

Version 0.5

Verteiler:

Name (alphab.)	Abteilung	Ort
Laszlo Arato	EMS	NTB, Buchs
Dr. Urs Graf	INF	NTB, Buchs

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

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	Ein	leitung	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	Ins	tallation	5
	2.1	Unterschiede "Altera Edition" und "Altera Starter Edition"	5
	2.2	Download von ModelSim-Altera	5
	2.3	Installation	5
3	Liz	enzierung	5
4	Ein	stellungen in Quartus für Modelsim	6
5	Ers	te 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	4.0
	Node	elSim // Verzeichnis unter "quartus/simulation/modelsim"	13
6	Mode Thr	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench	13 . 14
6	Thr 6.1	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench	13 . 14 14
6	Thr 6.1 6.2	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench	13 . 14 14 14
6	Thr 6.1 6.2 6.3	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter	13 14 14 14 15
6	Thr 6.1 6.2 6.3 6.4	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter	13 14 14 14 15 15
6	Mode Thr 6.1 6.2 6.3 6.4 6.5	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench	13 14 14 15 15 15
6	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench Aufruf von ModelSim und Starten der Simulation	13 14 14 15 15 17 19
6	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench. Die Testbench. Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench Aufruf von ModelSim und Starten der Simulation	13 14 14 15 15 15 17 19 21
7	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1	elSim // Verzeichnis unter "quartus/simulation/modelsim"	13 14 14 15 15 17 19 21
7	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.1	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench Aufruf von ModelSim und Starten der Simulation f-checking Testbench : Full Adder VHDL Source Code 1 full_add	13 14 14 15 15 15 17 19 21 21
6 7	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.1 7.2	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench Aufruf von ModelSim und Starten der Simulation f-checking Testbench : Full Adder VHDL Source Code 1 full_add	13 14 14 15 15 17 19 21 21 22
7	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.1 7.2 7.3	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench Aufruf von ModelSim und Starten der Simulation f-checking Testbench : Full Adder VHDL Source Code 1 full_add adder4 Einfache selbst-checkende Testbench für adder4	13 14 14 15 15 15 17 21 21 22 23
7	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.2 7.3 7.4	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench Aufruf von ModelSim und Starten der Simulation f-checking Testbench : Full Adder VHDL Source Code 1 full_add adder4 Einfache selbst-checkende Testbench für adder4 Erklärungen zur Testbench	13 14 14 15 15 15 17 19 21 21 22 23 24
6 7 8	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.2 7.3 7.4 7.4 Ans	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench Die Testbench Regeln für die Testbench VHDL Testbench für den ThreeBitCounter Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench Aufruf von ModelSim und Starten der Simulation f-checking Testbench : Full Adder VHDL Source Code 1 full_add adder4 Einfache selbst-checkende Testbench für adder4 Erklärungen zur Testbench spruchsvolle Testbench : Arcus Tangens CORDIC	13 14 14 15 15 15 17 19 21 21 21 22 23 24 25
6 7 8	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.1 7.2 7.3 7.4 Ans 8.1	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench. Die Testbench. Regeln für die Testbench. VHDL Testbench für den ThreeBitCounter. Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench. Aufruf von ModelSim und Starten der Simulation f-checking Testbench : Full Adder. VHDL Source Code .1 full_add adder4 Einfache selbst-checkende Testbench für adder4. Erklärungen zur Testbench : Arcus Tangens CORDIC VHDL Source Code	13 14 14 15 15 15 17 19 21 21 21 22 23 24 25
6 7 8	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.2 7.3 7.4 Ans 8.1 8.1	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench	13 14 14 15 15 15 17 19 21 21 21 22 23 24 25 25
6 7 8	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.1 7.2 7.3 7.4 Ans 8.1 8.1 8.1 8.1 8.1	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench. Die Testbench. Regeln für die Testbench VHDL Testbench für den ThreeBitCounter. Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench. Aufruf von ModelSim und Starten der Simulation if-checking Testbench : Full Adder. VHDL Source Code .1 full_add adder4 Einfache selbst-checkende Testbench für adder4. Erklärungen zur Testbench : Arcus Tangens CORDIC VHDL Source Code .1 attra Einfache selbst-checkende Testbench für adder4. Erklärungen zur Testbench . Arcus Tangens CORDIC VHDL Source Code .1 arctan_cordic.m.vhd. .2 barrel_shifter.m.vhd	13 14 14 15 15 15 17 21 21 21 21 23 24 25 25 30 31
6 7 8	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.2 7.3 7.4 Ans 8.1 8.1 8.1 8.2	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench	13 14 14 15 15 15 15 15 15 15 21 21 21 21 21 21 23 24 25 30 31 32
6 7 8	Mode Thr 6.1 6.2 6.3 6.4 6.5 6.6 Sel 7.1 7.2 7.3 7.4 Ans 8.1 8.1 8.1 8.1	elSim // Verzeichnis unter "quartus/simulation/modelsim" ree_bit_counter mit einer Testbench. Die Testbench. Regeln für die Testbench VHDL Testbench für den ThreeBitCounter. Erklärungen zur Testbench für den ThreeBitCounter Simulations-Einstellungen und Definition der Testbench. Aufruf von ModelSim und Starten der Simulation f-checking Testbench : Full Adder. VHDL Source Code .1 full_add. adder4 Einfache selbst-checkende Testbench für adder4. Erklärungen zur Testbench Spruchsvolle Testbench : Arcus Tangens CORDIC VHDL Source Code .1 arctan_cordic.m.vhd .2 barrel_shifter.m.vhd .3 cordic_rom.m.vhd	13 14 14 15 15 15 17 21 21 21 21 23 24 25 30 31



8.4	ModelSim	Wave Command File	wave.do
-----	----------	-------------------	---------



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/gts-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 <u>X:\Unterricht\arato\Altera_Source\11.0_modelsim_ae_windows</u> 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* \rightarrow *Options* ... \rightarrow *EDA Tool Options*

4 General	EDA Tool Options		
EDA Tool Options Fonts	Specify the location	of the tool executable for each third-party EDA	tool:
Headers & Footers Settings	EDA Tool	Location of Executable	
Libraries	LeonardoSpectrum		
License Setup	Precision Synthesis		
Preferred Text Editor Processing	Synplify		
Tooltip Settings	Synplify Pro		
Messages	Active-HDL		
Colors	Riviera-PRO		
Fonts	ModelSim		
	QuestaSim		
	ModelSim-Altera	C:\altera\11\modelsim_ae\win32aloem	
	NCSim	T)
	Use NativeLink w	ith a Synplify/Synplify Pro node-locked license	Help

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;
     USE IEEE.NUMERIC STD.ALL;
27
28
29
     PACKAGE three_bit_counter_pkg IS
30
         COMPONENT three_bit_counter IS
31
             PORT (
                        : IN STD_LOGIC;
: IN STD_LOGIC;
: OUT UNCT:
32
                 clk
33
                 enable
                           : OUT UNSIGNED (2 DOWNTO 0)
34
                 count
35
             );
         END COMPONENT three_bit_counter;
36
     END PACKAGE three_bit_counter_pkg;
37
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
                clk : IN STD_LOGIC;
enable : IN STD_LOGIC;
count : OUT INGTON
57
             PORT (
58
59
                           : OUT UNSIGNED (2 DOWNTO 0)
60
61
             );
     END ENTITY three_bit_counter;
62
63
64
     _____
65
     ARCHITECTURE Behavioral OF three_bit_counter IS
66
67
68
     SIGNAL internal_count : UNSIGNED (2 DOWNTO 0) := "000";
69
70
     BEGIN
         counter: PROCESS (clk, enable)
71
72
         BEGIN
             IF rising_edge(clk) AND enable ='1' THEN
73
74
                internal_count <= internal_count + 1 AFTER 15 ns;</pre>
75
             END IF;
76
         END PROCESS counter;
77
78
         Count <= internal_count;</pre>
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* \rightarrow *Run EDA Simulation Tool* \rightarrow *EDA RTL Simulation*

odel:	odelSim_Demo_ThreeBitCounter/quartus/ThreeBitCounter - ThreeBitCounter				
Т	Tools Window Help 🛡				
e	Run EDA Simulation Tool	EDA RTL Simulation			
ć	Run EDA Timing Analysis Tool EDA Gate Level Simulation				



Nach dem ModelSim "Splash-Screen" erscheint zuerst das Compile-Fenster ...

ModelSim ALTERA STA	ARTER EDIT	TION 6.6d	
File Edit View Com	pile Simu	ulate Add Library Tools Layout Window Help	
□ □ • 📾 🖾 🖉 Ū	X 🗓 🖌	総合合 🕹 監 電 🖬 💩 🏙 💯 🔯 😾 🖄 🗁 🕹 🐛 🕹 Layout Montestram 🚽 🛛 ColumnLayout [allColumns 🗨	
Library	1		
* Name	Type	Path	_
E 1 220model	Library	\$MODEL_TECH//altera/vhdl/220model	
220model_ver	Library	\$MODEL_TECH/altera/verilog/220m	
+ altera	Library	\$MODEL_LECH(altera/vhd/altera	
altera_insim	Library	WODEL_LECH././altera/vnoi/altera	
altera_msim_ver	Library		
altera mf ver	Library	WODEL_ECH	
+	Library	WODEL TECH, Jahra Jvenigo altera	
	Library	SMODEL TECH/, /altera/vhd/altoxb	
+- II altoxb lib	Library	SMODEL_TECH//altera/vhdl/altaxb	
+ altgxb_ver	Library	\$MODE_TECH//altera/verilog/altqxb	
+ arriagx	Library	\$MODEL_TECH//altera/vhdl/arriagx	
	Library	\$MODEL_TECH//altera/vhdl/arriagx	
	Library	\$MODEL_TECH//altera/verilog/arriag	
	Library	\$MODEL_TECH//altera/verilog/arriagx	
🖅 🕂 arriali	Library	\$MODEL_TECH//altera/vhdl/arriaii	
	Library	\$MODEL_TECH//altera/vhdl/arriaii_hssi	
	Library	\$MODEL_TECH//altera/verilog/arriaii	
	Library	\$MODEL_TECH//altera/vhd/arriai_p	
• arriai_pcie_hip_ver	Library	\$MODEL_TECH(altera/verilog/arriai	
Tallarnai ver	Librariv	GWENDEL LECTAL Dathers Avening Dariba	
A Transcript			
<pre># vmap work rtl_wor</pre>	k		<u>^</u>
# Copying C:\altera	\11.0_mo	odelsim_ase\modelsim_ase\win32aloem//modelsim.ini to modelsim.ini	
# ** Warning: Copie	d C:\alt	tera\11.0 modelsim ase\modelsim ase\win32aloem//modelsim.ini to modelsim.ini.	
# Updated :	modelsim	m.ini.	
+			
# vcom -93 -work wo	rk {D:/_	_Sandbox/Quartus Einfuehrung/ModelSim_Demo_ThreeBitCounter/src/three_bit_counter.m.vhd}	
# Model Technology	ModelSim	m ALIERA voom 6.6d Compiler 2010.11 Nov 2 2010	
# Loading packag	e std lo	ndi 1164	
# Loading packag	e numeri	ic_std	
# Compiling enti	ty three	e_bit_counter	
# Compiling arch	itecture	e behavioral of three_bit_counter	
# vcom -93 -work wo	rk {D:/	Sandbox/Quartus Einfuchrung/ModelSim Demo ThreeBitCounter/guartus//src/three bit counter.tb.vhd}	
# Model Technology	ModelSim	m ALTERA vcom 6.6d Compiler 2010.11 Nov 2 2010	
# Loading packag	e standa	ard	
# Loading packag	e std_lo	ogie_1164	
# Loading packag	three b		
# Compiling enti	tv three	e bit counter th	
# Compiling arch	itecture	e testbench of three_bit_counter_tb	
			_
<u> </u>			-
Loading			1.

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 ...

ModelSim ALTERA STARTER EDITION 6.6d				
<u>File Edit View Comp</u>	ile <u>S</u> imula	te A <u>d</u> d L <u>i</u> brary T <u>o</u> ols Layo <u>u</u> t <u>W</u> indo		
🗋 • 🗃 🖬 🛸 🚳	X 🖻 🍘	≙≙∣⊘-МЕЪЫ ⊗≝		
👖 Library 🚃				
₹ Name	Туре	Path		
□- / work	Library	rtl_work		
E three_bit_count	Entity	D:/Sandbox/Quartus Einfuehrung/M		
A behavioral	Architecture	1		
	Library	D:/Sandbox/Quartus Einfuehrung/M		
+− 1 220model	Library	\$MODEL_TECH//altera/vhdl/220model		
⊕ 220model_ver	Library	\$MODEL_TECH//altera/verilog/220m		
	Library	\$MODEL_TECH//altera/vhdl/altera		
	Library	\$MODEL_TECH//altera/vhdl/altera_l		
	Library	\$MODEL_TECH//altera/verilog/altera		
	Library	\$MODEL_TECH//altera/vhdl/altera_mf		
	Library	\$MODEL_TECH//altera/verilog/altera		



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".

M Define Clock
Clock Name
sim:/three_bit_counter/clk
offset Duty
0 50
Period Cancel
20 ns
Logic Values
High: 1 Low: 0
First Edge
OK Cancel



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".

Force Selected Signal
Signal Name: sim:/three_bit_counter/enable
Value: 1
Kind
Delay For: 0
Cancel After:
OK Cancel

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

	= + @ ×
ر المحصولات	
ر المحصول	
0000 ps	
654333	3 ps
	10000 ps (55433

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;
29
30
      ENTITY three_bit_counter_tb IS
31
      END ENTITY three_bit_counter_tb;
32
     ARCHITECTURE Testbench OF three_bit_counter_tb IS
33
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;</pre>
                                                         -- 20 ns cycle-time
                  sl clock <= '0'; WAIT FOR 10 ns;</pre>
                                                             = 50 MHz clock
51
                                                         ---
52
              END LOOP;
53
          END PROCESS tb_clock_proc;
54
55
          -- ##
                    TB Enable Signal Generation
          tb_main_proc : PROCESS
56
          BEGIN
57
              WAIT FOR 155 ns;
58
              sl_enable <= '1';</pre>
59
60
              WAIT FOR 400 ns;
                         <= '0';
61
              sl_enable
62
              WAIT FOR 150 ns;
              ASSERT false REPORT "End of simulation" SEVERITY FAILURE;
63
          END PROCESS tb_main_proc;
64
65
      END Testbench;
66
```

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:

Settings - ThreeBitCounter	
Category:	
General Files Libraries • Operating Settings and Conditions Voltage Temperature • Compilation Process Settings Early Timing Estimate Incremental Compilation Physical Synthesis Optimizations • EDA Tool Settings Design Entry/Synthesis Simulation Timing Analysis Formal Verification Board-Level • Analysis & Synthesis Settings VHDL Input Verilog HDL Input Default Parameters Fitter Settings TimeQuest Timing Analyzer Assembler Design Assistant SignalTap II Logic Analyzer Logic Analyzer Interface Power Play Power Analyzer Settings SSN Analyzer	Simulation Specify options for generating output files for use with other EDA tools. [Jool name: ModelSim-Altera Rung gate-level simulation automatically after compilation EDA Netlist Writer settings Format for output netlist: WHOL Time gcale: 100 us Output girectory: simulation/modelsim Generate Value Change Dump (VCD) file script Script Settings Design instance name: More EDA Netlist Writer Settings NativeLink settings Soript to set up simulation: More NativeLink Settings More NativeLink Settings More NativeLink Settings
	OK Cancel Apply Help

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 ...

disting test bench s	settings:				<u>N</u> ew
Name	Top Level Module	Design Instance	Run For	Test Bench File(s)	
hree_bit_counte	three_bit_counte	NA		/src/three_bit_counter.tb.vhd	Ealt

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

🥒 Edit Test Bench Setti	ngs	-		×
Edit test bench settings f	or the selected	test bench.		
Test bench name: three	_bit_counter_tb	0		
Top level <u>m</u> odule in test b	ench: three_b	it_counter_tb		
Use test bench to per	form VHDL timin	ng simulation		
Design instance name	e in test bench:	NA		
Simulation period				
Run simulation unti	all <u>v</u> ector stimu	li are used		
End simulation at:		s 🔻		
Test bench files				
Eile name:				Add
File Name	Library	HDL Version		Remove
/src/three_bit		Default		Un
				Down
				Properties
		ОК	Cancel	Help

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* \rightarrow *Run EDA Simulation Tool* \rightarrow *EDA RTL Simulation*

od	odelSim_Demo_ThreeBitCounter/quartus/ThreeBitCounter - ThreeBitCounter								
	Tools Window Help 💎								
e	Run EDA Simulation Tool	EDA RTL Simulation							
ć	Run EDA Timing Analysis Tool	EDA Gate Level Simulation							

Es erscheint zuerst das Compile-Fenster ...

ModelSim ALTERA STA	ARTER EDIT	TION 6.6d	- C X
File Edit View Com	pile Simu	ulate Add Library Tools Layout Window Help	
D • 🚘 🖬 🛸 🚳 I	¥. 🕅 🖌	11. つう 🕐 - 🏟 監 唱 🖬 🕼 🥴 🎬 😰 🔯 🖉 🚺 兴 (A) 🖄 Layout (NoDesign 🔍 ColumnLayout (AllColumns 🔍	
Library			
* Name	Type	Path	<u> </u>
€ 220model	Library	\$MODEL_TECH//altera/vhdl/220model	
	Library	\$MODEL_TECH/./altera/verilog/220m	
💽 🛨 altera	Library	\$MODEL_TECH//altera/vhdl/altera	
	Library	\$MODEL_TECH//altera/vhdl/altera	_
	Library	\$MODEL_TECH//altera/verilog/altera	
altera_mf	Library	\$MODEL_TECHJatera/whd/jatera_mt	
i itera_mf_ver	Library	\$MODEL_TECHaltera/verlog/altera	
± altera_ver	Library	swoodL_leCH,atterayVerlinggatera	
it aligxo	Library		
	Library		
	Library	SMOULT_ICLF]./alueta/verligg/alugau	
arriagy besi	Library	whore Terri / Janes dy magan loga	
arriagy hssi ver	Library		
+ arriagx ver	Library	WODEL TECH. Jahra Jering and an	
⊕_ fit arriai	Library	SMODE_TECH/_/altera/whil/artial	
	Library	SMODEL TECH/altera/vhd/arriaii hssi	
	Library	SMODEL TECH//altera/verilog/ariaii	
	Library	SMODEL TECH//altera/vhdl/arriali p	
. arriai_pcie_hip_ver	Library	\$MODEL_TECH//altera/verilog/arriaii	
III arriaii ver	Library	MODE TECH/ Jahera Kverlon Jeriai	-
A Transcript			
# ymap work rtl wor	sk.		
# Copying C:\altera	\11.0 mo	odelsim ase\modelsim ase\win32aloem//modelsim.ini to modelsim.ini	
# Modifying modelsi	m.ini		
# ** Warning: Copie	d C:\alt	tera\11.0_modelsim_ase\modelsim_ase\win32aloem//modelsim.ini to modelsim.ini.	
# Updated	modelsim	n.111.	
# vcom -93 -work wo	rk (D:/	Sandhox/Quartus Finfuebrung/ModelSim Demo ThreeRitCounter/arc/three bit counter.m.yhdl	
# Model Technology	ModelSim	m ALTERA vcom 6.6d Compiler 2010.11 Nov 2 2010	
# Loading packag	e standa	ard	
# Loading packag	e std_lo	ogic_1164	
# Loading packag	e numeri		
# Compiling enci	itecture	ounter a behavioral of three bit counter	
e e e e e e e e e e e e e e e e e e e			
# vcom -93 -work wo	rk {D:/_	_Sandbox/Quartus Einfuchrung/ModelSim_Demo_ThreeBitCounter/quartus//src/three_bit_counter.tb.vhd}	
# Model Technology	ModelSim	m ALTERA vcom 6.6d Compiler 2010.11 Nov 2 2010	
# Loading packag	e standa	and 1964	
# Loading packag	e suu_10 e numeri	0g12_1109 10 atd	
# Loading entity	three b	bit counter	
# Compiling enti	ty three	e_bit_counter_tb	
# Compiling arch	itecture	e testbench of three_bit_counter_tb	
			-
Loading			
licodding			

... 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:



Image: Second and Add Objects Tools Loog Window Hep Image: Second and Add Objects Tool Lo	If the first Verse Complex Soundare Add Object Table Longer Window Edge If the first Verse Soundare Add Object Table Longer Window Edge If the first Verse Soundare Add Object Table Longer Verse Soundare Control Table Longer Verse Soundare Add Object Table Longer Verse Soundare Verse Verse Soundare Verse Verse Soundare Verse Soundare Verse Soundar	ModelSim ALTERA STARTER EDITION 6.6d				- • ×
Commence for the study of the s	Image: Second	<u>Eile Edit View Compile Simulate Add C</u>	Objec <u>t</u> s T <u>o</u> ols Layo <u>u</u> t <u>W</u> indow <u>H</u> elp			
Commando Default Image: Defa	Contruent precaute Image: Status Image: St] D•≇∎©@(∦™®±2:2)(◎-#4 ฿ ฿ ቚ � ֎ ֎ ֎ ֎ @ 1	- ሩ 🛶 Ef 🛛 100 ps 🗲 El Eî El 🌌 😩 7	면 다 다 코 🌇 🕲 🛛 💥 🕅 🖻 🐐 🛛 Layout Simulate	
Image: Control of the X of X of X of Operator with the X of X of Operator X of X	Image: Intervention	ColumnLayout Default		<u> * + 7 7 7 7 7 1 4- 4 % 4 ≯ </u>	% \$<<<<<<>↓ □ ■ ■ ■ ■ 「 」 ■ ■ ■ ■ □ ■ □ ■ □ ■ □ ■ □ ■ □ ■ □ ■ □	
Tindarez Desgo unit Desgo unit Desgo unit Sego unit	Virtuation Design unit	💭 sim 🛨 🗗 🗙	💫 Objects 🛨 🖻 🗙	Wave		+ @ ×
A traded a traded a traded Padage and traded Padage a traded Padage a traded a traded padage traded a traded padage a trade	<pre>class tree bt contre_b tree bt.c. Architet</pre>	VInstance Design unit Design ur	Value Value	🚱 🗸 Msgs		V
Image: Set Constraint 0	<pre>(j = 0 _ close tree bit Archest</pre>	- three_bit_counter_tb_three_bit_c Architect	sl_dock 1	/three_bit_counter 1		
Brank rox Brank rox Perception etcoded standard Perception etcoded perception Perception etcoded perception <	<pre>sparager mere bit conter.ptil mere bit conter.</pre>	the dock process	<pre>> sl_enable 0 </pre>	/three_bit_counter 0		
tehnded is standard Podage at standard indexe, standard Podage numeric_std standard Nedage numeric_std standard (Nedage numeric_std standard Podage Processes (Active) → + if x Processes (Active) → + if x Process	<pre>interded istanded Possage interded Possage interded interded Possage interded interded Possage interded interded interded Possage interded int</pre>	tb_main_proc three_bit_c Process		/three_bit_counter 100 100		
stdjogic_lif4 stdjogic_lif4 padage numeric_std numeric_std Padage "Nome:std	<pre>Ind Job_1164 at Ubgr_L164 at Ubgr_L Pedage Index:_std numeric_std Pedage Index:_std numeric_std Pedage Index:_std numeric_std Pedage Index:_std numeric_std Pedage Index:_std Index</pre>	📕 standard standard Package				
numeric_std Padage	<pre>numeric_tid numeric_tid Package interest_tid P</pre>	std_logic_1164 std_logic_1 Package				
Interview of the second s	<pre></pre>	numeric_std numeric_std Package				
Processes (Active) The Type (filtered) 5t Byman, proc VHDL Process Ac Byman, proc VHDL Process Ac Cursor 1 Dos Type(filtered) 5t Cursor 1 Dos Type(filtered)	<pre></pre>					
Image: State of the state o	<pre> Store St</pre>					
Image: State of the s	<pre>image: processes (Active) image: proces</pre>					
Image: State in the s	<pre> Processe (Active) === \$ \$ \$ \$ X Processe (Active) === \$ \$ \$ \$ X Processe (Active) === \$ \$ \$ \$ \$ X Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ \$ Processe (Active) == \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$</pre>					
Image: State of the state	<pre>image: processe (Active) image: processe</pre>					
Image: State of the s	Processes (Active) = :::::::::::::::::::::::::::::::::::					
Processes (Actve) **** X * In x **** * In x ***** * In x ************************************	Processes (Active) Type (Niered St Whine th main_proc WHOL Process Ac th the main_proc WHOL Process Ac the main_proc whole active a					
	Image: Second					
Image: Type (filtered) [St Image:	Name Type (Miered) [St] Image: Droc WHDL Process Ac Image: Droc WHDL		🞲 Processes (Active) 🔤 🕅 🗶			
Image: product which process Image: product which process Image: product which process Image: product which pr	Image: Discontre process to main process to mai		Name Type (filtered) St			
Image: Second	<pre></pre>		U_mair_proc vhbc process Ac			
Image: State in the	Image: Statistic Breakpoint: Break in Process to main proc at D:/_Sandbox/Quartus Simulation Breakpoint: Break in Process to main proc at D:/_Sandbox/Quartus					
Image: Constraint of the	<pre></pre>					
▲ (more than the second sec	<pre> Simulation Breakpoint: Break in Process th main proc at D:/_Sandbox/Quartus Einfuehrung/ModelSim_Demo_ThreeBitCounter/quartus//src/three_bit_counter.tb.vhd line 71 vSM(paused)> </pre>					
	<pre> Case Now 705000 ps</pre>					
2 = 0 Now 705000 ps 70400 ps 7	<pre></pre>					
Carson 1: Ope Car	<pre> Case Now 70500 ps 70400 ps</pre>					
▲ Now 705000 ps 111111111111111111111111111111111111	Image: Simple in the intervence of the intervence					
Now 705000 ps 704000 ps 704	Iterary () im Iterary () im<					
Image: Second process of the second proces of the second process	<pre>Now 705000 ps 70400 ps 70</pre>					-
	Image: State of the state o			Now 705000 ps	704200 ps 704400 ps 704600 ps 704600 ps 704800 ps	705000 pe
	It Ubray Image: Simpletion Breakpoint: Break in Process to main proc at D:/_Sandbox/Quartus Einfuchrung/ModelSim_Demo_ThreeBitCounter/quartus//src/three_bit_counter.tb.vhd line 71 # MACRO ./ThreeBitCounter_run_msim_rtl_vhdl.do PAUSED at line 17 VSDM(paused)>	۲ ()		ਿ 🖉 🖉 🖉 🕞 🕞 🕞		
	A Transcipt ::::::::::::::::::::::::::::::::::::	Library 🔊 sim 👍				
	<pre># issup:</pre>					ابد الار الد ا
TellStopp	<pre># MACRO ./ThreeSitCounter_run_msim_rtl_vhdl.do PAUSED at line 17 VSDM(paused)></pre>	Simulation Breakpoint: Break in Proc	sets th main proc at D+/ Sandboy/Ouants	Finfuebrung/ModelSim Demo ThreeBitCounts	er/martus/ /src/three hit counter th whd line 71	
Y JANGE O'THEEBICOURCET RUN BIT L'UNDI OF BATSD AT LINE 17	VSDb(paused)>	# MACRO ./ThreeBitCounter_run_msim_rtl	L_vhdl.do PAUSED at line 17	a Einteentung, Houersim_Demo_InreeBitcounce	er/quarcus//src/onree_brc_counter.tb.vnd line /1	
	nzmulbarse0>	10774 (
Instandbarakeo'>	•	VSIM(paused)>				-
	Nov:705 ns Delta: 0 sm://tree_bit_counter_tb/tb_main_proc	Now: 705 ns Delta: 0 sim:/three_bit	t_counter_tb/tb_main_proc			

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 $Radix \rightarrow Unsigned$ 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 …

					_	
Object Declaration						
View Assertion						
Cover Directive View)	<u> </u>				
Radix	Ì		Global	Signal Radi	x	
Format)		Symbo	lic		_
Cut	Ctrl+X		Binary			
Сору	Ctrl+C		Octal			
Paste	Ctrl+V		Decima	l i		
Delete			Unsign	ed		
Group		1	Hexad	ecimal		
Ungroup			ASCII			
ongroup		-	Time			
Create/Modify Waveform	n		Fixed F	Point		
Map To Design Signal			Defaul	t		
Insert Divider			Numeri	c Enums		
Insert Breakpoint			Symbo	lic Enums		
Force		_				
NoForce						
Clock						
Desarching		-				
Properues						

Wave					_		X	=																= + ø	x
*	Msgs																								
/three_bit_counter_tb/sl_clock	1	ЪГ	IЛЛ	ЛЛ.	J	ιh	Л	Г	LГL	ГГ	Л	Л	Ц	บบ	Г	Л	Л	Т	ோ	L	பா	лл			•
/three_bit_counter_tb/sl_enable	0																		1				4		
+	4	0			(1	12)3)4),5	<u>)</u> 6)(7)0	<u>)</u> 1) <mark>2</mark>)3	14),5),6	7)0	1)2	3) <mark>4</mark>				Į.		
A≣⊛ Now	000 ps	ı ı DS			20	0000	n n ps		1		1	40	000	, , 0 ps		I.	1	ľ	600	0000) ps	1 1			Ī
🔓 🖉 😑 Cursor 1	587 ps																					704	587 ps		
	\mathbf{A}	▲																						Þ	



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;
     USE IEEE.STD_LOGIC_1164.ALL;
26
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
           PORT (
54
55
               u, v, carry_in: IN STD_LOGIC;
56
               sum, carry_out: OUT STD_LOGIC
57
           );
     END ENTITY full_add;
58
59
     _____
60
61
    ARCHITECTURE behavioral OF full_add IS
62
63
     BEGIN
                    <= u XOR v XOR carry_in;
64
        sum
65
                    <= (u AND v) OR ((u OR v) AND carry_in);
        carry_out
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
            PORT (
30
31
                      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
               a, b
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
     ENTITY adder4 IS
57
58
            PORT (
                      : IN STD LOGIC VECTOR(3 DOWNTO 0);
59
               a, b
60
                      : OUT STD LOGIC VECTOR(3 DOWNTO 0);
               s
                      : IN STD LOGIC;
61
               cin
                      : OUT STD_LOGIC
62
                cout
63
            );
64
     END ENTITY adder4;
65
     _____
66
67
     ARCHITECTURE structure OF full_add IS
68
        SIGNAL c : STD_LOGIC_VECTOR(2 DOWNTO 0);
69
70
     BEGIN
        u_adder_0 : full_add PORT MAP (a(0), b(0), cin, s(0), c(0));
71
        u_adder_1 : full_add PORT MAP (a(1), b(1), c(0), s(1), c(1));
72
        u_adder_2 : full_add PORT MAP (a(2), b(2), c(1), s(2), c(2));
73
        u_adder_3 : full_add PORT MAP (a(3), b(3), c(2), s(3), cout);
74
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
    ENTITY adder4 tb IS
30
31
     END ENTITY adder4 tb;
32
33
    ARCHITECTURE Testbench OF adder4 tb IS
34
    SIGNAL usig4_in_a: UNSIGNED(3 DOWNTO 0) := (OTHERS => '0');SIGNAL usig4_in_b: UNSIGNED(3 DOWNTO 0) := (OTHERS => '0');
35
36
     SIGNAL usigl_carry_in : UNSIGNED(0 DOWNTO 0) := (OTHERS => '0');
37
     SIGNAL slv4_sum : STD_LOGIC_VECTOR(3 DOWNTO 0);
38
39
     SIGNAL carry_out
                             : STD_LOGIC;
    SIGNAL i_count
                             : INTEGER
40
                                                      := 0;
41
42
    BEGIN
          -- ## Unit Under Test Instantiation
43
44
          u_dut : adder4 PORT MAP (
45
                       => STD_LOGIC_VECTOR(usig4_in_a),
              а
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
              VARIABLE vusig5 result
                                         : UNSIGNED(4 DOWNTO 0);
54
55
          BEGIN
56
                 Generate stimulus
              usig1 carry_in <= NOT usig1_carry_in;</pre>
57
              IF usig1_carry_in = "0" THEN
58
59
                 usig4_in_a <= usig4_in_a + 3;</pre>
                                                      -- Prime numbers added
60
                 IF usig4_in_a = "0000" THEN usig4_in_b <= usig4_in_b+7; END IF;
              END IF;
61
62
              WAIT FOR 10 ns;
63
64
              -- Check Result
              vusig5_result := '0'&usig4_in_a + usig4_in_b + usig1_carry_in;
65
66
67
              ASSERT slv4_sum = STD_LOGIC_VECTOR(vusig5_result(3 DOWNTO 0))
                  REPORT "Sum is wrong" SEVERITY ERROR;
68
              ASSERT carry_out = STD_LOGIC(vusig5_result(4))
69
                  REPORT "Carry is wrong" SEVERITY ERROR;
70
71
              i_count <= i_count + 1;</pre>
72
          END PROCESS tb_clock_proc;
73
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
                  isl_clock : IN std_logic;
isl_start : IN std_logic;
isigl2_input_x : IN signed (11 DOWNTO 0);
isigl2_input_y : IN signed (11 DOWNTO 0);
              PORT (
61
62
63
64
65
                   isigi2_input_y : IN signed (II
osl_output_valid : OUT std_logic;
66
                   osigl2_arctan_output : OUT signed (11 DOWNTO 0)
67
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;
     USE IEEE.numeric std.ALL;
76
77
78
     USE work.cordic rom pkg.ALL;
     USE work.barrel shifter pkg.ALL;
79
80
81
     ENTITY arctan_cordic IS
82
              PORT (
                  isl_clock : IN std_logic;
isl_start : IN std_logic;
isig12_input_x : IN signed (11 DOWNTO 0);
isig12_input_y : IN signed (11 DOWNTO 0);
osl_output_valid : OUT std_logic;
83
84
85
86
87
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
           sl_start_cordic_d1
 97
                                    : std_logic;
 98
            sl_finished
                                    : std_logic;
99
                                    : unsigned(3 DOWNTO 0);
            usig4_iteration_count
100
101
           sig19_cordic_adder_1
                                    : signed(18 DOWNTO 0);
           sig19_cordic_adder_3
102
                                    : signed(18 DOWNTO 0);
103
                                    : 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);
       END RECORD;
109
110
111
        TYPE t adder result IS RECORD
112
            sig18 sum
                                     : signed(17 DOWNTO 0);
113
                                     : std logic;
            sl carry
114
        END RECORD;
115
116
        SIGNAL r, r_next
                                    : t_cordic_reg;
117
        SIGNAL sig12_rom_inv5
118
                                    : signed(11 DOWNTO 0);
119
       SIGNAL sig18_input_x: signed(17 DOWNTO 0);SIGNAL sig18_input_y: signed(17 DOWNTO 0);
120
121
122
        SIGNAL sig18_barrel_shift_1_out : signed(17 DOWNTO 0);
123
        SIGNAL sig18_barrel_shift_2_out : signed(17 DOWNTO 0);
124
125
     _____
126
127
128
        -- Calculate bit sum using carry from previous step, then carry out
        FUNCTION add_w_carry (isig18_a, isig18_b: signed(17 DOWNTO 0);
129
                         isl_carry_in : std_logic) RETURN t_adder_result IS
            VARIABLE sl_carry : std_logic;
130
131
            VARIABLE r_result
                                    : t adder result;
        BEGIN
132
            sl_carry := isl_carry_in;
133
            FOR i IN 0 TO 17 LOOP
134
               r_result.sig18_sum(i) := isig18_a(i) XOR isig18_b(i)
135
                                                  XOR sl_carry;
136
                                    := (isig18_a(i) AND isig18_b(i))
                sl_carry
                                                   OR (isig18_a(i)
137
                               AND sl_carry) OR (isig18_b(i) AND sl_carry);
            END LOOP;
138
139
            r_result.sl_carry
                                 := sl_carry;
140
            RETURN r_result;
       END FUNCTION add_w_carry;
141
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))
	<pre>& isig12_input_y & (3 DOWNTO 0 => isig12_input_y(11));</pre>
150	
151	cordic_comb_proc : PROCESS (isl_clock, isl_start, r, r_next,
152	<pre>isig12_input_x, isig12_input_y,</pre>
153	<pre>sig18_input_x, sig18_input_y,</pre>
154	<pre>sig18_barrel_shift_1_out,</pre>
155	<pre>sig18_barrel_shift_2_out,</pre>
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	<pre>vsig18_adder_1_in_a := r.sig19_cordic_adder_1(17 DOWNTO 0);</pre>
171	<pre>vsig18_adder_1_in_b := r.sig19_cordic_adder_1(17 DOWNTO 0);</pre>
172	<pre>vsig18_adder_2_in_a := r.sig19_cordic_adder_2(17 DOWNTO 0);</pre>
173	<pre>vsig18_adder_2_in_b := r.sig19_cordic_adder_2(17 DOWNTO 0);</pre>
174	<pre>vsig18_adder_3_in_a := r.sig19_cordic_adder_3(17 DOWNTO 0);</pre>
175	<pre>vsig18_adder_3_in_b := r.sig19_cordic_adder_3(17 DOWNTO 0);</pre>
176	
177	v.sl_start_cordic_d1 := isl_start; Start on rising edge
178	
179	
180	First cycle in computation
181	<pre>IF r.sl_start_cordic_d1 = '0' AND isl_start = '1' THEN</pre>
182	
183	if numerator is equal to +0 or -0 then return directly 0
184	IF (isig12 input $y = B^{"}000000000000000000000000000000000000$
	OR isiq12 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 (isiq12 input $x = B^{"000000000000000000000000000000000000$
	OR isiq12 input x = B"111111111111") THEN
190	v.sl_finished := '1';



191	IF (isigl2_input_y(11) = '0') THEN 1440 == +90 degrees
192	v.sigl2_cordic_output := B"010110100000";
193	ELSE $1440 = -90$ degrees
194	v.sigl2_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 => 'O');
201	
202	Process first round
203	IF $isigl2_input_x(11) = '1'$ THEN sign of x
204	vsigl8_adder_l_in_a := NOT sigl8_input_x + 1;
205	vsigl8_adder_2_in_a := NOT sigl8_input_y + 1;
206	ELSE
207	vsigl8_adder_1_in_a := sigl8_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	vsigl8_adder_l_in_b := (OTHERS => '0');
215	<pre>vsigl8_adder_2_in_b := (OTHERS => '0');</pre>
216	
217	vsigl8_adder_3_in_a := (OTHERS => '0');
218	<pre>vsigl8_adder_3_in_b := (OTHERS => '0');</pre>
219	
220	END IF?
221	
222	Develop and a supervise for find and subserved above
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(1/ DOWNTO 0);
220	VSIGI8_adder_2_In_a := r.SIGI9_CordiC_adder_2(17_DOWNTO_0);
227	$vsigis_adder_s_in_a = r.sigis_cordic_adder_s(1) DOWNIO ();$
228	
229	TE r gigle gordig adder $2(17) = 111$ THEN
230	$\frac{1}{1} = \frac{1}{1} = \frac{1}$
231 222	vsigio_adder_1_in_b ·- Noi sigio_barret_shirt_1_out;
434 222	$vsigio_adder_2_in_b \rightarrow sigio_barrel_shirt_2_out;$
222	$v_{S1G10} = adder_{5} = m_{01} (000000 \& S1G12 = 100 = 1005);$
204 005	$\frac{1}{1000}$
235	vsigio_adder_1_in_b ·- sigio_barrel_shirt_1_out;
230	$v_{sigl0} = dder_2 = in_b = \frac{1}{2} = \frac{1000000}{100000} = \frac{1}{2} = \frac{1000000}{100000}$
227	VSIGIO_adder_5_III_D ·- 000000 & SIGIZ_FOM_IIIV57
230	
240	x usig 4 iteration count := r usig 4 iteration count + 1.
241	v.usigi_iteration_count ·- i.usigi_iteration_count + 1/
242	Check if the result has been found (num == 0)
243	TE (r sigl9 cordic adder $2 = -1$ OR r sigl9 cordic adder $2-0$) THEN
244	v_{usig4} iteration count := v^{8} :
245	END TF:



248	Reached the resu	lt
249	ELSE	
250		
251	v sigl? cordic outpu	t := r sigl9 cordic adder $3(13 \text{ DOWNTO } 2)$;
201	Override sign bit	
252		$+(11)$, \dots signal conditions $2(17)$.
252	v.sigi2_cordic_outpu	t(11) := r.sig19_cordic_adder_3(17);
253	v.usig4_iteration_co	unt := x"0";
254	v.sl_finished	:= ' <mark>1</mark> ';
255		
256	END IF;	
257		
258	Adder instatiati	07
250	ur addar ragult	dd w arrw(waial and ar 1 in a
239	vi_addei_iesuit ·- ad	u_w_carry(vsrgro_adder_r_rra,
0.50	V	sigis_adder_i_in_b, v.si_carry_i),
260	v.sigl9_cordic_adder_1	:= vr_adder_result.sig18_sum(1/)
		& vr_adder_result.sig18_sum;
261	v.sl_carry_1	<pre>:= vr_adder_result.sl_carry;</pre>
262		
263	vr adder result	:= add w carry(vsig18 adder 2 in a,
		sig18 adder 2 in b. v sl carry 2);
264	v sigl9 cordig adder 2	- yr adder regult gig18 gum(17)
201	V.SigiJ_coluic_addel_z	·- vi_adder_result.sigit_sum(i/)
0.65	1	« VI_adder_result.sigio_sum,
265	v.sl_carry_2	:= vr_adder_result.sl_carry;
266		
267	vr_adder_result	<pre>:= add_w_carry(vsig18_adder_3_in_a,</pre>
		<pre>vsig18_adder_3_in_b, v.sl_carry_3);</pre>
268	v.sig19 cordic adder 3	:= vr adder result.sig18 sum(17)
		& vr adder result sigl8 sum;
269	vel carry 3	·- yr adder regult al carry:
200	V.BI_Cally_5	·- vi_adder_result.si_cally/
270		Come enviching to simple
2/1	I_NEXU <= V;	Copy variables to signals
Z/Z END	PROCESS COrdic_Comb_proc	1
273		
274		
275		
276	cordic_reg_proc : PROCES	S (isl_clock)
277	BEGIN	
278	<pre>IF rising_edge(isl_c</pre>	<pre>lock) THEN r <= r_next; END IF;</pre>
279	END PROCESS cordic req p	roc;
280		
281		
201		
202		
283	Output assignments	
284	osl_output_valid	<= r.sl_tinished;
285	osig12_arctan_output	<= r.sigl2_cordic_output;
286		
287	u_shifter_1 : barrel_shi	fter PORT MAP (
288	usig4_shift value	<pre>=> r.usig4_iteration_count,</pre>
289	isig input	= r.sigl9 cordic adder 2(17 DOWNTO 0)
290	osia output	=> sigl& barrel shift 1 out
201) ·	· Sigto_Darrer_Durre_r_Out
202) /	
292		
293	u_sniiter_2 : barrel_shi	ITER PORT MAP (
294	usig4_shift_value	<pre>=> r.usig4_iteration_count,</pre>



```
____nput
osig_output
295
                                => r.sig19_cordic_adder_1(17 DOWNTO 0),
296
                                 => 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);
                                  : OUT signed (N-1 DOWNTO 0)
               osig_output
          );
     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
```

```
NTB
INTERSTAATLICHE HOCHSCHULE
FÜR TECHNIK BUCHS
```

```
sig_temp_1(N-1) <= isig_input(N-1);</pre>
               sig_temp_1(N-2 DOWNTO 0) <= isig_input(N-1 DOWNTO 1);</pre>
        ELSE
               sig_temp_1(N-1 DOWNTO 0) <= isig_input(N-1 DOWNTO 0);</pre>
        END TF;
  END PROCESS;
shift two : PROCESS (sig temp 1, usiq4 shift value(1))
BEGIN
    IF usig4_shift_value(1)='1' THEN
                                           -- shift-by-two
         sig_temp_2(N-1) <= sig_temp_1(N-1);</pre>
         sig_temp_2(N-2) <= sig_temp_1(N-1);</pre>
        sig_temp_2(N-3 DOWNTO 0) <= sig_temp_1(N-1 DOWNTO 2);</pre>
    FT.CF
        sig temp 2(N-1 DOWNTO 0) <= sig temp 1(N-1 DOWNTO 0);</pre>
    END IF;
END PROCESS;
shift_four : PROCESS (sig_temp_2, usig4_shift_value(2))
BEGIN
    IF usiq4 shift value(2)='1' THEN
                                           -- shift-by-four
         osig_output(N-1) <= sig_temp_2(N-1);</pre>
        osig_output(N-2) <= sig_temp_2(N-1);</pre>
        osig_output(N-3) <= sig_temp_2(N-1);</pre>
        osig_output(N-4) <= sig_temp_2(N-1);</pre>
        osig_output(N-5 DOWNTO 0) <= sig_temp_2(N-1 DOWNTO 4);</pre>
    ELSE
         osig_output(N-1 DOWNTO 0) <= sig_temp_2(N-1 DOWNTO 0);</pre>
    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)
               usig4_addr
       );
     END COMPONENT cordic_rom;
END PACKAGE cordic_rom_pkg;
_____
_____
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
```

NTB Einführung in INTERSTAATLICHE HOCHSCHULE FÜR TECHNIK BUCHS **ModelSim** 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 -- 1700 1 => x"6A4", -- 26,562 -- 898 2 => x"382", _ _ _ 14,031 -- 456 -- 7,125 3 => x"1C8", -- 229 -- 115 4 = x"0E5", -- 3,578 5 => x"073", 1,796 -- 57 6 => x"039", -- 0,890 -- 0,453 7 => x"01D" -- 29); BEGIN sig12_data_out <= ROM (to_integer(usig4_addr(2 DOWNTO 0)));</pre> 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 t_tb_result IS (GOOD, ERROR); TYPE SIGNAL tb result : t tb result := GOOD;



SIGNAL i_denumerator: integer := 5;SIGNAL i_numerator: integer : 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; : 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, => sl_req_sample, isl_start => sl_req_samp isigl2_input_x => sigl2_input isigl2_input_y => sigl2_input osl_output_valid => sl_output_valid, osigl2_arctan_output => sigl2_output_z isl_start => sig12_input_x, => sig12_input_y,); 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; -</pre> <= '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;</pre> sig12_input_x <= to_signed(int_rand_x, 12); --</pre> 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;</pre> sig12_input_y <= to_signed(int_rand_y, 12); -convert integer to std_logic_vector sl_req_sample <= '1';</pre> ELSE real_arc_tan <= arctan(real(i_numerator)/real(i_denumerator)) * 180.0 / 3.1415; sl req sample <= '0';</pre> 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 $+/-90^{\circ}/128 = +/-$ _ _ 0.352 real_diff_of_rtl <= abs(real_arc_tan -</pre> 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 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 1pm -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}
guietly 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}
```