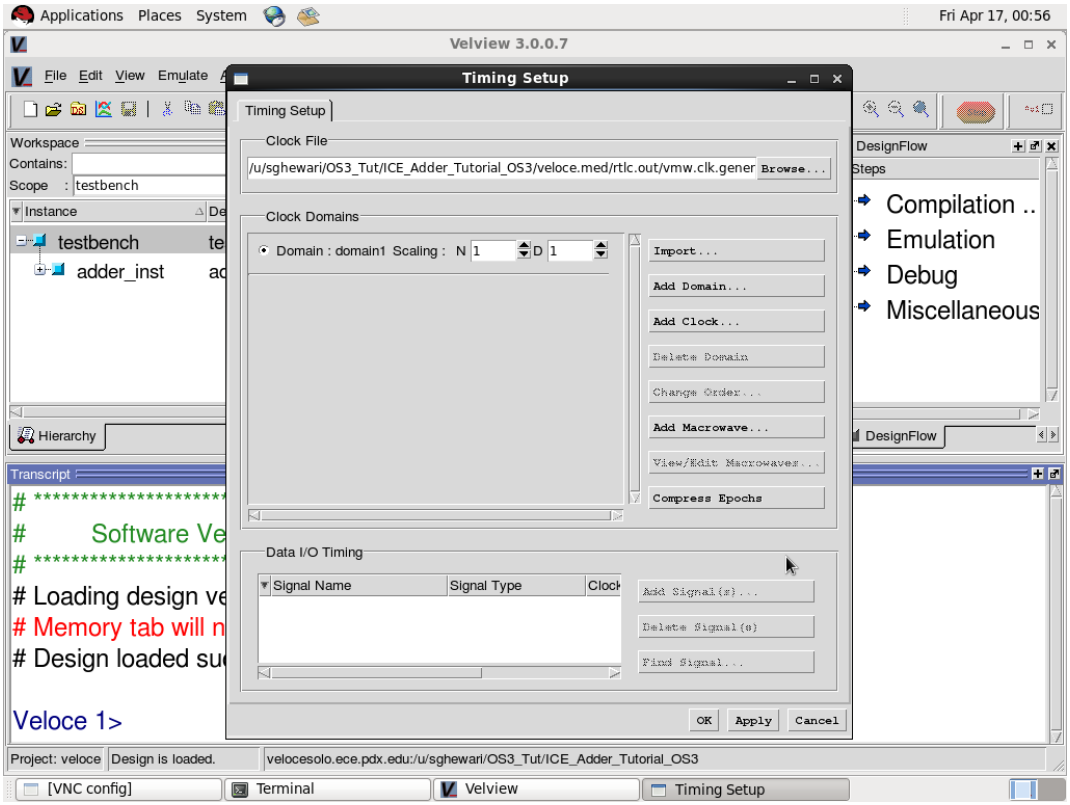
You should see a GUI like-



You should be able to see list of signals inside Signals Tab under Object Window.
If you don’t see this GUI, or if it complains that the design was not compiled until RTLC stage, then you haven’t successfully completed the Flow 1 until Task 5.

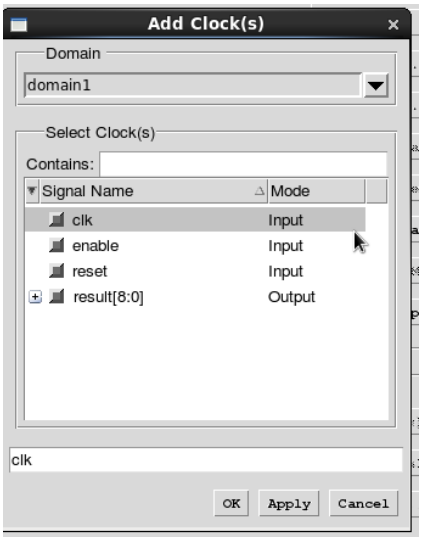
Step 2: Open Tools > Timing Setup

You should see the following window -



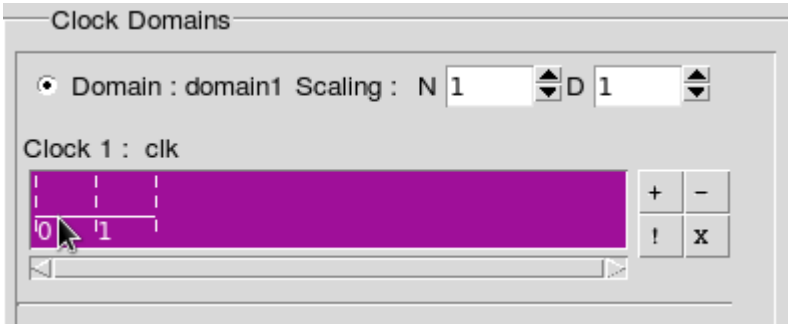
Step 3: Add Clock

In this step, we will specify the clock signal name in our design. Click on- Add Clock button on the right side of the window. Then choose the clock net name in your design, in this example it is “clk”. Click on OK.

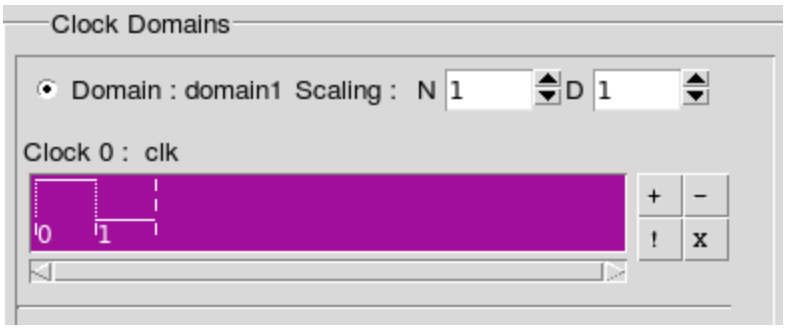


Step 4: Generate Clock

Next, we will generate the trigger for the clock. It must start with a positive edge. Once adding the clock as previous step, it looks like a flat line -



Click on the highlighted portion above, so that it looks like a clock starting with a posedge –

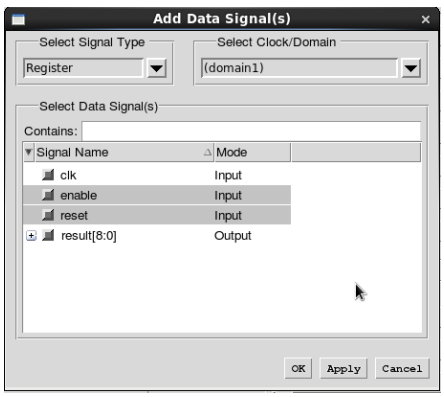


Step 5: Add Data I/O Timing

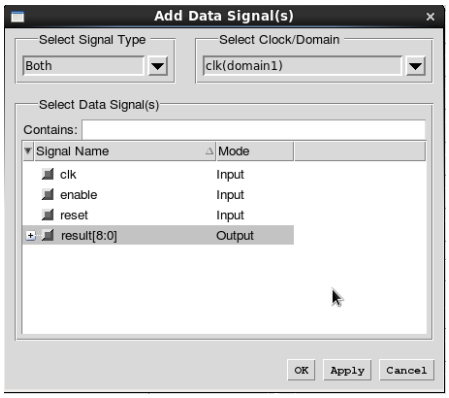
In this step we will make all “inputs” (except clock!) as “Register” type so that we can drive them to any value with a command. We will make all “outputs” as “Both” type so that they are logged on both edges of the clock. This is very similar to the way you will declare inputs as “reg” and outputs as “wire” in a traditional testbench.

In the bottom section of the Timing Setup window – Data I/O Timing section is present.

i. Adding All Inputs as Register
Click on Add Signal(s) option.
Select Signal Type is “Register”
Select all inputs (except clock)
Click OK

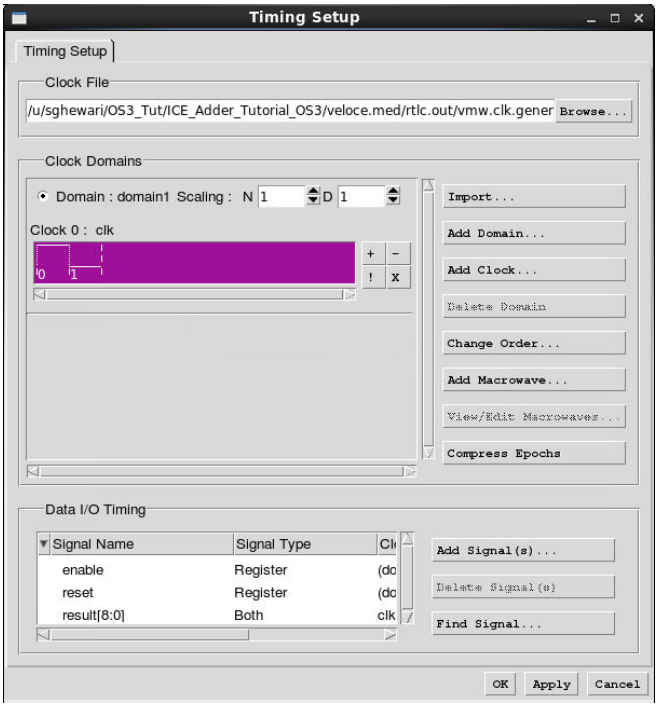


ii. Adding outputs as Both
Click on Add Signal(s) option
Select Signal Type as Both
Select all outputs
Click OK



Step 5: Save the vmw.clk.generated file

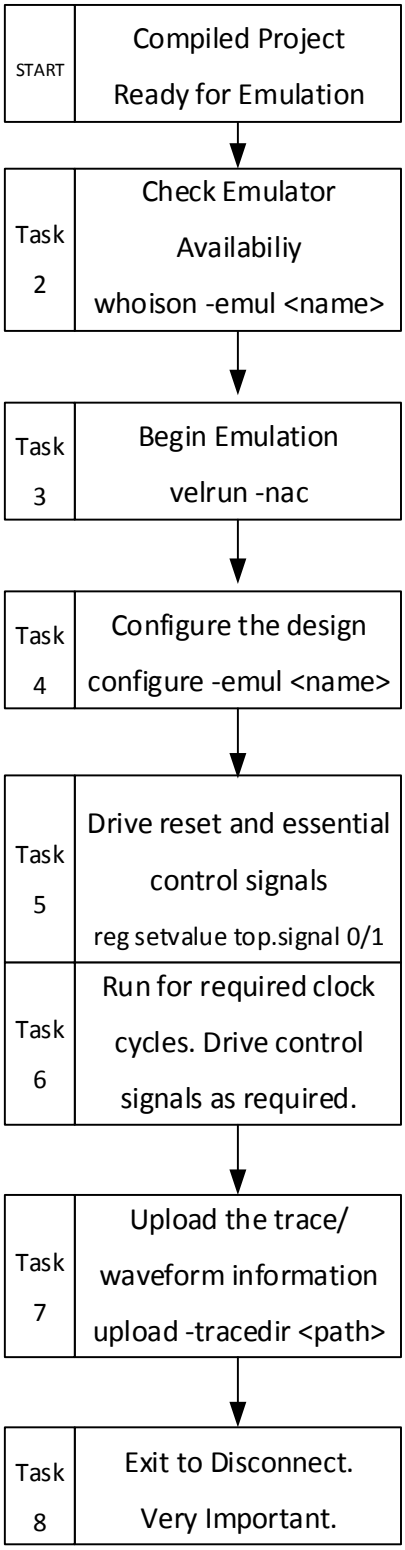
Finally, it should look similar to-



Now, click on OK. You should see the following message in the Transcript -
Clock file '/u/<Project_Folder_Path>/veloce.med/rtlc.out/vmw.clk.generated' saved.

You have now generated the Clock and I/O Specification file. To exit the velview GUI - Click on “X” in the top right and say “Yes” to exit, alternatively type “exit” in the transcript. You should be back on the terminal prompt. Now, resume compilation flow from Task 7 in the above flowchart.

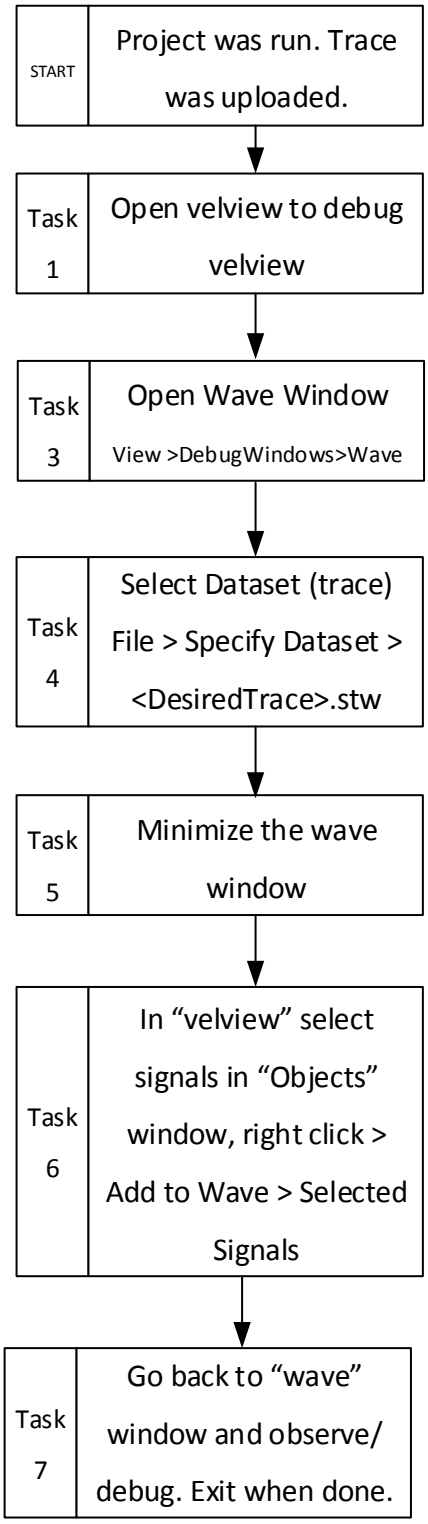
Flow 2: Run/Execute for Standalone



Prerequisites – Step 1: Compile flow is successful. You should have seen- "Info! [velcomp-25215]: : Compilation finished successfully." on terminal.

1. Begin with a compiled project. You need to be in the folder where you did Flow 1.
2. Check if anyone is connected to the emulator. Continue only if no one is connected.
   whoison –emul velocesolo1
3. Type velrun –nac -c, this launches velrun tool and open its own prompt - Veloce 1>
   Commands in all next tasks to be run in this prompt.
4. To configure the project on Veloce, type
   configure –emul velocesolo1
5. Drive control signals like "reset" and "enable" and run as required
   reg setvalue testbench.reset 1
   reg setvalue testbench.enable 0
   run 10
   reg setvalue testbench.reset 0
   run 10
   reg setvalue testbench.enable 1
   run 60000
6. In above steps we drove inputs and ran for required no of cycles.
   e.g. run 10 - runs for 10 cycles.
7. Once you run for required number of clocks, upload waveform data by-
   upload –tracedir ./veloce.wave/wave1
   (veloce.wave is preferred folder to upload wave. wave1 is the name of file)
8. **Disconnect!** Type-
   exit
   Do not stay connected once you have download traces.

Flow 3: Debug/View Waveform



Prerequisites – Step 2 : Run/Execute flow. You have run and uploaded trace.

1. Let’s begin with a project that was run and trace was uploaded.
2. Open velview by
   velview
3. Open the Wave window as specified
4. Select wave1.stw that we generated using Flow 2
5. Minimize the window, DO NOT close.
6. In objects window, select all signals and add them to the wave.
7. Go back to the Wave window (maximize) and observe the waveform and see if the design worked correctly.

Standalone Tutorial 1 – Adder DUT + Testbench on Veloce

(standalone_tutorial_1.tar.gz)

This example has a simple Adder and a counter based Testbench. Read the code and understand before executing. Note that testbench is synthesizable as well. In the above flows, the right hand side tells you the exact commands to be typed for this example. To get familiar with the flow, execute the flow step-by-step for the first round of execution.

Once you understand the flow, go ahead and use Makefile (type make). It also uses a do file, a standard way of providing list of commands to velrun as well as QuestaSim. Makefile and do files help us automate the tasks.

Standalone Tutorial 2 – Adder DUT + Testcases in Memory

(standalone_tutorial_2.tar.gz)

This examples uses $readmemh feature to download txt file containing testcases into a standard Verilog modeled memory. $readmemh is synthesizable for veloce. You still need to drive reset and enable signals. For this example, Makefile and do file are provided. You are encouraged to try step-by-step flow as well.

Standalone Tutorial 3 – Only Adder DUT

(standalone_tutorial_3.tar.gz)

This example uses Veloce as a big FPGA. There is no testbench written, instead the testcases are injected using “reg setvalue” command in do file. Note that the top level for this example is the Adder itself, unlike the previous two. The data inputs of Adder are to be made “Register” in the vmw.clk file through GUI. This is useful mode if you want to quickly try out synthesizing your design to check synthesis capability.