Generation of a .vcd-File for PowerPlay on Arria 10 Designs

G. Knittel
MPIfR Bonn, Germany
gknittel@mpifr-bonn.mpg.de
Ver. 1.1
Sep. 11, 2015

1 MOTIVATION

It seems that Quartus II 15.0 (Q15) does not support gate-level simulation for Arria 10 designs, neither functional nor timing. At least, to be precise, Q15 does not support these operations via NativeLink to the ModelSim-Altera Starter Edition. For an accurate power consumption estimation using PowerPlay, a precise description of toggle activities per node is required. These are normally recorded during gate-level simulation, which is not supported. Thus, one has to resort to less accurate default values.
This application note describes how to obtain a .vcd-file (*Value Change Dump*) despite this restriction. However, it is derived from functional simulation only, and might therefore not capture all transitions. The hope is that this is still more aligned with the physical behavior of the chip than a default toggle rate.
Note: a service request has been filed with Altera on Sep. 2, 2015, on this issue. No answer has been received so far, but any information from Altera may void this application note.

2 PROCEDURE

The steps are outlined using a small test design. Assumed we have set up a Q15 project for an Arria 10 10AX115U4F45I3SGES part with just one VHDL-file:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tripleadd is
  port ( clk : in  std_logic;
         a   : in  std_logic_vector (23 downto 0);
         b   : in  std_logic_vector (23 downto 0);
         c   : in  std_logic_vector (23 downto 0);
         d   : out std_logic_vector (23 downto 0) := x"000000");
end entity;

architecture rtl of tripleadd is

  signal areg, breg, creg : std_logic_vector(23 downto 0) := x"000000";

begin

process (clk)
begin
  if (rising_edge(clk)) then
    areg <= a;
    breg <= b;
    creg <= c;
    d    <= areg + breg + creg;
  end if;
end process;

end rtl;
```

Using Processing → Start → Start Test Bench Template Writer we generate a test bench skeleton and fill it arbitrarily:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tripleadd_vhd_tst IS
END tripleadd_vhd_tst;

ARCHITECTURE tripleadd_arch OF tripleadd_vhd_tst IS

SIGNAL clk : STD_LOGIC := '0';
SIGNAL a, b, c : STD_LOGIC_VECTOR(23 DOWNTO 0) := x"000000";
SIGNAL d : STD_LOGIC_VECTOR(23 DOWNTO 0);

COMPONENT tripleadd
PORT ( clk : IN  STD_LOGIC;
         a   : IN  STD_LOGIC_VECTOR(23 DOWNTO 0);
         b   : IN  STD_LOGIC_VECTOR(23 DOWNTO 0);
         c   : IN  STD_LOGIC_VECTOR(23 DOWNTO 0);
         d   : OUT STD_LOGIC_VECTOR(23 DOWNTO 0));
END COMPONENT;

BEGIN

clk <= NOT clk AFTER 25 ns;
inst1 : tripleadd PORT MAP ( a => a, b => b, c => c, clk => clk, d => d );

always : PROCESS (clk)
BEGIN
  IF (FALLING_EDGE(clk)) THEN
    a <= a + 1;
    b <= b + 3;
    c <= c + 5;
  END IF;
END PROCESS always;

END tripleadd_arch;
```

For NativeLink we need to fill in some forms. Afterwards the “Settings”-form should look like shown below. Note that we are using the built-in Altera-ModelSim Starter Edition.

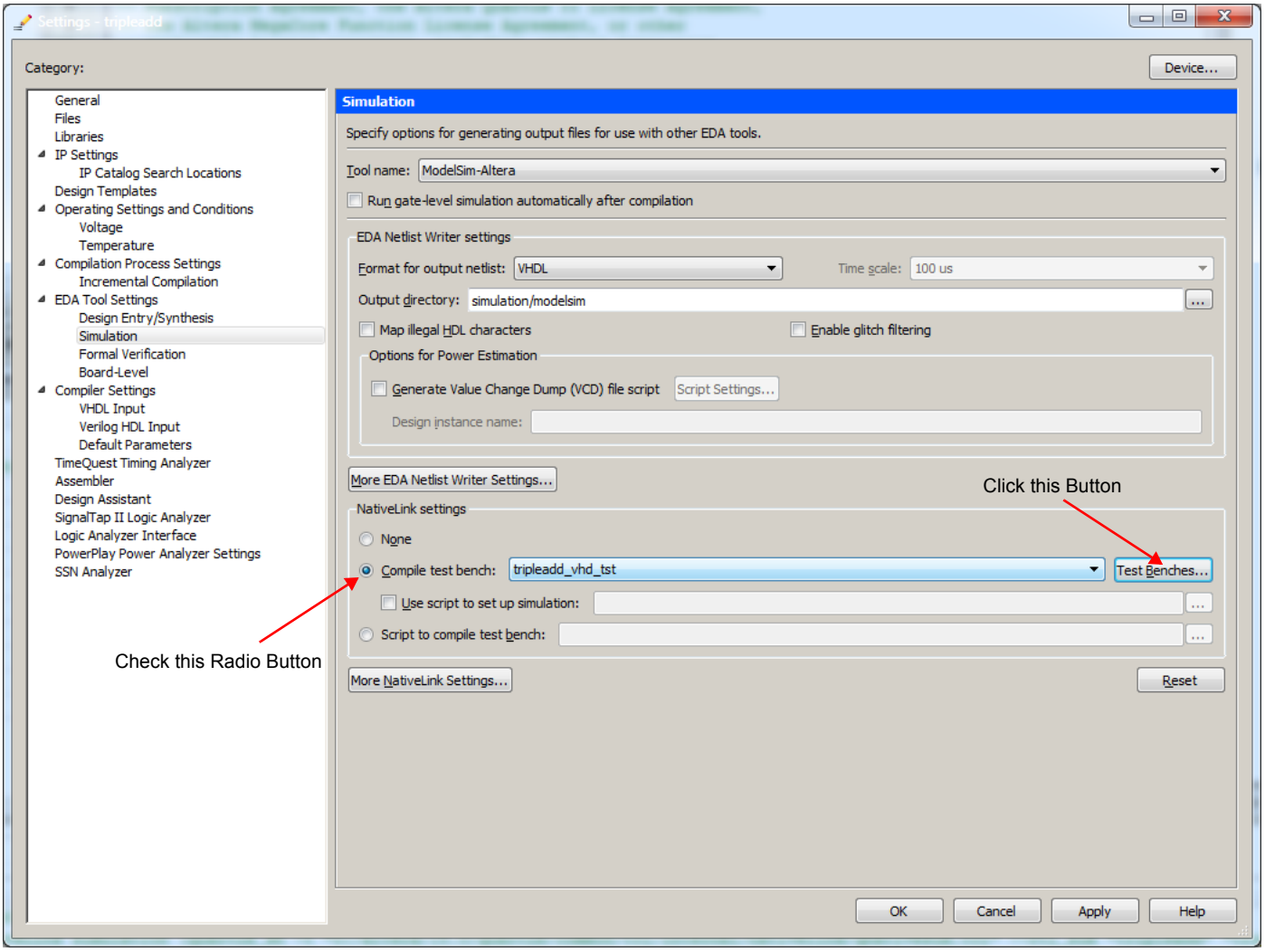


Figure 1: Settings for Simulation

Clicking on the “Test Benches...”-Button will open the following dialog box (shown after making changes):

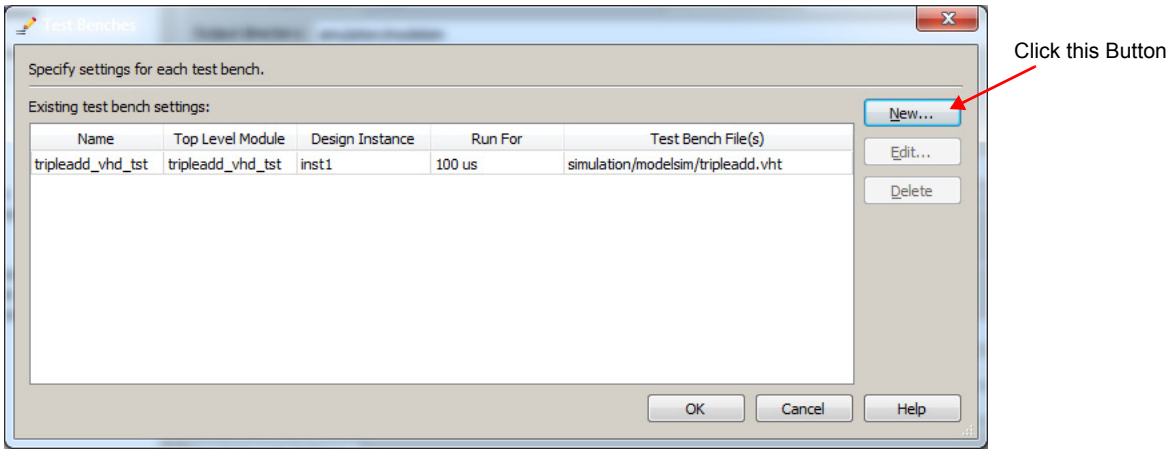


Figure 2: Test Bench Settings

Clicking on the “New..”-Button will open the final dialog box for this step:

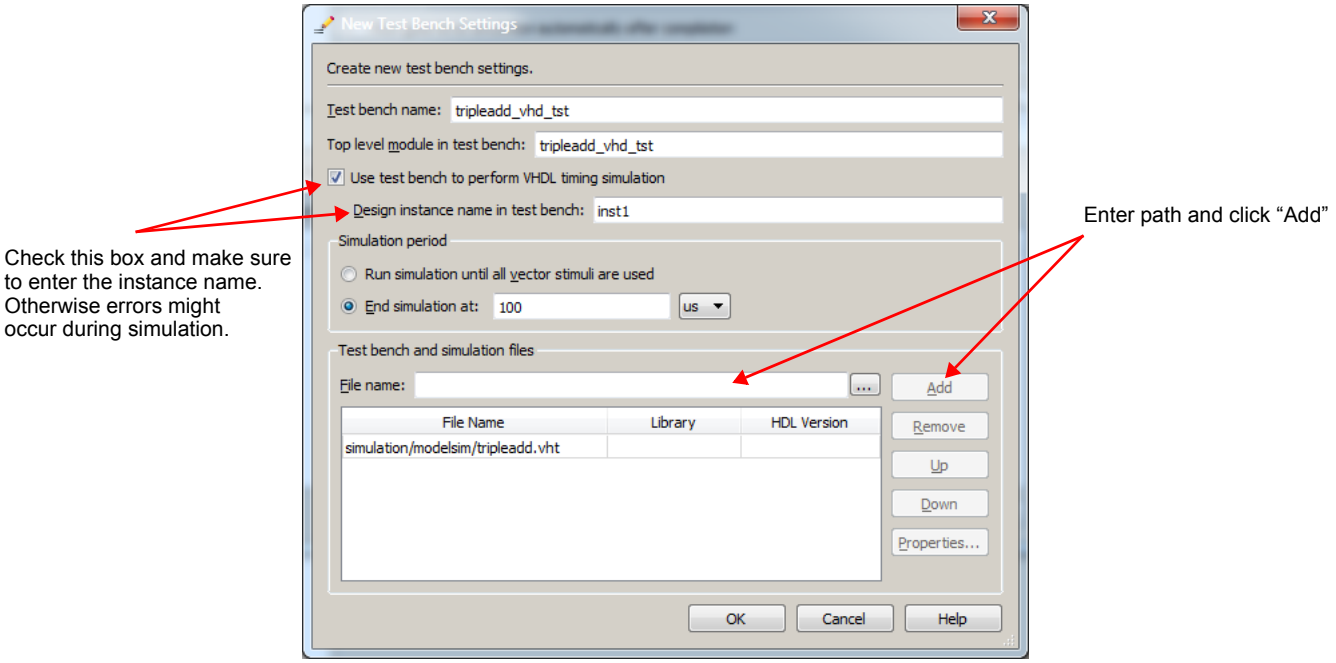


Figure 3: New Test Bench Settings

We could now perform an RTL simulation.
However, our goal is to run gate-level simulations so we need to make further changes. These relate to the EDA Netlist Writer. First we need to instruct it to generate a tcl-script which in turn instructs the simulator to record toggle activities of all nodes. Further we instruct it to generate a .do-script for a (non-existing) third-party simulation tool. After all the changes the dialog boxes look like shown in Figure 4 through Figure 6.

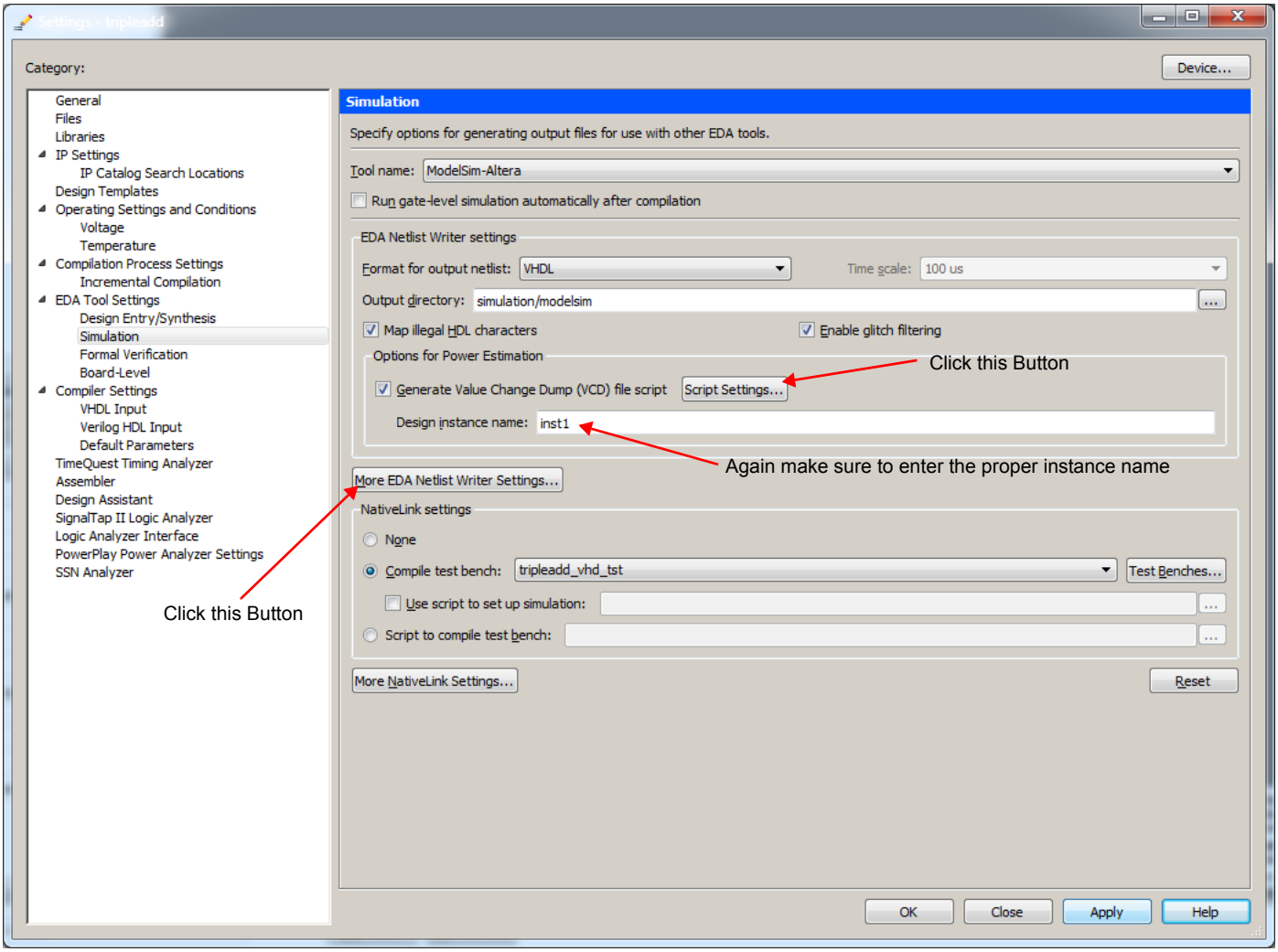


Figure 4: EDA Netlist Writer Settings

When clicking on “Script Settings”, a dialog box should pop up:

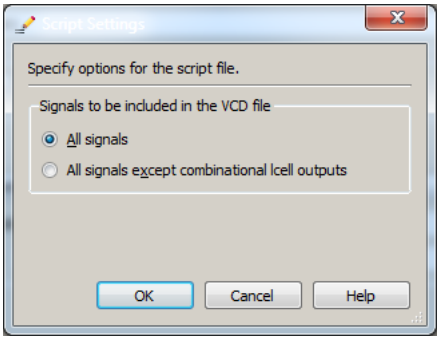


Figure 5: VCD File Script Settings

After clicking on “More EDA Netlist Writer Settings”, change the option “Generate third-party EDA tool command script for gate-level simulation” from Off to On.

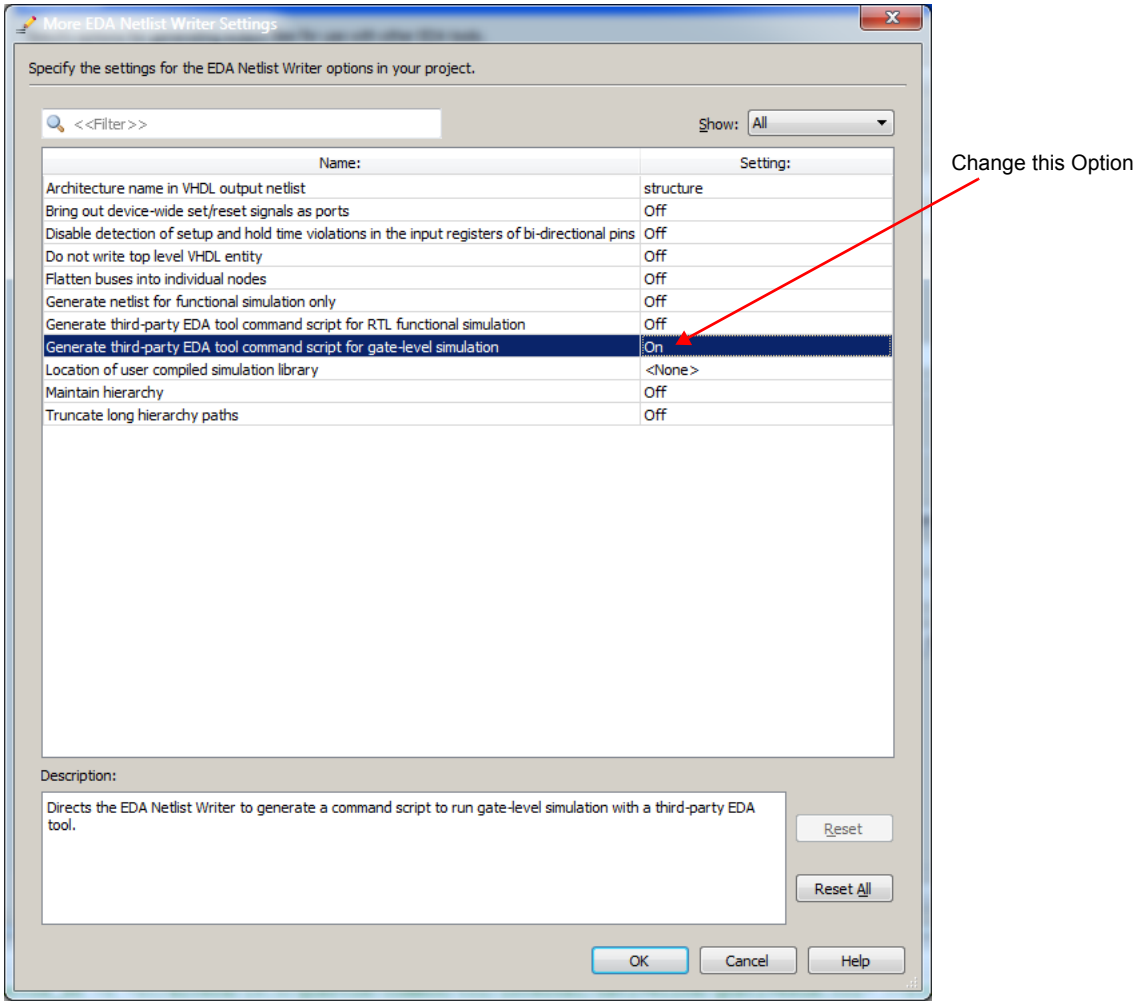


Figure 6: More EDA Netlist Writer Settings

Now the Q15 tools “Analysis & Synthesis”, “Fitter (Place and Route)” and “EDA Netlist Writer” must be executed. Before we do this, we should set up a minimal constraints file defining the clock and a few delays (so that the Timing Analyzer won’t complain):

```
# Create a simple 50ns clock
create_clock -period 50 -waveform {0 25} -name clk [get_ports clk]
set_input_delay  -clock { clk } 10 [get_ports {a[*]}]
set_input_delay  -clock { clk } 10 [get_ports {b[*]}]
set_input_delay  -clock { clk } 10 [get_ports {c[*]}]
set_output_delay -clock { clk } 10 [get_ports {d[*]}]
```

Additionally we might then want to make all I/O assignments. Now we can step through the tool flow above.

While running the EDA Netlist Writer will generate the following warning:

```
Warning (10905): Generated the EDA functional simulation files although EDA timing simulation option is chosen.
```

This is an indication that no timing-information (.sdo-files) have been generated. Trying to run a gate-level simulation (Tools → Run Simulation Tool → Gate Level Simulation) results in the following error message:

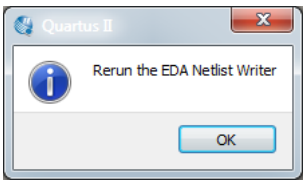


Figure 7: Error Message

The work-around is as follows.
The EDA Netlist Writer should have written (among others) two files:

- simulation/modelsim/tripleadd_dump_all_vcd_nodes.tcl, defining the nodes that should be monitored by the simulator, and
- simulation/modelsim/tripleadd_run_msim_gate_vhdl.do, which is the simulation script for ModelSim.

In this example the simulation script is as follows:

```
transcript on
if {[file exists gate_work]} {
vdel -lib gate_work -all
}
vlib gate_work
vmap work gate_work

vcom -93 -work work {tripleadd.vho}

vcom -93 -work work {<... your path ...>/TripleAdd/simulation/modelsim/tripleadd.vht}

vsim -t 1ps +transport_int_delays +transport_path_delays -sdftyp /inst1=tripleadd_vhd.sdo -L altera
-L altera_lnsim -L twentynm -L twentynm_hssi -L lpm -L sgate -L twentynm_hip -L gate_work -L work
-voptargs="+acc"  tripleadd_vhd_tst

source tripleadd_dump_all_vcd_nodes.tcl
add wave *
view structure
view signals
run 100 us
```

We can see that the .tcl-script for writing the .vcd-file will be executed. However, in the absence of any timing information we need to remove all timing-related options from the vsim command line. These are marked red.
After editing the command line we need to save it as a separate file, for example as

simulation/modelsim/tripleadd_run_msim_gate_vhdl_own.do.

In principle we can set the option in Figure 6 to “Off” again.
We can then instruct the system to use this script whenever a simulation run is desired, as shown in Figure 8. The effect of this check box is that the simulation scripts that are generated by the tools now include a line at the end to execute our own simulation script. For example, the script for doing an RTL simulation now looks like the following:

```
transcript on
if {[file exists rtl_work]} {
vdel -lib rtl_work -all
}
vlib rtl_work
vmap work rtl_work

vcom -93 -work work {<... your path ...>/TripleAdd/tripleadd.vhd}

vcom -93 -work work {<... your path ...>/TripleAdd/simulation/modelsim/tripleadd.vht}

vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L twentynm -L twentynm_hssi -L twentynm_hip
-L rtl_work -L work -voptargs="+acc"  tripleadd_vhd_tst

do <... your path ...>/TripleAdd/simulation/modelsim/tripleadd_run_msim_gate_vhdl_own.do
```

This means that after performing an RTL simulation, which the system allows us to do, automatically a functional gate-level simulation will follow. We could delete all lines except the last one to save the RTL simulation step, but the system will complain about a modified script before each run, which is not very convenient.
So, after re-executing the Quartus tool flow (Synthesis, Fitter, EDA Netlist Writer) we can then perform an RTL simulation, and implicitly a gate-level simulation, which was not possible before. After quitting ModelSim we will find a .vcd-file at

simulation/modelsim/tripleadd.vcd.

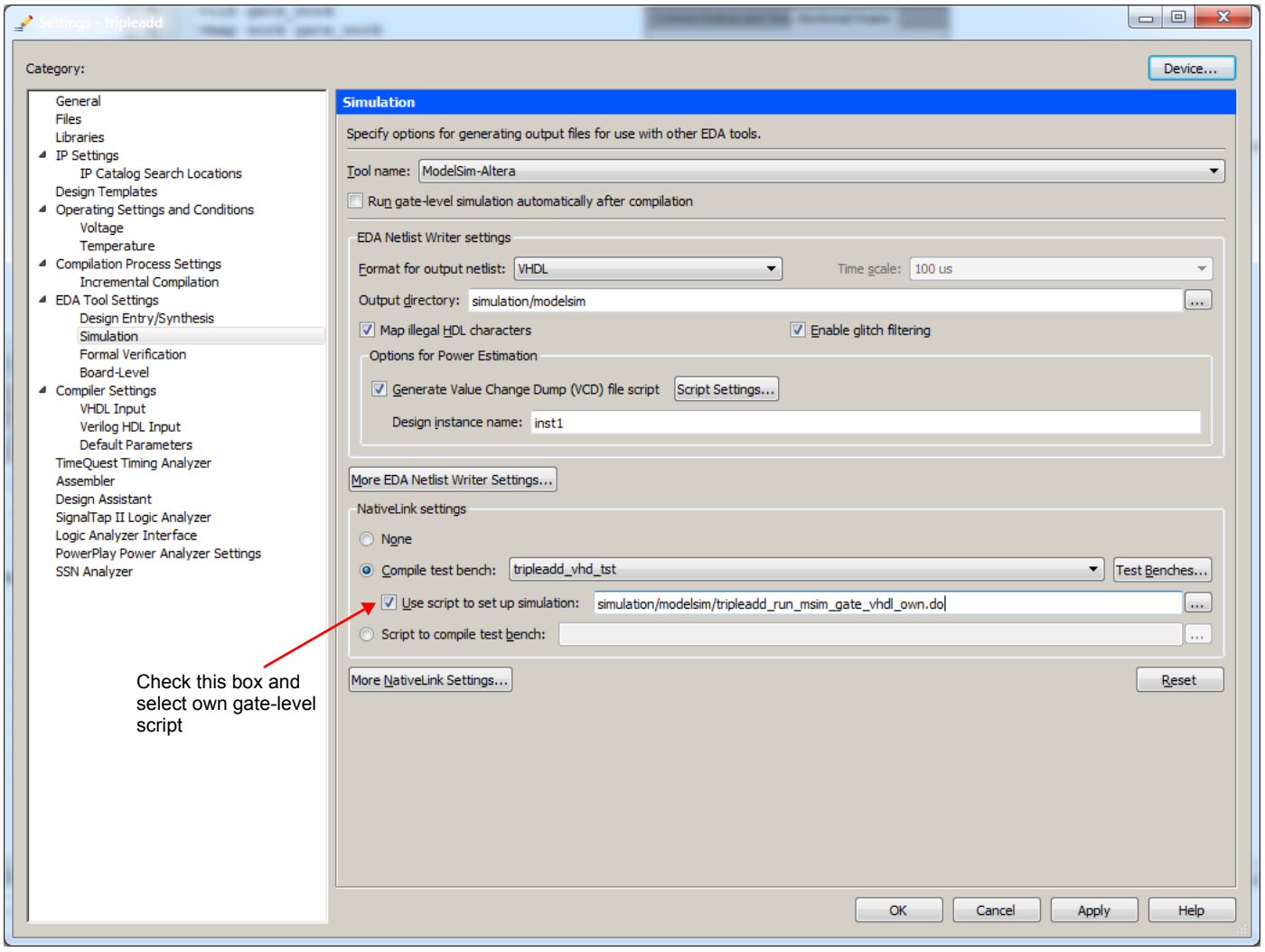


Figure 8: Settings with own .do-Script

Note: during gate-level simulation the Altera-ModelSim Starter Edition reports (for this small design already)

```
# ** Warning: Design size of 40546 statements exceeds ModelSim Altera Starter recommended capacity.
# Expect performance to be adversely affected.
```

Thus for larger designs no guarantee can be given that this method will work.

Using this .vcd-file we can now configure PowerPlay. The “Settings” dialog box should then look like in Figure 9. While running PowerPlay it will report:

```
Info (222002): Starting scan of VCD file simulation/modelsim/tripleadd.vcd (0 ns to End of File) for signal
static probabilities and transition densities
Info (222003): Finished scan of VCD file simulation/modelsim/tripleadd.vcd (0 ns to End of File) for signal
static probabilities and transition densities
```

When PowerPlay has finished execution it will report a high “Power Estimation Confidence”, which was the goal of this exercise (see Figure 10a). When using a default toggle rate instead, the confidence is low: “user provided insufficient toggle rate data” (see Figure 10b). Note also the fairly large dynamic power estimate differences.

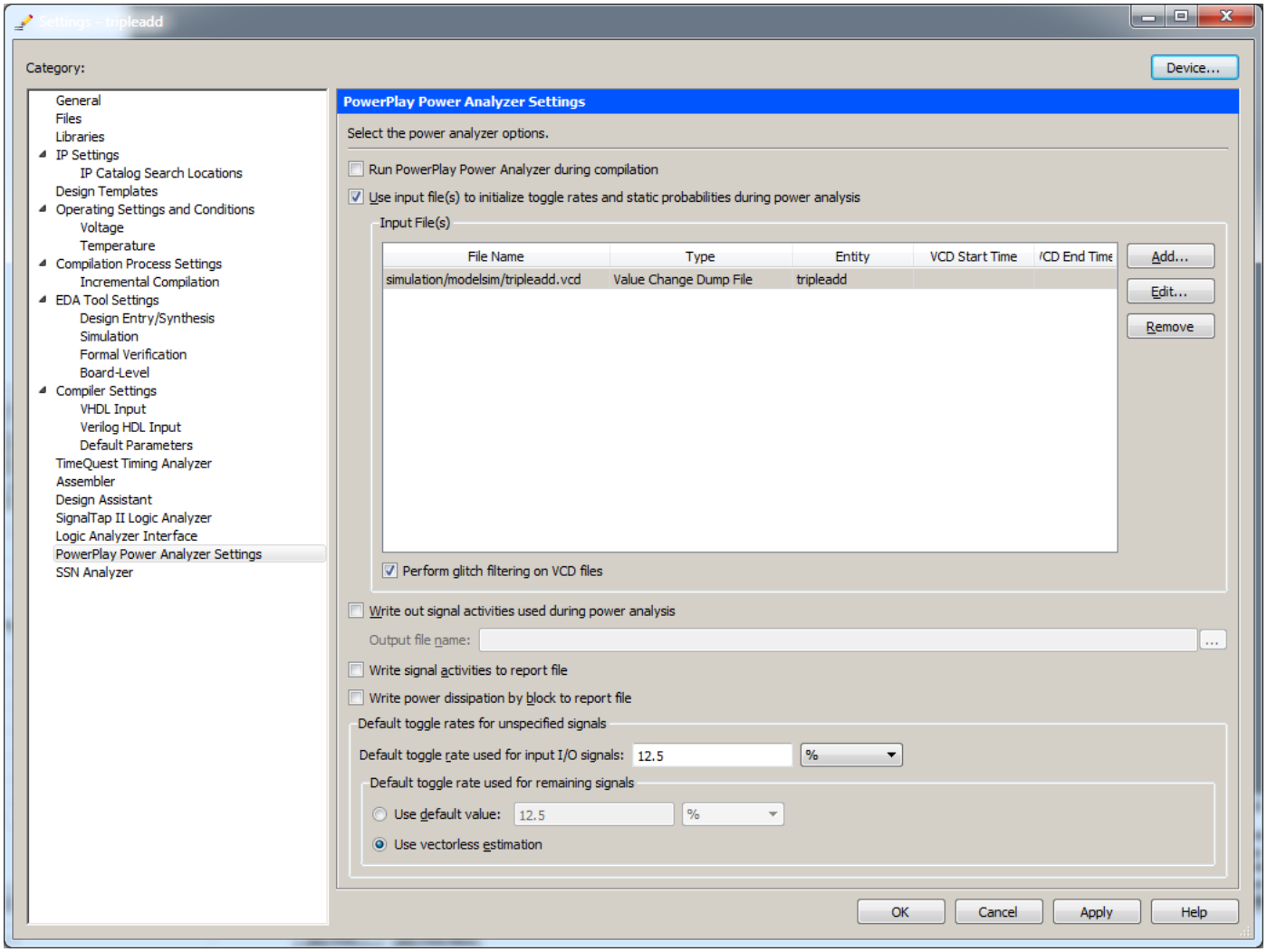
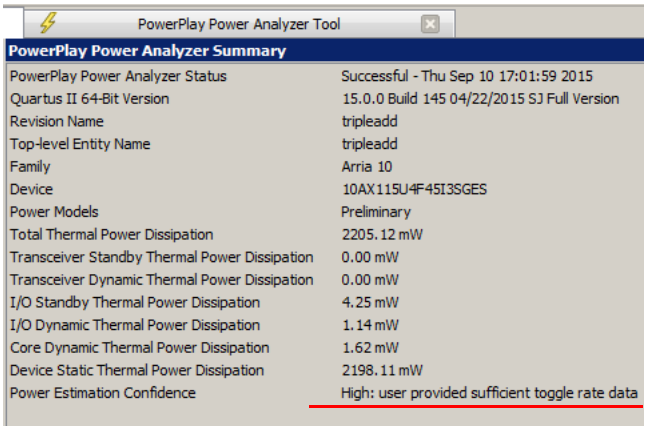
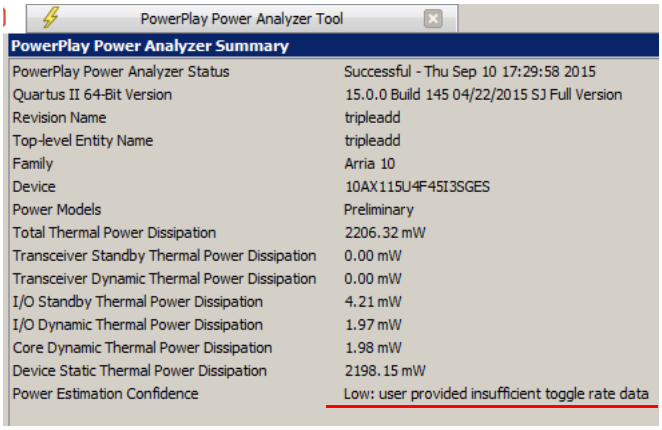


Figure 9: PowerPlay Settings



a.) Using the .vcd-file



b.) Using a default toggle rate of 12.5%

Figure 10: PowerPlay Power Analyzer Summary