Einführung in ModelSim

Version 0.9

Dokumentenverwaltung

Dokument-Historie

Version	Status	Datum	Verantwortlicher	Änderungsgrund
0.1	In Arbeit	9.9.2011	L. Arato	Start des Dokumentes
0.2	Draft	26.9.2011	L. Arato	Three_Bit_Counter
0.3	Draft	26.9.2011	L. Arato	Three_Bit_Counter mit direkten Stimuli
0.4	Draft	27.9.2011	L. Arato	Clean VHDL Code und 2. Beispiel
0.5	Release	28.9.2011	L. Arato	2. Beispiel nur „einfache“ Testbench
0.6	Extension	20.2.2012	L. Arato	Inhaltsverzeichnis und Kapitel 8
0.7	Korrektur	9.5.2012	L. Arato	Seite 23, END PROCESS Name
0.8	Korrektur	12.3.2013	L. Arato	Seite 15, „.ALL“ für DUT Use Statement
0.9	Korrektur	15.5.2014	L. Arato	Seite 15, _pkg für USE three_bit_counter, Seite 1, VERTEILER entfernt.

Änderungsberechtigte

Laszlo Arato Institut EMS NTB, Buchs
Dr. Urs Graf Institut INF NTB, Buchs

Dokument wurde mit folgenden Tools erstellt:

Microsoft WORD 2010

Inhaltsverzeichnis

1 Einleitung ... 4
1.1 Zweck des Dokuments ... 4
1.2 Gültigkeit des Dokuments ... 4
1.3 Begriffsbestimmungen und Abkürzungen ... 4
1.4 Zusammenhang mit anderen Dokumenten ... 4
2 Installation ... 5
2.1 Unterschiede „Altera Edition“ und „Altera Starter Edition“ ... 5
2.2 Download von ModelSim-Altera ... 5
2.3 Installation ... 5
3 Lizenzierung ... 5
4 Einstellungen in Quartus für Modelsim ... 6
5 Erste Simulation: ThreeBitCounter ... 7
5.1 VHDL Source Code ... 7
5.2 Erklärungen zum three_bit_counter VHDL Code ... 8
5.3 Einstellungen für ModelSim als Simulator ... 9
5.4 Starten der Simulation ... 9
5.5 Simulieren mit einzeln konfigurierten Signalen ... 11
5.6 Steuerung der Signale durch ein Skript-File ... 13
do ../../../sim/stimulus.do // Führt die Befehle im Skript-File aus. Der Pfad ist relativ zum ModelSim // Verzeichnis unter „quartus/simulation/modelsim“ ... 13
6 Three_bit_counter mit einer Testbench ... 14
6.1 Die Testbench ... 14
6.2 Regeln für die Testbench ... 14
6.3 VHDL Testbench für den ThreeBitCounter ... 15
6.4 Erklärungen zur Testbench für den ThreeBitCounter ... 15
6.5 Simulations-Einstellungen und Definition der Testbench ... 17
6.6 Aufruf von ModelSim und Starten der Simulation ... 19
7 Self-checking Testbench : Full Adder ... 21
7.1 VHDL Source Code ... 21
7.1.1 full_add ... 21
7.2 adder4 ... 22
7.3 Einfache selbst-checkende Testbench für adder4 ... 23
7.4 Erklärungen zur Testbench ... 24
8 Anspruchsvolle Testbench : Arcus Tangens CORDIC ... 25
8.1 VHDL Source Code ... 25
8.1.1 arctan_cordic.m.vhd ... 25
8.1.2 barrel_shifter.m.vhd ... 30
8.1.3 cordic_rom.m.vhd ... 31
8.2 arctan_cordic.tb.vhd ... 32
8.3 ModelSim Command File arctan_cordic_rtl_vhdl.do ... 35

8.4 ModelSim Wave Command File wave.do 36

1 Einleitung

1.1 Zweck des Dokuments

Diese Einführung soll Studenten und anderen interessierten Personen helfen, möglichst schnell und effizient die ModelSim Software von Mentor Graphics für FPGA Entwicklung zu nutzen.
Verwendet wird dabei das „ModelSim AE“, wobei das AE für „Altera Edition“ steht.
Nach bisheriger Erfahrung ist diese weitgehend identisch zur „ModelSim ASE“ (Altera Starter Edition).

1.2 Gültigkeit des Dokuments

Dieses Pflichtenheft ist für NTB internen Gebrauch.
Die Ausführungen gelten sowohl für die kostenlose Webedition Ausführung, wie auch für die Lizenzierte Vollversion. Dort wo Unterschiede bestehen, wird darauf explizit hingewiesen.

1.3 Begriffsbestimmungen und Abkürzungen

FPGA	Field Programmable Gate Array, ein programmierbarer Logikbaustein.
ALTERA	Altera Corporation ist ein Hersteller von FPGAs
Quartus II	Die offizielle Software von Altera für CPLD und FPGA Entwicklungen
VHDL	„Very High-Speed Hardware Description Language“

1.4 Zusammenhang mit anderen Dokumenten

Dieses Dokument ist die Ergänzung zum Dokument „Einführung in Quartus II“.

Folgende begleitende Dokumente sind geplant oder bereits in Arbeit:

- Einführung in VHDL Design
- Einführung in VHDL Testbench Design
- VHDL Design Guidelines

Weitere unterstützende Literatur:
- DE2_115_User_Manual.pdf

2 Installation

Die Software kann kostenlos von der Webseite von Altera heruntergeladen werden.
Dazu ist eine Registrierung erforderlich.

2.1 Unterschiede „Altera Edition“ und „Altera Starter Edition“

Die „Altera Starter Edition“ ist gratis, aber begrenzt auf Total 10‘000 Zeilen ausführbarem Code.

Die „Altera Edition“ ist Teil der NTB Altera Lizenzen. Wenn man diese Lizenz sonst kaufen will, kostet sie $ 945.- Dollar.

Die genauen Unterschiede findet man auf der Alter Webseite:
http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html

2.2 Download von ModelSim-Altera

ModelSim ist eine sehr starke Simulationsumgebung für digitale Schaltungen. Die ModelSim-Altera Version ist speziell eng mit Quartus II verbunden, kann aber auch einzeln genutzt werden.

Dabei ist die „Modelsim-Altera“ Version (ModelSim AE) von der kostenlosen „ModelSim Altera Starter Edition“ (ModelSim ASE) zu unterscheiden. Erstere funktioniert nur, wenn man z.B. über VPN Zugriff auf die NTB Lizenzfiles hat.

https://www.altera.com/download/software/modelsim/11.0 oder
https://www.altera.com/download/software/modelsim-starter/11.0

Hier kann man auch wieder die gewünschte Version und Betriebssystem auswählen.

Das File für Windows ist 341 MBytes gross.

2.3 Installation

Man kann für ModelSim die Installations-Files einzeln herunter laden, oder aber mit dem Altera-Installer arbeiten … es kommt auf das selbe heraus.

Wenn man mehrere PCs mit derselben Software aufsetzten will, dann ist es vielleicht einfacher, die Files einmal herunter zu laden, um dann mehrmals zu verwenden. Für das NTB befinden sich diese bereits in
X:\Unterricht\arato\Altera_Source\11.0_modelsim_ae_windows
X:\Unterricht\arato\Altera_Source\11.0_modelsim_ase_windows

3 Lizenzierung

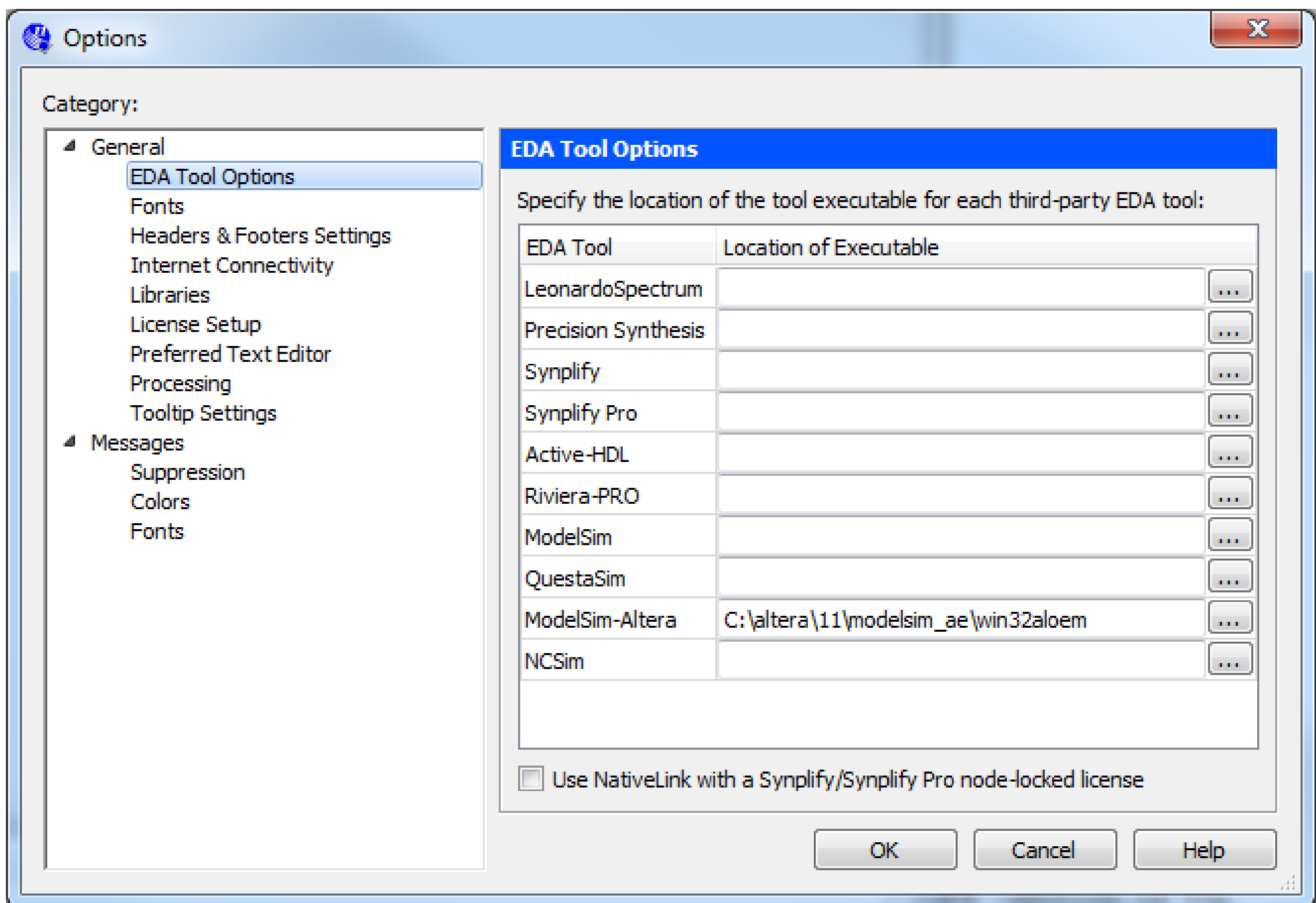
Dies ist in der allgemeinen Altera Quartus Lizenz enthalten. Genauere Beschreibung findet man im Dokument „Einführung in Quartus II“, Kapitel 3.

4 Einstellungen in Quartus für Modelsim

In Quartus muss eingestellt werden, welches Simulations-Werkzeug verwendet wird.

Da dies eine Projekt-spezifische Einstellung ist, muss sie bei einem komplett neuen Projekt immer neu von Hand eingegeben werden. Bei Projekten die auf einem bereits bestehenden Projekt aufbauen, ist dies nicht nötig.

Im Quartus das „EDA Tool Options“ Menu öffnen mit
Tools → Options … → EDA Tool Options



Dort muss in der Zeile “Modelsim-Altera” der Pfad zum ModelSim Programm eingegeben werden.

Für die volle Modelsim Version wäre dies der Pfad „C:\altera\11\modelsim_ae\win32aloem“

Für die Starter-Edition von Modelsim wäre dies „C:\altera\11\modelsim_ase\win32aloem“

Gebraucht wird der Pfad zum Verzeichnis mit dem (seltsamen) Namen **win32aloem** !!!

5 Erste Simulation: ThreeBitCounter

Anhand eines sehr einfachen Beispiels werden die Funktionen aufgezeigt und erklärt.

5.1 VHDL Source Code

Die Target-Funktion ist ein sehr einfacher 3-Bit Zähler mit einem Enable-Signal:

```
25      LIBRARY IEEE;
26      USE IEEE.STD_LOGIC_1164.ALL;
27      USE IEEE.NUMERIC_STD.ALL;
28
29      PACKAGE three_bit_counter_pkg IS
30          COMPONENT three_bit_counter IS
31              PORT (
32                  clk         : IN  STD_LOGIC;
33                  enable      : IN  STD_LOGIC;
34                  count       : OUT UNSIGNED (2 DOWNTO 0)
35              );
36          END COMPONENT three_bit_counter;
37      END PACKAGE three_bit_counter_pkg;
38
39      ----------------------------------------------------------------------

52      LIBRARY IEEE;
53      USE IEEE.STD_LOGIC_1164.ALL;
54      USE IEEE.NUMERIC_STD.ALL;
55
56      ENTITY three_bit_counter IS
57              PORT (
58                  clk         : IN  STD_LOGIC;
59                  enable      : IN  STD_LOGIC;
60                  count       : OUT UNSIGNED (2 DOWNTO 0)
61              );
62      END ENTITY three_bit_counter;
63
64      ----------------------------------------------------------------------
65
66      ARCHITECTURE Behavioral OF three_bit_counter IS
67
68      SIGNAL internal_count   : UNSIGNED (2 DOWNTO 0) := "000";
69
70      BEGIN
71          counter: PROCESS (clk, enable)
72          BEGIN
73              IF rising_edge(clk) AND enable ='1' THEN
74                  internal_count <= internal_count + 1 AFTER 15 ns;
75              END IF;
76          END PROCESS counter;
77
78          Count <= internal_count;
79
80      END Behavioral;
```

5.2 Erklärungen zum three_bit_counter VHDL Code

Zeilen 25 – 27: Deklaration verwendeten Bibliotheken
Wir verwenden für die Schnittstellen nach aussen Signale vom Typ „STD_LOGIC“ und „UNSIGNED“.
Diese Typen sind in der Bibliothek (LIBRARY) des IEEE Standards definiert, und zwar in den Paketen „STD_LOGIC_1164“ und „NUMERIC_STD“.

Zeilen 29 – 37: Definition der Komponente
Die Komponenten-Definition (COMPONENT) könnte auch in der nächst höheren Hierarchiestufe stehen, denn erst dort wird sie zur Instantiierung dieses Moduls benötigt. Da jedoch dieses Modul (three_bit_counter) in mindestens zwei höheren Modulen verwendet wird (VHDL Design und Testbench) ist es immer von Vorteil wenn man die Komponenten-Definition beim Modul selbst behält, und über ein „PACKAGE“ den anderen Modulen zur Verfügung stellt.

Zeilen 52 – 54: Definition verwendeter Bibliotheken
Dies muss für die Entity und Architektur an dieser Stelle wiederholt werden … die Anweisungen in Zeilen 25 bis 27 gelten nur für das PACKAGE.

Zeilen 56 – 62: Definition der ENTITY
Es mag zwar wenig sinnvoll erscheinen, dass man jedes Mal praktisch identisch die COMPONENT und die ENTITY definieren muss, aber in VHDL is es halt so. Man kann es zum Teil mit der Definition einer Funktion in Software vergleichen, wenn der Funktions-Aufruf mit allen notwendigen Parametern nicht nur im .c File definiert ist, sondern nochmals identisch (aber ohne Funktionsinhalt) im .h File.

Zeile 66: Definition der Architektur
Die Architektur kann fast jeden beliebigen Namen tragen. Da man aber zu jeder Entity mehrere Architekturen definieren kann, ist es sinnvoll der Architektur immer einen aussagekräftigen, sinnvollen Namen zu geben.
In diesem Zusammenhang bedeutet der Name „Behavioral“ dass es sich um eine Architektur handelt die sich in allen Aspekten so verhält wie (später) die „richtige“ Implementation, aber dass diese Architektur NICHT dafür vorgesehen ist, synthetisiert und implementiert zu werden (z.B. wegen dem „AFTER 15 ns“).
Eine Architektur die Synthetisiert werden kann, nennt man z.B. „RTL“ oder „Struct“ oder „ALTERA“.

Zeile 68: Signal-Definition
Das interne Signal wird benötigt, weil in VHDL ein Ausgangs-Signal nicht innerhalb des Moduls selbst wieder gelesen werden kann. Deshalb verwenden wir ein internes Signal für den Zähler, und kopieren dessen Wert kontinuierlich (in Zeile 78) auf den Ausgang.

Zeilen 70 – 80: Die Architektur

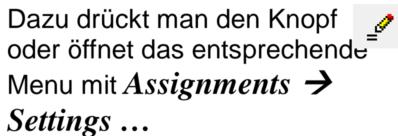
Zeilen 71 – 76: Der registrierte Prozess
Bedingungen (IF) oder Schleifen (LOOP) können nur innerhalb eines Prozesses verwendet werden.
Die „sensitivity list“ in der Prozess-Deklaration definiert auch, auf welche Signale dieser Prozess in der Simulation reagieren soll.

Zeile 74: Die eigentliche Zuweisung
Hier wird tatsächlich gezählt … allerdings jeweils erst mit 15 Nanosekunden Verzögerung. Dies kann eine langsame Schaltung annähern, aber sie verhindert auch, dass diese Architektur exakt synthetisiert werden und in einem FPGA verwendet werden kann.

Zeilen 78: Kopieren des internen Zählerstandes auf den Ausgang
Da das Ausgangssignal „count“ beim Zählen in Zeile 74 zur Verfügung steht, müssen wir ein internes Signal verwenden und dieses hier auf den Ausgang kopieren.

5.3 Einstellungen für ModelSim als Simulator

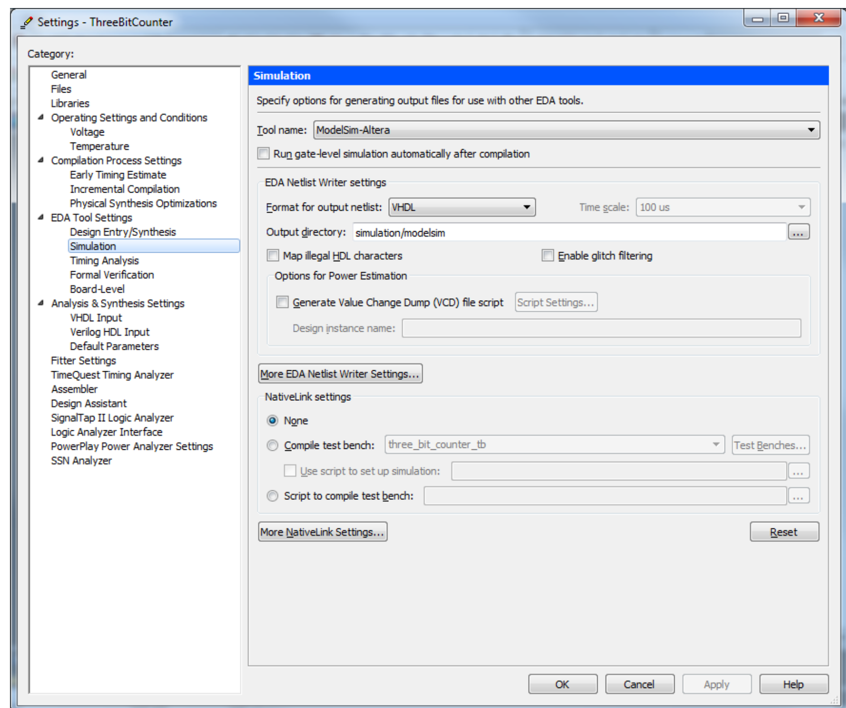
Damit Quartus II weiss, wie die Simulation laufen soll, muss man dies entsprechend definieren:

Dazu drückt man den Knopf  oder öffnet das entsprechende Menu mit ***Assignments → Settings …***

Es erscheint ein neues Fenster, wobei man auf der linken Seite den Punkt „Simulation“ im Bereich „EDA Tool Settings“ anwählen muss. Dafür erscheint dann dieser Abschnitt:

Hier muss man jetzt rechts in der obersten Zeile bei „Tool name“ die Auswahl „ModelSim-Altera“ einstellen.

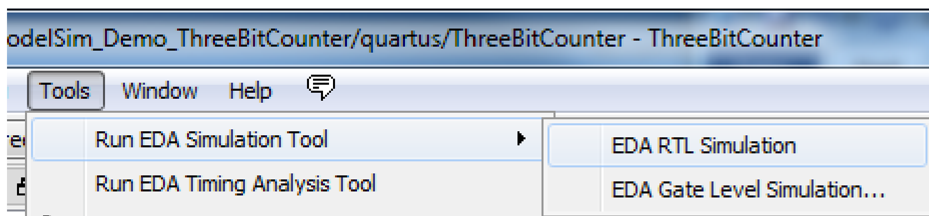
Bei NativeLink settings muss der Knopf für „None“ angewählt sein.



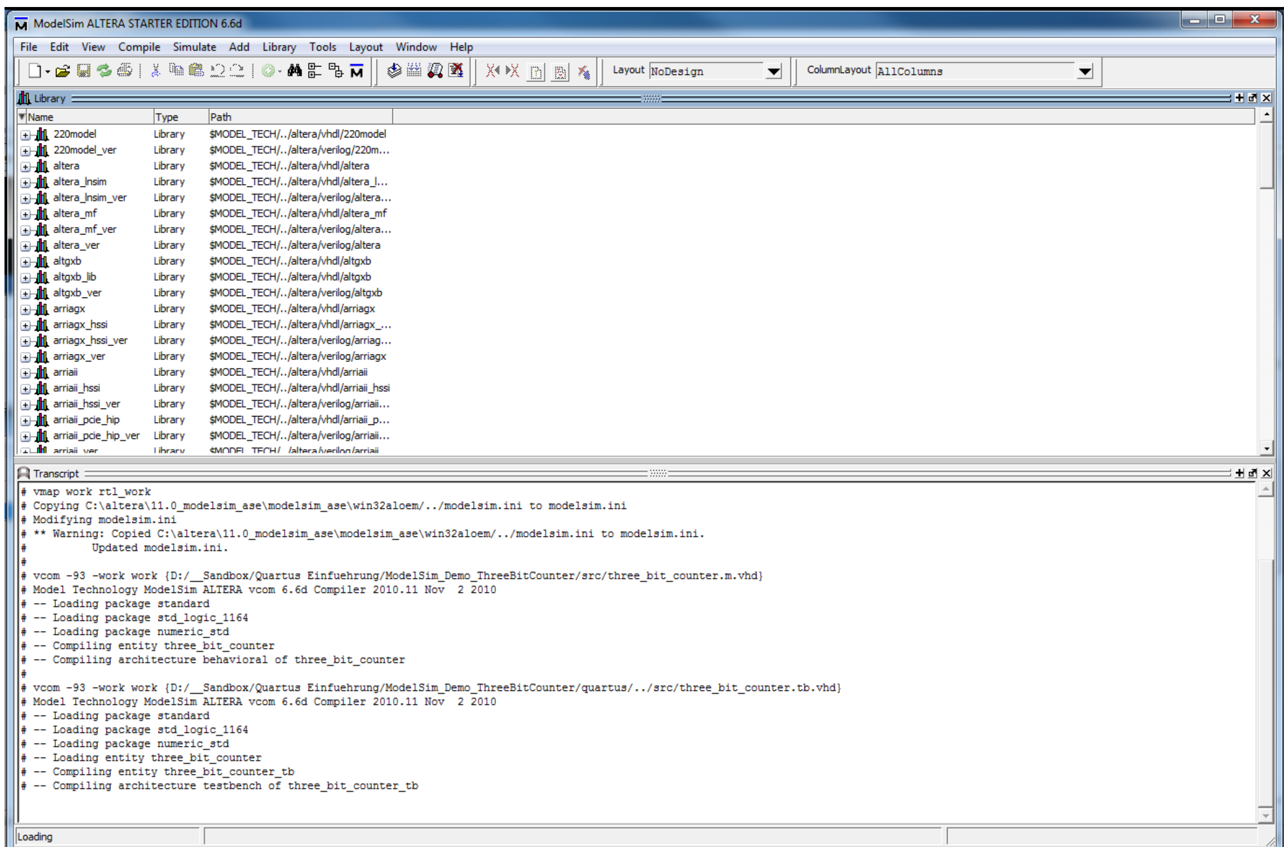
5.4 Starten der Simulation

In Quartus startet man die Simulation mit Menu

Tools → Run EDA Simulation Tool → EDA RTL Simulation

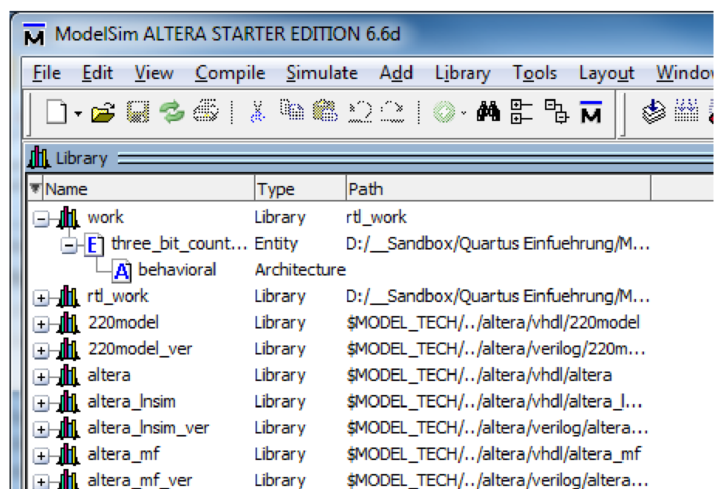


Nach dem ModelSim „Splash-Screen“ erscheint zuerst das Compile-Fenster …



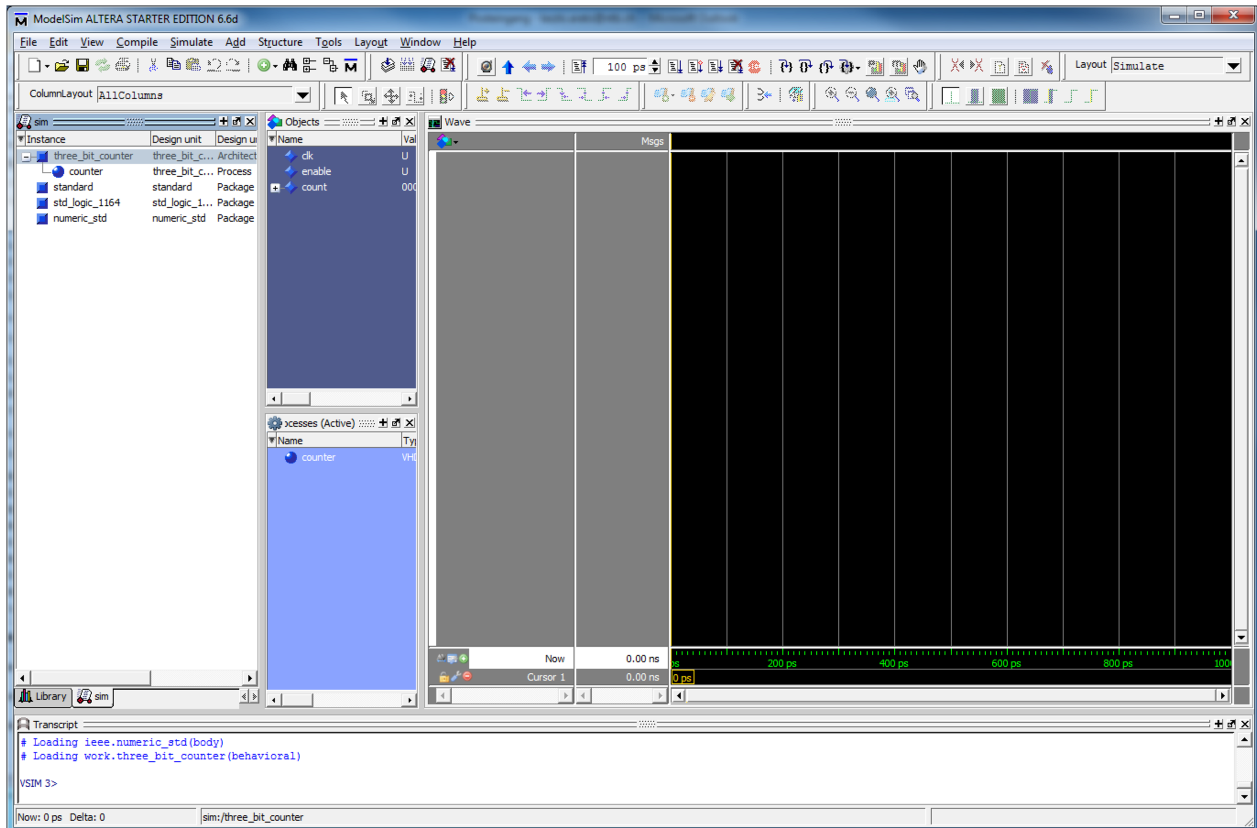
Hier muss man jetzt oben links bei den Bibliotheken zuerst die Library „work“ öffnen (Klick auf das „+“ Zeichen), dann die Entity „three_bit_counter“ ausweiten (Klick auf das „+“ Zeichen).

Um die Simulation für die Architektur dieses Moduls zu starten muss man jetzt mit der Maus zweimal auf die Architektur „behavioral“ klicken …



5.5 Simulieren mit einzeln konfigurierten Signalen

Auf den Doppel-Klick zum Starten der Simulation öffnet sich die Simulation, die etwa so aussehen sollte:



Es sind noch keine Signale definiert, und auch noch keine „Simulationszeit“ vergangen …

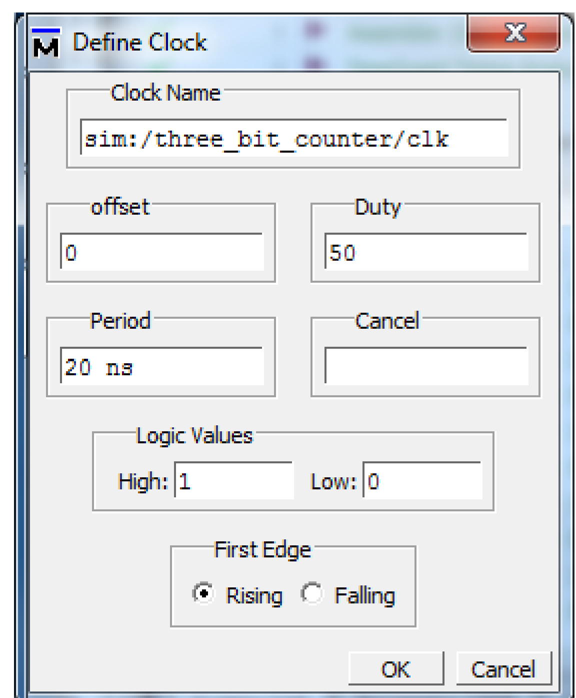
Auswählen der Signale

Im Mittleren Bereich „Objects“ sieht man die Signale des three_bit_counters. Diese kann man gemeinsam oder einzeln anwählen und in das „Wave“ Fenster rechts ziehen.

Anlegen des Taktes

Wählen sie im „Objects“ oder „Wave“ Fenster das Signal „clk“. Mit Rechts-Klick öffnet sich das Menu, wo sie auf „Clock …“ klicken.

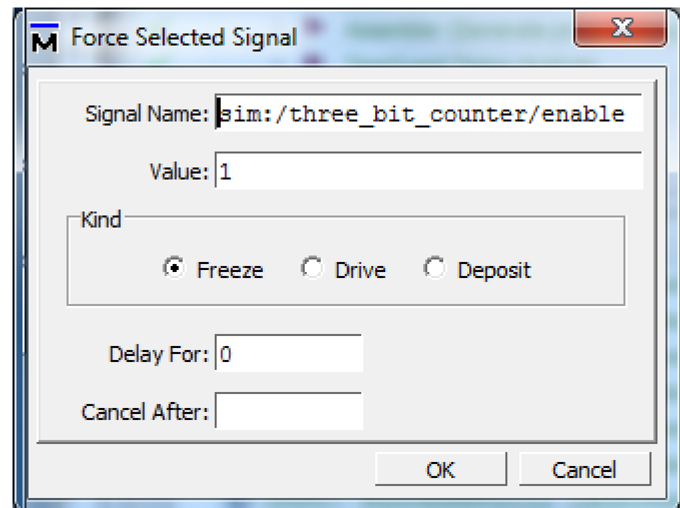
Hier stellt man jetzt die Periode auf „20 ns“ ein, und klickt auf den Knopf „OK“.



Kontrolle des Enable Signals

Wählen Sie im „Objects“ oder „Wave“ Fenster das Signal „enable“, und klicken Sie auf die rechte Maustaste.

Im Menu wählen Sie dann „Force“ und stellen im sich öffnenden Fenster das „Value“ auf „0“ und klicken „ok“.



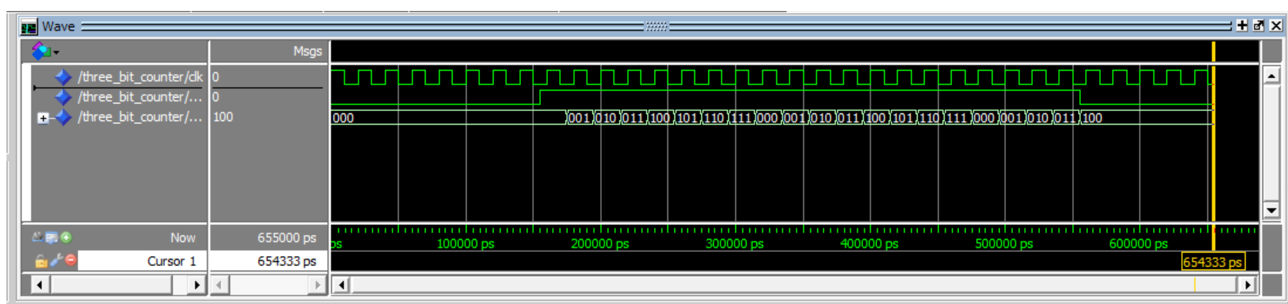
Laufen lassen der Simulation
Schreiben sie ins untere Fenster („Transcript“) den Befehl „run 155 ns“.

Ändern Sie jetzt den Wert des Signals „enable“ auf „1“

Simulieren sie erneut für 400 ns mit „run 400 ns“

Ändern sie das Signal „enable“ wieder auf „0“ und simulieren sie weitere 100 ns.

Am Ende sollten Sie in etwa diese Grafik als Resultat erhalten



Schlussfolgerungen

Es ist grundsätzlich möglich, ein Design durch direkte Kontrolle der Signale zu simulieren … aber …

- es ist mühsam und aufwendig
- die Verwendung von Eingabe-Fenstern ist intuitiv aber nicht sehr effizient
- auch nach kleinen Änderungen oder bei jedem Neustart der Simulation muss man alles wiederholen

5.6 Steuerung der Signale durch ein Skript-File

Wie Sie vielleicht bemerkt haben, erscheint im „Transcript“ Fenster nach jeder Signal-Konfiguration per Maus eine Kommando-Zeile. Wie Sie richtig vermuten, genügen auch diese Befehle um das selbe zu erreichen.

Man kann jetzt diese Befehle in ein geeignetes Text-File kopieren, und dann nacheinander automatisch ausführen lassen.

- Öffnen Sie ein neues Text-File im Verzeichnis ThreeBitCounter\sim mit dem Namen „stimulus.do“ in einem Text-Editor.
- Kopieren Sie die verschiedenen bisherigen Befehle in das File
- Speichern sie dieses Text-File

Dieses File sollte jetzt etwa folgenden Inhalt haben:

```
force -freeze sim:/three_bit_counter/clk 1 0, 0 {10000 ps} -r {20 ns}
force -freeze sim:/three_bit_counter/enable 0 0
run 155 ns
force -freeze sim:/three_bit_counter/enable 1 0
run 400 ns
force -freeze sim:/three_bit_counter/enable 0 0
run 100 ns
```

Wenn Sie jetzt wieder in das “Transcript” Fenster von ModelSim Klicken, können sie dort die folgenden Befehle ausführen:

restart –f	// Die Simulation wird zurückgesetzt, alle Werte werden gelöscht
do ../../../sim/stimulus.do	// Führt die Befehle im Skript-File aus. Der Pfad ist relativ zum ModelSim // Verzeichnis unter „quartus/simulation/modelsim“

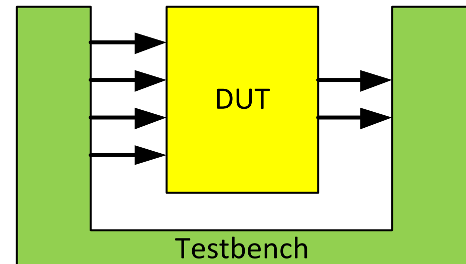
Schlussfolgerungen

Es ist möglich, die Simulation mit einem Skript-File durchzuführen. Vor allem wenn man die Simulation wiederholen will, ist das sehr praktisch … aber …

- es ist mühsam und aufwendig das Simulations-File zu erstellen
- die Verwendung von Eingabe-Fenstern ist intuitiv aber immer noch nicht sehr effizient
- die direkte Verwendung von ModelSim Befehlen ist etwas kryptisch und gewöhnungsbedürftig

6 Three_bit_counter mit einer Testbench

6.1 Die Testbench



Eine Testbench ist ein Modul, welches die zu prüfende Einheit umschliesst, und dadurch sowohl die Signal-Eingänge wie Ausgänge kontrollieren und überwachen kann.

Die zu prüfende Einheit nennt man in der Regel DUT (Device under Test).

Die Testbench wird in der Regel in der gleichen Sprache geschrieben wie das zu simulierende Objekt, weil dann der Simulator nur eine Sprache unterstützen muss und deshalb günstiger und schneller ist. Aber es ist durchaus möglich, die Testbench in Verilog, oder sogar in SystemC oder Java zu schreiben. Ein anderer Vorteil der gleichen Sprache ist natürlich auch, dass dann der Entwickler nur eine Sprache wirklich beherrschen muss.

6.2 Regeln für die Testbench

Im Gegensatz zum DUT muss die Testbench nicht synthetisierbar sein, sie wird immer nur simuliert und wird nicht auf dem FPGA implementiert. Deshalb kann man

- Ohne Bedenken „WAIT FOR" Befehle mit Zeiten bis Milli- oder Pico-Sekundenbereich benützen
- Nach Belieben „LOOP" Schleifen verwenden
- Signale vom Typ Integer und Real verwenden
- Synchrone und asynchrone Prozesse nach Bedarf mischen

6.3 VHDL Testbench für den ThreeBitCounter

Diese Testbench ist spezifisch für den Baustein ThreeBitCounter aus Kapitel 5.1 geschrieben.

```
25    LIBRARY IEEE;
26    USE IEEE.STD_LOGIC_1164.ALL;
27    USE IEEE.NUMERIC_STD.ALL;
28    USE work.three_bit_counter_pkg.ALL;
29
30    ENTITY three_bit_counter_tb IS
31    END ENTITY three_bit_counter_tb;
32
33    ARCHITECTURE Testbench OF three_bit_counter_tb IS
34
35    SIGNAL  sl_clock, sl_enable   : std_logic := '0';
36    SIGNAL  usig3_count           : unsigned(2 DOWNTO 0);
37
38    BEGIN
39        --      ##    Unit Under Test Instantiation
40        u_three_bit_counter : three_bit_counter PORT MAP (
41            clk        => sl_clock,
42            enable     => sl_enable,
43            count      => usig3_count
44        );
45
46        --      ##    TB Clock Process
47        tb_clock_proc : PROCESS
48        BEGIN
49            WHILE (true) LOOP
50                sl_clock <= '1'; WAIT FOR 10 ns;     -- 20 ns cycle-time
51                sl_clock <= '0'; WAIT FOR 10 ns;     --     = 50 MHz clock
52            END LOOP;
53        END PROCESS tb_clock_proc;
54
55        --  ##    TB Enable Signal Generation
56        tb_main_proc : PROCESS
57        BEGIN
58            WAIT FOR 155 ns;
59            sl_enable    <= '1';
60            WAIT FOR 400 ns;
61            sl_enable    <= '0';
62            WAIT FOR 150 ns;
63            ASSERT false REPORT "End of simulation" SEVERITY FAILURE;
64        END PROCESS tb_main_proc;
65
66    END Testbench;
```

6.4 Erklärungen zur Testbench für den ThreeBitCounter

Diese Testbench besteht aus folgenden Teilen:

Zeilen 30 – 31: Deklaration der Entity
Wie jedes VHDL Modul, braucht auch dieses eine Deklaration der Entity. Da es aber keine Signale nach aussen gibt, ist die PORT Liste leer und kann weggelassen werden.

Zeile 33: Architecture
Wie jedes VHDL Modul hat auch die Testbench eine Architektur, eine innere Struktur und Inhalt.

Zeilen 35 – 36: Definition der Signale
Wir benötigen innerhalb der Testbench auch Signale, welche hier definiert werden. In unserem einfachen Fall sind dies nur die Schnittstellen-Signale. Bei komplexeren Testbenches können dies auch die inneren Zustandssignale, Flags, Zähler und ähnliches sein.

Zeile 38: BEGIN
Hier fängt nun endlich die Architektur der Testbench wirklich an.

Zeilen 39 – 44: Instantiierung der DUT
Das zu testende Modul wird hier als hierarchisch tiefer gelegene Komponente instantiiert, und die Signale der Testbench den Signalen des Moduls zugewiesen.

Zeilen 46 – 53: Clock Process
Hier wird ein fortwährendes Taktsignal mit 50 MHz erzeugt. Jeweils 10 ns hoch, dann 10 ns tief, und es fängt endlos wieder von vorne an.

Zeilen 55 – 62: Erzeugung des Enable Signals
Das Enable Signal soll erst nach 155 ns der Simulation eingeschaltet werden, und dann nach weiteren 400 ns wieder ausgeschaltet werden. Dabei erwarten wir, dass der Zähler nur während der Zeit von 155ns bis 555 ns zählt, und anschliessend wieder stabil bei seinem letzten Wert bleibt.

Zeile 63: Abbruch der Simulation
Durch einen ASSERT Befehl der immer getriggert wird (Bedingung ist immer „false“) und der die Stufe „FAILURE“ hat wird die Simulation an dieser Stelle unterbrochen. Ein kleiner Nachteil dieser Art des Beendens der Simulation ist die „Fehlermeldung“ im Report-Fenster von Modelsim. Dafür hat man den Vorteil, dass man keine feste Simulationslänge eingeben muss, sondern die Simulation jederzeit wieder automatisch angepasst wird wenn neue Funktionen getestet werden.

6.5 Simulations-Einstellungen und Definition der Testbench

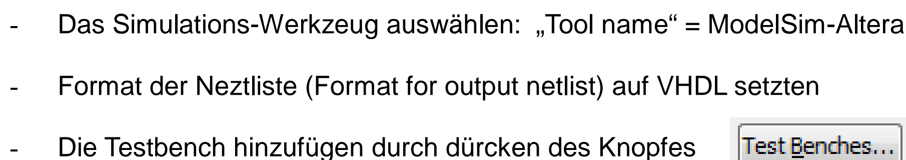
Damit Quartus II weiss, wie die Simulation laufen soll, muss man dies entsprechend definieren:

Dazu drückt man den Knopf oder öffnet das entsprechende Menu mit ***Assignments → Settings …***

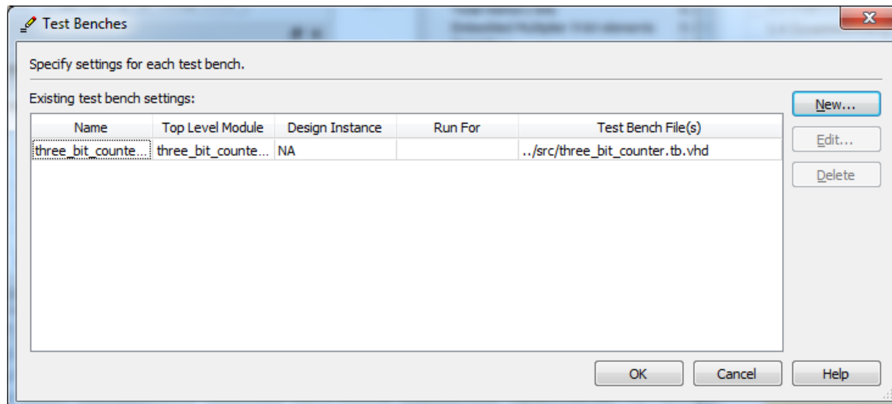
Es erscheint ein neues Fenster, wobei man auf der linken Seite den Punkt „Simulation“ im Bereich „EDA Tool Settings“ anwählen muss. Dafür erscheint dann dieser Abschnitt:

Jetzt muss man hier

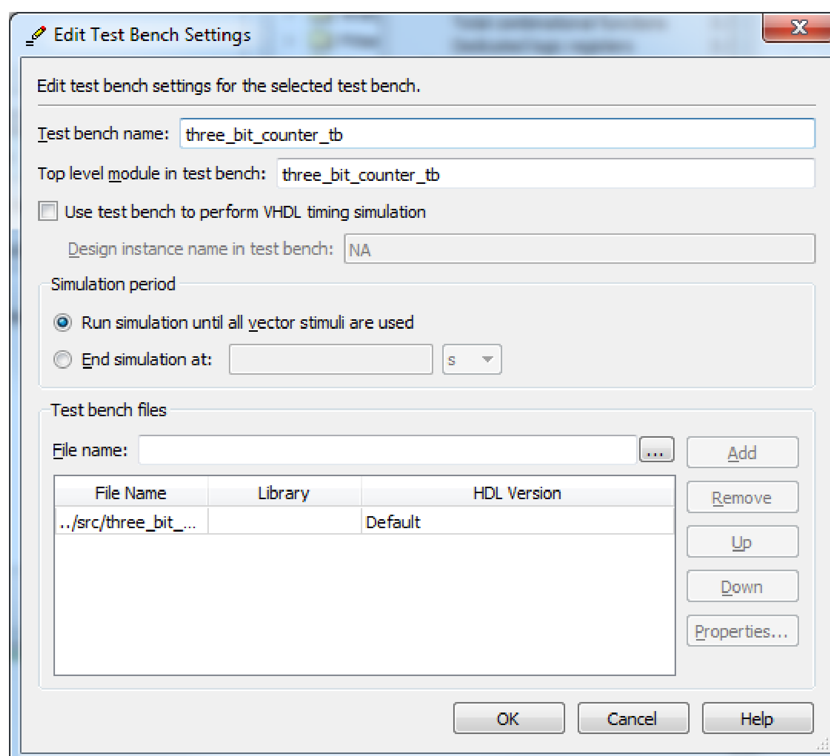
- Das Simulations-Werkzeug auswählen: „Tool name“ = ModelSim-Altera
- Format der Neztliste (Format for output netlist) auf VHDL setzten
- Die Testbench hinzufügen durch dürcken des Knopfes Test Benches…



- Es erscheint das Eingabe Fenster für die Auswahl der Testbenches … es könnte ja auch mehr als nur eine geben …
-



- In diesem neu erscheinenden Fenster mit dem Knopf „New …“ die neue Testbench hinzufügen … Dazu erscheint das Fenster für die Testbench-Einstellungen:



Hier muss man einen Namen für die Testbench geben … dieser Name ist mit Vorteil der gleiche wie der Name der Datei …

Dieser Name ist mit Vorteil auch gleich der Name des Top-Level Moduls in der Testbench.

Da wir die Simulation mit einem ASSERT Befehl kontrolliert beenden, müssen wir bei der Simulations-Länge keine Zeit definieren sondern können einfach „Run simulation until all vector stimuli are used“ wählen.

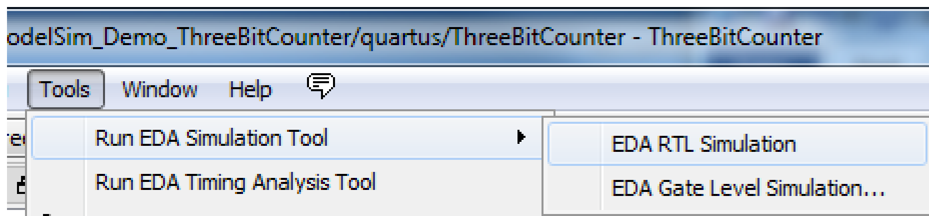
Natürlich muss bei File name“ das Testbench-File ausgewählt werden.

Für die Testbench-Datei (File name) muss man entweder den Pfad und Namen der Datei eingeben, oder sucht dieses im Verzeichniss mit dem Knopf .

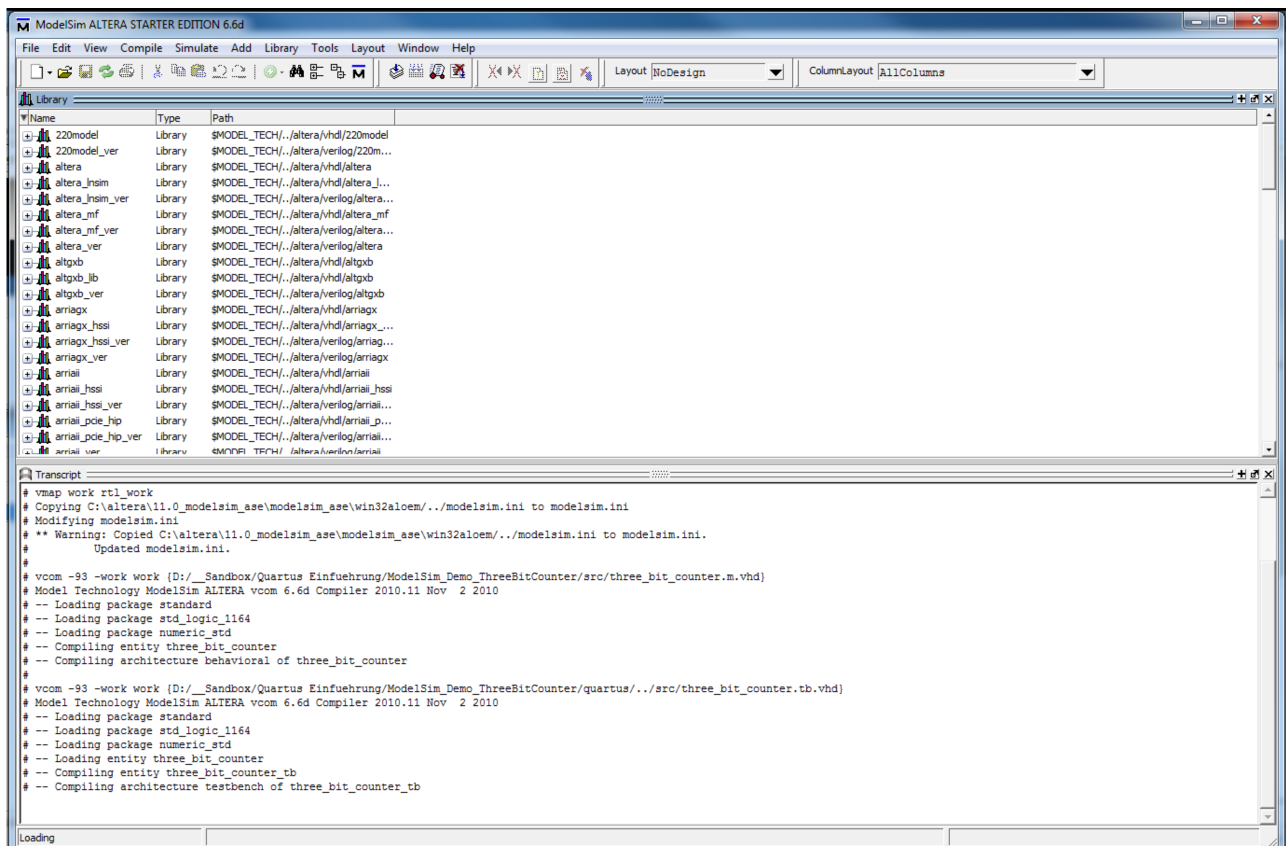
6.6 Aufruf von ModelSim und Starten der Simulation

In Quartus II startet man ModelSim und die Simulation mit dem Menu

Tools → Run EDA Simulation Tool → EDA RTL Simulation



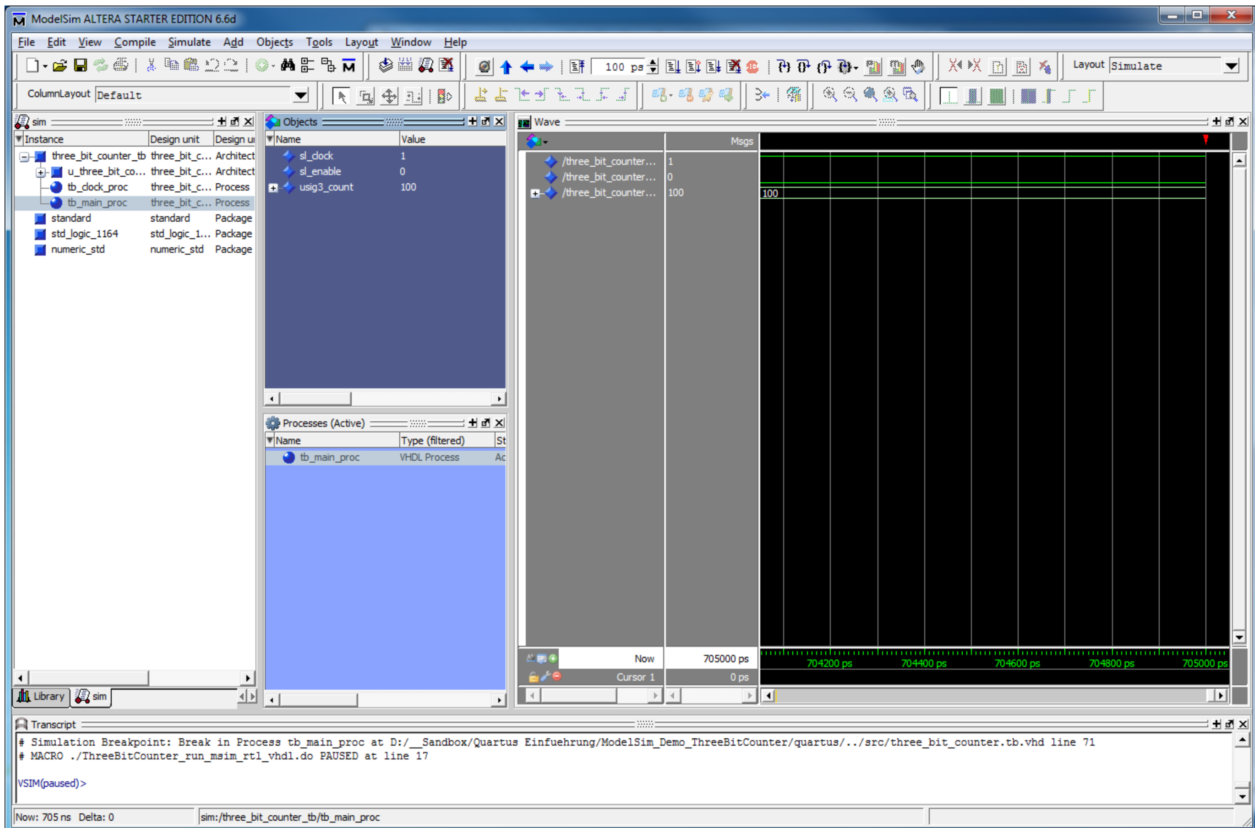
Es erscheint zuerst das Compile-Fenster …



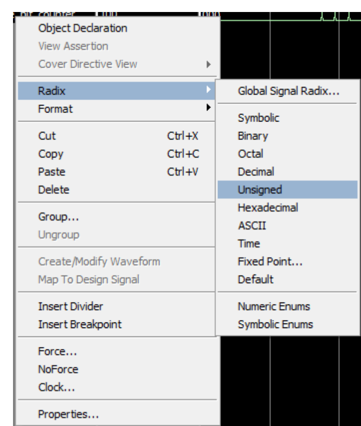
… und anschliessend, wenn es keine Fehler hat, das eigentliche Simulations-Fenster:

Wenn alles richtig kodiert und eingestellt wurde, läuft ModelSim bis zum Ende der Simulation durch, und bleibt dann mit der Meldung „paused“ stehen.

Dabei sollte jetzt das ModelSim Fenster etwa wie folgt aussehen:



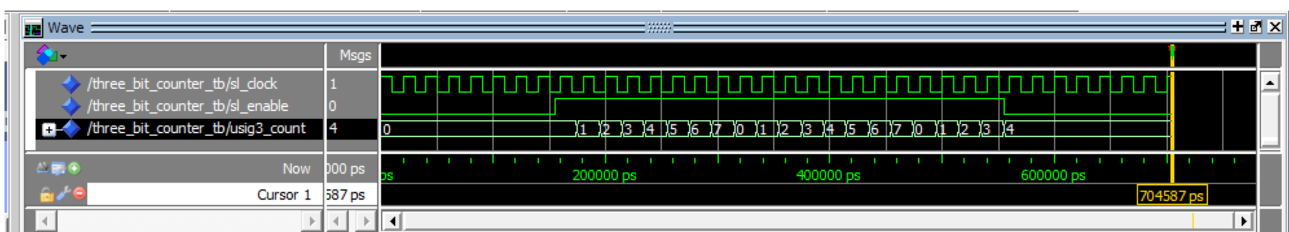
Oben rechts sieht man das Fenster für die Top-Level Signale, mit den Signalnamen im grauen Feld und den Signalen selbst mit schwarzem Hintergrund. Die hellgrüne Zeitskala am unteren Rand zeigt die Zeit von 704‘200 ps bis 705000 ps.



Durch klicken in das Waveform Fenster und Druck auf den Knopf auf die gesamte Simulation gezoomt.

Durch Rechts-Klick auf den Namen des untersten Signals und Auswahl ***Radix → Unsigned*** erhalten wir eine dezimale Darstellung ohne negative Zahlen (statt der lästigen binären Schreibweise).

Mit etwas verschieben der Ränder und anpassen der Breite der Zeilen für die Namen erhält man dann das folgende „Fertige“ Bild: Der Zähler, der nur bei aktivem „Enable“-Signal von 0 bis 7 zählt und dann wieder bei 0 anfängt …



7 Self-checking Testbench : Full Adder

Eine gute Testbench kann nicht nur Signale generieren, sondern kann das Resultat auch selbst überprüfen.

Für dieses Beispiel verwenden wir einen einfachen 4-Bit Addierer mit Carry, der aus 4 einzelnen Volladdierer aufgebaut ist.

7.1 VHDL Source Code

7.1.1 full_add

Dies ist ein simple Full-Adder mit 3 Eingängen und 2 Ausgängen.

Um die Instantiierung zu vereinfachen besitzt dieses Modul sein eigenes PACKAGE und COMPONENT Deklaration … damit man diese nicht bei jeder Verwendung wieder neu schreiben muss.

```
25    LIBRARY IEEE;
26    USE IEEE.STD_LOGIC_1164.ALL;
27
28    PACKAGE full_add_pkg IS
29        COMPONENT full_add IS
30            PORT (
31                u, v, carry_in: IN  STD_LOGIC;
32                sum, carry_out: OUT STD_LOGIC
33            );
34        END COMPONENT full_add;
35    END PACKAGE full_add_pkg;
36
37    ------------------------------------------------------------------------

50    LIBRARY IEEE;
51    IEEE.STD_LOGIC_1164.ALL;
52
53    ENTITY full_add IS
54            PORT (
55                u, v, carry_in: IN  STD_LOGIC;
56                sum, carry_out: OUT STD_LOGIC
57            );
58    END ENTITY full_add;
59
60    -------------------------------------------------------------------------
61
62    ARCHITECTURE behavioral OF full_add IS
63    BEGIN
64        sum           <= u XOR v XOR carry_in;
65        carry_out     <= (u AND v) OR ((u OR v) AND carry_in);
66    END behavioral;
```

7.2 adder4

Dieser 4-Bit Volladdierer verwendet 4 mal das Modul full_add, und verbindet diese nur.
Deshalb nennt man diese Architektur „struct“ oder „structure“, da auf dieser Ebene keine Verknüpfungen oder Bedingungen existieren, sondern nur Verbindungen.

```
25    LIBRARY IEEE;
26    USE IEEE.STD_LOGIC_1164.ALL;
27
28    PACKAGE adder4_pkg IS
29        COMPONENT adder4 IS
30            PORT (
31                a, b    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
32                s       : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
33                cin     : IN  STD_LOGIC;
34                cout    : OUT STD_LOGIC
35            );
36        END COMPONENT adder4;
37    END PACKAGE adder4_pkg;
38
39    ------------------------------------------------------------------------

52    LIBRARY IEEE;
53    USE IEEE.STD_LOGIC_1164.ALL;
54
55    USE work.full_add_pkg.ALL;
56
57    ENTITY adder4 IS
58            PORT (
59                a, b    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
60                s       : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
61                cin     : IN  STD_LOGIC;
62                cout    : OUT STD_LOGIC
63            );
64    END ENTITY adder4;
65
66    ------------------------------------------------------------------------
67
68    ARCHITECTURE structure OF full_add IS
69        SIGNAL   c       : STD_LOGIC_VECTOR(2 DOWNTO 0);
70    BEGIN
71        u_adder_0 : full_add PORT MAP (a(0), b(0), cin,  s(0), c(0));
72        u_adder_1 : full_add PORT MAP (a(1), b(1), c(0), s(1), c(1));
73        u_adder_2 : full_add PORT MAP (a(2), b(2), c(1), s(2), c(2));
74        u_adder_3 : full_add PORT MAP (a(3), b(3), c(2), s(3), cout);
75    END structure;
```

7.3 Einfache selbst-checkende Testbench für adder4

```
25    LIBRARY IEEE;
26    USE IEEE.STD_LOGIC_1164.ALL;
27    USE IEEE.NUMERIC_STD.ALL;
28    USE work.adder4_pkg.ALL;
29
30    ENTITY adder4_tb IS
31    END ENTITY adder4_tb;
32
33    ARCHITECTURE Testbench OF adder4_tb IS
34
35    SIGNAL  usig4_in_a      : UNSIGNED(3 DOWNTO 0)  := (OTHERS => '0');
36    SIGNAL  usig4_in_b      : UNSIGNED(3 DOWNTO 0)  := (OTHERS => '0');
37    SIGNAL  usig1_carry_in  : UNSIGNED(0 DOWNTO 0)  := (OTHERS => '0');
38    SIGNAL  slv4_sum        : STD_LOGIC_VECTOR(3 DOWNTO 0);
39    SIGNAL  carry_out       : STD_LOGIC;
40    SIGNAL  i_count         : INTEGER               := 0;
41
42    BEGIN
43        --  ##  Unit Under Test Instantiation
44        u_dut : adder4 PORT MAP (
45            a          => STD_LOGIC_VECTOR(usig4_in_a),
46            b          => STD_LOGIC_VECTOR(usig4_in_b),
47            s          => slv4_sum,
48            cin        => usig1_carry_in(0),
49            cout       => carry_out
50        );
51
52        --  ##  TB Main Process
53        tb_main_proc : PROCESS
54            VARIABLE vusig5_result      : UNSIGNED(4 DOWNTO 0);
55        BEGIN
56            --  Generate stimulus
57            usig1_carry_in <= NOT usig1_carry_in;
58            IF usig1_carry_in = "0" THEN
59               usig4_in_a <= usig4_in_a + 3;         --  Prime numbers added
60               IF usig4_in_a = "0000" THEN usig4_in_b <= usig4_in_b+7; END IF;
61            END IF;
62
63            WAIT FOR 10 ns;
64            --  Check Result
65            vusig5_result := '0'&usig4_in_a + usig4_in_b + usig1_carry_in;
66
67            ASSERT slv4_sum = STD_LOGIC_VECTOR(vusig5_result(3 DOWNTO 0))
68                REPORT "Sum is wrong" SEVERITY ERROR;
69            ASSERT carry_out = STD_LOGIC(vusig5_result(4))
70                REPORT "Carry is wrong" SEVERITY ERROR;
71
72            i_count <= i_count + 1;
73        END PROCESS tb_main_proc;
74
75        --  ##  End simulation when enough samples are checked
76        ASSERT i_count < 20 REPORT "End of simulation" SEVERITY FAILURE;
77    END Testbench;
```

7.4 Erklärungen zur Testbench

Zeilen 52 – 73: Prozess zur Signalerzeugung und Überprüfung
Wenn sie diesen Code mit dem Code der ersten Testbench vergleichen (Seite 14), stellen Sie vielleicht fest dass hier die LOOP Endlos-Schleife fehlt. Tatsächlich ist es nicht notwendig, eine explizite Schleife innerhalb des Prozesses zu definieren, da dieser Prozess von keinem Signal abhängig ist (leere Sensitivity-Liste) und so ganz automatisch immer durchlaufen wird. Jeder Durchlauf dauert 10 Nanosekunden, und kaum ist er fertig beginnt der Prozess von neuem.

Zeilen 56 – 61: Stimulus-Erzeugung
Hier geht es darum, auf möglichst einfache Art möglichst viele mögliche Zustände zu erzeugen.

- Zuerst wird das Carry-In Bit hin- und her geschaltet.
- Jedes zweite Mal wird dann der Eingang A erhöht.
- Erst wenn der Eingang A wieder auf „0000“ steht, wird Eingang B erhöht.

Für den Fall dass die Eingänge A und B jeweils nur um „1“ erhöht werden, ist es offensichtlich dass alle 512 möglichen Signalzustände erzeugt werden (16 * 16 * 2). Nur muss man dann den Test mindestens 128 + 1 Zyklen lang laufen lassen, bevor alle Bits getestet werden.

Wenn man jedoch zu den Eingänge A und B jeweils eine ungeraden Zahl addiert, dann erreicht man auch alle Zustände, aber nicht mehr in wachsender Reihenfolge, sondern über eine Pseudo-Zufalls-Sequenz. Dadurch werden bereits sehr viel schneller alle Bits in den Test einbezogen.

Zeilen 65 – 70: Überprüfung der Resultate
Jeweils 10 ns nach dem Anlegen der Stimuli wird das Resultat der DUT überprüft. Dabei wird in Zeile 65 das Resultat ausgerechnet, und dann in Zeile 67 für die Summe, und Zeile 69 für das Carry-Bit überprüft.

Die Überprüfung erfolgt hier nicht mit einer IF Anweisung, sondern mit einem ASSERT Befehl. Dabei wird der REPORT und SEVERITY Teil des Befehls wird erst ausgeführt, wenn die ASSERT-Bedingung falsch ist.

Zeile 76: Abbruch der Simulation nach 20 Werten
Nach jeder Überprüfung eines Resultates wird der Zähler i_count um eins erhöht. Sobald 20 Werte überprüft wurde, bricht die Simulation ab. Natürlich könnten es auch viel mehr Werte sein – aber man sollte einen sinnvollen Kompromiss zwischen vollständiger Kontrolle und zeitlicher Effizienz finden. Speziell im Hinblick auf zukünftig grössere und viel umfangreichere Module und Funktionen …

Spezielle Tricks in dieser Testbench

In Zeile 65 wird das Resultat intern ausgerechnet. Dabei sollen zwei 4-Bit Zahlen und eine 1-Bit Zahl (Carry) addiert werden um eine 5-Bit Zahl zu erhalten. Dazu werden zuerst einmal die beiden 4-Bit Input Werte nicht als STD_LOGIC_VECTOR, sondern von Anfang an als UNSIGNED definiert. Bei der Übergabe an das DUT werden sie dann noch zu STD_LOGIC_VECTOR umgeformt.

Das Carry-Bit ist noch etwas hartnäckiger … denn die eingebauten arithmetischen Funktionen erkennen ein einzelnes Bit nicht als Zahl an. Deshalb wird das Carry-Bit als ein UNSIGNED Array mit Länge 1 definiert. Bei der Übergabe an das DUT (in Zeile 48) muss dann explizit spezifiziert werden, dass man nur genau das erste-und-einzige Bit übergeben will. Ohne diese Deklaration mit „(0)“ gäbe es einen Fehler in der Zuweisung von einem Array zu einem Bit … VHDL merkt nicht dass ein Bit-Array mit Länge 1 auch nur genau ein einzelnes Bit ist.

8 Anspruchsvolle Testbench : Arcus Tangens CORDIC

Hier ist noch ein anspruchsvolles Beispiel, welches die Stärken einer Self-Checking Testbench durch alternative Berechnung der Resultate im Simulations-Bereich zeigt.

8.1 VHDL Source Code

Dieser auf dem CORDIC Algorithmus basierende Block zur Berechung des Arcus Tangens eines Winkels aus Ankathete und Gegenkathete basiert auf einer iterativen Näherung benötigt praktisch so viele Takt-Zyklen wie das Resultat dann Bit-Genauigkeit haben soll. Dies ist die Grundlage des CORDIC, und kann auf dem Internet nachgelesen werden.

8.1.1 arctan_cordic.m.vhd

```
55  LIBRARY IEEE;
56  USE IEEE.std_logic_1164.ALL;
57  USE IEEE.numeric_std.ALL;
58
59  PACKAGE arctan_cordic_pkg IS
60      COMPONENT arctan_cordic IS
61          PORT (
62              isl_clock               : IN  std_logic;
63              isl_start               : IN  std_logic;
64              isig12_input_x          : IN  signed (11 DOWNTO 0);
65              isig12_input_y          : IN  signed (11 DOWNTO 0);
66              osl_output_valid        : OUT std_logic;
67              osig12_arctan_output    : OUT signed (11 DOWNTO 0)
68          );
69      END COMPONENT arctan_cordic;
70  END PACKAGE arctan_cordic_pkg;
71
72  --------------------------------------------------------------------------
73
74  LIBRARY IEEE;
75  USE IEEE.std_logic_1164.ALL;
76  USE IEEE.numeric_std.ALL;
77
78  USE work.cordic_rom_pkg.ALL;
79  USE work.barrel_shifter_pkg.ALL;
80
81  ENTITY arctan_cordic IS
82          PORT (
83              isl_clock               : IN  std_logic;
84              isl_start               : IN  std_logic;
85              isig12_input_x          : IN  signed (11 DOWNTO 0);
86              isig12_input_y          : IN  signed (11 DOWNTO 0);
87              osl_output_valid        : OUT std_logic;
88              osig12_arctan_output    : OUT signed (11 DOWNTO 0)
89          );
90  END ENTITY arctan_cordic;
91
92  -------------------------------------------------------------------------
```

```
 94  ARCHITECTURE rtl OF arctan_cordic IS
 95
 96      TYPE t_cordic_reg IS RECORD
 97          sl_start_cordic_d1          : std_logic;
 98          sl_finished                 : std_logic;
 99          usig4_iteration_count       : unsigned(3 DOWNTO 0);
100
101          sig19_cordic_adder_1        : signed(18 DOWNTO 0);
102          sig19_cordic_adder_2        : signed(18 DOWNTO 0);
103          sig19_cordic_adder_3        : signed(18 DOWNTO 0);
104          sl_carry_1                  : std_logic;
105          sl_carry_2                  : std_logic;
106          sl_carry_3                  : std_logic;
107
108          sig12_cordic_output         : signed(11 DOWNTO 0);
109      END RECORD;
110
111      TYPE t_adder_result IS RECORD
112          sig18_sum                   : signed(17 DOWNTO 0);
113          sl_carry                    : std_logic;
114      END RECORD;
115
116      SIGNAL r, r_next                : t_cordic_reg;
117
118      SIGNAL sig12_rom_inv5           : signed(11 DOWNTO 0);
119
120      SIGNAL sig18_input_x            : signed(17 DOWNTO 0);
121      SIGNAL sig18_input_y            : signed(17 DOWNTO 0);
122
123      SIGNAL sig18_barrel_shift_1_out : signed(17 DOWNTO 0);
124      SIGNAL sig18_barrel_shift_2_out : signed(17 DOWNTO 0);
125
126  --------------------------------------------------------------------------
127
128      -- Calculate bit sum using carry from previous step, then carry out
129      FUNCTION add_w_carry (isig18_a, isig18_b: signed(17 DOWNTO 0);
                          isl_carry_in : std_logic) RETURN t_adder_result IS
130          VARIABLE sl_carry           : std_logic;
131          VARIABLE r_result           : t_adder_result;
132      BEGIN
133          sl_carry  := isl_carry_in;
134          FOR i IN 0 TO 17 LOOP
135              r_result.sig18_sum(i)   := isig18_a(i) XOR isig18_b(i)
                                                      XOR sl_carry;
136              sl_carry                := (isig18_a(i) AND isig18_b(i))
                                                       OR (isig18_a(i)
137                                  AND sl_carry) OR (isig18_b(i) AND sl_carry);
138          END LOOP;
139          r_result.sl_carry           := sl_carry;
140          RETURN r_result;
141      END FUNCTION add_w_carry;
142
143  -------------------------------------------------------------------------
```

```
144
145   BEGIN
146
147       --  Resize input from <12.0> to <14.4> format for better accuracy
148       sig18_input_x <=    (17 DOWNTO 16 => isig12_input_x(11))
                              &  isig12_input_x & (3 DOWNTO 0 => isig12_input_x(11));
149           sig18_input_y   <=     (17 DOWNTO 16 => isig12_input_y(11))
                              &  isig12_input_y & (3 DOWNTO 0 => isig12_input_y(11));
150
151       cordic_comb_proc : PROCESS (isl_clock, isl_start, r, r_next,
152                                   isig12_input_x, isig12_input_y,
153                                   sig18_input_x, sig18_input_y,
154                                   sig18_barrel_shift_1_out,
155                                   sig18_barrel_shift_2_out,
156                                   sig12_rom_inv5
157                                  )
158
159           VARIABLE  v : t_cordic_reg;
160
161           VARIABLE  vsig18_adder_1_in_a, vsig18_adder_1_in_b
                                  : signed(17 DOWNTO 0);
162           VARIABLE  vsig18_adder_2_in_a, vsig18_adder_2_in_b
                                  : signed(17 DOWNTO 0);
163           VARIABLE  vsig18_adder_3_in_a, vsig18_adder_3_in_b
                                  : signed(17 DOWNTO 0);
164
165           VARIABLE  vr_adder_result   : t_adder_result;
166
167       BEGIN
168           v := r;                 --    Keep variables stable
169
170           vsig18_adder_1_in_a   := r.sig19_cordic_adder_1(17 DOWNTO 0);
171           vsig18_adder_1_in_b   := r.sig19_cordic_adder_1(17 DOWNTO 0);
172           vsig18_adder_2_in_a   := r.sig19_cordic_adder_2(17 DOWNTO 0);
173           vsig18_adder_2_in_b   := r.sig19_cordic_adder_2(17 DOWNTO 0);
174           vsig18_adder_3_in_a   := r.sig19_cordic_adder_3(17 DOWNTO 0);
175           vsig18_adder_3_in_b   := r.sig19_cordic_adder_3(17 DOWNTO 0);
176
177           v.sl_start_cordic_d1  := isl_start; -- Start on rising edge
178
179
180           --  First cycle in computation
181           IF r.sl_start_cordic_d1 = '0' AND isl_start = '1' THEN
182
183               -- if numerator is equal to +0 or -0 then return directly 0
184               IF (isig12_input_y = B"000000000000"
                                     OR isig12_input_y = B"111111111111") THEN
185                   v.sl_finished := '1';
186                   v.sig12_cordic_output  := (OTHERS => '0');
187
188               -- if donumerator is equal to +0 or -0 then return directly
                      +90 or -90 according to the sign of numerator
189               ELSIF (isig12_input_x = B"000000000000"
                                    OR isig12_input_x = B"111111111111") THEN
190                   v.sl_finished := '1';
```

```
191                     IF (isig12_input_y(11) = '0') THEN  -- 1440 == +90 degrees
192                         v.sig12_cordic_output   := B"010110100000";
193                     ELSE                                -- -1440 == -90 degrees
194                         v.sig12_cordic_output   := B"101001011111";
195                     END IF;
196
197     -- if no extreme case, start 1st iteration of regular cordic processing
198                 ELSE
199                     v.sl_finished := '0';
200                     v.usig4_iteration_count   := (OTHERS => '0');
201
202                     -- Process first round
203                     IF isig12_input_x(11) = '1' THEN      -- sign of x
204                         vsig18_adder_1_in_a          := NOT sig18_input_x + 1;
205                         vsig18_adder_2_in_a          := NOT sig18_input_y + 1;
206                     ELSE
207                         vsig18_adder_1_in_a          := sig18_input_x;
208                         vsig18_adder_2_in_a          := sig18_input_y;
209                     END IF;
210                     v.sl_carry_1 := '0';
211                     v.sl_carry_2 := '0';
212                     v.sl_carry_3 := '0';
213
214                     vsig18_adder_1_in_b := (OTHERS => '0');
215                     vsig18_adder_2_in_b := (OTHERS => '0');
216
217                     vsig18_adder_3_in_a := (OTHERS => '0');
218                     vsig18_adder_3_in_b := (OTHERS => '0');
219
220                 END IF;
221
222
223             --  Regular cordic processing for 2nd and subsequent steps
224             ELSIF r.usig4_iteration_count < 8 THEN
225                 vsig18_adder_1_in_a := r.sig19_cordic_adder_1(17 DOWNTO 0);
226                 vsig18_adder_2_in_a := r.sig19_cordic_adder_2(17 DOWNTO 0);
227                 vsig18_adder_3_in_a := r.sig19_cordic_adder_3(17 DOWNTO 0);
228
229
230             IF r.sig19_cordic_adder_2(17) = '1' THEN
231                 vsig18_adder_1_in_b     := NOT sig18_barrel_shift_1_out;
232                 vsig18_adder_2_in_b     := sig18_barrel_shift_2_out;
233                 vsig18_adder_3_in_b     := NOT ("000000" & sig12_rom_inv5);
234             ELSE
235                 vsig18_adder_1_in_b     := sig18_barrel_shift_1_out;
236                 vsig18_adder_2_in_b     := NOT sig18_barrel_shift_2_out;
237                 vsig18_adder_3_in_b     := "000000" & sig12_rom_inv5;
238             END IF;
239
240             v.usig4_iteration_count     := r.usig4_iteration_count + 1;
241
242             -- Check if the result has been found (num == 0)
243             IF (r.sig19_cordic_adder_2 = -1 OR r.sig19_cordic_adder_2=0) THEN
244                 v.usig4_iteration_count := x"8";
245             END IF;
```

```
248        --      Reached the result
249        ELSE
250
251            v.sig12_cordic_output     := r.sig19_cordic_adder_3(13 DOWNTO 2);
               -- Override sign bit
252            v.sig12_cordic_output(11) := r.sig19_cordic_adder_3(17);
253            v.usig4_iteration_count   := x"0";
254            v.sl_finished              := '1';
255
256        END IF;
257
258        --      Adder instatiation
259        vr_adder_result     := add_w_carry(vsig18_adder_1_in_a,
                                   vsig18_adder_1_in_b, v.sl_carry_1);
260        v.sig19_cordic_adder_1    := vr_adder_result.sig18_sum(17)
                                        & vr_adder_result.sig18_sum;
261        v.sl_carry_1              := vr_adder_result.sl_carry;
262
263        vr_adder_result           := add_w_carry(vsig18_adder_2_in_a,
                                   vsig18_adder_2_in_b, v.sl_carry_2);
264        v.sig19_cordic_adder_2    := vr_adder_result.sig18_sum(17)
                                        & vr_adder_result.sig18_sum;
265        v.sl_carry_2              := vr_adder_result.sl_carry;
266
267        vr_adder_result           := add_w_carry(vsig18_adder_3_in_a,
                                       vsig18_adder_3_in_b, v.sl_carry_3);
268        v.sig19_cordic_adder_3    := vr_adder_result.sig18_sum(17)
                                        & vr_adder_result.sig18_sum;
269        v.sl_carry_3              := vr_adder_result.sl_carry;
270
271        r_next <= v;                       --     Copy variables to signals
272    END PROCESS cordic_comb_proc;
273
274    ------------------------------------------------------------------------
275
276        cordic_reg_proc : PROCESS (isl_clock)
277        BEGIN
278            IF rising_edge(isl_clock) THEN r <= r_next; END IF;
279        END PROCESS cordic_reg_proc;
280
281    ------------------------------------------------------------------------
282
283        --  Output assignments
284        osl_output_valid          <= r.sl_finished;
285        osig12_arctan_output      <= r.sig12_cordic_output;
286
287        u_shifter_1 : barrel_shifter PORT MAP (
288            usig4_shift_value     => r.usig4_iteration_count,
289            isig_input            => r.sig19_cordic_adder_2(17 DOWNTO 0),
290            osig_output           => sig18_barrel_shift_1_out
291        );
292
293        u_shifter_2 : barrel_shifter PORT MAP (
294            usig4_shift_value     => r.usig4_iteration_count,
```

```
295            isig_input            => r.sig19_cordic_adder_1(17 DOWNTO 0),
296            osig_output           => sig18_barrel_shift_2_out
297        );
298
299        u_cordic_rom : cordic_rom PORT MAP (
300            usig4_addr            => r.usig4_iteration_count,
301            sig12_data_out        => sig12_rom_inv5
302        );
303
304    END ARCHITECTURE rtl;
```

8.1.2 barrel_shifter.m.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;

PACKAGE barrel_shifter_pkg IS
      COMPONENT barrel_shifter IS
            GENERIC (N : integer := 18);
            PORT (
                  usig4_shift_value : IN  unsigned(3 DOWNTO 0);
                  isig_input              : IN  signed (N-1 DOWNTO 0);
                  osig_output             : OUT signed (N-1 DOWNTO 0)
            );
      END COMPONENT barrel_shifter;
END PACKAGE barrel_shifter_pkg;


--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY barrel_shifter is
    GENERIC (N : integer := 18);
    PORT (
                  usig4_shift_value : IN  unsigned(3 DOWNTO 0);
                  isig_input              : IN  signed (N-1 DOWNTO 0);
                  osig_output             : OUT signed (N-1 DOWNTO 0)
    );
END barrel_shifter;

--------------------------------------------------------------------------------

ARCHITECTURE rtl OF barrel_shifter IS

    SIGNAL sig_temp_1, sig_temp_2: signed (N-1 DOWNTO 0);

BEGIN
      shift_one : PROCESS (isig_input, usig4_shift_value(0))
      BEGIN
            IF usig4_shift_value(0)='1' THEN      -- shift-by-one
```

```
                  sig_temp_1(N-1) <= isig_input(N-1);
                  sig_temp_1(N-2 DOWNTO 0) <= isig_input(N-1 DOWNTO 1);
            ELSE
                  sig_temp_1(N-1 DOWNTO 0) <= isig_input(N-1 DOWNTO 0);
            END IF;
      END PROCESS;

    shift_two : PROCESS (sig_temp_1, usig4_shift_value(1))
    BEGIN
        IF usig4_shift_value(1)='1' THEN     -- shift-by-two
            sig_temp_2(N-1) <= sig_temp_1(N-1);
            sig_temp_2(N-2) <= sig_temp_1(N-1);
            sig_temp_2(N-3 DOWNTO 0) <= sig_temp_1(N-1 DOWNTO 2);
        ELSE
            sig_temp_2(N-1 DOWNTO 0) <= sig_temp_1(N-1 DOWNTO 0);
        END IF;
    END PROCESS;

    shift_four : PROCESS (sig_temp_2, usig4_shift_value(2))
    BEGIN
        IF usig4_shift_value(2)='1' THEN     -- shift-by-four
            osig_output(N-1) <= sig_temp_2(N-1);
            osig_output(N-2) <= sig_temp_2(N-1);
            osig_output(N-3) <= sig_temp_2(N-1);
            osig_output(N-4) <= sig_temp_2(N-1);
            osig_output(N-5 DOWNTO 0) <= sig_temp_2(N-1 DOWNTO 4);
        ELSE
            osig_output(N-1 DOWNTO 0) <= sig_temp_2(N-1 DOWNTO 0);
        END IF;
    END PROCESS;

END ARCHITECTURE rtl;
```

8.1.3 cordic_rom.m.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE IEEE.numeric_std.ALL;

PACKAGE cordic_rom_pkg IS
      COMPONENT cordic_rom IS
            PORT (
                  usig4_addr         : IN  unsigned(3 downto 0);
                  sig12_data_out     : OUT signed(11 DOWNTO 0)
        );
      END COMPONENT cordic_rom;
END PACKAGE cordic_rom_pkg;

--------------------------------------------------------------------------------
-------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
```

```
USE IEEE.numeric_std.ALL;

ENTITY cordic_rom IS
            PORT (
                  usig4_addr         : IN  unsigned(3 downto 0);
                  sig12_data_out     : OUT signed(11 DOWNTO 0)
            );
END ENTITY cordic_rom;

--------------------------------------------------------------------------------
-------------------

ARCHITECTURE structural OF cordic_rom IS

      CONSTANT DIMROM: natural := 8;
      CONSTANT DIMWORD: natural := 12;
      TYPE   ROM_IMAGE IS ARRAY (integer RANGE 0 TO DIMROM-1) OF signed(DIMWORD-
1 DOWNTO 0);

      CONSTANT ROM : ROM_IMAGE := ( -- INTEGER value -- FRACTIONAL value
            0 => x"B40",                 -- 2880                 -- 45
            1 => x"6A4",                 -- 1700                     -- 26,562
            2 => x"382",                       -- 898                        --
14,031
            3 => x"1C8",                       -- 456                   -- 7,125
            4 => x"0E5",                       -- 229                     -- 3,578
            5 => x"073",                       -- 115                        --
      1,796
            6 => x"039",                       -- 57                      -- 0,890
            7 => x"01D"                        -- 29              -- 0,453
      );

BEGIN
    sig12_data_out <= ROM (to_integer(usig4_addr(2 DOWNTO 0)));
END ARCHITECTURE structural;
```

8.2 arctan_cordic.tb.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.math_real.ALL; -- for UNIFORM, TRUNC

USE work.arctan_cordic_pkg.ALL;

ENTITY arctan_cordic_tb IS
END ENTITY arctan_cordic_tb;

ARCHITECTURE behavioral OF arctan_cordic_tb IS


      TYPE   t_tb_result IS (GOOD, ERROR);

      SIGNAL tb_result                   : t_tb_result := GOOD;
```

```
      SIGNAL i_denumerator                  : integer := 5;
      SIGNAL i_numerator                          : integer := 5;

      SIGNAL real_arc_tan                         : real;
      SIGNAL real_arc_tan_rtl_out   : real;
      SIGNAL real_diff_of_rtl               : real;

      SIGNAL sig12_input_x                  : signed (11 downto 0); -- WRITE HERE
THE VALUE FOR DEN
      SIGNAL sig12_input_y                  : signed (11 downto 0); -- WRITE HERE
THE VALUE FOR NUM
      SIGNAL sig12_output_z                 : signed (11 downto 0);

      SIGNAL sl_output_valid                : std_logic;
      SIGNAL sl_output_valid_d1             : std_logic;
      SIGNAL sl_req_sample                  : std_logic := '0';

      SIGNAL sl_clock                             : std_logic := '0';
      SIGNAL sl_reset, sl_reset_d1  : std_logic := '0';

BEGIN

      --    ##    Instantiate Device Under Test
      --    ##
      --    ################################
    my_rtl_cordic : arctan_cordic PORT MAP (
            isl_clock                       => sl_clock,
            isl_start                       => sl_req_sample,
            isig12_input_x                  => sig12_input_x,
            isig12_input_y                  => sig12_input_y,
            osl_output_valid          => sl_output_valid,
            osig12_arctan_output      => sig12_output_z
      );
      real_arc_tan_rtl_out          <=
real(to_integer(signed(sig12_output_z)))/16.0;


      --    ##    sl_clock and sl_reset SIGNALs
      --    ################################
      sl_clock    <= NOT sl_clock after 100 ns; -- 50 MHz
      sl_reset    <= '1' after 200 ns,'0' after 600 ns; --, '1' after 9us, '0'
after 9.3us ;


      --    ##    Random Stimulus Generation
      --    ##
      --    ################################
      random_stim_gen_proc : PROCESS (sl_clock) --sl_reset, egress_valid)
            VARIABLE seed1    : positive := 2564;           -- Seed values for
random generator
            VARIABLE seed2    : positive := 6542;           -- Seed values for
random generator
            VARIABLE rand: real;                            -- Random real-number
value in range 0 to 1.0
            VARIABLE int_rand_x : integer;                  -- Initialise seed1,
```

```
seed2 if you want -
            VARIABLE int_rand_y : integer;                  -- otherwise they're
initialised to 1 by default

      BEGIN
            IF rising_edge(sl_clock) THEN
                  sl_reset_d1                         <= sl_reset;
                  sl_output_valid_d1                  <= sl_output_valid;

                  --    Act upon falling edge of sl_reset, or rising edge of
output_valid
                  IF (sl_reset = '0' AND sl_reset_d1 = '1')
                  OR (sl_output_valid = '1' AND sl_output_valid_d1 = '0') THEN

                        UNIFORM(seed1, seed2, rand);
      -- generate random value, range 0 .. 1
                        int_rand_x := INTEGER(TRUNC(rand*4096.0-2048.0));    --
convert to integer, range 0 - 4095
                        i_denumerator     <= int_rand_x;
                        sig12_input_x           <= to_signed(int_rand_x, 12); --
convert integer to std_logic_vector

                        UNIFORM(seed1, seed2, rand);
      -- generate random value, range 0 .. 1
                        int_rand_y := INTEGER(TRUNC(rand*4096.0-2048.0));    --
convert to integer, range 0 - 4095
                        i_numerator <= int_rand_y;
                        sig12_input_y           <= to_signed(int_rand_y, 12); --
convert integer to std_logic_vector

                        sl_req_sample <= '1';
                  ELSE
                        real_arc_tan      <=
arctan(real(i_numerator)/real(i_denumerator)) * 180.0 / 3.1415;
                        sl_req_sample <= '0';
                  END IF;

            END IF;
      END PROCESS random_stim_gen_proc;


      --    ##    Self-Testing the results
      --    ##
      --    ################################
      check_result_proc : PROCESS (sl_clock)
      BEGIN
            IF rising_edge(sl_clock) THEN
                  IF (sl_output_valid = '1' AND sl_output_valid_d1 = '0')THEN
                        --    Check for correct result
                        --    Even though the output is 12 bits, the current
cordic implementation does only 8 rounds.
                        --    Therefore the accuracy is only +/- 90°/128 = +/-
0.352
                        real_diff_of_rtl <= abs(real_arc_tan -
real_arc_tan_rtl_out);
                        IF abs(real_arc_tan - real_arc_tan_rtl_out) > 0.52 THEN
```

```
                              tb_result   <= ERROR;
                        END IF;
                  END IF;
            END IF;
      END PROCESS check_result_proc;

END ARCHITECTURE behavioral;
```

8.3 ModelSim Command File arctan_cordic_rtl_vhdl.do

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

vcom -93 -work work {../../../src/barrel_shifter.m.vhd}
vcom -93 -work work {../../../src/cordic_rom.m.vhd}
vcom -93 -work work {../../../src/arctan_cordic.m.vhd}

vcom -93 -work work {../../../sim/arctan_cordic.tb.vhd}

vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim
        -L cycloneive -L rtl_work -L work -voptargs="+acc" arctan_cordic_tb

do ../../../sim/wave.do

view structure
view signals
run 40 us
wave zoom full
```

8.4 ModelSim Wave Command File wave.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -divider Input
add wave -noupdate /arctan_cordic_tb/i_denumerator
add wave -noupdate /arctan_cordic_tb/i_numerator
add wave -noupdate -divider Output
add wave -noupdate /arctan_cordic_tb/real_arc_tan
add wave -noupdate /arctan_cordic_tb/real_arc_tan_rtl_out
add wave -noupdate /arctan_cordic_tb/real_diff_of_rtl
add wave -noupdate -divider Result
add wave -noupdate /arctan_cordic_tb/tb_result
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {3177267 ps} 0}
configure wave -namecolwidth 321
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {0 ps} {42 us}
```