

Tutorial VHDL met Quartus 13.0sp1 en ModelSim-Altera 10.1d

DE HAAGSE
HOGESCHOOL

J.E.J. op den Brouw
De Haagse Hogeschool
Opleiding Elektrotechniek
30 januari 2019
J.E.J.opdenBrouw@hhs.nl

Versiehistorie

Rev.	Datum	Aut.	Beschrijving
1.0	16-02-2014	JodB	initial TEX version Word 2003 v013
1.1	19-02-2014	JodB	small updates
1.2	17-05-2014	JodB	v014) added Chap 5, minor fixed
1.3	00-00-0000	JodB	Not published
1.4	17-07-2014	JodB	consistent numbering version/filename changed filename xxxchip.png & xxxlayout7seg.png for consistency with INLDIG tutorial
1.5	12-02-2015	JodB	changed layout to that of schematic entry tutorial removed line numbers from tb_*.vhd and tb_*.do moved comment about concatenation of PIN_
1.6	20-04-2015	JodB	replaced 'de naam van de testbench' with 'de naam van het commandobestand' uniformly use of 'commandobestand'
1.7	02-06-2015	JodB	added "instance not bound" tips and tricks... changed HHS logo to PDF version
1.8	23-03-2016	JodB	added commands to delete db, incremental_db, simulation and output_files to the remove-script reordered some sections in Tips & Tricks modified block diagram DE0 Board
1.81	05-02-2018	JodB	changed modelsim executable path
1.82	30-01-2019	JodB	changed logo, parskip set to v1

Voor suggesties en/of opmerkingen over deze tutorial kan je je wenden tot J. op den Brouw, kamer D1.047, of je kunt email versturen naar J.E.J.opdenBrouw@hhs.nl.

Inhoudsopgave

1	Inleiding	6
2	Practicumomgeving	8
2.1	Ontwikkelbord	8
2.2	Software-versies en web-edition	9
3	Ontwikkelbordje	10
3.1	Blokschema	10
3.2	Altera Cyclone III	12
4	Tutorial met VHDL	14
4.1	Quartus starten	14
4.2	Project aanmaken	16
4.3	Invoeren van VHDL-broncode	20
4.4	Simulatie VHDL-broncode	25
4.5	Pinnen toekennen	33
4.6	Compilatie	37
4.7	Configureren van de Cyclone III	38
4.8	Gegenereerde hardware	40
5	Tips, tricks & troubleshoot	42
5.1	Foutmelding “Top-level ... undefined”	42
5.2	Instellen pad naar ModelSim	43
5.3	Smart compilation	44
5.4	Quartus blijft hangen	44
5.5	Ingestelde pad naar ModelSim wordt niet opgeslagen	45
5.6	Gebruik USB-Blaster onder Linux	45
5.7	Design Rule S102	46
5.8	Naamgeving bestanden en entity’s	46
5.9	Pinnen worden niet getoond in de Pin Planner	47
5.10	Verkeerde pinnen worden getoond	47
5.11	ModelSim stopt na enige tijd	47
5.12	“Instance ... is not bound”	47
5.13	Opruimen van een Quartus-project	48
A	Knoppen en sneltoetsen	49
B	Pinbenaming EP3C16F484C-6N	50

Lijst van figuren

1.1	De VHDL Design Flow	7
2.1	Het DE0-ontwikkelbord	8
3.1	Het DE0-ontwikkelbord met benoeming van periferie	10
3.2	Blokschema ontwikkelbord	11
3.3	Foto Cyclone III	11
3.4	Floor Plan van de Cyclone III	13
4.1	Quartus II pictogram	14
4.2	Quartus II opstartscherm	15
4.3	Openingsscherm project navigator	15
4.4	Project wizard	16
4.5	Introductie project wizard	17
4.6	Invullen mapnaam, projectnaam en naam van de top-level.	17
4.7	De map bestaat niet en moet worden aangemaakt.	18
4.8	Toevoegen van al bestaande bestanden.	18
4.9	Invoeren van device-gegevens.	19
4.10	Invoeren van EDA tools.	19
4.11	Opsomming van gemaakte keuzen in de project wizard.	20
4.12	De top-level naam verschijnt in de Project Navigator.	20
4.13	Aanmaken nieuw bestand.	20
4.14	Keuze bestandstype.	21
4.15	Overzicht Quartus IDE na aanmaken nieuw VHDL-bestand.	22
4.16	Screenshot van de in te voeren VHDL-code.	22
4.17	Opslaan VHDL-bestand.	23
4.18	Opgeven bestandsnaam VHDL-bestand.	23
4.19	Het opgeslagen bestand staat in de lijst van bestanden.	23
4.20	Starten Analysis & Synthesis vanuit het menu.	24
4.21	De analyse is gelukt.	24
4.22	De analyse is mislukt.	24
4.23	Aanmaken VHDL-testbench-bestand.	27
4.24	Aanpassen bestandseigenschappen.	27
4.25	Bestandstype veranderd naar VHDL-testbench.	27
4.26	Overzicht bestanden en bestandstypen.	28
4.27	Aanmaken ModelSim commando-script.	28
4.28	Naamgeving ModelSim commando-script aanmaken.	30
4.29	Project settings aanpassen.	30
4.30	Invoeren settings voor simulatie met ModelSim	31
4.31	Foutmelding dat synthese nog niet gelukt is.	31
4.32	Overzicht ModelSim IDE na het uitvoeren van het commando-script.	32

4.33	Timingsdiagram van de VHDL-code.	33
4.34	Opnieuw uitvoeren van het commando-script.	33
4.35	Starten van de Pin Planner	34
4.36	Overzicht van de Pin Planner IDE met pinbenaming ingevuld.	35
4.37	Afsluiten Pin Planner.	35
4.38	Starten van de Device Settings.	35
4.39	Overzicht IDE van de Device Settings.	36
4.40	Selecteren van de opties voor de niet-gebruikte pinnen.	36
4.41	Starten van de compilatie.	37
4.42	De compilatie is gelukt.	37
4.43	Overzicht van het resultaat van de compilatie.	38
4.44	Starten van de programmer.	39
4.45	Overzicht van de Programmer IDE.	39
4.46	Selecteren van de USB-Blaster download-hardware.	40
4.47	Selecteren van de RTL Viewer.	40
4.48	Overzicht van de gegenereerde hardware in primitieven.	41
5.1	De top-level entity is niet gevonden.	42
5.2	Selectie van top-level entity.	43
5.3	De simulator kan niet worden gestart.	43
5.4	Instellen van pad naar ModelSim.	44
5.5	Instellen van van de optie Smart compilation.	44
5.6	Foutmelding bij programmeren onder Linux.	45
B.1	Layout 7-segment displays	51

Lijst van tabellen

3.1	Enige gegevens over de EP3C16F484C-6N	12
4.1	Koppelingen ingangen aan pinnen en schakelaars	34
4.2	Koppelingen uitgangen aan pinnen en schakelaars	34
A.1	Knoppen en sneltoetscombinaties.	49
B.1	Pinbenamingen FPGA, deel 1.	50
B.2	Pinbenamingen FPGA, deel 2.	51

Listings

4.1	VHDL-testbench	26
4.2	ModelSim command script	29
5.1	Windows opruimscrip	48

1. Inleiding

Vroeger was het de gewoonte om schakelingen op te bouwen uit losse componenten zoals transistoren, weerstanden en condensatoren. Naarmate de schakelingen complexer werden nam echter de kans op slechte verbindingen toe en de betrouwbaarheid af. Daarom is men er toe over gegaan meerdere componenten op één siliciumchip te integreren; voor digitale schakelingen begon dit met poorten en flipflops (Small Scale Integration), ging verder met tellers, decoders, multiplexers et cetera (Medium Scale Integration), en het einde is met de geavanceerde microprocessors en geheugens nog niet in zicht (Very Large Scale Integration). Het inwendige van zo'n geïntegreerde schakeling is echter niet te veranderen; de functionaliteit ligt vast. Een nieuwe generatie van componenten, de zogenaamde configureerbare logica, biedt de mogelijkheid zelf een schakeling te ontwikkelen. Deze componenten bevatten een groot aantal basisschakelingen (poorten, flipflops en losse verbindingen) die door de gebruiker willekeurig met elkaar en met in- en uitgangspennen kunnen worden verbonden. Bovendien zijn er recepten om bijvoorbeeld in één klap een gehele 16-bit teller te configureren (bibliotheekelementen). De taak van een ontwerper verschuift dus van solderen naar beschrijven.

Hoe gaat dit nu in zijn werk? Het eigenlijke beschrijven gebeurt met behulp van een PC. Dit is dus de onmisbare schakel in het geheel. De ontwerper bedenkt eerst een schakeling op papier. Als het probleem niet in één keer te overzien is, worden er deelontwerpen gemaakt die onderling verbonden zijn. Elk deelontwerp bevat een schakeling of, als het probleem nog niet te overzien is, weer deelontwerpen. We noemen dit principe hiërarchisch ontwerpen.

Nadat alle deelontwerpen zijn bedacht wordt overgegaan tot het invoeren van de deelschakelingen. Hier hebben we een verscheidenheid aan keuzes. We kunnen schakelingen invoeren als een schema met behulp van een tekenpakket, maar ook met behulp van een speciale taal die beschrijft wat de schakeling moet doen, bijvoorbeeld VHDL. Daarnaast zijn er nog invoermogelijkheden via toestandsmachines, booleaanse functies en waarheidstabellen.

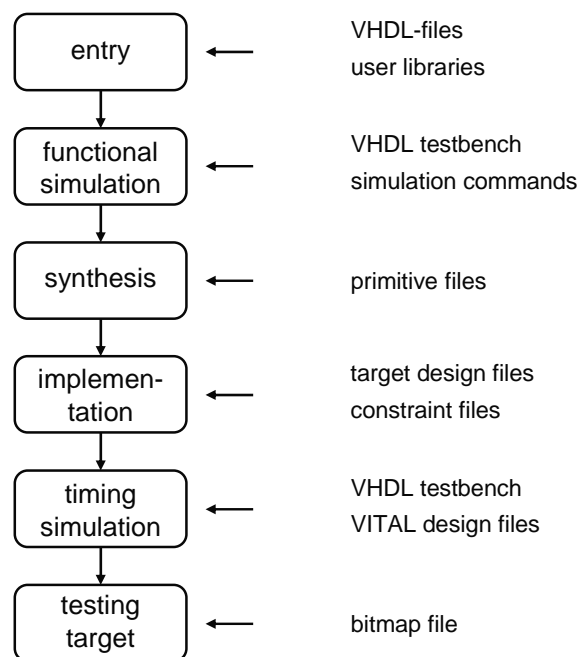
Wanneer alle deelontwerpen gemaakt zijn, moeten ze worden omgezet naar een bestand met gegevens die in de component geladen moet worden. Dit proces heet synthetiseren. Als tijdens dit omzetten een fout wordt geconstateerd, bijvoorbeeld twee uitgangen aan elkaar, wordt dit gemeld aan de ontwerper en moet de fout hersteld worden. Treden er geen fouten op dan levert de software een bruikbaar configuratiebestand op. LET OP: dit betekent nog niet dat het ontwerp precies doet wat het moet doen! Er kan nog best een functionele fout in het ontwerp zitten. Denk hierbij aan een programmeertaal. De compiler vindt geen syntax-fouten maar dat geeft geen garantie dat het programma doet wat het moet doen.

De laatste stap is het daadwerkelijk configureren van de component. Daarvoor is een hardware-programmer nodig. Die zorgt ervoor dat het configuratiebestand in de configureerbare component wordt gestopt.

Het ontwerptraject (een af te leggen weg van handelingen) wordt nu als volgt:

- bedenken van de schakeling,
- invoeren van het ontwerp in VHDL-code,
- functionele simulatie
- omzetten van de VHDL-beschrijving naar een voor de configureerbare component geschikt bestand, eventuele fouten moeten eerst aangepast worden,
- eventueel timing simulatie
- testen van de schakeling.

In figuur 1.1 is het ontwerptraject nog eens schematisch aangegeven.



Figuur 1.1: *De VHDL Design Flow*

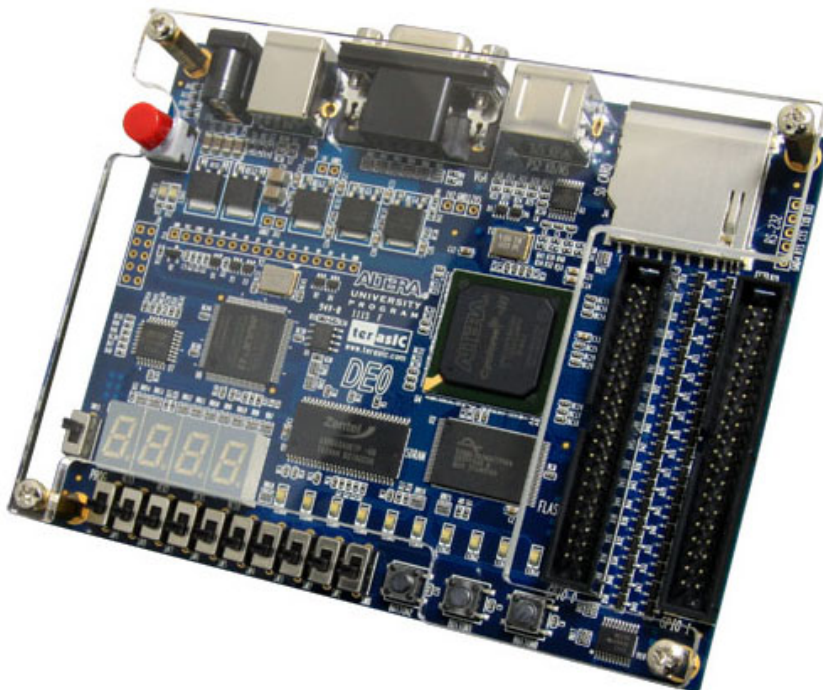
Het bedenken van de schakeling is een creatief proces. Ervaring en goede kennis van digitale systemen helpt je hierbij op weg. Het opzetten van een goede simulatie hoort bij de vaardigheden van de ontwerper. De rest wordt eigenlijk door de tools van Quartus afgehandeld. Het is voor de ontwerper niet meer interessant om op laag niveau de hardware te bekijken. Je moet er dan wel zeker van zijn dat je de hardware met de juiste regels beschreven hebt: combinatoriek en flankgevoelige geheugenelementen. Latches zijn uit den boze.

2. Practicumomgeving

In dit hoofdstuk wordt de practicumomgeving toegelicht.

2.1 Ontwikkelbord

Het practicum maakt gebruik van een ontwikkelbordjes, de DE0. Op dit bordje is een FPGA van Altera geplaatst. Hieronder is een foto weergegeven.



Figuur 2.1: *Het DE0-ontwikkelbord*

Daarnaast zijn ook nog schakelaars, leds en 7-segment displays aanwezig. Zie hoofdstuk 3 voor meer informatie.

Naast het bordje wordt een softwarepakket van de fabrikant gebruikt genaamd Quartus II. In de volgende paragraaf wordt een korte beschrijving gegeven van de software. Je hoeft niet alles in één keer te kennen. Verderop in deze handleiding is een tutorial opgenomen die je stap voor stap door het ontwerptraject loodt.

Om de component te configureren is een (hardware-)programmer nodig. Deze is op het ontwikkelbord geplaatst. Er is alleen een USB-kabel nodig voor een verbinding tussen een PC en het ontwikkelbord.

Quartus II

Quartus II is een alles-in-één pakket voor het ontwikkelen van digitale schakelingen en het configureren van (Altera) componenten. Het bestaat uit een viertal delen:

- het invoergeedeelte - d.m.v. schema's, VHDL, toestandsdiagrammen
- synthesizer - dit deel vertaalt de invoer naar een netlist,
- implementation - genereert een bit-file die je in de component kunt laden,
- programmer - dit deel configureert de component via de USB-interface.

ModelSim

ModelSim is een VHDL-simulator die direct vanuit de broncode simuleert. Er zijn in principe geen tussenstappen nodig zoals synthese. Je kan echter ook de uitvoer van de synthese simuleren. Hierdoor krijg je inzicht in de vertragingstijden. Met ModelSim is het mogelijk abstracte beschrijvingen van digitale schakelingen te simuleren. Deze schakelingen zijn niet synthetiseerbaar.

ModelSim kan als stand-alone pakket gebruikt worden. Wij zullen ModelSim gebruiken als onderdeel van Quartus en ModelSim starten vanuit Quartus.

2.2 Software-versies en web-edition

Het Quartus-pakket komt in twee smaken. Er is een volledige betaalde versie waarbij een licentie-server nodig is en er is een zogenaamde *Web Edition*. De eerstgenoemde is de meest krachtige versie: alle *devices* van Altera zijn hiermee te configureren. Daarnaast heb je hier nog optie-pakketten voor DSP-ontwikkeling en digitale filters. De Web Edition is gratis, heeft geen licentie-server nodig, maar kan niet synthetiseren voor alle beschikbare IC's. Er zijn geen optie-pakketten beschikbaar.

Van ModelSim bestaan ook twee versies: de volledige, betaalde Altera-versie en de zogenaamde *Altera Starter Edition*. De typering "Altera" geeft aan dat het specifiek ontwikkeld is om met Quartus samen te werken. De betaalde versie heeft geen beperkingen, de Altera Starter Edition kan maximaal 10000 VHDL-coderegels simuleren en verwerkt de code langzamer dan de betaalde versie.

De Web Edition van Quartus (met ModelSim geïntegreerd) kan gevonden worden op

<http://dl.altera.com/13.0sp1/?edition=web#tabs-1>

De software draait op Windows™ én Linux™, er is geen OS X-versie. Op het practicum wordt Windows gebruikt.

Let erop dat het practicum wordt uitgevoerd met versie 13.0sp1 resp. versie 10.1d.

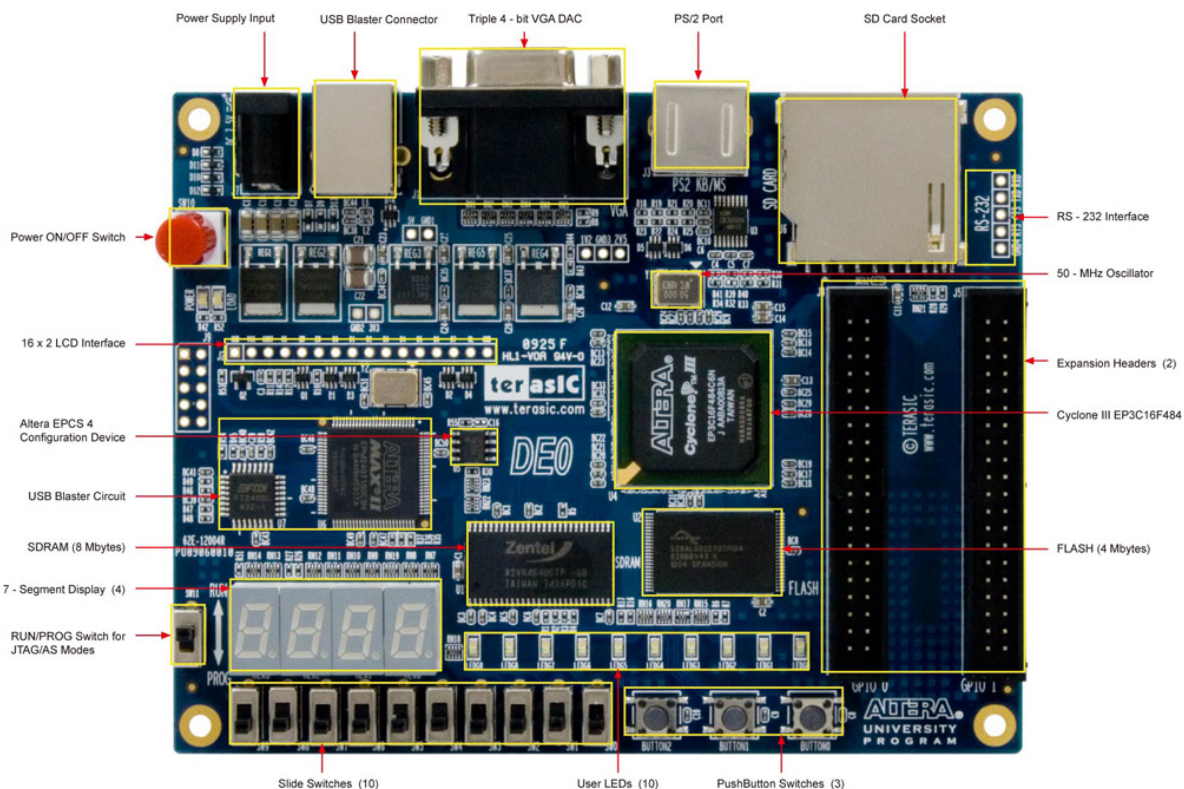
Noot: aangeraden wordt om versie 13.0sp1 te installeren. Hogere versies ondersteunen de FPGA's (Cyclone II en III) op de gebruikte ontwikkelbordjes niet. Versie 13.0sp1 is prima geschikt om deze tutorial te doorlopen. Let er op dat de pictogrammen kunnen afwijken t.o.v. oudere projecten. De in het practicum gemaakte Quartus-projecten kunnen zonder problemen door zowel versie 13.0sp1 als 11.1sp1 geopend worden. ModelSim is geïntegreerd in het installatiepakket.

3. Ontwikkelbordje

In dit hoofdstuk wordt het ontwikkelbordje nader beschreven. Eerst wordt een blokschema getoond en worden de diverse onderdelen kort beschreven. Daarna volgt een paragraaf over de configureerbare component, de Altera Cyclone III, met specificaties van het gebruikte type.

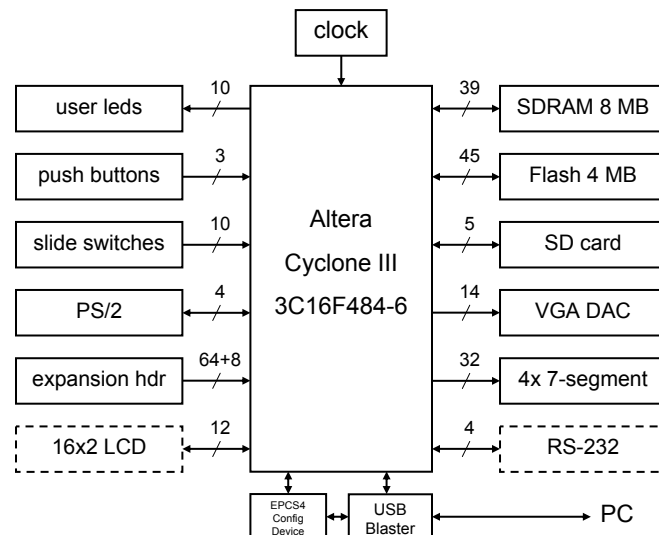
3.1 Blokschema

Het ontwikkelbord is ontwikkeld door het bedrijf Terasic in samenwerking met Altera, de fabrikant van de Cyclone III. Een gedetailleerde beschrijving is niet nodig; we geven slechts een blokschema van de onderdelen. Alles wat je nodig hebt tijdens het practicum staat hier beschreven. In figuur 3.1 is een foto afgebeeld met de diverse onderdelen.



Figuur 3.1: Het DE0-ontwikkelbord met benoeming van periferie

In figuur 3.2 is een blokschema weergegeven. Bij elk onderdeel staat vermeld met hoeveel signalen het onderdeel verbonden is met de Cyclone III. De voedingslijnen zijn niet meegenomen.



Figuur 3.2: Blokschema ontwikkelbord

Cyclone III

Het hart van het ontwikkelbord wordt gevormd door de Cyclone III EP3C16F484-6N (zie figuur 3.3). Dit is een configureerbaar IC waarin een digitale schakeling kan worden geplaatst.



Figuur 3.3: Foto Cyclone III

Push buttons

Het ontwikkelbord heeft drie drukknoppen genaamd BUTTON2 t/m BUTTON0. Deze drukken geven een laag logisch niveau (0) af als de knop is ingedrukt en een hoog logisch niveau (1) als de knop niet is ingedrukt. Verder zijn deze drie knoppen ontddenderd en zijn te gebruiken als kloksignaal.

Slide switches

Het ontwikkelbord heeft tien schuifschakelaars genaamd SW9 t/m SW0. Een schakelaar geeft een laag logisch niveau (0) af als de schakelaar naar beneden is geschoven (het dichtst bij de rand van de print) en een hoog logisch niveau (1) als de schakelaar naar boven is geschoven. Deze schakelaars zijn niet ontddenderd en zijn alleen voor niveaugevoelige ingangen bedoeld.

Leds

Het ontwikkelbord heeft tien groene leds LEDG9 t/m LEDG0. Een led brandt als een hoog logisch niveau (1) wordt aangeboden en is uit als een laag logisch niveau (0) wordt aangeboden.

7-segment displays

Het ontwikkelbord heeft vier 7-segment displays waarmee getallen van verschillend formaat kunnen worden gemaakt. Elk display bestaat uit zeven leds waarmee een cijfer kan worden gevormd. Daarnaast heeft elk display een punt. Een led brandt als een laag logisch niveau (0) wordt aangeboden en is uit als een hoog logisch niveau (1) wordt aangeboden.

Clock

Het ontwikkelbord heeft één 50 MHz klokoscillator aan boord. Het kloksignaal kan dienen als (direct) kloksignaal voor het klokken van flipflops of als invoer voor een Phase Locked Loop (PLL).

Een overzicht van de pinaansluitingen en de layout van de 7-segment displays is te vinden in bijlage B.

Overige aansluitingen

Het ontwikkelbord bevat verder nog een 8 MD SDRAM, een 4 MB Flash, een VGA-uitgang, een LCD-interface, een SD-card interface, een PS/2-interface, een RS-232-interface en een expansion header met 64 I/O-lijnen en 8 kloklijnen. Dit wordt verder niet besproken.

3.2 Altera Cyclone III

De Cyclone III FPGA (Field Programmable Gate Array) is opgebouwd uit zogenaamde logische elementen (Logic Elements, LE). Met deze elementen kan je een digitale schakeling bouwen. Het gebruikte type heeft er 15408 aan boord. Om een indruk te geven wat dat inhoudt: een volledige 32-bits processor (NIOS/II) gebruikt zo'n 1800 elementen. Je kan er dus acht processoren in kwijt.

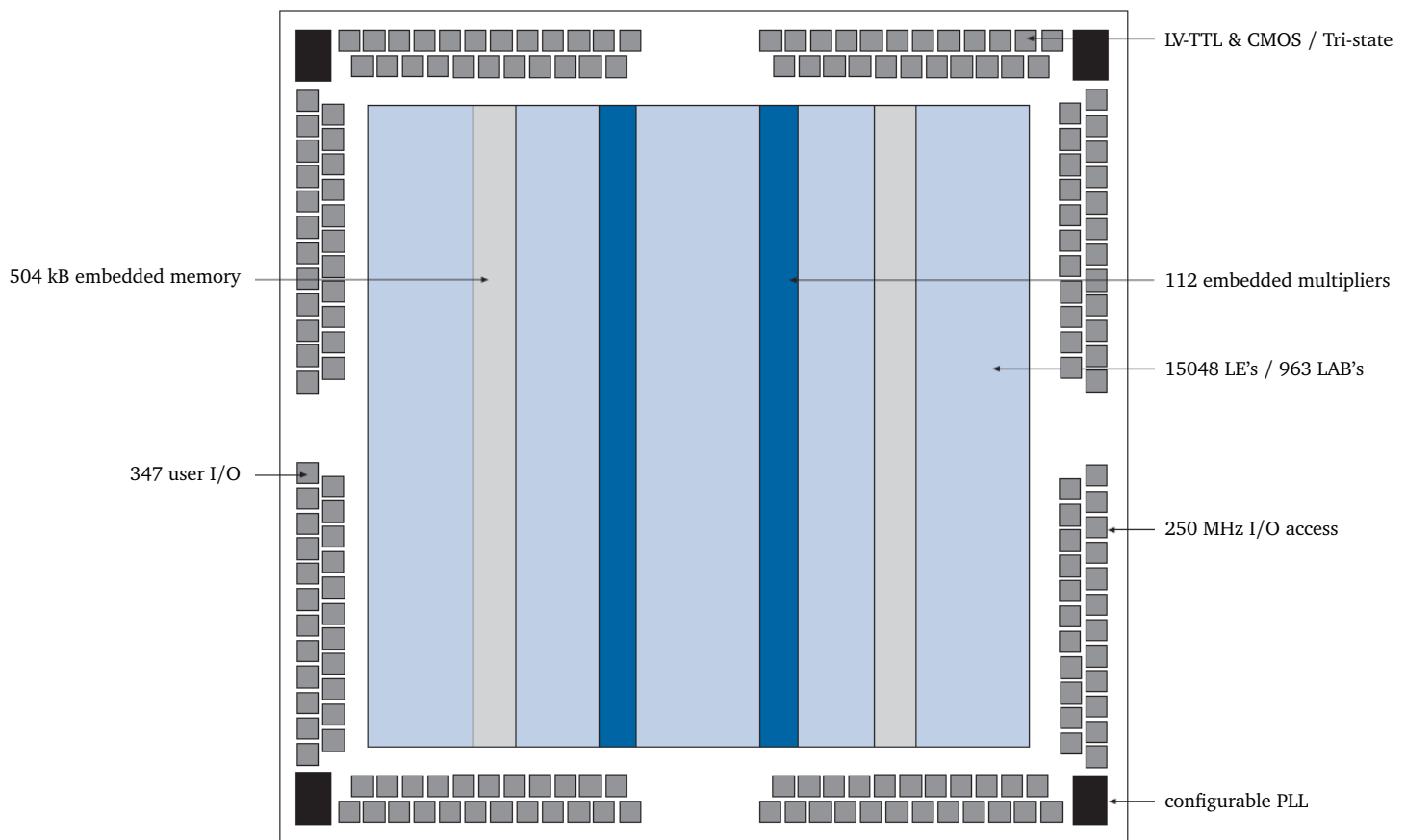
In tabel 3.1 zijn wat gegevens te vinden.

Tabel 3.1: Enige gegevens over de EP3C16F484C-6N

Aansluitpinnen (user I/O)	484 (347)
Logische elementen	15408
Geheugenelementen	15408 (één per element)
RAM-bits	516096
Vermenigvuldigers	112 (9x9 bit) / 56 (18x18 bit)
Phase Locked Loops	4
Global Clock Networks	20
$t_{PD(pin-to-pin)}$	6 ns
f_{MAX} (I/O, stand alone)	250 MHz

Figuur 3.4 geeft een Floor Plan van de bij het practicum gebruikte 3C16. Daarnaast heeft elke Cyclone RAM en vermenigvuldigers aan boord. De vermenigvuldigers zijn sneller dan

wanneer ze met LE's worden opgebouwd. Je kan ze bijvoorbeeld gebruiken bij digitale signaalbewerking.



Figuur 3.4: *Floor Plan van de Cyclone III*

Logic Element en Logic Array Block

Elke LE bestaat uit een 4-input look-up table (LUT) en een D-flipflop. De LE kan een combinatorische of sequentiële functie vervullen. De LUT kan elke combinatorische schakeling van vier variabelen nabootsen, de D-flipflop kan één bit onthouden. Indien je schakeling te groot is om in één LE te stoppen wordt dat door de software (synthese, mapper) verdeeld over meerdere LE's. Een Logic Array Block bestaat uit 16 LE's en snelle interconnect.

Routing en interconnect

Elke LE kan maar een klein deel van schakeling bevatten. Een schakeling zal dus uit meerdere LE's bestaan. Tussen de LE's is dus informatie-uitwisseling nodig. Dat gebeurt door de routing en interconnect. Realiseer je dat zo'n 50% van het chipoppervlak alleen maar routing is! De vertragingstijd van combinatorische schakelingen komt voor 2/3 voor rekening van de routing! Binnen een LAB is snelle interconnect mogelijk.

I/O Banks

De I/O banks (deze chip heeft er acht) zijn verantwoordelijk voor verbindingen tussen de buitenwereld en het interne gedeelte van de chip. Het levert de externe signalen netjes af aan de routing en signalen van routing worden netjes aan de buitenwereld afgeleverd. Tot de mogelijkheden horen: LV-TTL, LV-CMOS input, tri-state output, programmable slew rate.

4. Tutorial met VHDL

We gaan ons nu bezighouden met het ontwerpen van digitale schakelingen in VHDL. VHDL is een beschrijvingstaal; het heeft het uiterlijk van een programmeertaal maar in plaats van machinetaal of binaire code beschrijf je hoe de hardware moet werken. VHDL leren is een traject apart en komt niet in deze tutorial aan bod.

Deze tutorial zal je stap voor stap door de diverse onderdelen van Quartus en Modelsim leiden waarna je zelf een aantal opdrachten moet uitwerken.

De tutorial behandelt slechts een klein gedeelte van alle mogelijkheden die in Quartus en Modelsim voor handen zijn. Je zal zelf meer functies moeten onderzoeken die je voor de opdrachten nodig hebt.

We zullen de volgende stappen doorlopen:

- project aanmaken
- VHDL code invoeren m.b.v. de HDL editor
- simuleren van de VHDL code op functioneel niveau
- compilatie (synthese en implementatie) van de VHDL code naar een configuratiebestand
- downloaden van de configuratiebestand in de Cyclone III.

Niet aan bod komt verificatie. Verificatie valt in twee stukken uiteen: timing simulatie en timing analyse. De eerste is eigenlijk niets anders dan functionele simulatie maar dan met inbegrip van timing aspecten. Timing analyse kan je gebruiken om bijvoorbeeld het langste pad te vinden.

In bijlage A is een tabel opgenomen met een aantal knoppen en sneltoetscombinaties. Niet alle knoppen worden in deze tutorial gebruikt.

NB: gebruik geen spaties, leestekens of "vreemde"tekens in mapnamen en bestandsnamen!

4.1 Quartus starten

Quartus wordt gestart door op het pictogram te klikken of via het Startmenu. Zie figuur 4.1.



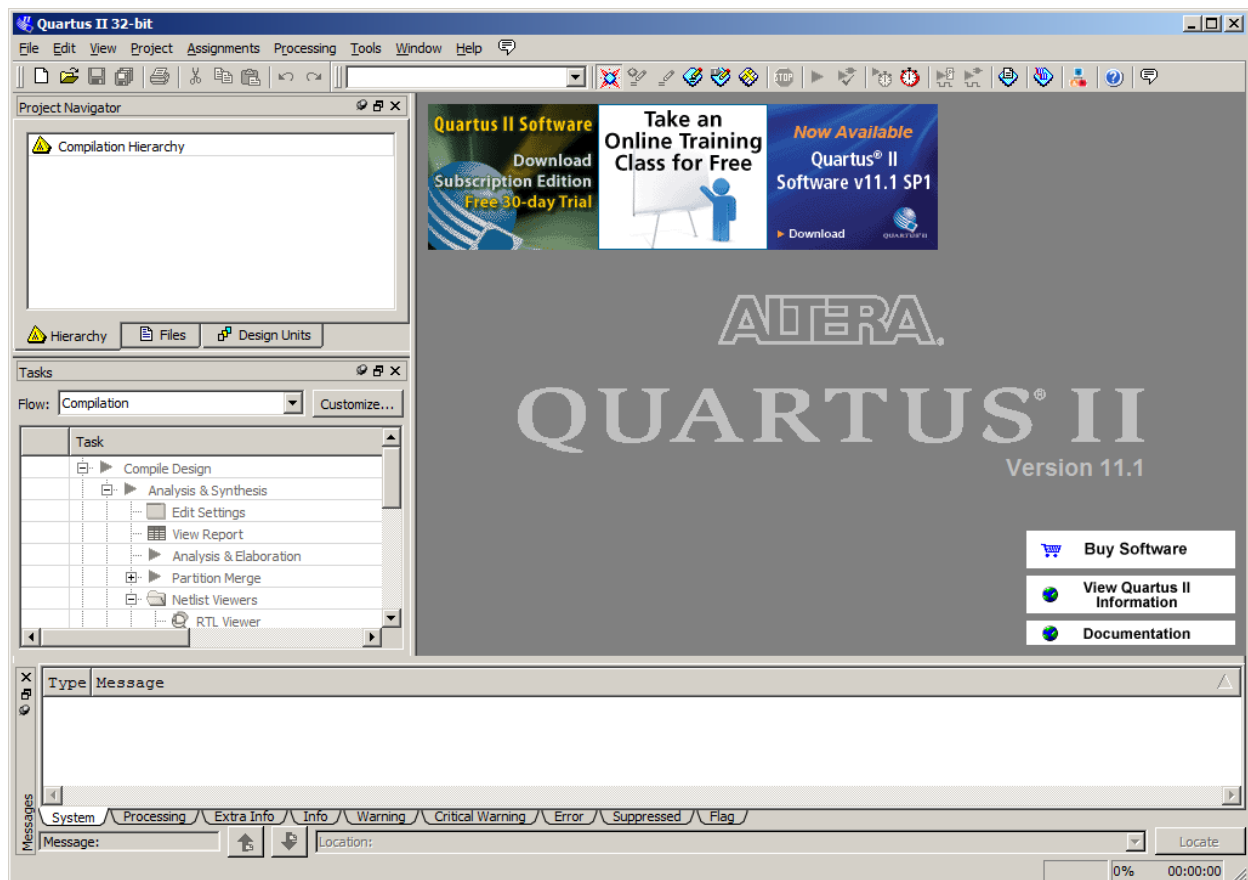
Figuur 4.1: *Quartus II pictogram*

Na het opstarten *kan* figuur 4.2 verschijnen. Via deze Getting Started kan je snel een project aanmaken of openen. We slaan dit scherm over. Klik hiervoor op het kruisje rechtsboven.



Figuur 4.2: Quartus II opstartscherm

De Project Manager wordt gestart (figuur 4.3).

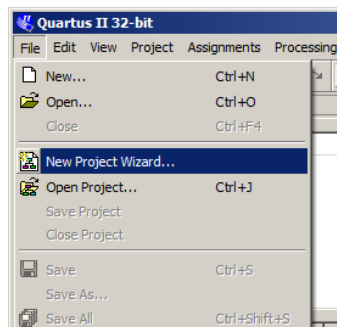


Figuur 4.3: Openingsscherm project navigator

Indien van toepassing, wordt het laatst geopende project geopend. De Project Manager bestaat uit menu's, knoppenbalk en vier vensters. Linksboven is de Project Navigator. Hierin worden alle (broncode-)bestanden en de onderlinge afhankelijkheden zichtbaar gemaakt, zoals hiërarchieën en testbenches. Daaronder is de Tasks-venster. Hierin worden de mogelijke taken bij een geselecteerd bestand weergegeven, zoals synthetiseren of implementeren. Rechts is het edit-venster waar bestanden bewerkt en bekeken kunnen worden. Onderaan is het Message-venster waar uitvoer van de diverse opdrachten wordt afgedrukt.

4.2 Project aanmaken

Quartus II werkt met zogenaamde projecten als beheerseenheid. In een project worden alle bestanden ondergebracht die je aanmaakt of die de software aanmaakt (bijv. output van de synthesizer). Een project is in feite niets anders dan een map op de harde schijf met daarin alle bijbehorende bestanden. Voor deze tutorial maken we een nieuw project aan. Klik in de Project Manager op **File→New Project Wizard**. Zie figuur 4.4.



Figuur 4.4: Project wizard

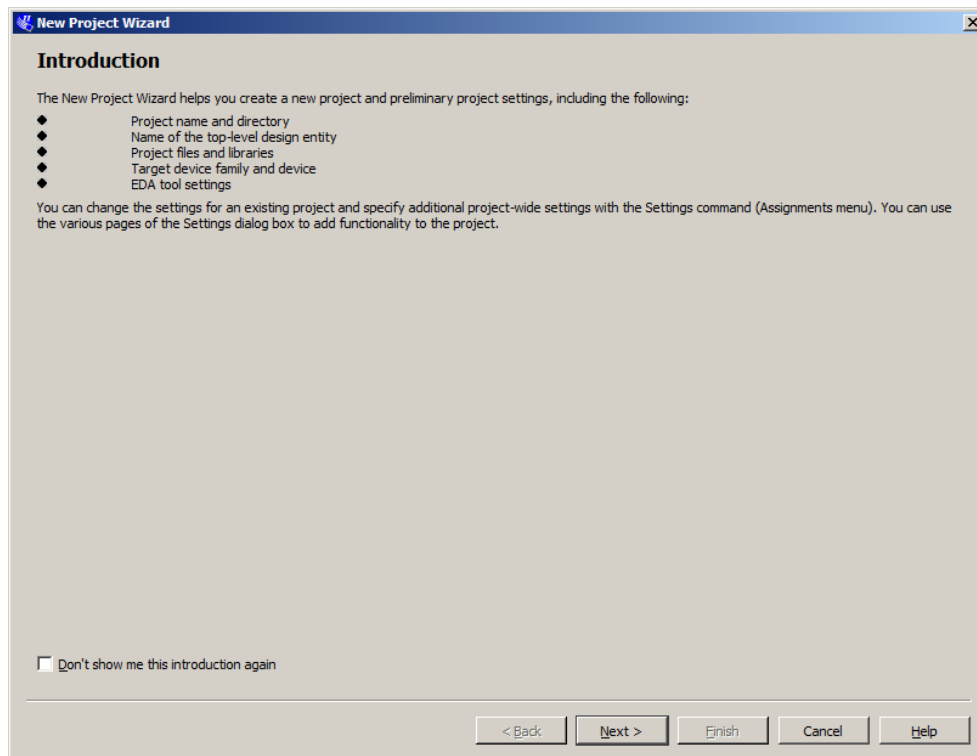
Er verschijnt een dialoogscherm zoals in figuur 4.5. Dit scherm geeft een introductie over de stappen die gaan volgen. Klik op **Next**.

Nu volgt een nieuw scherm (figuur 4-6). Hierin kan je de projectnaam, projectmap en projecttype instellen. Zorg ervoor dat je de projecten aanmaakt op de H-schijf en niet op een USB-stick; Quartus maakt veel bestanden aan en werken via stick verloopt dan traag.

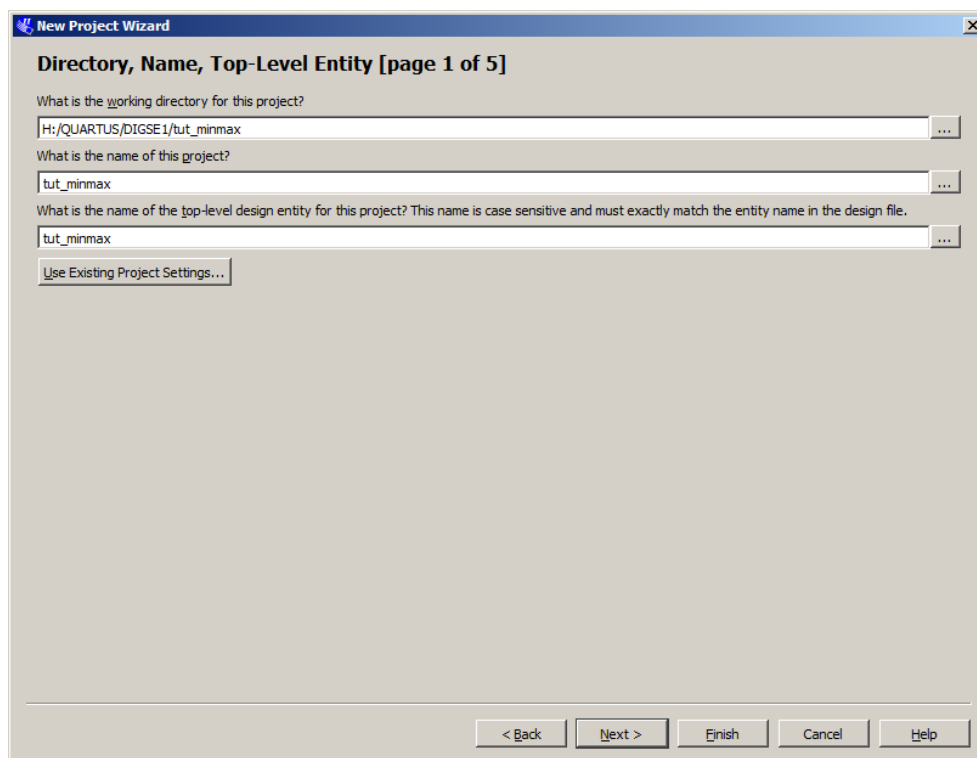
Gebruik geen spaties, leestekens of "vreemde"tekens in mapnamen en bestandsnamen!
Gebruik geen USB-stick.
Sla je bestanden op de H-schijf op.

Vul de velden in figuur 4.6 in zoals is aangegeven.

Let er op dat je de naam in het laatste veld goed invoert. Die naam heb je nodig bij de entity-beschrijving van je VHDL-beschrijving.

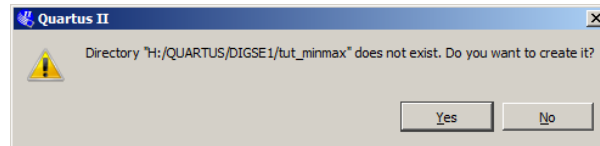


Figuur 4.5: Introductie project wizard



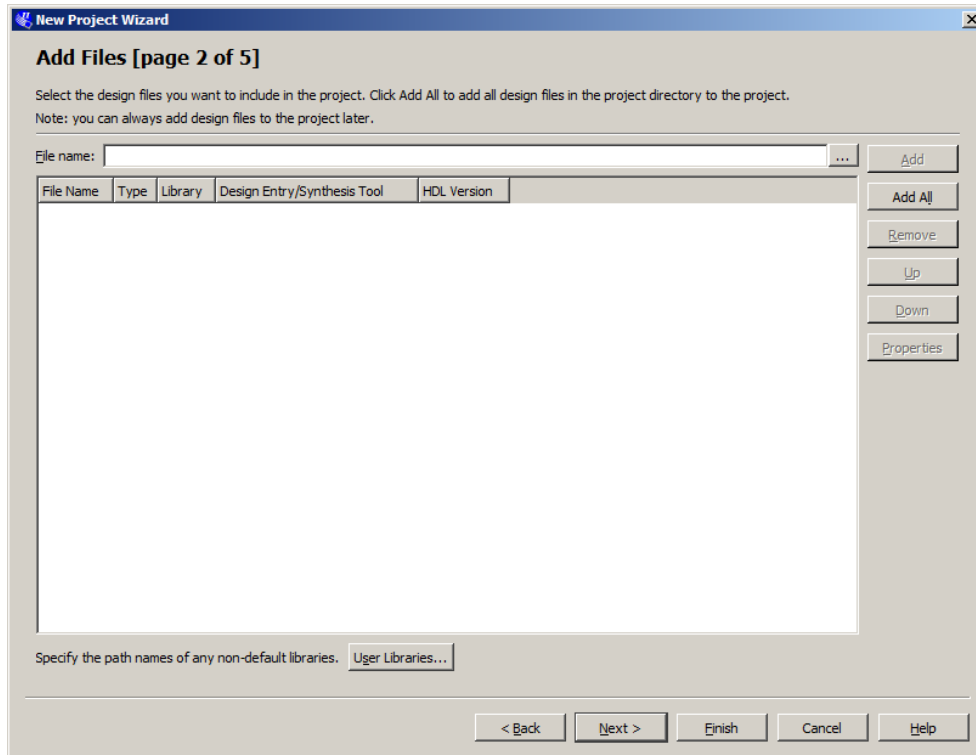
Figuur 4.6: Invullen mapnaam, projectnaam en naam van de top-level.

Na een klik op **Next** wordt nog gemeld dat de opgegeven map niet bestaat en wordt gevraagd of deze aangemaakt mag worden. Zie figuur 4.7.



Figuur 4.7: De map bestaat niet en moet worden aangemaakt.

Klik nu op **Yes**. In het volgende scherm kan je bestaande bestanden of bibliotheken (libraries) opgeven die je in het project wil opnemen (figuur 4.8). We maken hier geen gebruik van. Klik op **Next**.



Figuur 4.8: Toevoegen van al bestaande bestanden.

Nu wordt een vervolgscherm geopend waarin de device settings worden gevraagd. Het gebruikte type is een Cyclone III, FBGA (Fine Ball Grid Array) met 484 pinnen en Speed Grade 6. Vul eerst de velden in bij Device Family, Target Device en Show in 'Available devices' list zoals aangegeven in figuur 4.9. Daarna selecteer je in het veld Available Devices het type EP3C16F484C6, de eerste uit de lijst. Klik daarna op **Next**.

In het volgende scherm (figuur 4.10) kies je als simulatie-tool ModelSim-Altera met VHDL als taal. Zorg dat het vinkje bij Run gate level simulation automatically after compilation uit staat. Klik daarna op **Next**.

New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family

Family:

Devices:

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package:

Pin count:

Speed grade:

Name filter:

☒ Show advanced devices ☐ HardCopy compatible only

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL	3al Clo
EP3C16F484C6	1.2V	15408	347	516096	112	4	20
EP3C40F484C6	1.2V	39600	332	1161216	252	4	20
EP3C55F484C6	1.2V	55856	328	2396160	312	4	20
EP3C80F484C6	1.2V	81264	296	2810880	488	4	20

Companion device

HardCopy:

☐ Limit DSP & RAM to HardCopy device resources

< Back Next > Finish Cancel Help

Figuur 4.9: Invoeren van device-gegevens.

New Project Wizard

EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

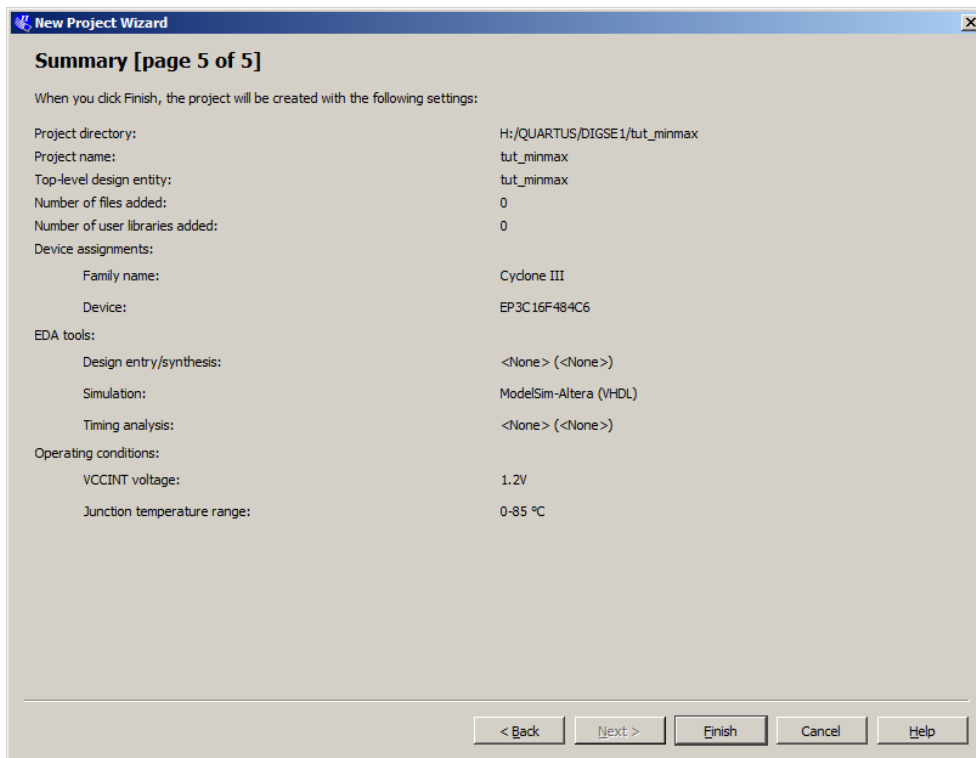
EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	<input type="checkbox"/> Run gate-level simulation automatically after compilation
Timing Analysis	<None>	<None>	<input type="checkbox"/> Run this tool automatically after compilation
Formal Verification	<None>	<None>	
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

< Back Next > Finish Cancel Help

Figuur 4.10: Invoeren van EDA tools.

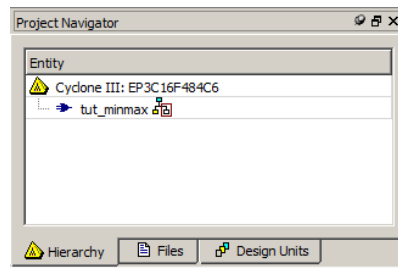
Het laatste scherm van de Project Wizard geeft een overzicht van de gemaakte keuzen. Zie figuur 4.11. Als er iets niet klopt kan je nog terug. Veel keuzen zijn in een later stadium nog aan te passen. Klik op **Finish** om de wizard af te sluiten.



Figuur 4.11: Opsomming van gemaakte keuzen in de project wizard.

In de titelbalk van de Project Manager verschijnt de naam van project. In de Project Navigator linksboven verschijnt de gebruikte Altera-component en daaronder het top-level design name. Zie figuur 4.12.

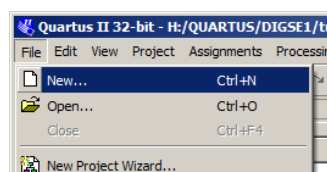
Opmerking: je kan later de top-level design name nog veranderen. Zie paragraaf 5.1.



Figuur 4.12: De top-level naam verschijnt in de Project Navigator.

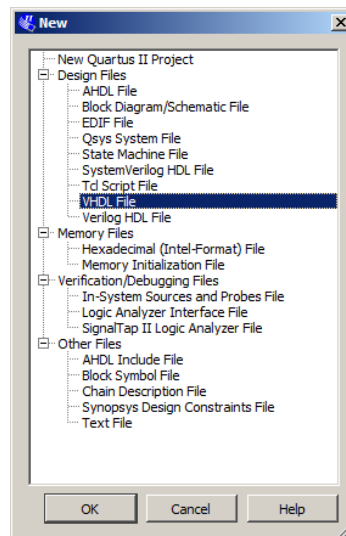
4.3 Invoeren van VHDL-broncode

Als eerste zullen we een VHDL-bestand in het project aanmaken. Klik in de Project Manager op **File**→**New** (figuur 4.13).



Figuur 4.13: Aanmaken nieuw bestand.

Nu verschijnt er een scherm waarin je het type van het nieuwe bestand kan kiezen (figuur 4.14). Kies voor VHDL file.



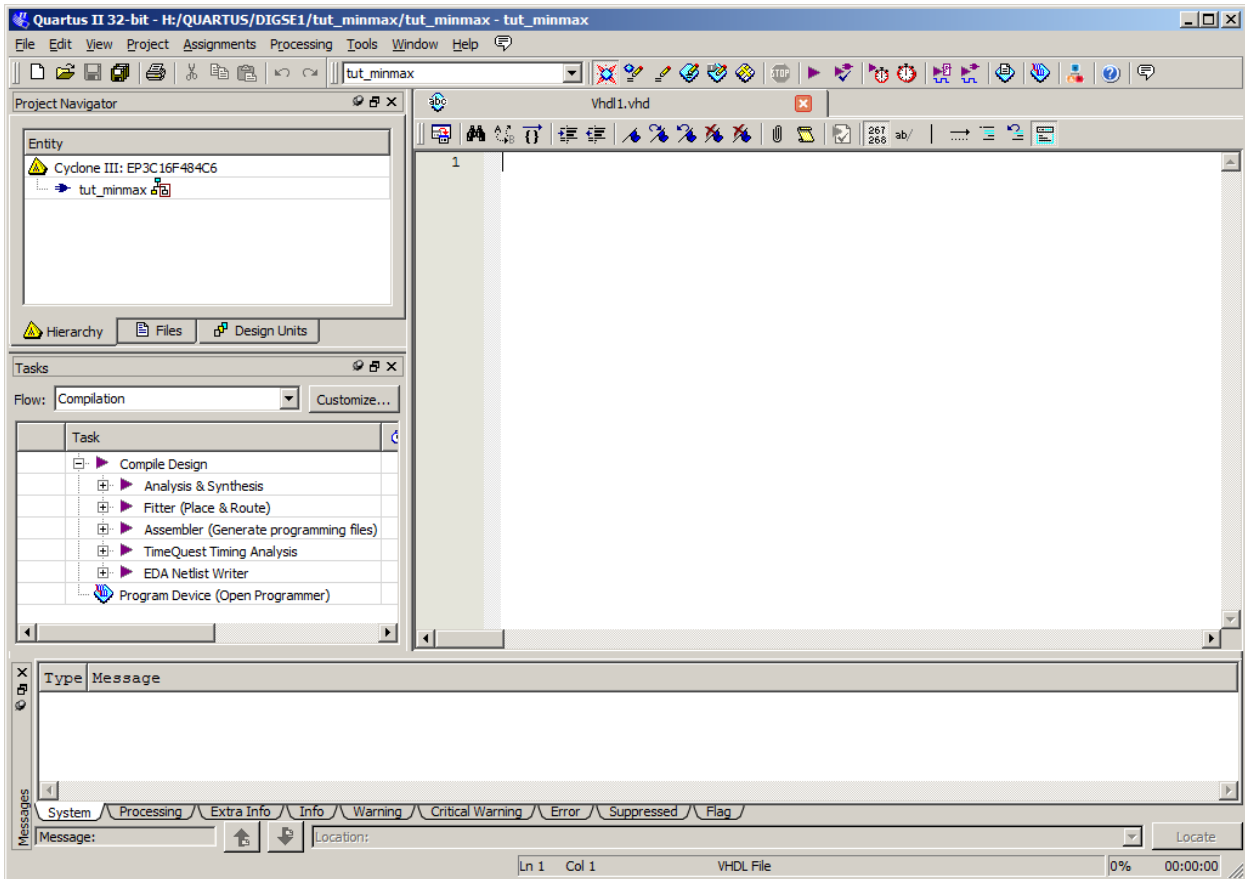
Figuur 4.14: *Keuze bestandstype.*

De Project Manager opent nu een leeg bestand. Dit is te zien in figuur 4.15. Merk op dat het bestand de tijdelijk naam `Vhd1.vhd` heeft. Bij het opslaan van het bestand kan alsnog een andere naam worden ingevoerd.

Voer nu de code in zoals is weergegeven in figuur 4.16. De VHDL-code in het edit-venster wordt overzichtelijk gehouden door het gebruik van kleuren. Groen is voor commentaar, blauw is voor zogenaamde keywords, roze is voor alles wat met het type `std_logic` te maken heeft, rood is voor getallen en zwart is voor de overige tekens en woorden.

Merk op dat de entity de naam `tut_minmax` heeft. Deze is eerder opgegeven als zogenaamde top level design entity (zie figuur 4.6).

Heb je enig idee wat de werking is van deze in VHDL beschreven hardware? Dat maakt het doorlopen van deze tutorial nog leuker.



Figuur 4.15: Overzicht Quartus IDE na aanmaken nieuw VHDL-bestand.

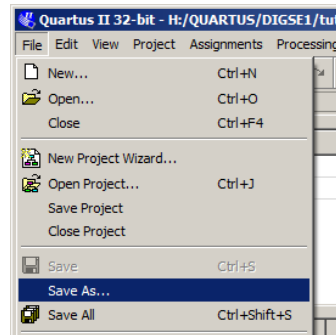
```

1  -- Name:      tut_minmax
2  -- Description: selects the minimum or maximum of two 4-bit integers
3  -- Date:      <the date>
4  -- Author:    <your name>
5  --
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9
10 entity tut_minmax is
11     port (a, b : in std_logic_vector(3 downto 0);
12           minmax : in std_logic;
13           f : out std_logic_vector(3 downto 0)
14           );
15 end entity tut_minmax;
16
17 architecture csa of tut_minmax is
18     signal min, max : std_logic_vector(3 downto 0);
19 begin
20     min <= a when a<b else b;
21     max <= a when a>b else b;
22     f <= min when minmax = '0' else max;
23 end architecture csa;
24

```

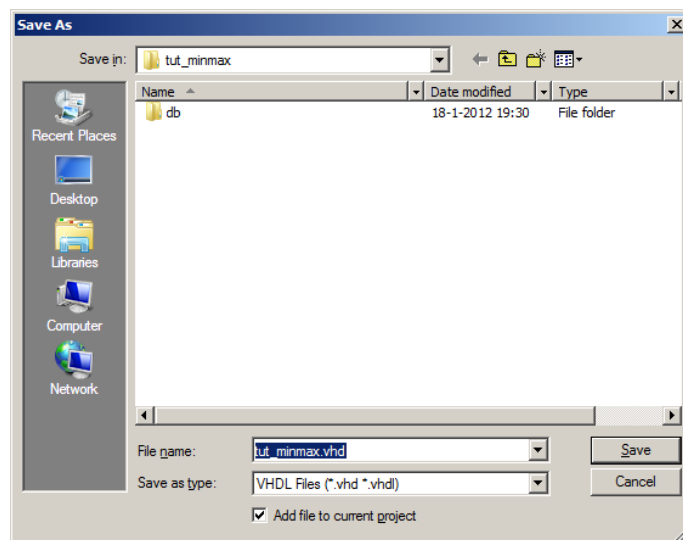
Figuur 4.16: Screenshot van de in te voeren VHDL-code.

Nu de code is ingevoerd, moet het bestand opgeslagen worden. Klik in de Project Manager op **File→Save** (zie figuur 4.17).



Figuur 4.17: Opslaan VHDL-bestand.

Er wordt een scherm geopend waarin de juiste map en de bestandsnaam ingevuld kunnen worden. Zie figuur 4.18. Er wordt een bestandsnaam voorgesteld, dit kan je wijzigen. Let op het vinkje bij Add file to current project.

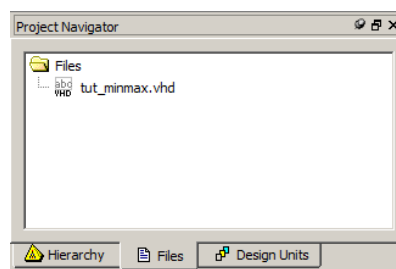


Figuur 4.18: Opgeven bestandsnaam VHDL-bestand.

Noot: je mag geen spaties, leestekens of "vreemde"tekens in de bestandsnaam opnemen!

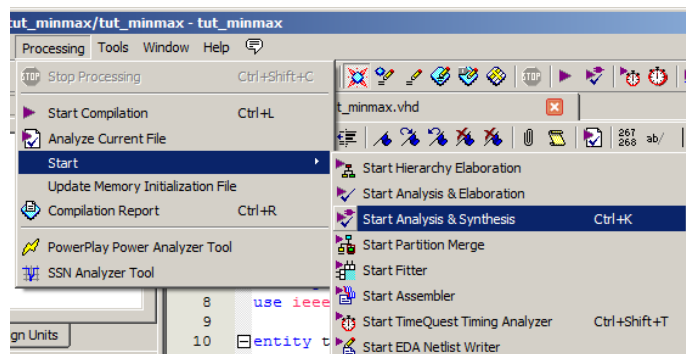
Noot: let goed op de map waarin het bestand opgeslagen wordt. Quartus wil nog wel eens de map van een eerder geopend project presenteren.

Het bestand is nu terug te vinden in de Project Navigator onder het tabblad Files. Zie figuur 4.19.



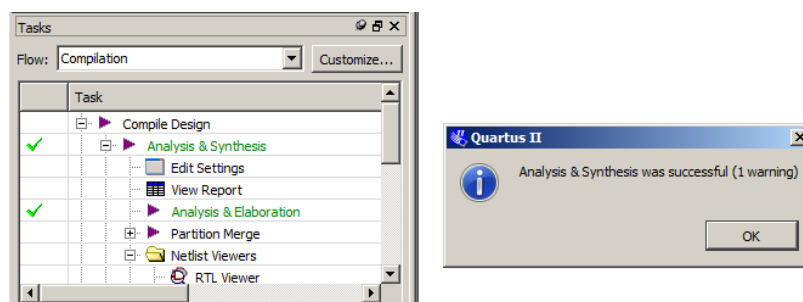
Figuur 4.19: Het opgeslagen bestand staat in de lijst van bestanden.

Om te controleren of de VHDL-code syntactisch correct is en er hardware voor kan worden gegenereerd zullen we de synthesizer starten. Klik in de Project Manager op **Processing**→**Start**→**Start Analysis & Synthesis** of gebruik de sneltoetscombinatie **Ctrl+K**. Zie figuur 4.20.



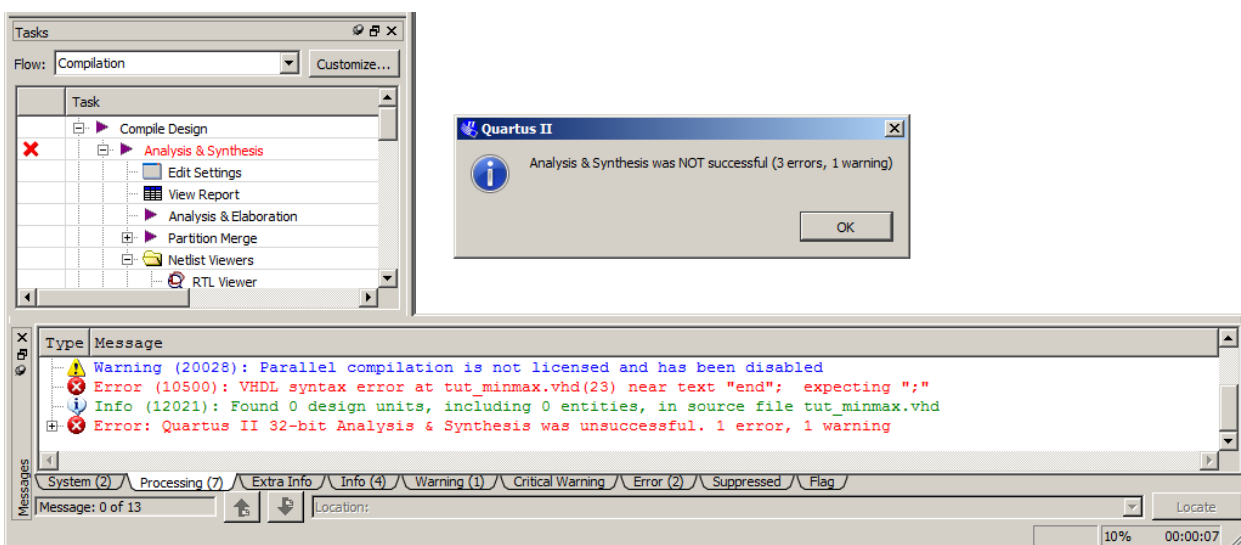
Figuur 4.20: Starten Analysis & Synthesis vanuit het menu.

Als deze stap gelukt is zie je onder Tasks een paar vinkjes verschijnen en je krijgt een melding (figuur 4.21). Klik op **OK** om verder te gaan.



Figuur 4.21: De analyse is gelukt.

Mocht je een fout hebben gemaakt, bijvoorbeeld het vergeten van een punt-komma, dan zal de analyser dat melden, zie figuur 4.22.



Figuur 4.22: De analyse is mislukt.

4.4 Simulatie VHDL-broncode

Nu de code geschreven is, dient het gesimuleerd te worden. Dit droogzwemmen is bedoeld om te kunnen verifiëren of de schakeling, en dus de VHDL-code, werkt volgens de specificaties. Het Quartus II pakket gebruikt hiervoor de externe simulator ModelSim van firma ModelTech. We gebruiken de simulator om aan te tonen dat onze VHDL-code functioneel correct is. Het kan namelijk best zijn dat de VHDL-code gesynthetiseerd kan worden, maar dat de schakeling niet doet wat het moet doen. Bij deze simulatie worden geen vertragingstijden meegenomen.

Voor simulatie zijn twee bestanden nodig: een testbench en een commandobestand. Een testbench is een VHDL-bestand met daarin stimuli dat voor simulatiedoeleinden is geschreven. Je beschrijft hier de waarden van de ingangen (ports) in de tijd gezien. Het is een sequentie van toekenning. Het tweede bestand is een zogenaamd commandobestand: hierin staan opdrachten voor de simulator. De commando's zijn onderdeel van de script-taal Tcl ("Tickle"). Je kan deze opdrachten ook interactief op een commandoregel invoeren, maar vaak wil je de simulatie een paar keer opnieuw draaien. Dan is het steeds invoeren van de (zelfde reeks) commando's een tijdrovende zaak.

Noot: de hier beschreven wijze van simuleren wijkt een klein beetje af van de gebruikelijke methode zoals dit normaal in Quartus wordt gedaan. Verderop in de tutorial wordt hier nog wat meer over beschreven.

De testbench begint met een wait-statement van 10 ns (nanoseconden; simulatietijd, geen echte tijd). Merk op dat eerst gewacht wordt; de ingangen van de VHDL-beschrijving zijn dan nog niet gedefinieerd. Dat is straks ook terug te zien in de simulatieresultaten. Na 10 ns wordt ingang A op 1011_2 (11_{10}) gezet (merk op dat dit vier bits zijn) en ingang B op 1001_2 (9_{10}) gezet. Ingang minmax wordt op 0 gezet. Er wordt weer 10 ns gewacht. Zo worden alle combinaties met A, B en minmax gesimuleerd. De laatste wachtopdracht zorgt ervoor dat de simulator zal stoppen.

Maak een nieuw broncodebestand aan via het menu **File**→**New** en kies VHDL File (zie de figuren 4.13 en 4.14). Er wordt nu een leeg scherm geopend. Vul hier de testbenchcode uit listing 4.1 op pagina 26 in. Geef het bestand de naam `tb_tut_minmax.vhd` en sla het op. Zie figuur 4.23.

```

-- Name:          tb_tut_minmax
-- Description: testbench for tut_minmax
-- Date:          <the date>
-- Author:        <your name>
--

library ieee;
use ieee.std_logic_1164.all;

-- Empty entity
entity tb_tut_minmax is
end tb_tut_minmax;

architecture sim of tb_tut_minmax is
-- Top level signals
signal a, b      : std_logic_vector(3 downto 0);
signal minmax    : std_logic;
signal f         : std_logic_vector(3 downto 0);

-- Component declaration
component tut_minmax is
    port (a, b      : in std_logic_vector(3 downto 0);
          minmax    : in std_logic;
          f         : out std_logic_vector(3 downto 0)
        );
end component tut_minmax;

begin
    -- Component instantiation
    dut : tut_minmax
        port map (a => a, b => b, minmax => minmax, f => f);

    -- Process that asserts the stimuli
    stim: process is
    begin
        wait for 10 ns;
        a <= "1011"; b <= "1001"; minmax <= '0';
        wait for 10 ns;
        minmax <= '1';
        wait for 10 ns;
        a <= "0011"; b <= "1100"; minmax <= '0';
        wait for 10 ns;
        minmax <= '1';
        wait;    -- wait forever
    end process;

end sim;

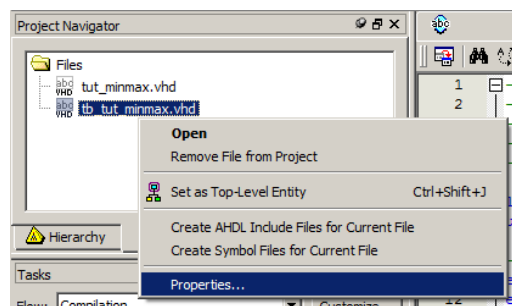
```

Listing 4.1: VHDL-testbench



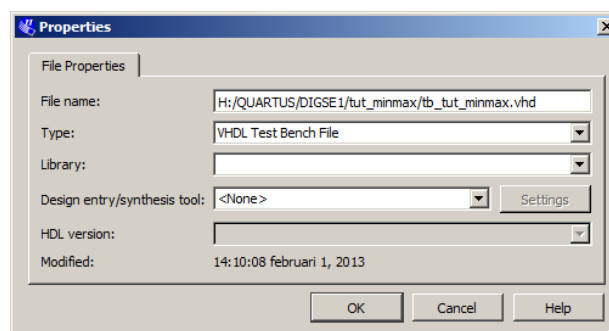
Figuur 4.23: Aanmaken VHDL-testbench-bestand.

Het bestand wordt nu echter gezien als een VHDL-broncodebestand en niet als een testbench. Hiervoor moeten we het type wijzigen. Klik in de Project navigator met de **rechter** muisknop op de naam `tb_tut_minmax.vhd` om een menu te openen. Selecteer **Properties** in het menu (figuur 4.24).



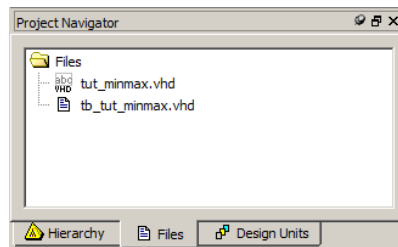
Figuur 4.24: Aanpassen bestandseigenschappen.

Selecteer in het veld Type: de optie VHDL Test Bench File. Klik dan op **Ok** (figuur 4.25).



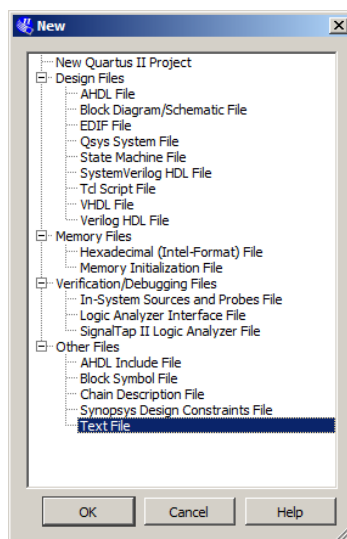
Figuur 4.25: Bestandstype veranderd naar VHDL-testbench.

In de Project Navigator is nu zichtbaar dat het type is veranderd (figuur 4.26).



Figuur 4.26: *Overzicht bestanden en bestandstypen.*

De volgende stap is het aanmaken van het commandobestand. Klik in de Project Manager op menu **File**→**New** en kies Text File (zie de figuren 4.13 en 4.27). Er wordt nu een leeg scherm geopend.



Figuur 4.27: *Aanmaken ModelSim commando-script.*

Vul de code in zoals is weergegeven in listing 4.2 op pagina 29.

Noot: in de code zijn twee padnamen opgenomen die beginnen met punten (`../..`). Dit is een verwijzing naar twee hoger gelegen mappen. Quartus plaatst allerlei bestanden in de map genaamd `simulation/modelsim` die onder de project-map geplaatst is en start de simulatie vanaf dat punt. Wij hebben echter onze testbench en commandobestand in project-map geplaatst. Vandaar dat bij de compileeropdrachten dus een dubbele verwijzing naar een hoger gelegen padnaam moet worden gebruikt.

Sla het bestand op, maar let op! De extensie moet in `.do` veranderd worden! Zie figuur 4.28.

```

# Name:          tb_tut_minmax.do
# Description:   script for running simulation
# Date:          <the date>
# Author:        <your name>

# Set transcript on
transcript on

# Recreate the work directory and map to work
if {[file exists rtl_work]} {
    vdel -lib rtl_work -all
}
vlib rtl_work
vmap work rtl_work

# Compile the Tutorial description and testbench. Note the double
# parent references in the path name
vcom -93 -work work ../../tut_minmax.vhd
vcom -93 -work work ../../tb_tut_minmax.vhd

# Start the simulator with 1 ns time resolution
vsim -t 1ns -L rtl_work -L work -voptargs="+acc" tb_tut_minmax

# Log all signals in the design, good if the number of signals is small.
add log -r *

# Add all toplevel and simulated device signals to the list view
add list *
add list dut/*

# Add all toplevel signals and a number of signals inside
# the simulated design to the wave view
add wave -divider "Inputs"
add wave a
add wave b
add wave minmax
add wave -divider "Internals"
add wave dut/min
add wave dut/max
add wave -divider "Outputs"
add wave f

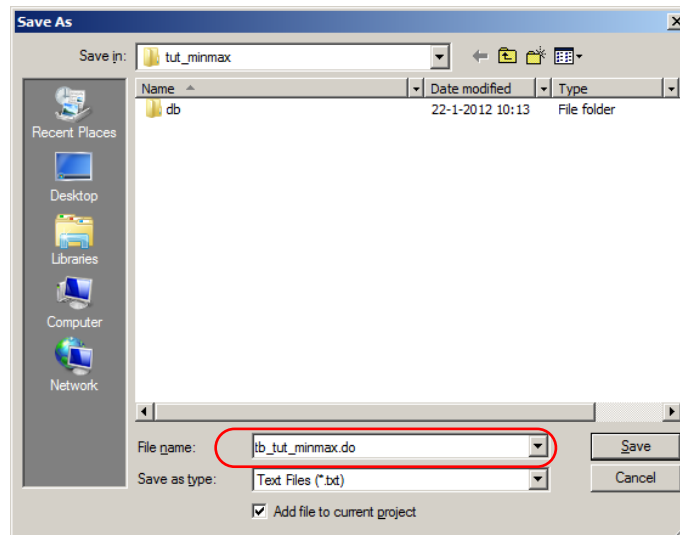
# Open the List and Waveform window
view list
view wave

# Run simulation for 50 ns
run 50 ns

# Fill up the waveform in the window
wave zoom full

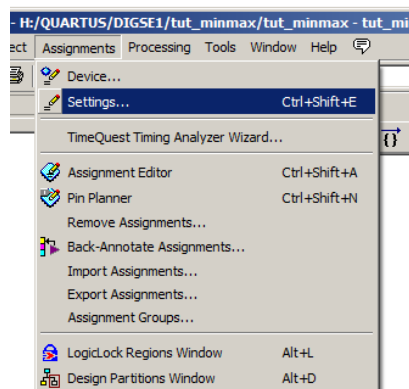
```

Listing 4.2: ModelSim command script



Figuur 4.28: Naamgeving ModelSim commando-script aanmaken.

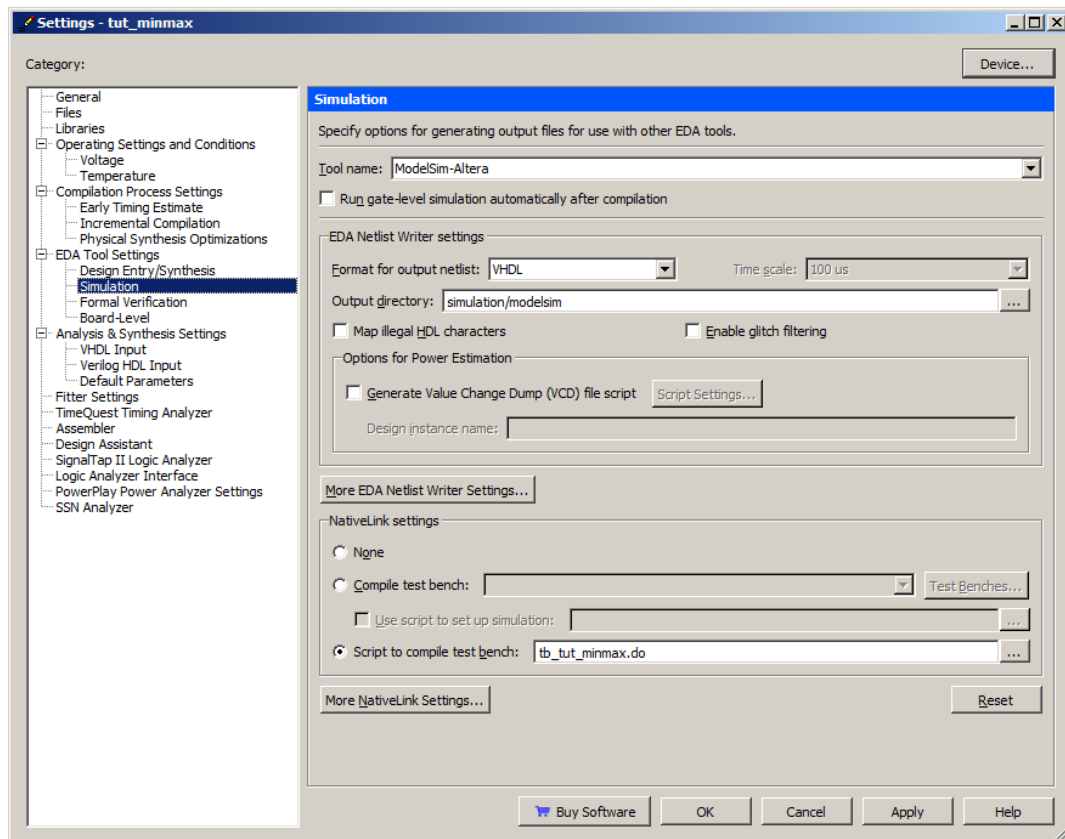
Nu moeten we de simulatie-omgeving instellen. Selecteer via het menu **Assignments**→**Settings** of gebruik de sneltoetscombinatie **Ctrl+Shift+E**. Zie figuur 4.29.



Figuur 4.29: Project settings aanpassen.

Er wordt nu een nieuw scherm geopend waarin de instellingen van het project gewijzigd kunnen worden. Selecteer in het gedeelte Category de optie EDA Tool Settings - Simulation (zie figuur 4.30). Aan de rechterkant verschijnen de instellingen voor de simulatie. De toolnaam moet op ModelSim-Altera staan de optie **Run gate-level simulation** moet uit staan.

Vink onder NativeLink Settings de optie Script to compile testbench aan. In het bijbehorende veld moet je de naam van het commandobestand invullen. Je kunt ook het bestand selecteren door op de knop met de drie puntjes te klikken. Er wordt dan een venster geopend waarin je een bestand kan selecteren (geen figuur).

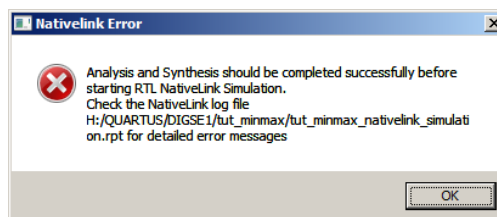


Figuur 4.30: Invoeren settings voor simulatie met ModelSim

Klik daarna op **Apply** en **OK**. Het scherm wordt afgesloten.

Nu moet de simulatie gestart worden. Klik in de Project Manager op de menu-optie **Tools→Run Simulation Tool→RTL Simulation**. De simulator wordt gestart via Quartus.

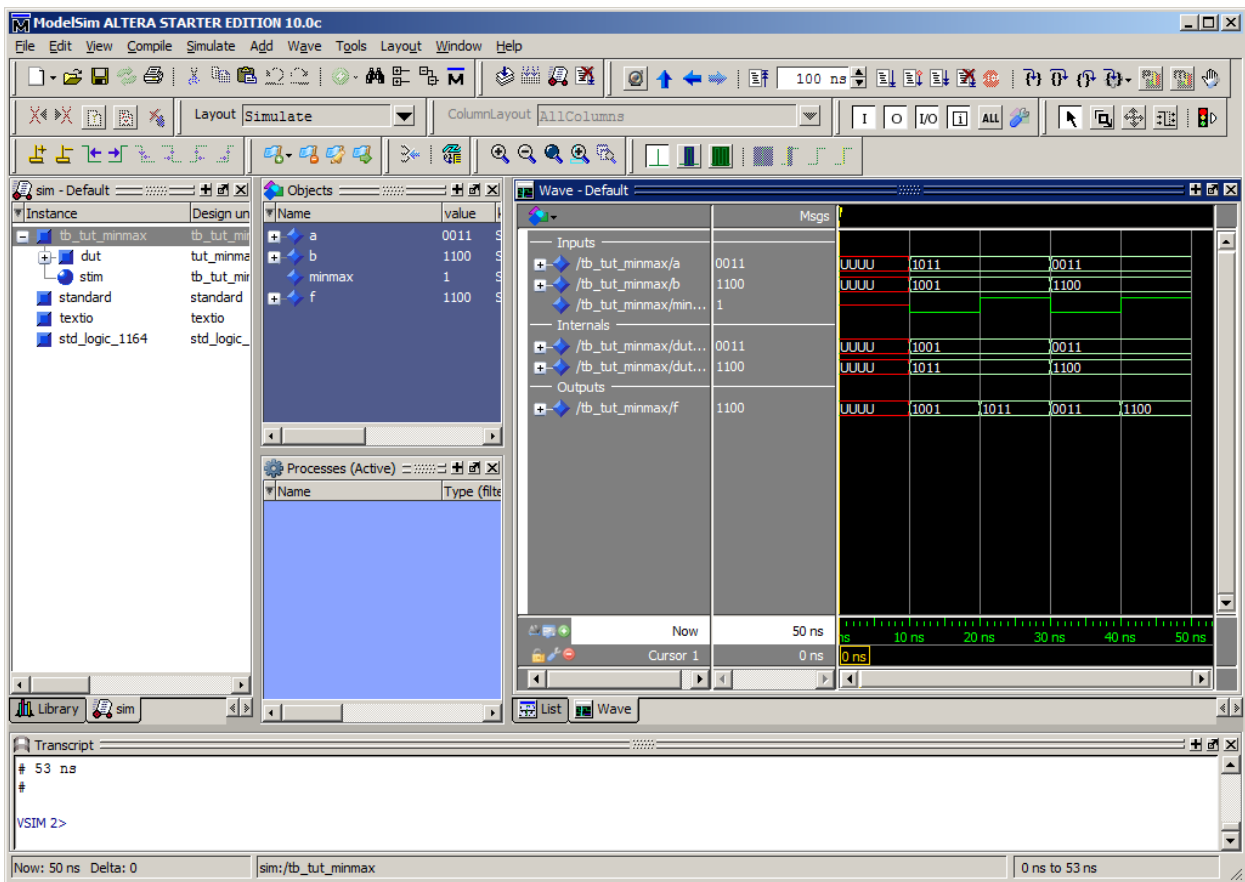
Noot: voordat de simulator gestart kan worden, moet de beschrijving geanalyseerd en gesynthetiseerd zijn. Zie de figuren 4.20 tot en met 4.22. Als dat niet gebeurt is krijg je de volgende foutmelding (figuur 4.31).



Figuur 4.31: Foutmelding dat synthese nog niet gelukt is.

Noot: voordat de simulator gestart kan worden, moet je eerst het pad naar de simulator instellen, anders volgt een foutmelding. Zie paragraaf 5.2.

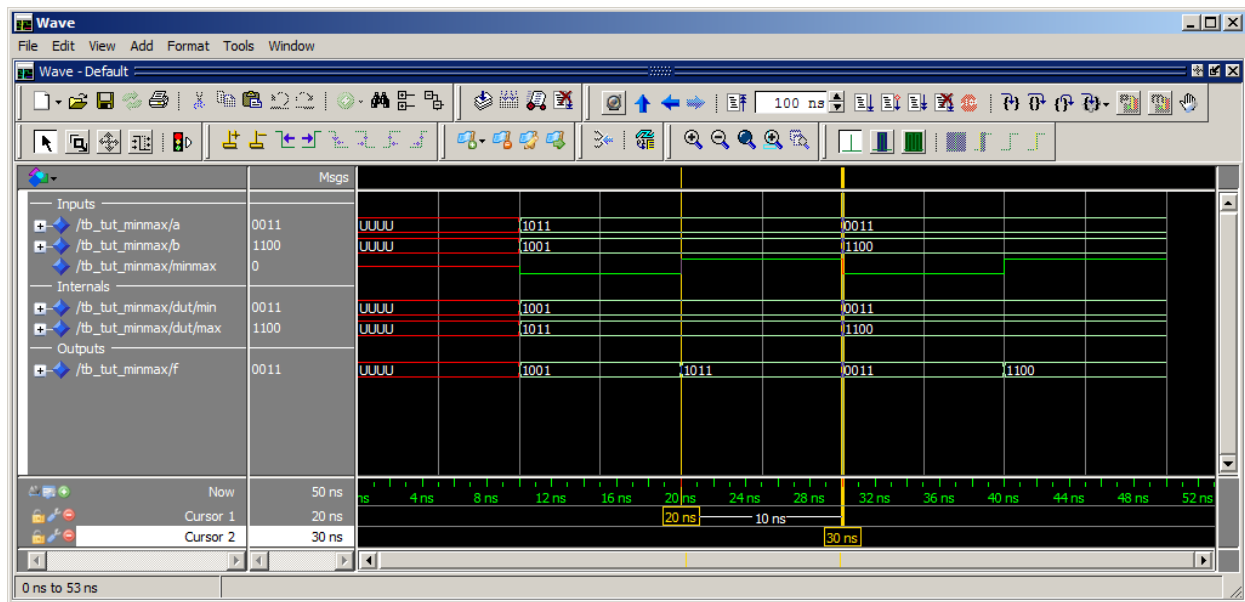
Tijdens het starten van Modelsim wordt een *splashscreen* getoond. De simulator wordt gestart (dit kost enige tijd) en er worden vijf vensters geopend (zie figuur 4.32).



Figuur 4.32: Overzicht ModelSim IDE na het uitvoeren van het commando-script.

Onderaan is het *Transcript Window*. Hier kan je ook losse commando's geven (voor gevorde-
ren). Rechts is het *Waveform Window*. De overige drie zijn niet nu niet van belang.

In figuur 4.33 is de *Waveform Window* te zien. Hierin wordt een tijddiagram afgebeeld van
de simulatie. Het beste kan je dit venster maximaliseren, zeker als er veel signalen zijn en er
voor langere tijd gesimuleerd wordt.



Figuur 4.33: Timingsdiagram van de VHDL-code.

Gebruik de menuoptie **Wave**→**Zoom**→**Zoom Full** om het venster geheel te vullen met het tijdsdiagram. Je sluit de simulator af door het hoofdvenster af te sluiten.

Je kan het commandobestand nogmaals uitvoeren door in het transcript window op de ↑-toets te drukken. Het laatst ingevoerde commando verschijnt dan. Druk op de **enter**-toets om dat commando uit te voeren. Let op de vreemde naam van het commandobestand. Dit komt door de wijze waarop Quartus en Modelsim samenwerken. Zie figuur 4.34.

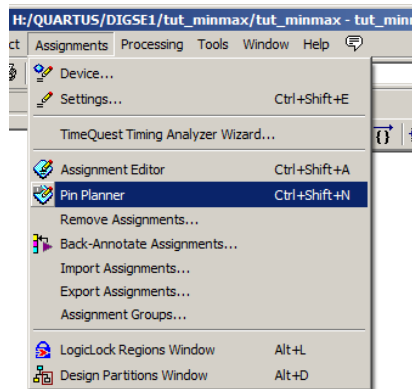


Figuur 4.34: Opnieuw uitvoeren van het commando-script.

4.5 Pinnen toekennen

Voordat we aan de implementatie gaan beginnen, moeten we eerst de fysieke pinnen koppelen aan de in- en uitgangen van de VHDL-beschrijving. Dit gebeurt met het onderdeel *Pin Planner*. Helaas weet de Pin Planner nog niet welke in- en uitgangen het ontwerp heeft. Pas na synthese zijn deze bekend. Gelukkig hebben we dit al eerder gedaan. Zie de figuren 4.20 tot en met 4.22.

Start nu de Pin Planner via de menuoptie **Assignments**→**Pin Planner** in de Project Manager (zie figuur 4.35) of gebruik de sneltoetscombinatie **Ctrl+Shift+N**.



Figuur 4.35: *Starten van de Pin Planner*

In de Pin Planner kunnen de fysieke pinnen aan de in- en uitgangen van de VHDL-beschrijving gekoppeld worden. De fysieke pinnen zijn zelf weer verbonden met schakelaars en leds. In de twee onderstaande tabellen zijn de koppelingen weergegeven (tabellen 4.1 en 4.2).

Tabel 4.1: *Koppelingen ingangen aan pinnen en schakelaars*

Ingang	Pinnaam	Schakelaar
a[3]	D2	SW9
a[2]	E4	SW8
a[1]	E3	SW7
a[0]	H7	SW6
b[3]	G4	SW3
b[2]	H6	SW2
b[1]	H5	SW1
b[0]	J6	SW0
minmax	J7	SW5

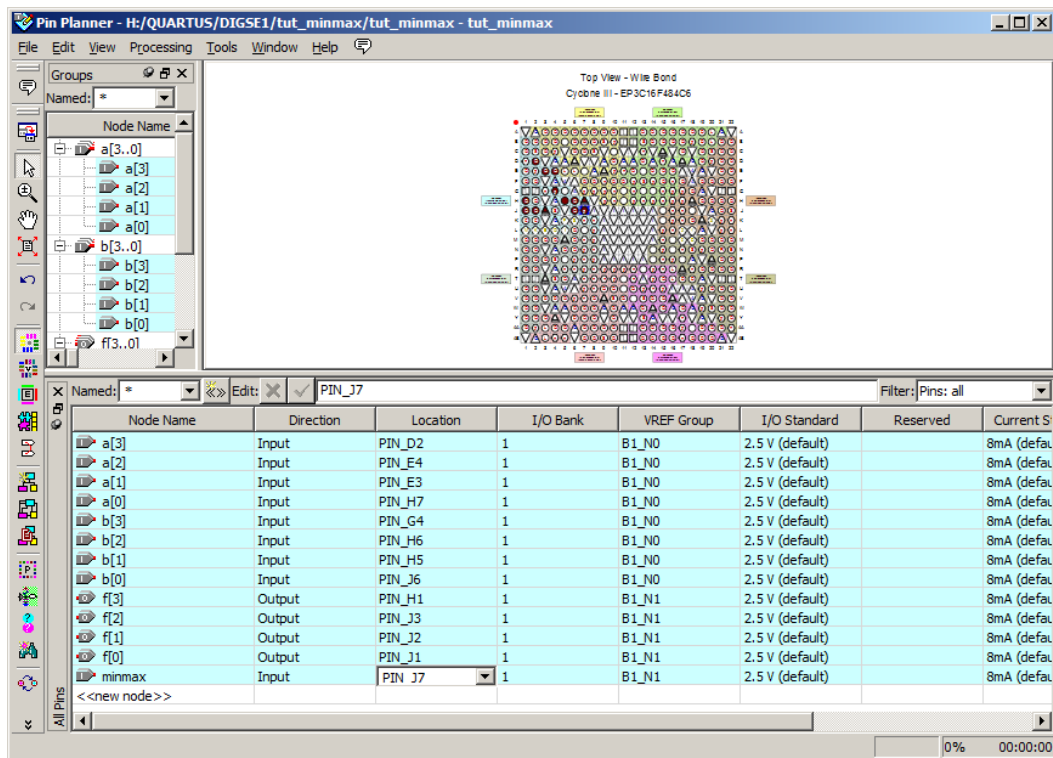
Tabel 4.2: *Koppelingen uitgangen aan pinnen en schakelaars*

Uitgang	Pinnaam	Schakelaar
f[3]	H1	LEDG3
f[2]	J3	LEDG2
f[1]	J2	LEDG1
f[0]	J1	LEDG0

Vul de pinnamen in volgens de tabellen 4.1 en 4.2. Zie figuur 4.36.

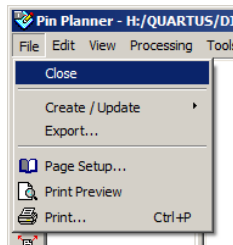
Opmerking: je hoeft alleen het laatste deel van een naam in te voeren, dus D2, E4 etc. Quartus zal het woord PIN_ voor de pinnaam zetten, dus J2 wordt dan PIN_J2.

Noot: in bijlage B is een complete lijst opgenomen met daarin de koppelingen tussen de fysieke pinnen en de schakelaars, drukknoppen, leds en 7-segments displays.



Figuur 4.36: Overzicht van de Pin Planner IDE met pinbenaming ingevuld.

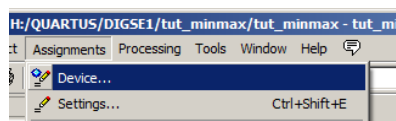
Sluit de Pin Planner af via **File**→**Close**. Je hoeft de invoer niet op te slaan, dat gebeurt automatisch (er is trouwens geen save as-optie). Zie figuur 4.37.



Figuur 4.37: Afsluiten Pin Planner.

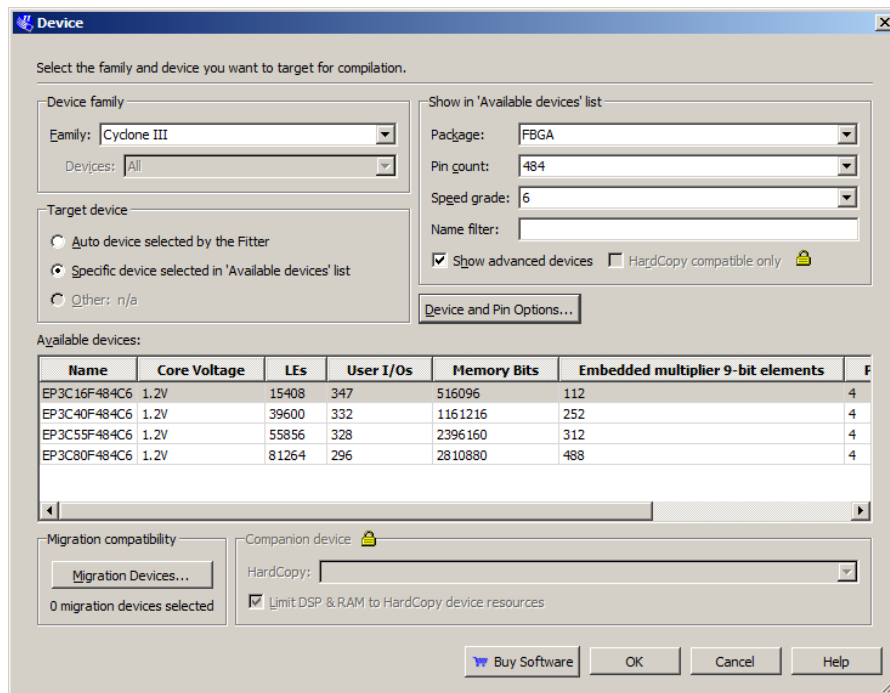
Van alle pinnen worden er maar een paar gebruikt. De ongebruikte pinnen worden standaard in tri-state met pull-up gezet. Dat is een veilige toestand maar sommige leds gaan dan zwak branden. Dat kan tot verwarring leiden. De standaard toestand van de pinnen moet veranderd worden in tri-state.

Selecteer in de Program Manager **Assignments**→**Device**. Zie figuur 4.38.



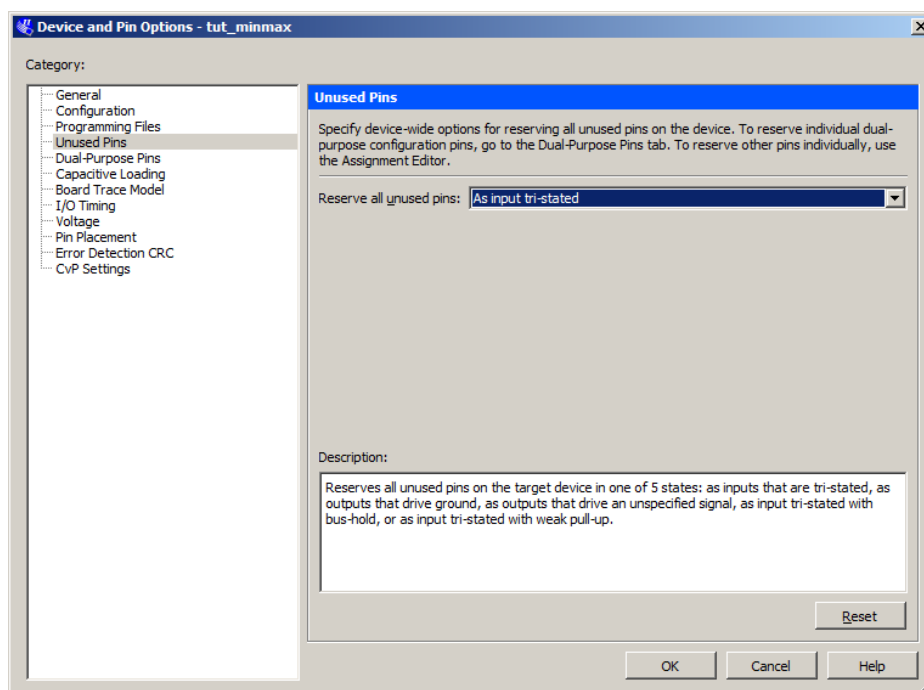
Figuur 4.38: Starten van de Device Settings.

Er wordt een dialoog geopend (zie figuur 4.39). Selecteer daar de knop **Device and Pin Options**.



Figuur 4.39: Overzicht IDE van de Device Settings.

Er wordt een nieuw dialoog geopend, zie figuur 4.40. Selecteer onder Category: de optie Unused Pins. Aan de rechterkant in nu een dropdown-box zichtbaar met de naam Reserve all unused pins. Selecteer de optie As input tri-stated. Klik op **OK**. De dialoog wordt afgesloten en je komt weer terug bij het eerste dialoog. Sluit deze af door op **OK** te klikken.

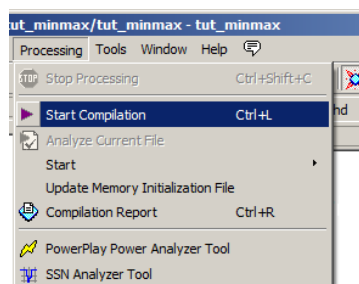


Figuur 4.40: Selecteren van de opties voor de niet-gebruikte pinnen.

4.6 Compilatie

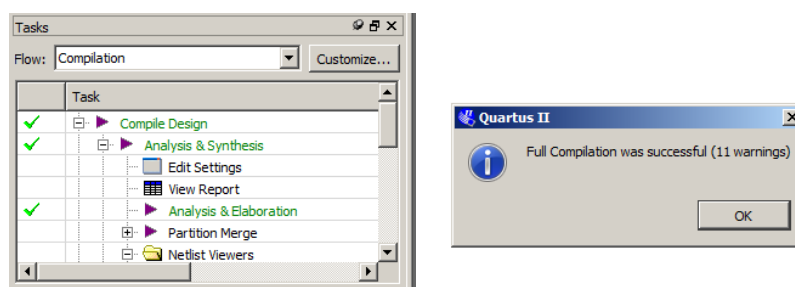
Nu de pinaansluitingen ingevoerd zijn, wordt de VHDL-beschrijving gecompileerd. Compileren valt uiteen in twee delen: synthese en implementatie. Synthese houdt in dat de VHDL-code wordt vertaald met als resultaat een netlist; een beschrijving van de digitale logica in primitieven. Denk hierbij aan poorten, flipflops, LUTs (LookUp Table, ROM) of speciale voorzieningen zoals Phase Locked Loops of klokbuffers. Deze primitieven zijn voor elk configureerbaar type weer anders. Bij het aanmaken van een nieuw project hebben we al opgegeven welke component we gaan gebruiken. Implementatie houdt in dat de primitieven worden *gemapped* op de LE's. In deze fase wordt ook rekening gehouden met pinaansluitingen en timing.

Start de compilatie via de menuoptie **Processing→Start Compilation** of gebruik de sneltoetscombinatie **Ctrl+L**. Zie figuur 4.41.



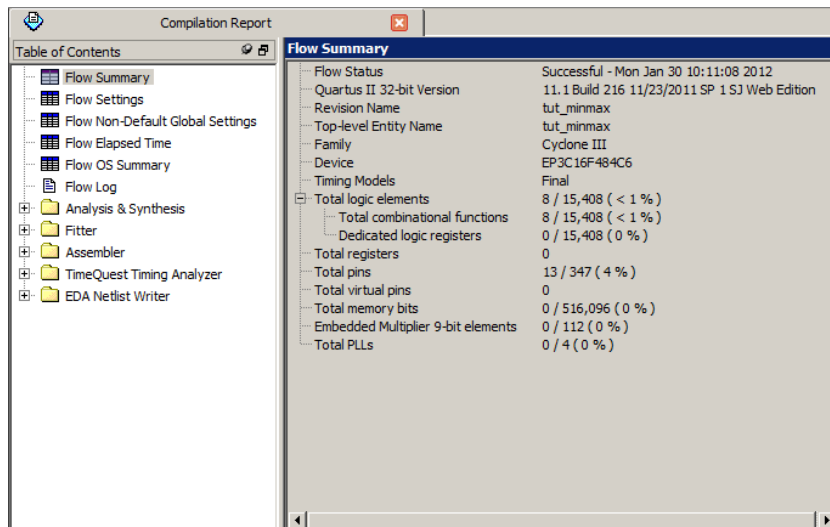
Figuur 4.41: Starten van de compilatie.

De compilatie wordt nu gestart. Dat kan afhankelijk van het ontwerp enige tijd duren. Je kan de voortgang in het **Tasks**-venster. Als de compilatie geen fouten oplevert krijg je een melding zoals in figuur 4.42. In het algemeen zijn de waarschuwingen niet problematisch, maar kijk de melding toch even na.



Figuur 4.42: De compilatie is gelukt.

Je krijgt na compilatie een rapport met alle verrichte werkzaamheden. Een interessant onderdeel hiervan is het aantal gebruikte LE's. Hieraan kan je zien hoe groot je ontwerp is. Zie figuur 4.43.



Figuur 4.43: Overzicht van het resultaat van de compilatie.

De programmatuur is ontwikkeld door Quartus. Er zijn ook andere *tool chains* verkrijgbaar die VHDL-code met hoog abstractieniveau kunnen compileren. Die van Quartus kan dat niet; alleen RTL-beschrijvingen (Register Transfer Level) zijn mogelijk. We zullen ons hier aan houden.

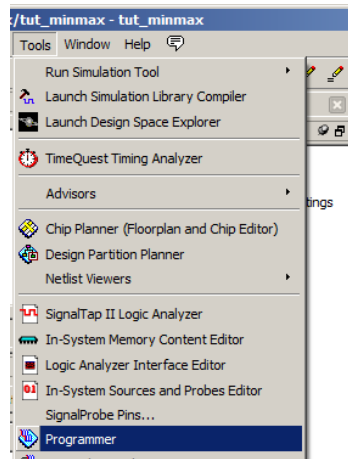
4.7 Configureren van de Cyclone III

De compilatiestap is nu afgerond: we kunnen verder gaan met het configureren van de Cyclone III. We gaan dus het configuratiebestand in deze component laden. Dat gaat volgens het JTAG- protocol. Dit protocol stelt de gebruiker in staat een hele keten van componenten te configureren. Dit is erg handig omdat je zo updates in de componenten kan laden, ook al zijn ze al op een print gemonteerd.

Zorg ervoor dat de USB-kabel juist is aangesloten en het ontwikkelbord is aangezet. Als je dit vergeet, zal de software een foutmelding geven.

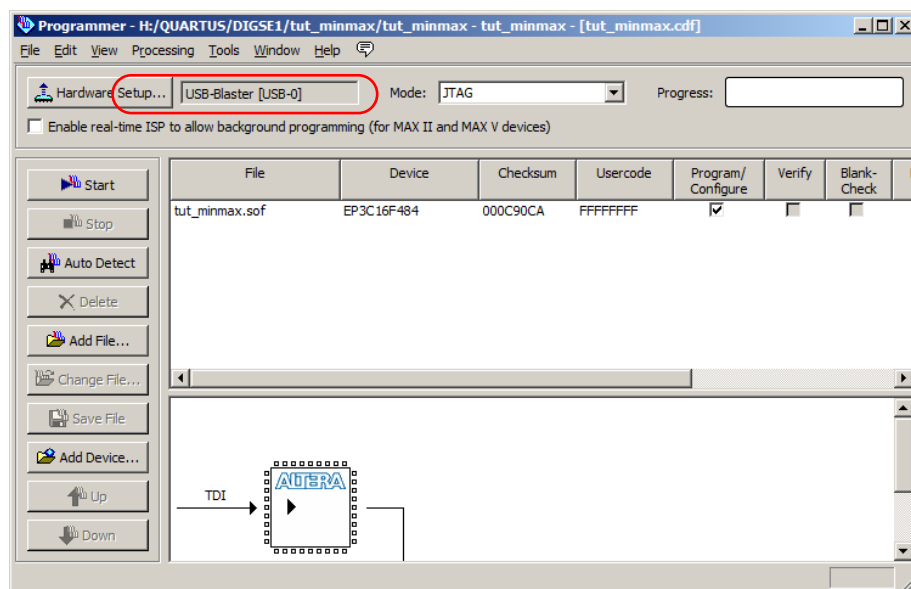
Raadpleeg de docent voordat je de apparatuur aansluit en aanzet!

Selecteer in de Project Manager de menuoptie **Tools**→**Programmer** (figuur 4.44).



Figuur 4.44: Starten van de programmer.

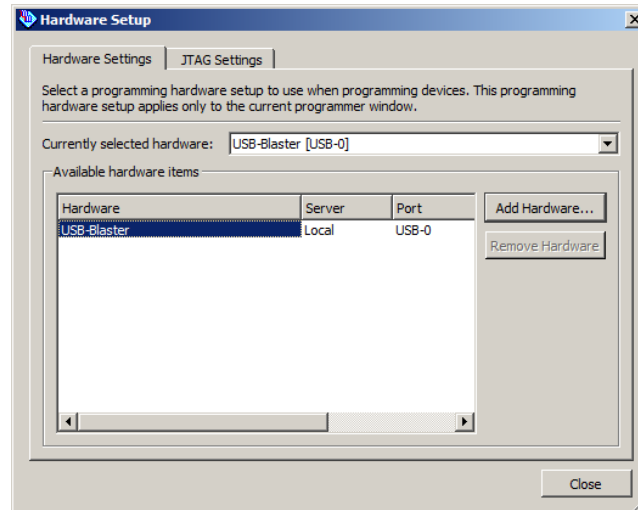
De programmer wordt gestart (figuur 4.45).



Figuur 4.45: Overzicht van de Programmer IDE.

Je kan zien of de programmer het ontwikkelbord heeft gevonden als in het veld naast Hardware Setup de regel USB-Blaster [USB-0] staat. Zo niet, klik dan op **Hardware Setup** en selecteer de Blaster. Je ziet ook dat de programmer een bestand tut_minmax.sof heeft geselecteerd. Dit is het configuratiebestand dat in de Cyclone III geladen moet worden.

Als in het veld naast Hardware Setup de opmerking No Hardware staat, klik dan op de knop **Hardware Setup**. **Dubbelklik** in het geopende dialoogvenster op USB-Blaster in de lijst bij Available Hardware Items en klik daarna op de knop **Close**. Zie figuur 4.46.

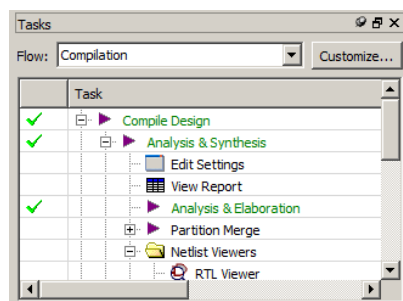


Figuur 4.46: Selecteren van de USB-Blaster download-hardware.

Druk op Start om de Cyclone III te configureren. Daarna kan je je ontwerp testen door middel van de schakelaars.

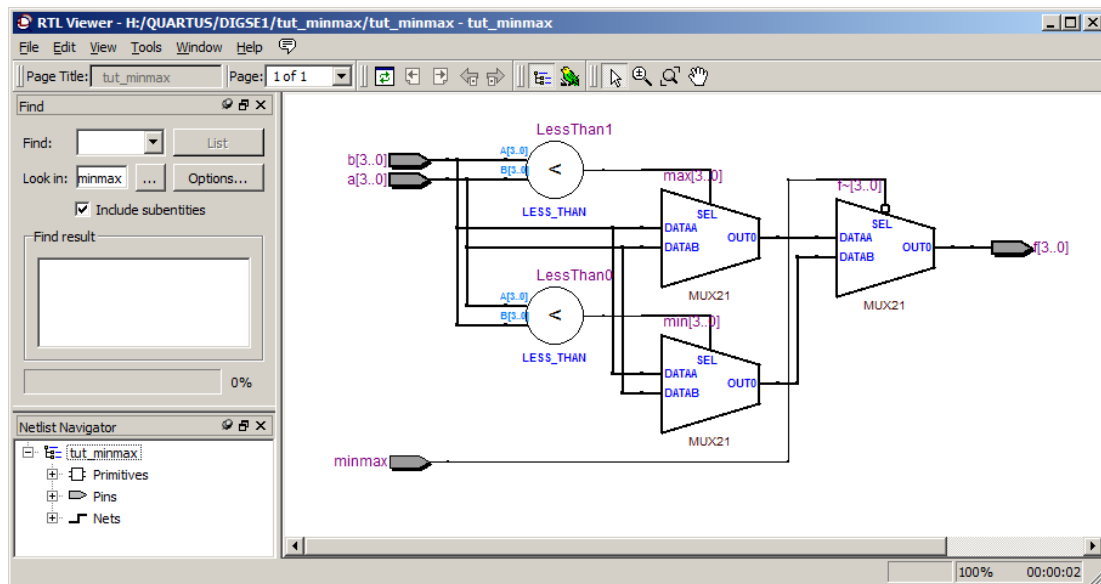
4.8 Gegenerateerde hardware

Het is natuurlijk interessant om te bekijken wat de synthese nu eigenlijk voor hardware heeft opgeleverd. Dit kan je zien met behulp van de RTL viewer. Dubbelklik in het Tasks-venster op de regel **RTL Viewer**. Zie figuur 4.47.



Figuur 4.47: Selecteren van de RTL Viewer.

Er wordt een nieuw programma gestart met daar de gegenereerde hardware (figuur 4.48). Let wel: dit is nog niet de werkelijke implementatie, alleen primitieven.



Figuur 4.48: *Overzicht van de gegenereerde hardware in primitieven.*

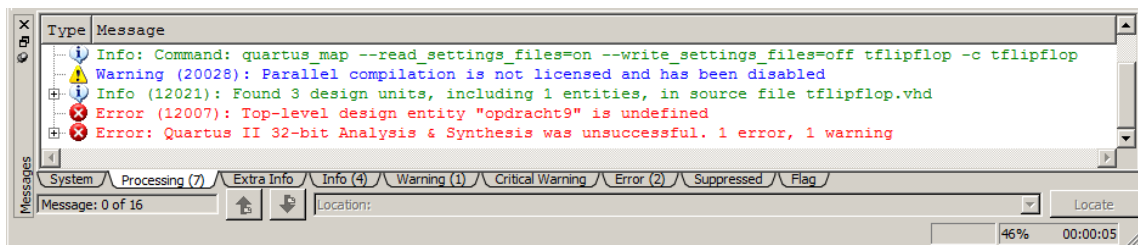
De tutorial is nu ten einde. Veel succes met het practicum.

5. Tips, tricks & troubleshoot

In dit hoofdstuk wordt een aantal veel voorkomende problemen toegelicht en hoe je ze kunt verhelpen. Daarnaast natuurlijk de tips & tricks.

5.1 Foutmelding “Top-level ... undefined”

Onderstaande foutmelding geeft aan (zie figuur 5.1) dat de ingestelde *top-level design entity* niet gedefinieerd is. De kwalificatie *top-level* slaat op de allerhoogste design entity (denk aan VHDL-entity) en die kan niet gevonden worden of is niet gedefinieerd.

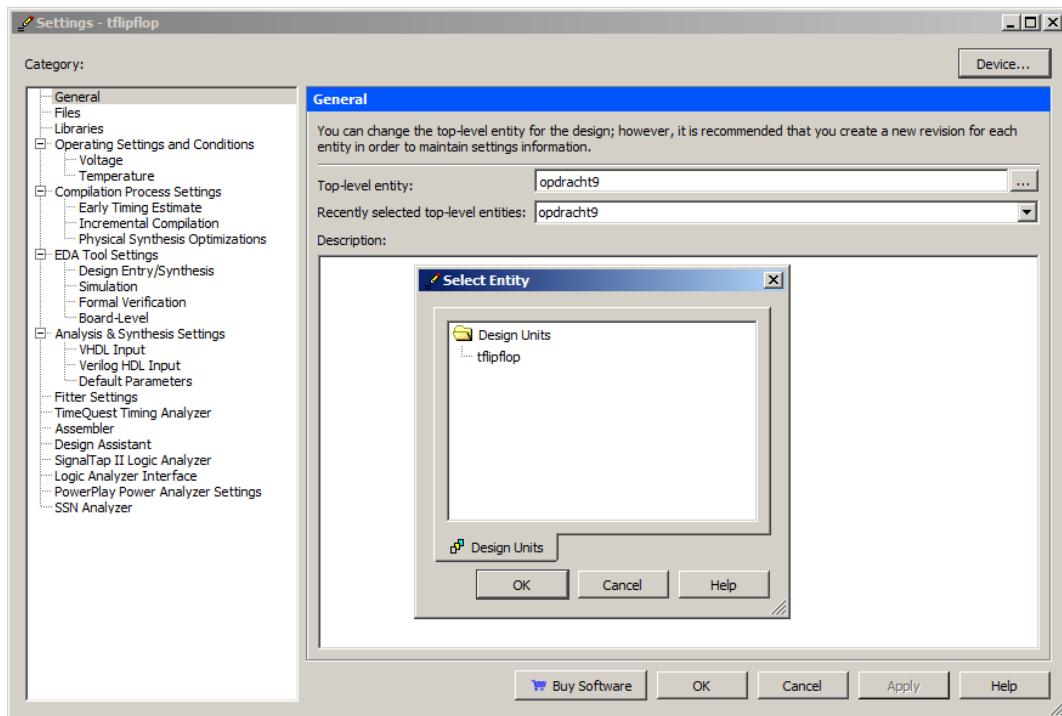


Figuur 5.1: De top-level entity is niet gevonden.

Dit gebeurt meestal bij het aanmaken van een nieuw project omdat de verkeerde naam is ingevuld (zie figuur 4.6). In het voorbeeld is de naam opdracht9 ingevuld terwijl bij de entity-beschrijving in VHDL een andere naam is gebruikt. Gelukkig kan in Quartus een andere entity als top-level worden ingesteld. Selecteer via het menu **Assignments**→**Settings** of gebruik de sneltoets **Ctrl+Shift+E** (zie ook figuur 4.29). Klik in het nieuw geopende dialoog links boven op General.

Rechts kan een ander top-level gekozen worden op de knop met de drie puntjes te klikken achter het veld met Top level entity: waarna in een klein venster de nieuwe naam gekozen kan worden. Klik dan op **Ok**. Het kleine venster wordt afgesloten. Klik in de dialoog eerst op knop **Apply** en daarna op knop **Ok**. De dialoog wordt afgesloten. Zie figuur 5.2.

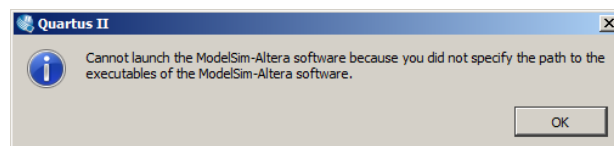
In de Program Navigator is nu de nieuwe top-level entity te vinden.



Figuur 5.2: Selectie van top-level entity.

5.2 Instellen pad naar ModelSim

Als in Quartus het pad naar ModelSim niet correct is ingesteld, krijg je de volgende foutmelding (figuur 5.3).



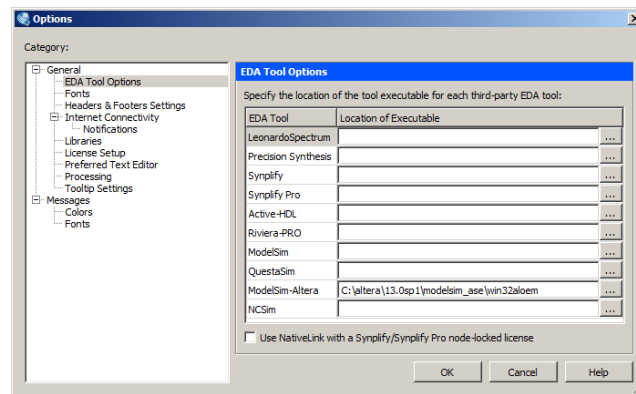
Figuur 5.3: De simulator kan niet worden gestart.

Ga als volgt te werk:

- Open in de Project manager het menu **Tools→Options**
- In het venster dat geopend wordt kies je **EDA Tool Options**
- Aan de rechterkant kan je in het veld ModelSim-Altera het pad opgeven.
- Voor de PC's op school is dat C:\altera\13.0sp1\modelsim_ase\win32aloem

Op je eigen PC of laptop hangt dat af van het installatie-pad. Zie figuur 5.4.

Tip: bij gebruik van Quartus v13.1 moet je een \ ("backslash") achter de padnaam zetten.

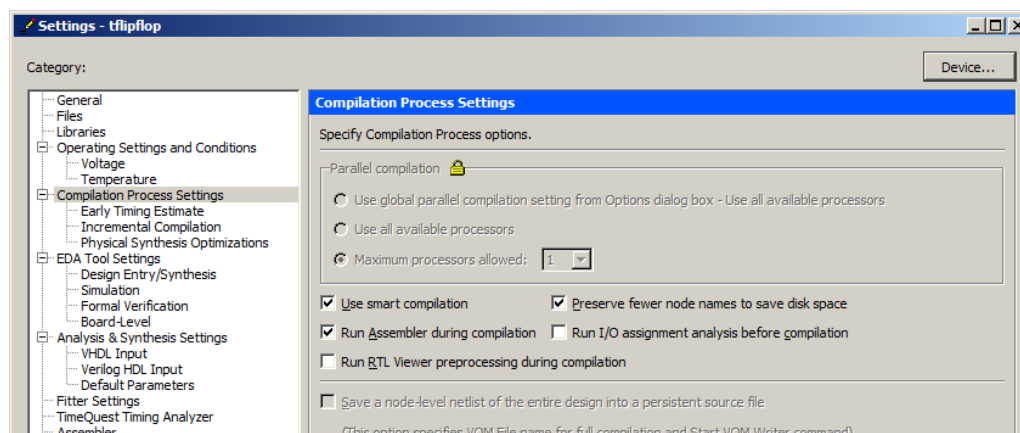


Figuur 5.4: Instellen van pad naar ModelSim.

5.3 Smart compilation

Quartus heeft de neiging om bij een compilatie-opdracht alle stappen te doorlopen, ook als dat niet nodig is. Denk bijvoorbeeld aan het instellen van een nieuwe top-level entity. Dan is synthese niet nodig, die is al een keer uitgevoerd. Als een project meerdere VHDL-bestanden bevat, is het niet nodig om alle bestanden opnieuw te synthetiseren, alleen de bestanden die aangepast zijn.

Quartus heeft een optie die *Smart compilation* wordt genoemd en alleen die stappen doorloopt die nodig zijn voor het eindresultaat. Open via het menu **Assignments**→**Settings**. Selecteer in het gedeelte Category de optie **Compilation Process Settings**. Aan de rechterkant verschijnen de instellingen voor compilatie. Zet een vinkje bij de optie **Use smart compilation** en sluit het venster af door eerst op knop **Apply** te drukken en daarna op knop **Ok**. Zie figuur 5.5.



Figuur 5.5: Instellen van van de optie Smart compilation.

5.4 Quartus blijft hangen

Er is een aantal situaties waardoor Quartus blijft hangen en alleen maar via de Task Manager van Windows kan worden afgesloten. Hieronder de lijst met bekende problemen:

- Bij het aanmaken van een nieuw project is als projectmap een map gekozen waar je als

gebruiker geen schrijfrechten voor hebt. Een mooi voorbeeld is de map `C:\altera\11.1`, de installatiemap van Quartus. Deze map wordt standaard opgegeven bij het aanmaken van een nieuw project. De oplossing is uiteraard eenvoudig: selecteer een andere map.

- De bestanden van het project staan opgeslagen op de H:-schijf. Soms is deze schijf tijdelijk niet beschikbaar, bijvoorbeeld als gevolg door veel gebruikers. De oplossing: gewoon even wachten, na een tijdje reageert Quartus weer.

5.5 Ingestelde pad naar ModelSim wordt niet opgeslagen

Het is een paar keer gebleken dat, ondanks dat het pad naar de ModelSim executable ingesteld is, ModelSim niet gestart kan worden. Dit komt vooral voor bij versie 13.1 van Quartus. Het is mogelijk om handmatig een verwijzing in te stellen naar de ModelSim executable.

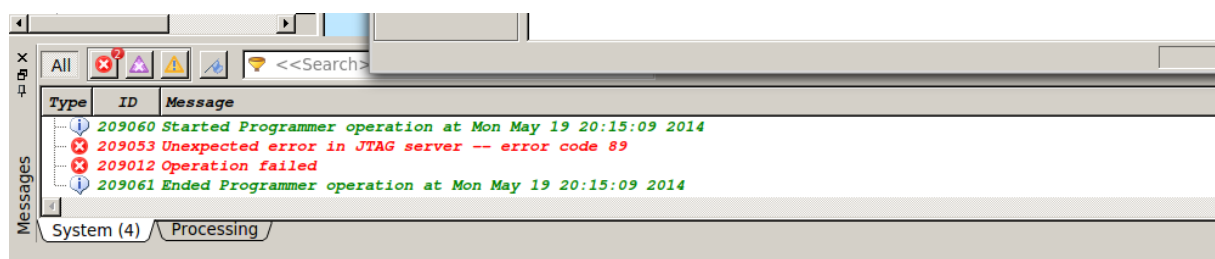
- Open de map waar het gebruikerprofiel is opgeslagen, meestal iets in de trant van `C:\Users\<gebruikersnaam>`
- Open het initialisatie-bestand `quartus2.ini`
- Voeg de volgende code toe, uiteraard met de juiste padnaam

```
[EDA_Tool_Paths 13.1]
EDA_TOOL_PATH_MODELSIM ALTERA = C:\altera\13.1\modelsim_ae\win32aloem
```

- Sluit het bestand
- Start Quartus opnieuw op

5.6 Gebruik USB-Blaster onder Linux

Als je onder Linux als gewone gebruiker inlogt, kan je niet direct gebruik maken van de USB-aansluiting. Je krijgt dan een foutmelding zoals te zien is in figuur 5.6.



Figuur 5.6: Foutmelding bij programmeren onder Linux.

Voer de volgende handelingen uit om als gewone gebruiker de USB-Blaster te kunnen gebruiken. De handelingen zijn getest op CentOS 6.5. De handelingen moet je als gebruiker root uitvoeren.

- Maak een bestand `40-usbblaster.rules` aan in de map `/etc/udev/rules.d`
- Plaats in het bestand de volgende code:

```
# USB-Blaster

SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="09fb", ATTRS{
    idProduct}=="6001", MODE="0666", SYMLINK+="usbbaster/%k"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="09fb", ATTRS{
    idProduct}=="6002", MODE="0666", SYMLINK+="usbbaster/%k"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="09fb", ATTRS{
    idProduct}=="6003", MODE="0666", SYMLINK+="usbbaster/%k"

# USB-Blaster II
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="09fb", ATTRS{
    idProduct}=="6010", MODE="0666", SYMLINK+="usbbaster2/%k"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="09fb", ATTRS{
    idProduct}=="6810", MODE="0666", SYMLINK+="usbbaster2/%k"
```

- Sluit het bestand
- Herlees de udev-regels met `udevadm control --reload-rules`
- Trigger de updates met `udevadm trigger`
- Herstart de programmer-software

5.7 Design Rule S102

Als je de melding S102: Synchronous Port and Asynchronous Port of the Same Register Should Not Be Driven by the Same Signal Source krijgt, heb je waarschijnlijk een asynchrone én synchrone reset in je ontwerp gebruikt. Zoek alle deelontwerpen af naar deze twee vormen en verander alle synchrone resets in asynchrone resets. Zie http://quartushelp.altera.com/13.0/mergedProjects/verify/da/comp_file_rules_synch_reset.htm en klik op de *buttons* om de schema's te openen.

5.8 Naamgeving bestanden en entity's

Een *entity* is een hardware-eenheid en levert bij synthese dus hardware op. Een *bestandsnaam* is de naam van het bestand waarin de hardware beschreven of getekend is. De entity-naam is onafhankelijk van het gebruikte besturingssysteem, de bestandsnaam is wel afhankelijk.

In Quartus zijn schemabestanden met de extensie *.bdf* gekoppeld aan de entity-naam: de bestandsnaam zonder de extensie is gelijk ook de entity-naam.

Bij beschrijvingstalen als VHDL en Verilog ligt dat anders: de bestandsnaam hoeft niet hetzelfde te zijn als de entity-naam. In feite is het bestand een *container* met daarin de beschrijving van de hardware. ModelSim gebruikt de bestandsnaam om de beschrijving te compileren (bv. met *vcom* en *vlog*) maar gebruikt de entity-naam bij het starten van de simulatie (m.b.v. *vsim*).

Let op: entity-namen mogen *niet* beginnen met een cijfer of een leesteken. In feite gelden voor de entity-namen dezelfde regels als voor variabelen. Schemabestandsnamen mogen dus ook niet met een cijfer of een leesteken beginnen, VHDL-bestandsnamen wel.

Het is verstandig om de bestandsnaam en de entity-naam hetzelfde te houden. Dat voorkomt allerlei problemen. Plaats maximaal één design (entity en architecture) in één bestand.

Gebruik voor testbenches dezelfde naamgeving als de entity die gesimuleerd moeten worden met `tb_` ervoor. Doe dat ook voor de entitynaam van de testbench. Modelsim command scripts eindigen op de extensie `.do` en starten over het algemeen de simulator met als top level de testbench. Zie onderstaand stukje code.

```
# This file is named 'tb_entityname.do'
...
vcom entityname.vhd      # file containing design entity 'entityname'
vcom tb_entityname.vhd   # file containing testbench entity 'tb_entityname'
vsim tb_entityname       # Simulator top level is 'tb_entityname'
...
```

5.9 Pinnen worden niet getoond in de Pin Planner

Als de pinnen niet worden getoond in de Pin Planner, heb je waarschijnlijk geen *Device Type* opgegeven en staat het type op *Auto*. Ga via menu **Assignments**→**Device** en vul de juiste gegevens in. Zie ook figuur 4.9.

5.10 Verkeerde pinnen worden getoond

Dan heb je de verkeerde entity als top-level ingesteld. Zie paragraaf 5.1.

5.11 ModelSim stopt na enige tijd

Als ModelSim na enige tijd stopt voordat de simulatie werkelijk gestart wordt, dan ben je vergeten om de do-file op te geven (in Quartus). Zie de figuren 4.29 en 4.30.

5.12 “Instance ... is not bound”

Deze foutmelding wordt gegenereerd door ModelSim. Het betekent dat binnen de simulatie-omgeving geen entity met die naam is gedefinieerd en er kan dus geen instantie van worden aangemaakt. De simulatie gaat wel verder. Dat levert uiteraard problemen op. De werking is van het te simuleren systeem is niet correct, want er mist een deel. Interne signalen van de niet-geïnstantiëerde entity zijn niet beschikbaar. Mogelijke oorzaken zijn:

- Het VHDL-bestand met de betreffende entity wordt niet gecompileerd (de `vcom`-opdracht is niet correct of vergeten).
- Bij instantiëring wordt een entity-naam opgegeven die in geen van de gecompileerde VHDL-bestanden is opgegeven.

5.13 Opruimen van een Quartus-project

Quartus heeft de neiging om tijdens compilatie ontzettend veel, vooral kleine bestanden aan te maken. Je kan heel veel van die bestanden en mappen gewoon verwijderen als je project is afgerond.

Onderstaand script kan je draaien in een Windows-command box en verwijdt bijna elk bestand dat niet nodig is inclusief een aantal mappen die door Quartus en ModelSim aangemaakt worden.

```
@echo off
rem
rem
echo.
echo This program will delete all unnessecary files from Quartus projects.
echo.
echo Please be VERY carefull! Press Ctrl-C to break.
echo.
pause
echo.

FOR /D /r %%G in ("db*") DO rd /s/q %%G
FOR /D /r %%G in ("incremental_db*") DO rd /s/q %%G
FOR /D /r %%G in ("simulation*") DO rd /s/q %%G
FOR /D /r %%G in ("output_files*") DO rd /s/q %%G

del /s *.done
del /s *.rpt
del /s *.sof
del /s *.pof
del /s *.summary
del /s *.jdi
del /s vsim.wlf
del /s *.bak
del /s transcript
del /s *assignment_defaults.qdf
del /s *.pin
del /s modelsim.ini
del /s *.qws
del /s *.smsg
del /s *.map
del /s *.cdf
del /s *.dpf
del /s PLLJ_PLLSPE_INFO.txt

















echo.
echo Done.
echo.
pause
```

Listing 5.1: *Windows opruimscript*

A. Knoppen en sneltoetsen

In deze bijlage is een tabel opgenomen met een aantal knoppen en sneltoetscombinaties. Niet alle knoppen worden in deze tutorial gebruikt.

Tabel A.1: *Knoppen en sneltoetscombinaties.*

Knop	Benaming	Menu	Sneltoets
	View Project Navigator	View→Utility Windows→Project Navigator	Alt+0
	Device Assignments	Assignments→Device	-
	Settings Assignments	Assignments→Settings	Ctrl+Shift+E
	Assignment Editor*	Assignments→Editor	Ctrl+Shift+A
	Pin Planner	Assignments→Pin Planner	Ctrl+Shift+N
	Floor Planner*	Tools→Chip Planner	-
	Start Compilation	Processing→Start Compilation	Ctrl+L
	Start Analysis	Processing→Start→Start Analysis & Synthesis	Ctrl+K
	Start TimeQuest Timing Analyser*	Processing→Start→ Start TimeQuest Timing Analyser	Ctrl+Shift+T
	Open TimeQuest Timing Analyser*	Tools→TimeQuest Timing Analyser	-
	RTL Simulation	Tools→Run Simulation Tools→RTL Simulation	-
	Gate Level Simulation*	Tools→Run Simulation Tools→Gate Level Simulation	-
	Compilation Report	Processing→Compilation Report	Ctrl+R
	Programmer	Tools→Programmer	-
	Analyse Current File	Processing→Analyse Current File	-
	Insert Template	Edit→Insert Template	-

* Wordt niet gebruikt tijdens deze tutorial.

B. Pinbenaming EP3C16F484C-6N

In deze bijlage vind je de pinbenaming terug. Er staan ook opmerkingen bij.

Tabel B.1: *Pinbenamingen FPGA, deel 1.*

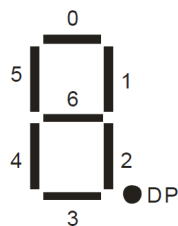
Type	Naam	Pinnaam	Opmerking
Clock	CLOCK_50	G21	Global Clock 1
Push Buttons	BUTTON[0]	H2	Ontdenderd, actief laag
	BUTTON[1]	G3	
	BUTTON[2]	F1	
Switches	SW[0]	J6	Niet ontdenderd, actief hoog
	SW[1]	H5	
	SW[2]	H6	
	SW[3]	G4	
	SW[4]	G5	
	SW[5]	J7	
	SW[6]	H7	
	SW[7]	E3	
	SW[8]	E4	
	SW[9]	D2	
Leds	LEDG[0]	J1	Actief hoog
	LEDG[1]	J2	
	LEDG[2]	J3	
	LEDG[3]	H1	
	LEDG[4]	F2	
	LEDG[5]	E1	
	LEDG[6]	C1	
	LEDG[7]	C2	
	LEDG[8]	B2	
	LEDG[9]	B1	

Dit zijn de benamingen zoals ze in de documentatie van het DE0-bordje worden gebruikt. Je kan ook je eigen namen gebruiken. Quartus zal, wanneer van toepassing, het woord PIN_ voor de pinnaam zetten, dus J2 wordt dan PIN_J2.

Op de volgende pagina staan de pingegevens van de 7-segment displays. Tevens is de layout gegeven.

Tabel B.2: Pinbenamingen FPGA, deel 2.

Type	Naam	Pinnaam	Opmerking
7-segment	HEX0_D[0]	E11	Alle actief laag
	HEX0_D[1]	F11	
	HEX0_D[2]	H12	
	HEX0_D[3]	H13	
	HEX0_D[4]	G12	
	HEX0_D[5]	F12	
	HEX0_D[6]	F13	
	HEX0_DP	D13	
	HEX1_D[0]	A13	
	HEX1_D[1]	B13	
	HEX1_D[2]	C13	
	HEX1_D[3]	A14	
	HEX1_D[4]	B14	
	HEX1_D[5]	E14	
	HEX1_D[6]	A15	
	HEX1_DP	B15	
	HEX2_D[0]	D15	
	HEX2_D[1]	A16	
	HEX2_D[2]	B16	
	HEX2_D[3]	E15	
	HEX2_D[4]	A17	
	HEX2_D[5]	B17	
	HEX2_D[6]	F14	
	HEX2_DP	A18	
	HEX3_D[0]	B18	
	HEX3_D[1]	F15	
	HEX3_D[2]	A19	
	HEX3_D[3]	B19	
	HEX3_D[4]	C19	
	HEX3_D[5]	D19	
	HEX3_D[6]	G15	
	HEX3_DP	G16	



Figuur B.1: Layout 7-segment displays