

# **TUTORIAL SCHEMATIC ENTRY**

## **MET**

## **QUARTUS 11.1**

## **EN**

## **MODELSIM-ALTERA 10.0**

# INHOUDSOPGAVE

<b>1</b>	<b>Inleiding</b>	<b>5</b>
<b>2</b>	<b>Practicumomgeving</b>	<b>7</b>
2.1	Ontwikkelbord	7
2.2	Software-versies en web-edition	8
<b>3</b>	<b>Ontwikkelbordje</b>	<b>9</b>
3.1	Blokschema	9
3.2	Altera Cyclone III	11
<b>4</b>	<b>Tutorial Schematic Entry</b>	<b>13</b>
4.1	Installatie project Tutorial	14
4.2	Project en naamgeving	15
4.3	Project Tutorial starten	15
4.4	Aanmaken schemabestand	18
4.5	Aanmaken van symbool van het huidige bestand	23
4.6	Aanmaken van een tweede schemabestand	24
4.7	Eerste synthese	25
4.8	Simulatie poortschakeling	26
4.9	Compilatie	28
4.10	Configureren van de Cyclone III	29
<b>5</b>	<b>Tips, tricks &amp; troubleshoot</b>	<b>32</b>
5.1	Instellen pad naar ModelSim	32
5.2	Smart compilation	33
5.3	Quartus blijft hangen	34
5.4	Ingestelde pad naar ModelSim wordt niet opgeslagen	34
5.5	Gebruik USB-Blaster onder Linux	34
5.6	Bestandsnaam en entity-naam	35
5.7	Opruimen van een Quartus-project	35
<b>A</b>	<b>Knoppen en sneltoetsen</b>	<b>37</b>
<b>B</b>	<b>Pinbenaming EP3C16F484C-6N</b>	<b>38</b>
<b>C</b>	<b>INLDIG-flow onder Linux</b>	<b>40</b>
<b>D</b>	<b>INLDIG-flow onder Windows</b>	<b>41</b>

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](mailto:J.E.J.opdenBrouw@hhs.nl).

# LIJST VAN FIGUREN

1.1	Het ontwerptraject. . . . .	6
2.1	Het DE0-ontwikkelbord . . . . .	7
3.1	Het DE0-ontwikkelbord met benoeming van periferie . . . . .	9
3.2	Blokschema ontwikkelbord . . . . .	10
3.3	Foto Cyclone III . . . . .	10
3.4	Floor Plan van de Cyclone III . . . . .	12
4.1	De inhoud van de map common. . . . .	14
4.2	Inhoud van de map tutorial. . . . .	16
4.3	Quartus II opstartscherm . . . . .	16
4.4	Openingsscherm Project Manager . . . . .	17
4.5	Selecteren van de INLDIG-flow. . . . .	17
4.6	Het Files-tabblad. . . . .	18
4.7	Aanmaken nieuw bestand. . . . .	18
4.8	Keuze bestandstype. . . . .	18
4.9	Overzicht Quartus IDE na aanmaken nieuw BDF-bestand. . . . .	19
4.10	Overzicht van de knoppen. . . . .	19
4.11	Selectie component. . . . .	20
4.12	Selectie AND2-poort. . . . .	20
4.13	Selectie input-poort. . . . .	21
4.14	Alle geplaatste componenten. . . . .	21
4.15	Alle geplaatste componenten met verbindingen. . . . .	21
4.16	Schema met nieuwe pinnamen. . . . .	22
4.17	Bestand opslaan. . . . .	22
4.18	Opgeven bestandsnaam BDF-bestand. . . . .	22
4.19	Het opgeslagen bestand staat in de lijst van bestanden. . . . .	23
4.20	Aanmaken nieuw symbool. . . . .	23
4.21	Bestandsnaam nieuw symbool. . . . .	24
4.22	Het bestand is opgeslagen. . . . .	24
4.23	Selectie nieuwe component. . . . .	24
4.24	Het bestand is opgeslagen. . . . .	25
4.25	Starten Analysis & Synthesis vanuit het menu. . . . .	25
4.26	De analyse is gelukt. . . . .	25
4.27	De analyse is mislukt. . . . .	26
4.28	Starten van de simulatie. . . . .	27
4.29	Het resultaat van de simulatie. . . . .	27
4.30	Het transcript-window. . . . .	28
4.31	ModelSim afsluiten. . . . .	28
4.32	Starten van de compilatie. . . . .	28

4.33 De compilatie is gelukt. . . . .	29
4.34 Overzicht van het resultaat van de compilatie. . . . .	29
4.35 Starten van de programmer. . . . .	30
4.36 Overzicht van de Programmer IDE. . . . .	30
4.37 Selecteren van de USB-Blaster download-hardware. . . . .	31
5.1 De simulator kan niet worden gestart. . . . .	32
5.2 Instellen van pad naar ModelSim. . . . .	33
5.3 Instellen van van de optie Smart compilation. . . . .	33
5.4 Foutmelding bij programmeren onder Linux. . . . .	34
B.1 Layout 7-segment displays . . . . .	39

## LIJST VAN TABELLEN

3.1 Enige gegevens over de EP3C16F484C-6N . . . . .	11
4.1 Betekenis bestandsnaamextensies. . . . .	15
A.1 Knoppen en sneltoetscombinaties. . . . .	37
B.1 Pinbenamingen FPGA, deel 1. . . . .	38
B.2 Pinbenamingen FPGA, deel 2. . . . .	39

## LISTINGS

5.1 Windows opruimsript . . . . .	36
-----------------------------------	----

# 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.

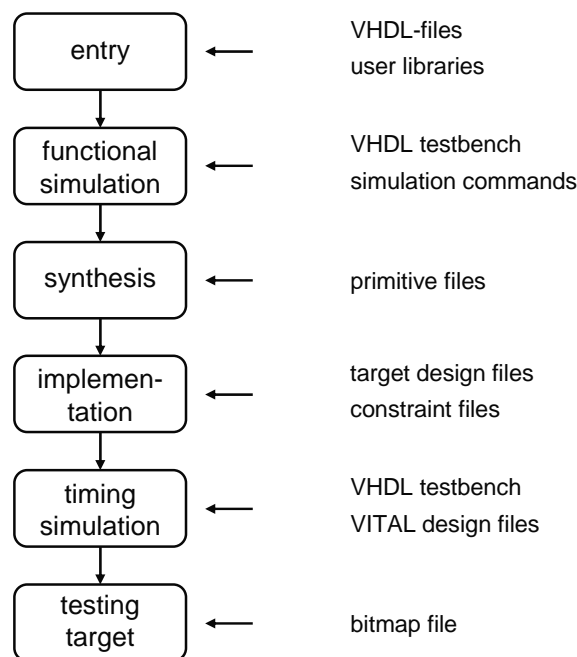
Wanneer alle deelontwerpen gemaakt zijn, moeten ze worden omgezet naar een bestand met gegevens die in de component geladen moet worden. Dit proces heet synthetiseren. Als tijdens dit omzetten een fout wordt geconstateerd, bijvoorbeeld twee uitgangen aan elkaar, wordt dit gemeld aan de ontwerper en moet 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 met poorten,
- functionele simulatie
- omzetten van de poortschakeling 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:** *Het ontwerptraject.*

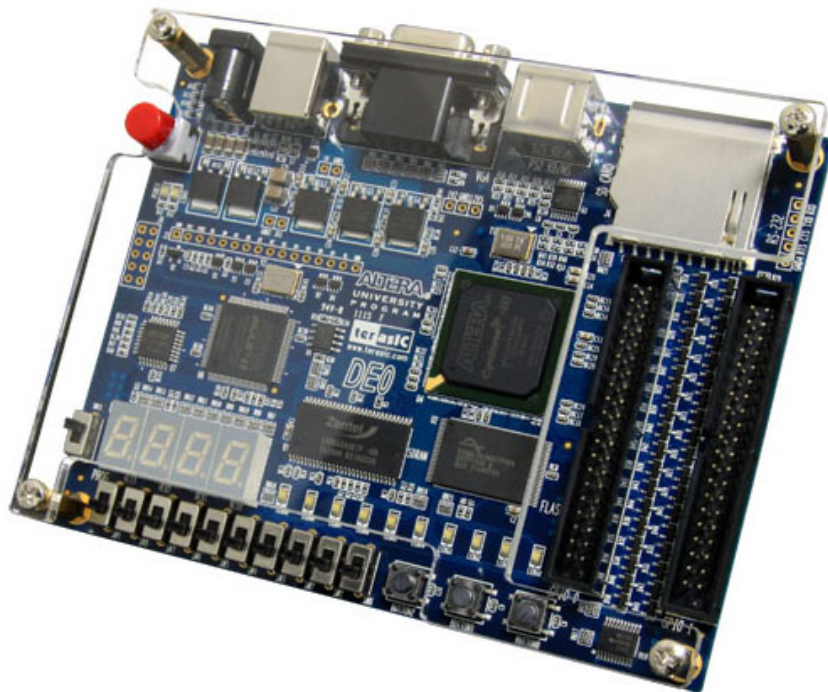
Het bedenken van de schakeling is een creatief proces. Ervaring en goede kennis van digitale systemen helpt je hierbij op weg. Het opzetten van een goede simulatie hoort bij de vaardigheden van de ontwerper. De rest wordt eigenlijk door de tools van Quartus afgehandeld. Het is voor de ontwerper niet 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 loodst.

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 invoergedeelte - 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 optiepakketten 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 optiepakketten 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 11.1sp1 resp. versie 10.0c.

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. versie 11.1sp1. De in het practicum gemaakte Quartus-projecten kunnen zonder problemen door zowel versie 13.0sp1 als 11.1sp1 geopend worden. Vanaf versie 13.0 is ModelSim geïntegreerd in het installatiepakket.

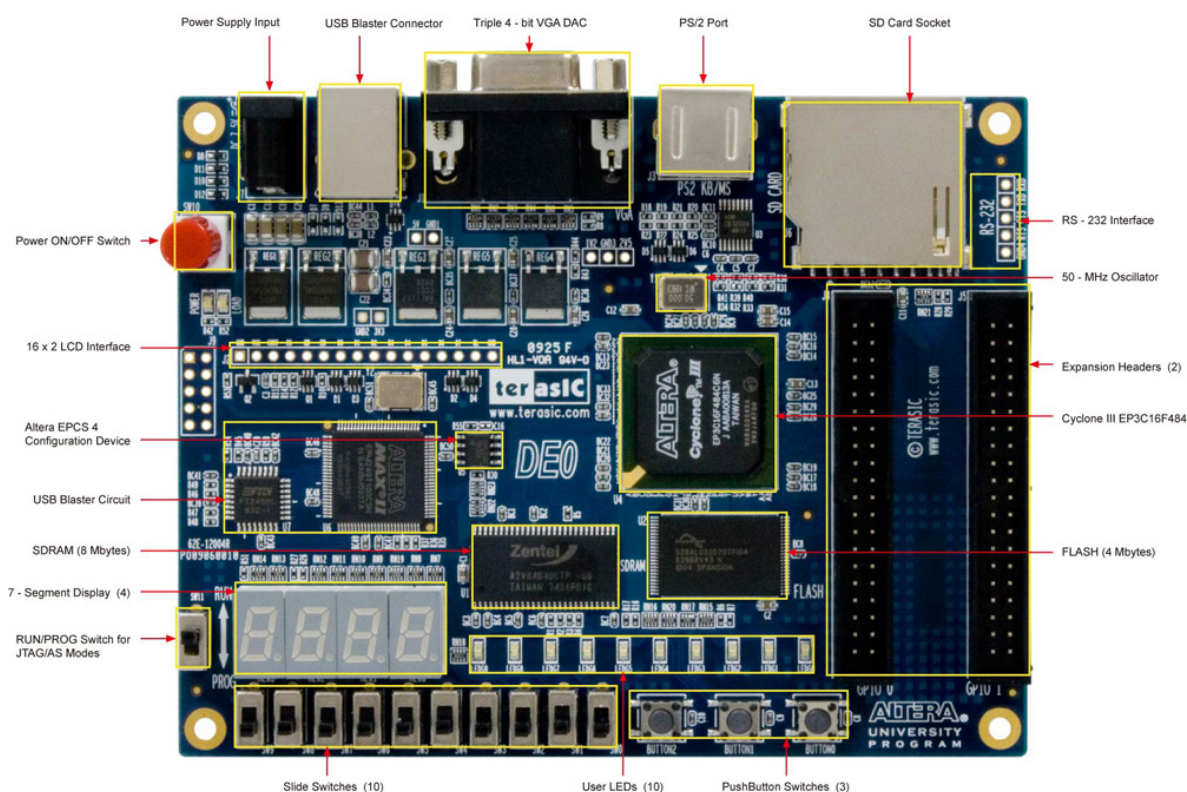


### 3. ONTWIKKELBORDJE

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.

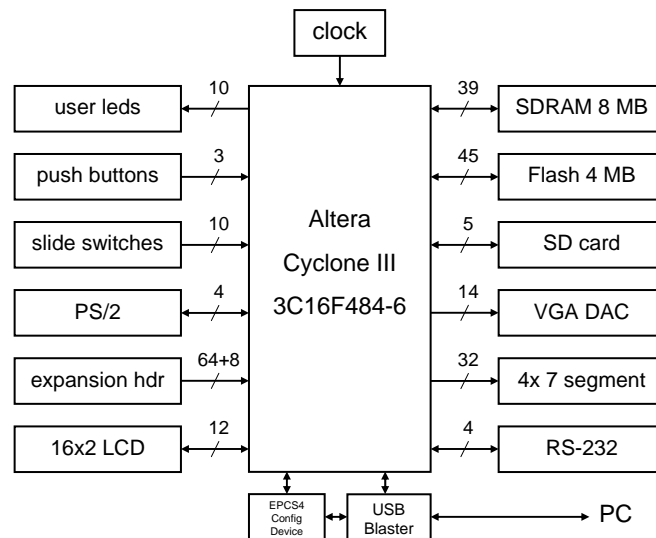
#### 3.1 Blokschema

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 ontdenderd 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 ontdenderd 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)<sup>1</sup> 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.

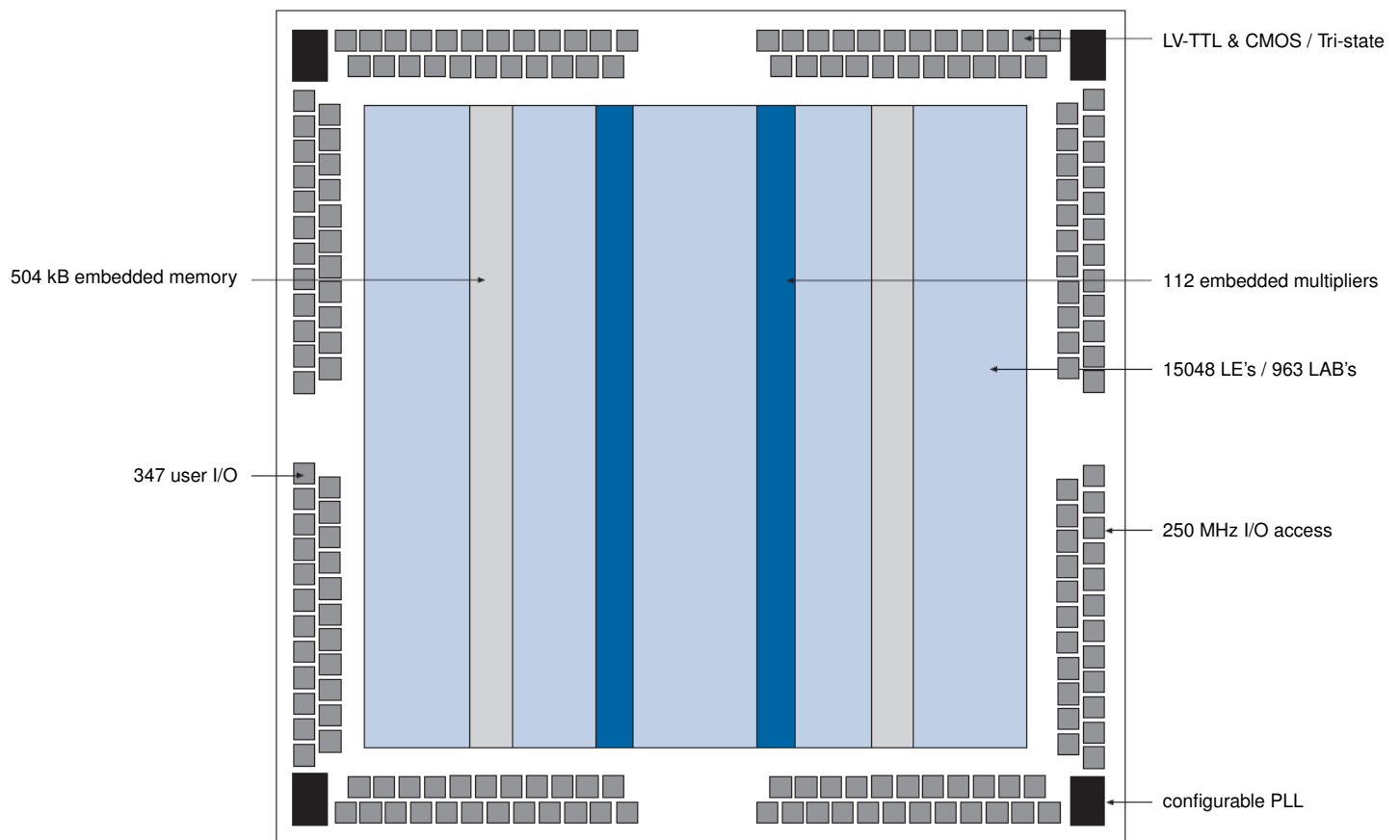
In tabel 3.1 zijn wat gegevens te vinden.

**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 de workshop 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.

<sup>1</sup> Het is wel raar dat een FPGA configureerbaar heet, terwijl de naam suggereert dat ze programmeerbaar zijn.



**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 SCHEMATIC ENTRY

In deze tutorial gaan we ons bezighouden met het invoeren, simuleren en implementeren van digitale schakelingen met schematische invoer. Andere stappen zoals het aanmaken van een project en pintoewijzing worden in een andere tutorial behandeld.

Zoals bekend kan elke digitale schakeling worden opgebouwd met de bekende poorten zoals AND, OR en NOT. Daarnaast zijn er veel andere poorten zoals de EXOR, NAND en NOR. Tijdens deze tutorial leer je hoe je een schema dat is opgebouwd uit poorten moet invoeren. Daarnaast maak je kennis met hiërarchisch ontwerpen: het schema van een deelschakeling kan je gebruiken als een “poort” in een ander schema. Hierdoor kan je snel grotere schakelingen maken. Het ontwerpen van digitale schakelingen met poorten is een traject apart en komt niet in deze tutorial aan bod.

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 openen
- schema-invoer d.m.v. Schematic Editor.
- simuleren van de schakeling op functioneel niveau
- compilatie (synthese en implementatie) van de poortschakeling naar een configuratiebestand
- downloaden van de configuratiebestand in de Cyclone III.

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

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

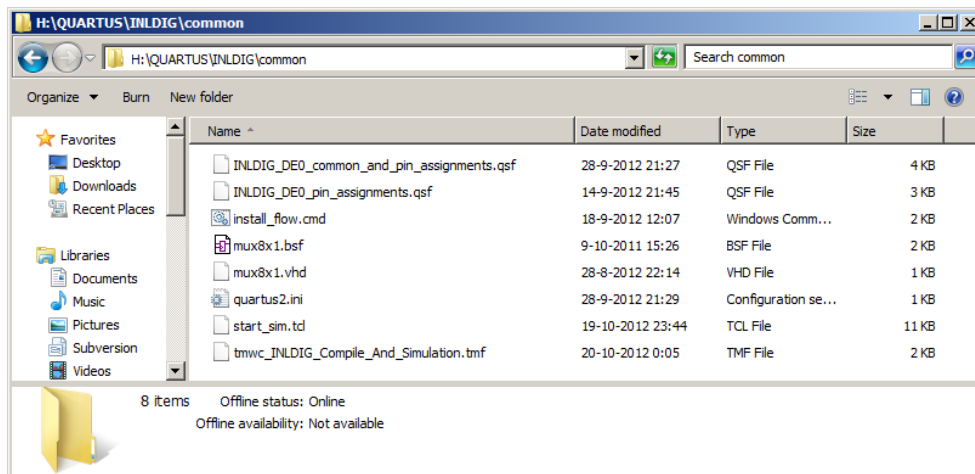
**NB: gebruik geen spaties, leestekens of "vreemde"tekens in mapnamen en bestandsnamen! Bestandsnamen niet met een cijfer beginnen!**

Noot: het doel van deze tutorial is het leren omgaan met de software (zogenaamde tools), niet het leren van digitale techniek. Over de werking van de schakeling die wordt ingevoerd (wat doet het) wordt geen uitleg gegeven.

## 4.1 Installatie project Tutorial

Voordat we echt kunnen beginnen, moeten we eerst de projectomgeving inrichten. Hiervoor moet je toegang hebben tot de BlackBoard Course INLDIG. De bestanden kunnen ook gevonden worden op <http://ds.opdenbrouw.nl/inldig.html>.

1. Maak op de H:-schijf direct onder de *root*<sup>2</sup> een map aan met de naam QUARTUS. Je hebt dan een pad H:\QUARTUS.
2. Maak in het pad H:\QUARTUS een map aan met de naam INLDIG. Je hebt nu een pad met de naam H:\QUARTUS\INLDIG.
3. Download van BlackBoard<sup>3</sup> het bestand `inldig_common_tutorial.zip` en plaats dit bestand in H:\QUARTUS\INLDIG.
4. Pak het bestand uit in deze map. Let op: bij uitpakken maakt Windows automatisch de map `inldig_common_tutorial` aan! Dat is niet de bedoeling!
5. Je vindt nu twee mappen: `tutorial` en `common`. Navigeer naar de map `common`. Hierin vindt je een bestand met de naam `install_flow.cmd`. Zie figuur 4.1.



Figuur 4.1: De inhoud van de map `common`.

6. Dubbelklik op het bestand `install_flow.cmd`. Er wordt een scherm gestart met als uitvoer

```
Installing INLDIG Flow...
1 file(s) copied.
1 file(s) copied.
Press any key to continue . . .
```

7. Druk op een toets om het scherm te sluiten. Noot: als je iets anders ziet dan hierboven, raadpleeg de docent.

De flow is nu geïnstalleerd. We kunnen beginnen met invoeren van de schakeling.

Noot: het is ook mogelijk om de bestanden handmatig in een andere map te installeren, zie de bijlagen C en D.

<sup>2</sup> Zie [http://en.wikipedia.org/wiki/Root\\_directory](http://en.wikipedia.org/wiki/Root_directory)

<sup>3</sup> Een alternatief is [http://ds.opdenbrouw.nl/inldig/inldig\\_common\\_tutorial.zip](http://ds.opdenbrouw.nl/inldig/inldig_common_tutorial.zip)

## 4.2 Project en naamgeving

Een project bestaat uit niets anders dan een verzameling bestanden. De belangrijkste bestanden hebben de volgende extensies:

**Tabel 4.1:** *Betekenis bestandsnaamextensies.*

Extensie	Volledige naam	Betekenis
.qpf	Quartus Project File	Projectbestand met naam en algemene informatie
.qsf	Quartus Settings File	Instellingen bij een bepaalde <i>revisie</i> , meestal is er maar één revisie
.bdf	Block Design File	Schema-bestand, zoals poorten, en componenten waardoor hiërachiën mogelijk zijn.
.bsf	Block Schematic File	Een component ("poort") gemaakt van een bdf-bestand of vhd-bestand
.do	ModelSim Command File	Bestand met commando's voor de simulator
.vhd	VHDL File	Bestand met VHDL-code
.v	Verilog File	Bestand met Verilog-code, wordt tijdens het practicum niet gebruikt
.sof	SRAM Object File	Bestand met "code" voor de FPGA, informatie die in de FPGA geladen moet worden

Quartus maakt in het simulatie- en synthesesetraject nog veel meer bestanden met extensies aan, die zijn niet van belang. ModelSim-testbenches beginnen met tb\_.

**Een project kan automatisch in Quartus worden geopend door de dubbelklikken op het betreffende .qpf-bestand.**

**Niet op een .bdf-bestand dubbelklikken, want dan is er geen projectomgeving beschikbaar en kan er niet gesimuleerd of gesynthetiseerd worden.**

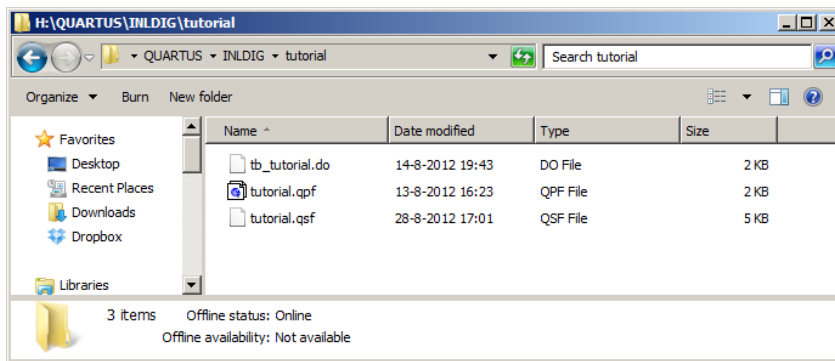
**Gebruik geen spaties, leestekens of "vreemde" tekens in bestandsnamen en mapnamen!**

**Gebruik geen minteken in bestandsnamen, underscores zijn wel toegestaan.**

## 4.3 Project Tutorial starten

We starten Quartus op via Explorer. Navigeer naar H:\QUARTUS\INLDIG\tutorial en dubbelklik op het bestand tutorial.qpf. Zie figuur 4.2.





**Figuur 4.2:** Inhoud van de map *tutorial*.

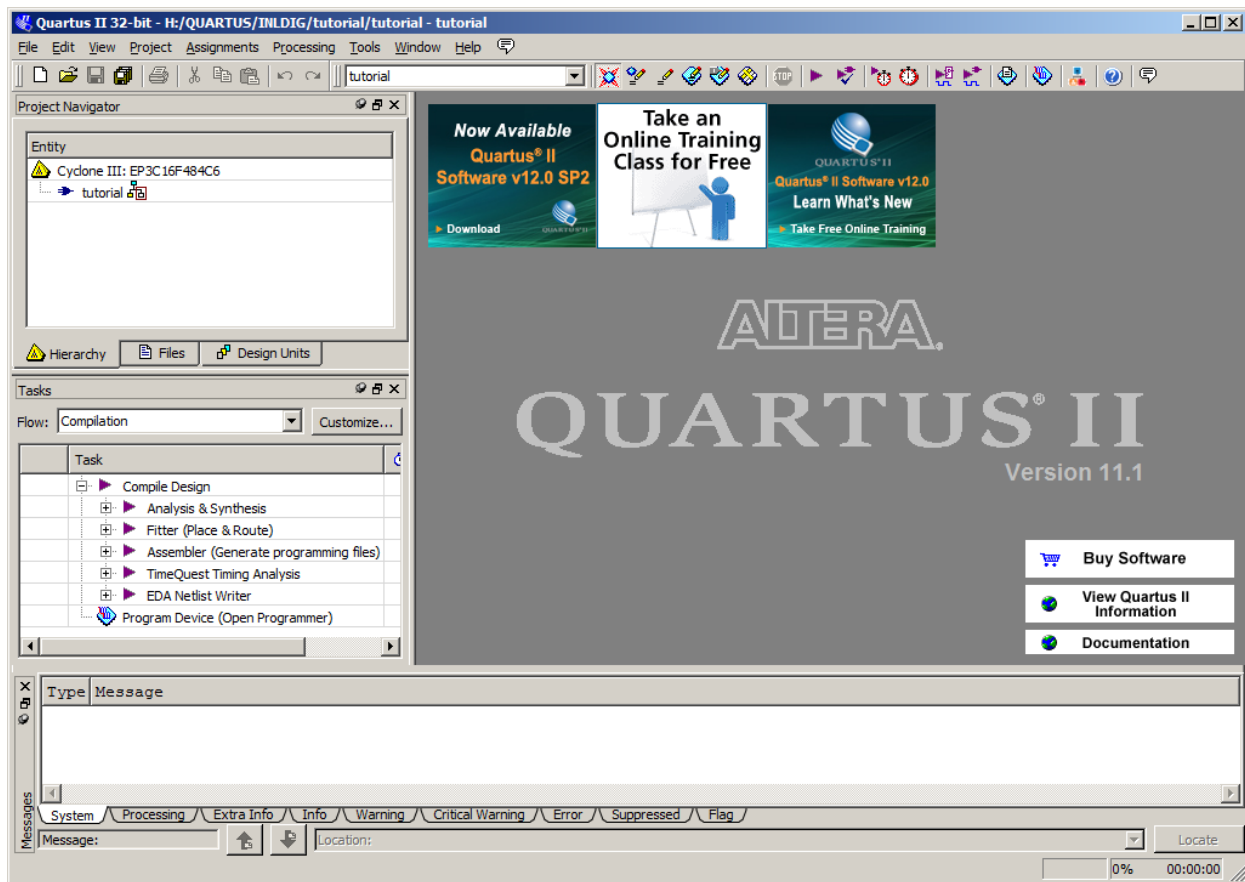
Na het opstarten kan figuur 4.3 verschijnen. Via deze Getting Started kan je snel een project aanmaken of openen. We slaan dit scherm over. Klik hiervoor op het kruisje rechtsboven.



**Figuur 4.3:** Quartus II opstartscherm

De *Project Manager* wordt gestart (figuur 4.4). De grootte en indeling van het scherm kan iets afwijken van de figuur. Probeer dit aan te passen volgens de figuur.

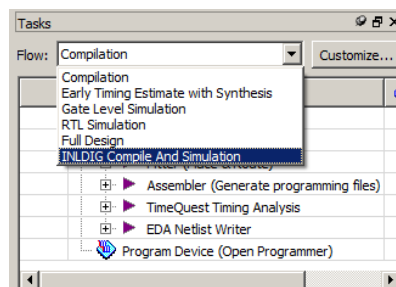




**Figuur 4.4:** Openingsscherm Project Manager

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.

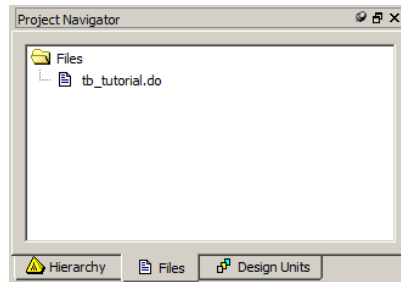
Voor de juiste werking moet eerst de *flow* worden ingesteld. In het venster *Tasks* kan je bij het onderdeel *Flow* kiezen voor een flow. Kies hier de flow *INLDIG Compile And Simulation*. Zie figuur 4.5.



**Figuur 4.5:** Selecteren van de INLDIG-flow.

Kies in het venster *Project Navigator* voor het tabblad *Files*. Hier zie je een overzicht van de

bestanden die tot het project behoren. Het project bestaat uit slechts één bestand genaamd `tb_tutorial.do`. Zie figuur 4.6.

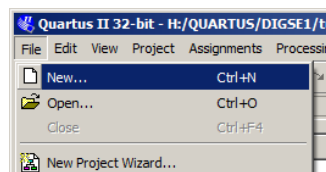


**Figuur 4.6:** *Het Files-tabblad.*

## 4.4 Aanmaken schemabestand

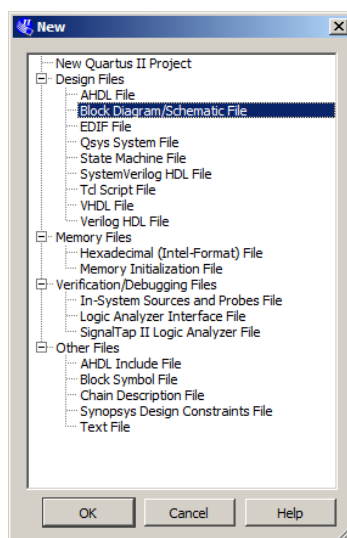
We gaan nu het eerste schema invoeren. Dit schema gaat later gebruikt worden als “poort” in een ander schema.

Als eerste zullen we een schemabestand in het project aanmaken. Klik in de Project Manager op **File**→**New** (figuur 4.7). Als alternatief kan de toetscombinatie **Ctrl+N** gebruikt worden.



**Figuur 4.7:** *Aanmaken nieuw bestand.*

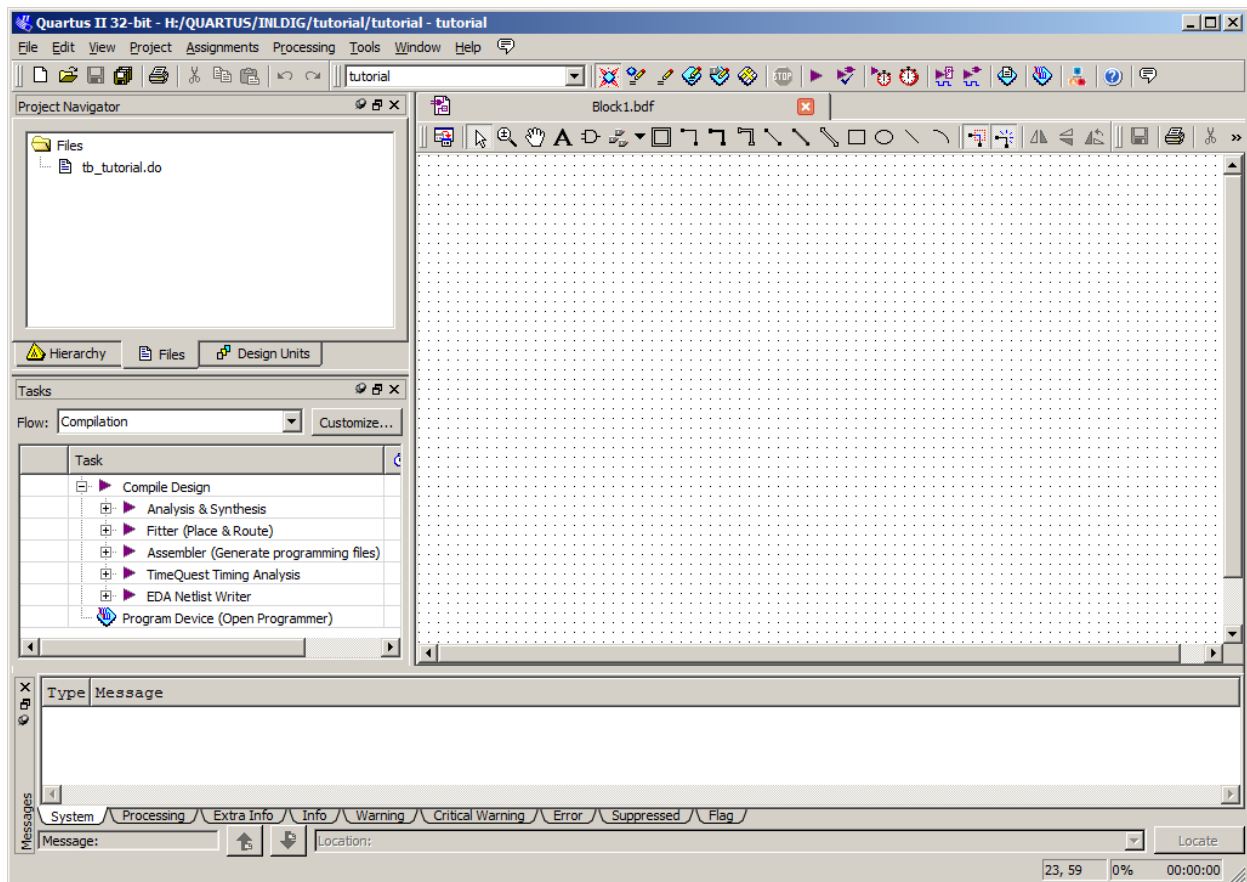
Nu verschijnt er een scherm waarin je het type van het nieuwe bestand kan kiezen (figuur 4.8). Kies voor **Block Diagram/Schematic File**.



**Figuur 4.8:** *Keuze bestandstype.*

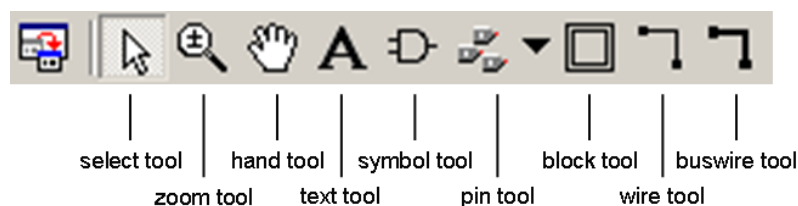
De Project Manager opent nu een leeg bestand. Dit is te zien in figuur 4.9. Merk op dat het

bestand de tijdelijk naam Block1.bdf heeft. Het sterretje achter de naam geeft aan dat het bestand gewijzigd en nog niet opgeslagen is. Bij het opslaan van het bestand kan alsnog een andere naam worden ingevoerd.



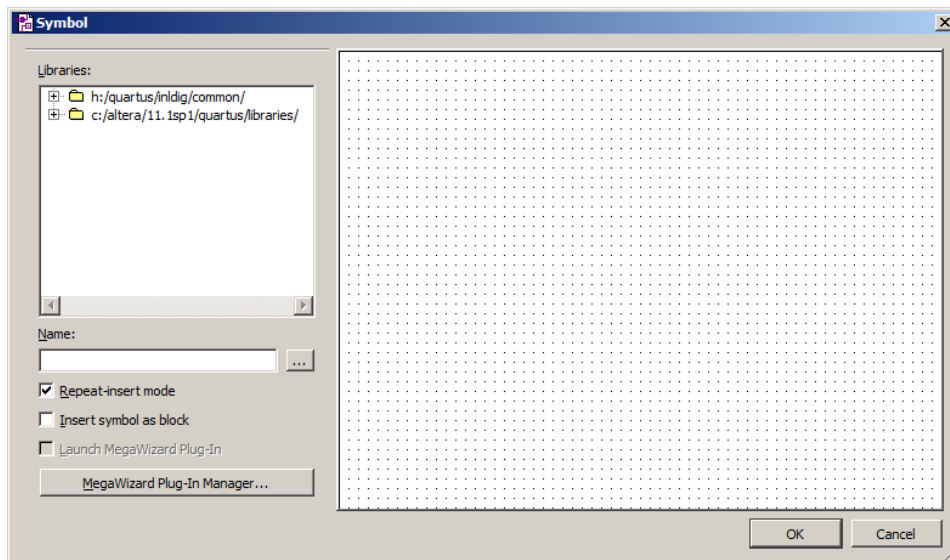
**Figuur 4.9:** *Overzicht Quartus IDE na aanmaken nieuw BDF-bestand.*

Bovenaan het edit-venster staat een rij knoppen. De twee belangrijkste zijn de poortinvoer-knop (symbol tool, ziet eruit als een AND-poort) en de pin-invoerknop (pin tool). Zie figuur 4.10.



**Figuur 4.10:** *Overzicht van de knoppen.*

Klik nu op het AND-poortje. Er wordt een dialoogvenster geopend. Zie figuur 4.11.

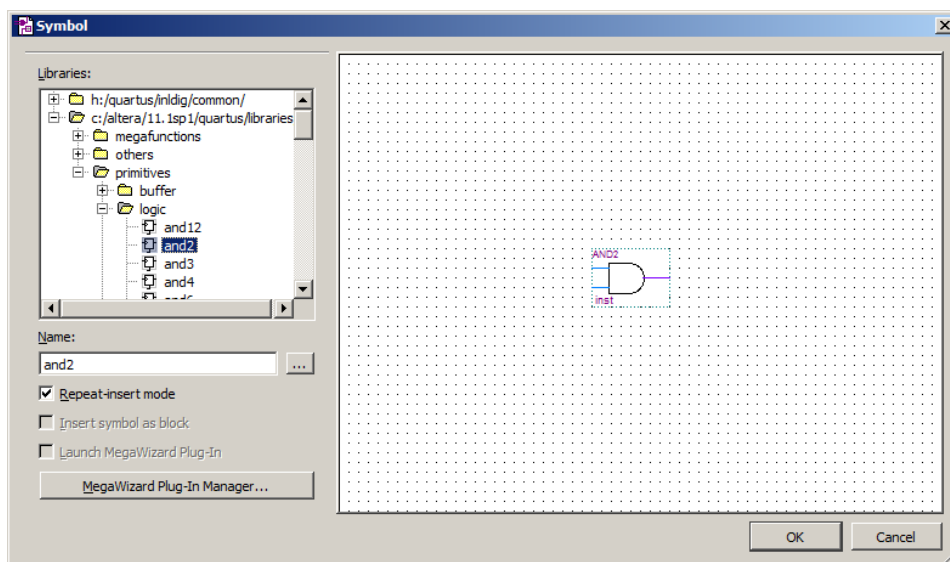


**Figuur 4.11:** *Selectie component.*

Linksboven staan de *libraries* (bibliotheken) vermeld. Hierin zijn de poorten opgenomen die we gaan gebruiken.

Noot: als de eerste regel in figuur 4.11 niet ziet, dan heb je bij het installeren van de projectomgeving niet de juiste mappenstructuur aangemaakt. Zorg ervoor dat de mappen `common` en `tutorial` in de map `H:\QUARTUS\INLDIG\` geplaatst zijn.

Open nu de tweede library tot op het niveau van `logic` en selecteer de AND2-poort. Zie figuur 4.12.



**Figuur 4.12:** *Selectie AND2-poort.*

Klik op **OK**. De AND2-poort “hangt” nu aan de cursor. Plaats de poort ergens in het midden van het edit-venster.

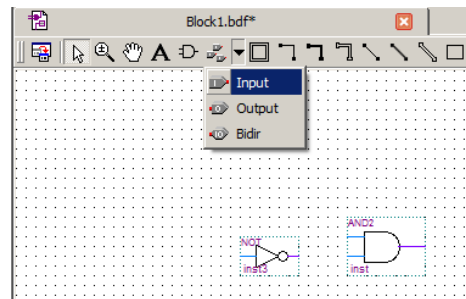
Noot: na het plaatsen “hangt” er opnieuw een AND2-poort aan de cursor. Je kan er dus nog één plaatsen. Dit wordt de *Repeat-insert Mode* genoemd. Druk op de **ESC**-toets om dit af te

breken.

Noot: in figuur 4.12 zie je een niveau genaamd others. Gebruik **geen** poorten uit dit deel van de bibliotheek.

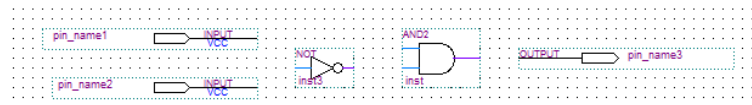
Voeg nu op gelijke wijze een NOT-poort in. Deze poort zit iets lager in dezelfde kolom.

Nu moeten de ingangs- en uitgangspoorten nog worden geplaatst. Dit kan op twee manieren: via de symbol tool (maar dan onder het niveau pin) of via de pin tool. Klik hiervoor op het driehoekje van pin tool pictogram en selecteer Input. Zie figuur 4.13.



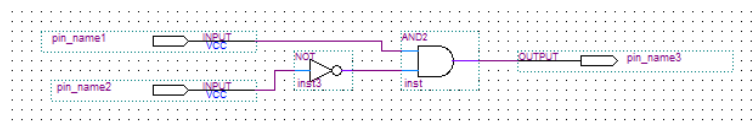
**Figuur 4.13:** Selectie input-poort.

Plaats nu twee input-pinnen in het edit-venster. Selecteer via de pin tool nu Output en plaats een output-pin. Zie figuur 4.14.



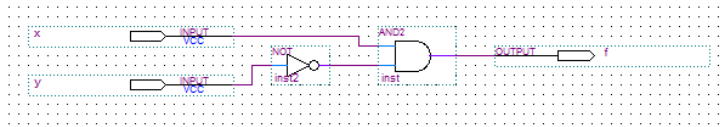
**Figuur 4.14:** Alle geplaatste componenten.

Nu moeten de diverse pinnen en poorten aan elkaar verbonden worden. Dat gaat vrij makkelijk. Plaats de cursor boven het eindpunt van de bovenste input-pin. Druk op de linker muisknop en houdt deze ingedrukt. Sleep nu naar vlak voor de bovenste ingang van de AND2-poort en laat de muisknop los. Er verschijnt nu een blauwe lijn. Maak nu een kleine lijn haaks naar beneden en vervolgens een lijn naar AND2-poort. Klik dan even in het edit-veld zodat de blauwe lijn gedeselecteerd wordt. De lijn wordt nu paars. Verbind vervolgens ook de andere componenten zoals aangegeven in figuur 4.15.



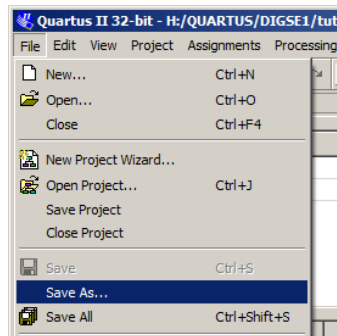
**Figuur 4.15:** Alle geplaatste componenten met verbindingen.

Als laatste stap moeten de ingangen en uitgangen nog een nieuwe naam krijgen. Dubbelklik precies op de naam pin\_name1. De naam wordt nu geselecteerd. Verander dit in x (kleine letter). Vervang nu ook pin\_name2 door y en pin\_name3 door f. Het geheel is te zien in figuur 4.16.



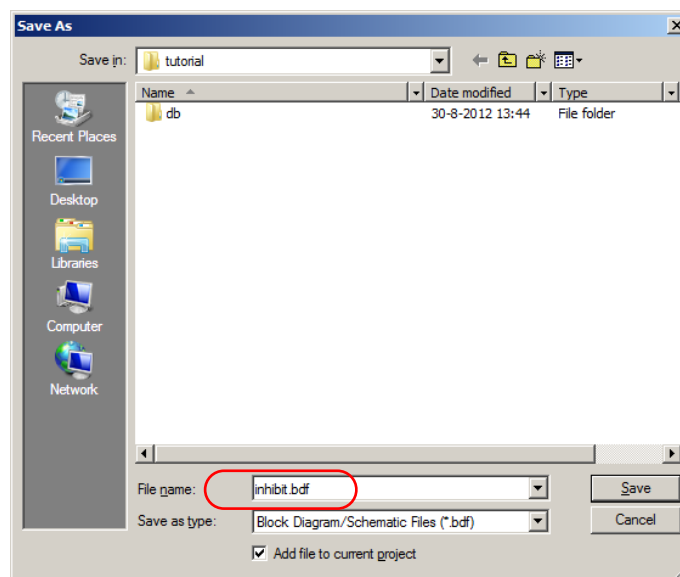
**Figuur 4.16:** Schema met nieuwe pinnamen.

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



**Figuur 4.17:** Bestand opslaan.

Er wordt een scherm geopend waarin de juiste map en de bestandsnaam ingevuld kunnen worden. Zie figuur 4.18. Er wordt een bestandsnaam voorgesteld, wijzig dit in `inhibit.bdf`. Let op het vinkje bij Add file to current project.



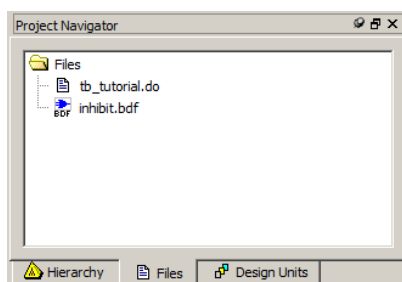
**Figuur 4.18:** Opgeven bestandsnaam BDF-bestand.

**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.

**Noot:** bestandsnaam *niet* met een cijfer beginnen!

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

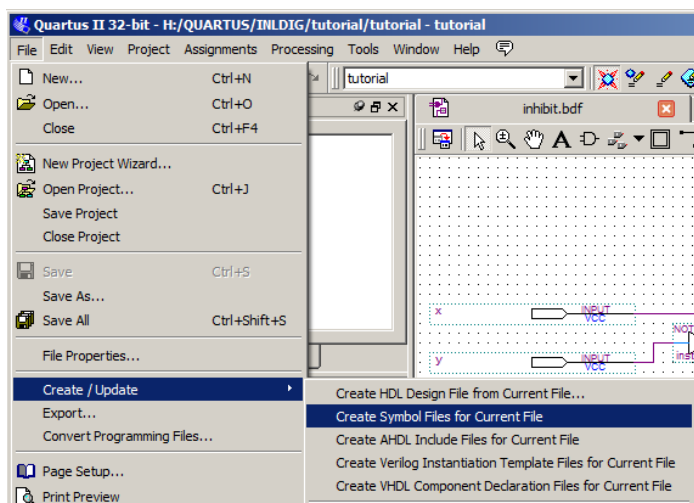


**Figuur 4.19:** Het opgeslagen bestand staat in de lijst van bestanden.

## 4.5 Aanmaken van symbool van het huidige bestand

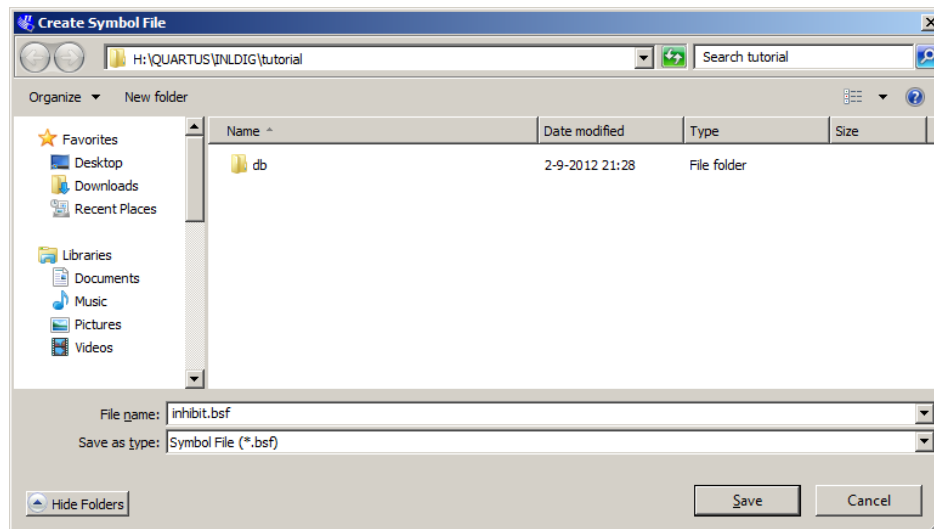
We gaan het schema dat net is opgeslagen in een ander schema gebruiken. Hiervoor moet eerst een *symbol* worden aangemaakt van het schema. Dit symbol komt dan terug in de *library*.

Open het bestand `inhibit.bdf` of, als het bestand al geopend is, selecteer in het edit-venster het tabblad van het bestand. Selecteer via het menu **File**→**Create/Update**→**Create Symbol Files for Current File**. Zie figuur 4.20.



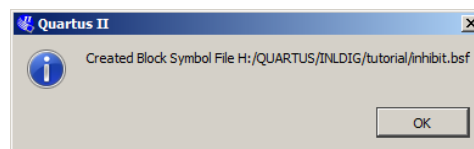
**Figuur 4.20:** Aanmaken nieuw symbol.

Er wordt een dialoogvenster geopend waarin een bestandsnaam kan worden ingevoerd. De voorgestelde naam `inhibit.bsf` is goed. Klik op **OK** om het bestand op te slaan. Zie figuur 4.21.



**Figuur 4.21:** Bestandsnaam nieuw symbool.

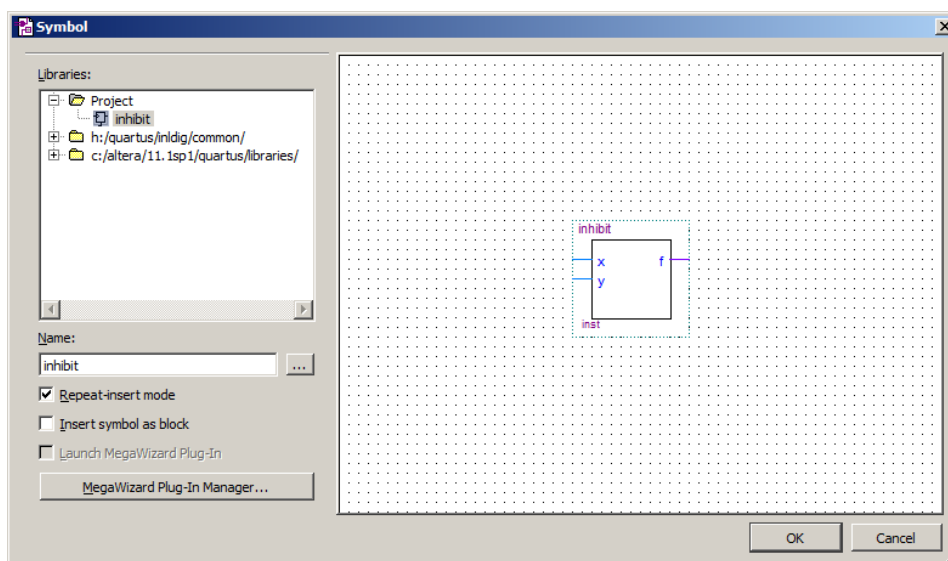
Quartus komt met de melding dat het bestand is aangemaakt. Zie figuur 4.22.



**Figuur 4.22:** Het bestand is opgeslagen.

## 4.6 Aanmaken van een tweede schemabestand

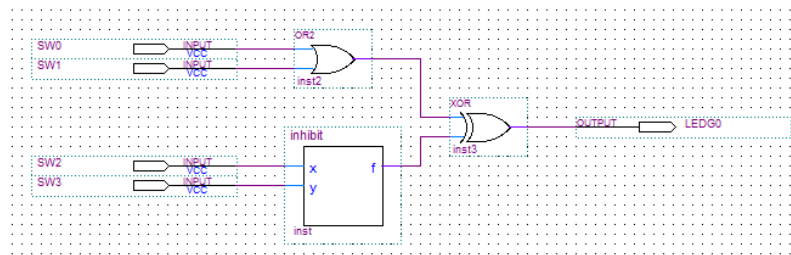
We maken nu op dezelfde wijze een nieuw schemabestand aan (zie figuren 4.7 en 4.8). Plaats daarin de net aangemaakt “poort”. Deze kan je vinden onder onder de library Project. Zie figuur 4.23.



**Figuur 4.23:** Selectie nieuwe component.



Plaats deze “poort” ergens in het edit-venster. Maak het schema verder af zoals in aangegeven in figuur 4.24. Let goed op de poorten; er moeten een OR2- en een XOR-poort geplaatst worden.



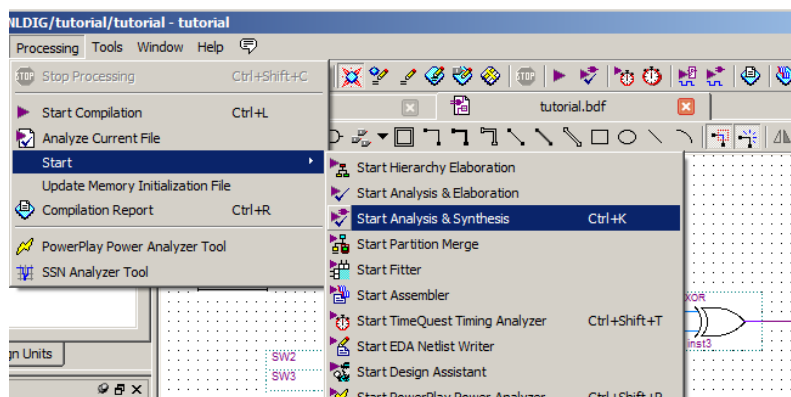
**Figuur 4.24:** Het bestand is opgeslagen.

Let goed op de namen van de ingangen en uitgang (hoofdletters!).

Sla het bestand op onder de naam `tutorial.bdf`. De naam van het bestand komt nu in de lijst van bestanden bij de Project Navigator.

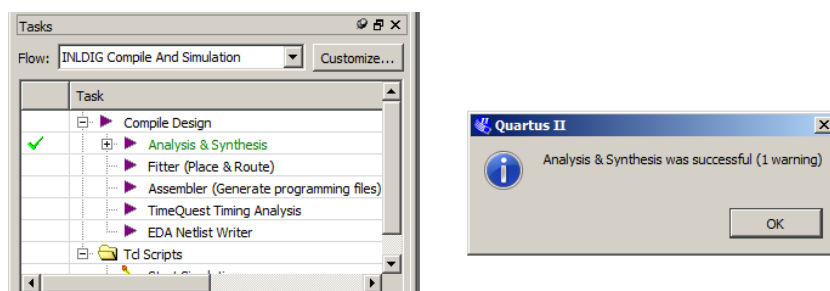
## 4.7 Eerste synthese

Om te controleren of het schema *syntactisch correct* is en er hardware voor kan worden gegene-reerd zullen we de synthesizer starten. Klik in de Project Manager op **Processing**→**Start**→**Start Analysis & Synthesis** of gebruik de sneltoetscombinatie **Ctrl+K**. Zie figuur 4.25.



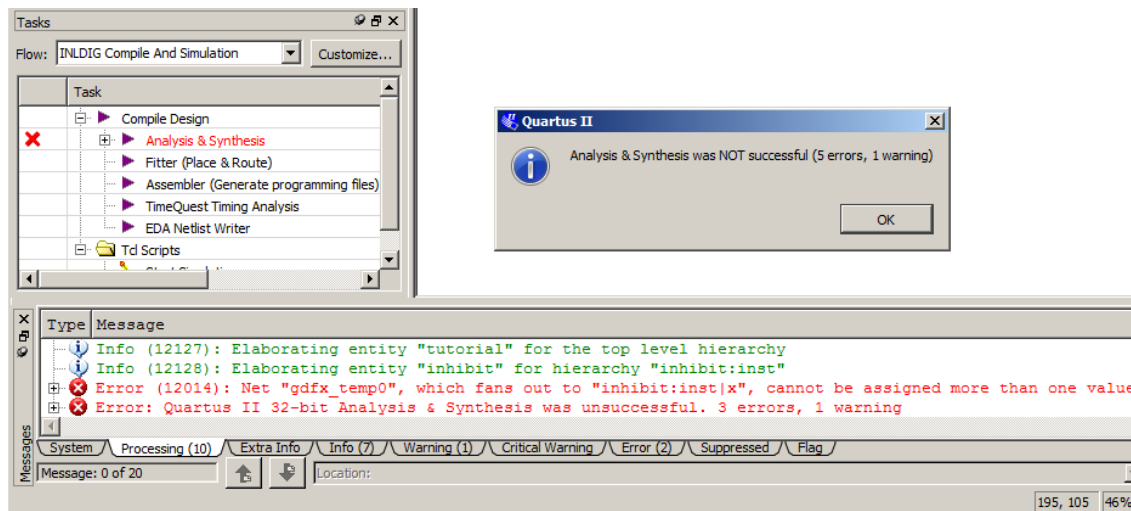
**Figuur 4.25:** Starten Analysis & Synthesis vanuit het menu.

Als deze stap gelukt is zie je onder Tasks een paar vinkjes verschijnen en je krijgt een melding (figuur 4.26). Klik op **OK** om verder te gaan.



**Figuur 4.26:** De analyse is gelukt.

Mocht je een fout hebben gemaakt, bijvoorbeeld twee ingangen aan elkaar verbonden, dan zal de synthesizer dat melden, zie figuur 4.27. Merk op dat de fout in de schakeling inhibit zit.



Figuur 4.27: De analyse is mislukt.

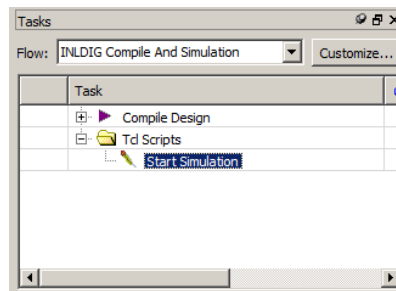
## 4.8 Simulatie poortschakeling

Nu de schema's ingevoerd zijn, moet het geheel gesimuleerd te worden. Dit droogzwemmen is bedoeld om te kunnen verifiëren of de schakeling, en dus de schema's, werkt volgens de specificaties. Het Quartus II pakket gebruikt hiervoor de externe simulator ModelSim van firma Mentor Graphics. We gebruiken de simulator om aan te tonen dat onze schakeling functioneel correct is. Het kan namelijk best zijn dat de schakeling gesynthetiseerd kan worden, maar dat de schakeling niet doet wat het zou moeten doen. Bij deze simulatie worden geen vertragingstijden meegenomen.

Voor simulatie is een zogenaamd scriptbestand nodig. Hierin staan opdrachten voor de simulator. 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. In het scriptbestand staat een aantal opdrachten die de simulator gebruikt om de ingangen mee aan te sturen. Dit zijn de zogenaamde *stimuli*. De simulator gebruikt deze waarden om de schakeling door te rekenen.

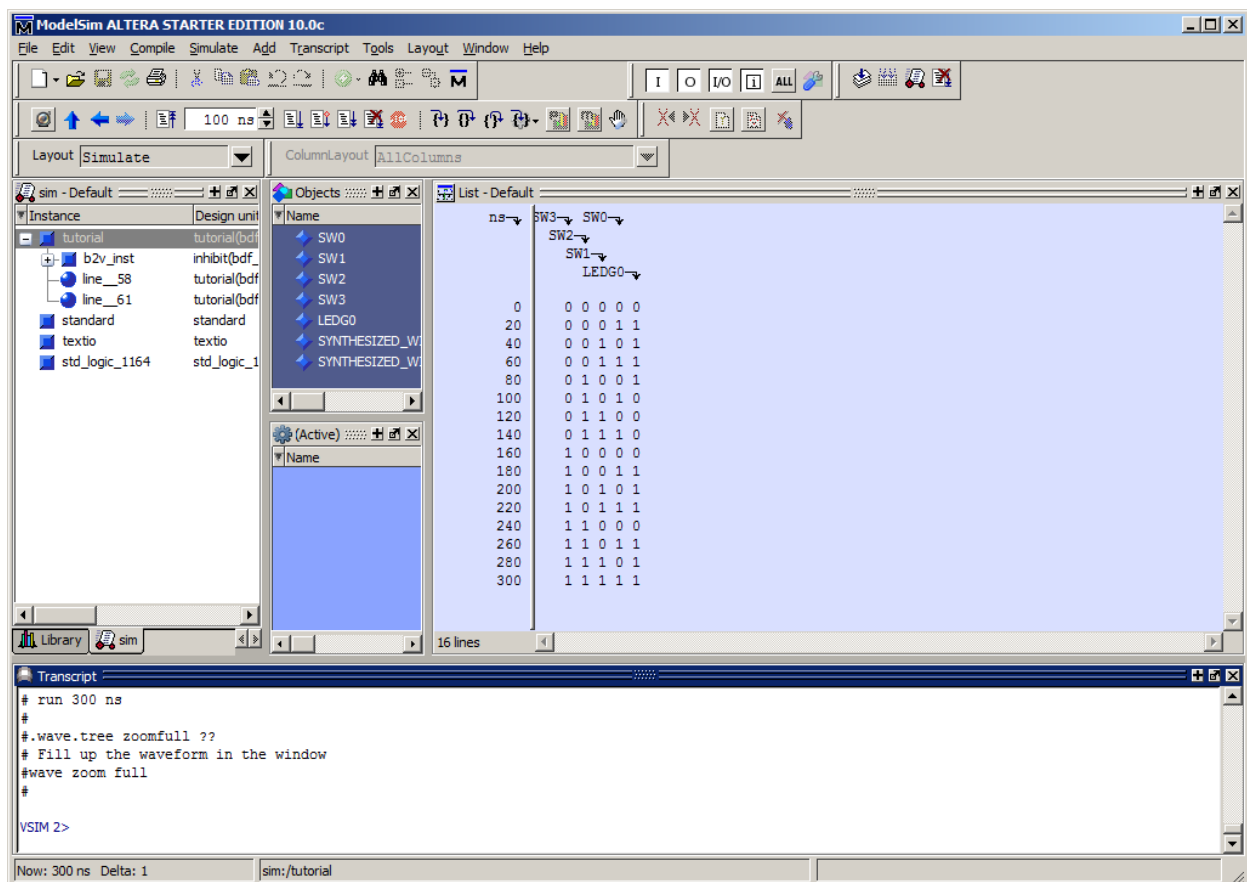
Noot: de commando's staan in het bestand `tb_tutorial.do` dat al is aangemaakt. Je kan de inhoud van het bestand bekijken, maar laat de inhoud ongewijzigd anders kan de simulatie mislukken.

Dubbelklik in het Tasks-venster op `Start Simulation`, zie figuur 4.28.



**Figuur 4.28:** Starten van de simulatie.

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 4.29).



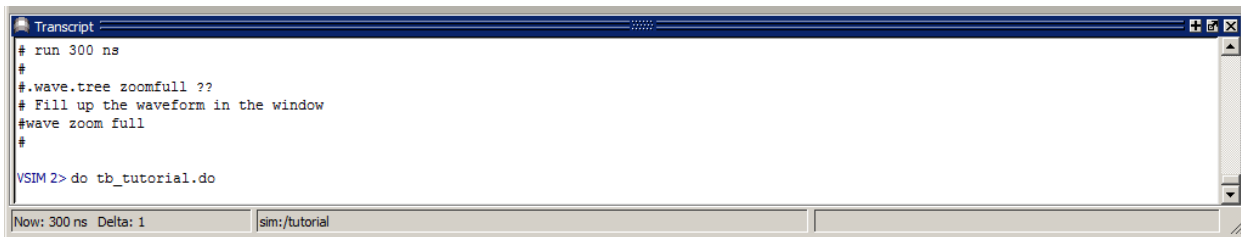
**Figuur 4.29:** Het resultaat van de simulatie.

Onderaan is de *Transcript Window*. Hier kan je ook losse commando's geven (voor gevorderden). Rechts is de *List Window*. De overige drie zijn op dit moment niet van belang.

In de List Window wordt de uitkomst van de simulatie weergegeven. Je ziet een aantal rijen. Een rij wordt voorafgegaan aan een getal (dat is een simulatietijd) en daarna vijf nullen of enen. De eerste vier stellen de waarden voor van de vier ingangen SW3 t/m SW0, de laatste de waarde van de uitgang LEDG0. Deze namen komen overeen met de schakelaars en leds van het ontwikkelbordje. Zie hoofdstuk 3.

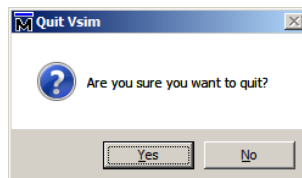
Je kan het commando-script nogmaals uitvoeren door in de transcript window op de ↑-toets

te drukken. Het laatst ingevoerde commando verschijnt dan. Druk op de **enter**-toets om dat commando uit te voeren. Zie figuur 4.30.



Figuur 4.30: Het transcript-window.

De simulatie is nu ten einde. Sluit de simulator af via het menu **File**→**Quit**. Er wordt een dialoogvenster geopend, zie figuur 4.31. Klik op **Yes**.

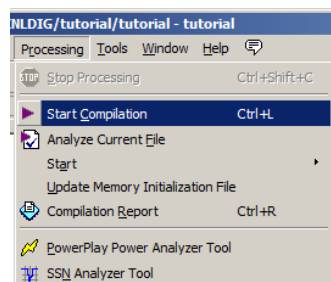


Figuur 4.31: ModelSim afsluiten.

## 4.9 Compilatie

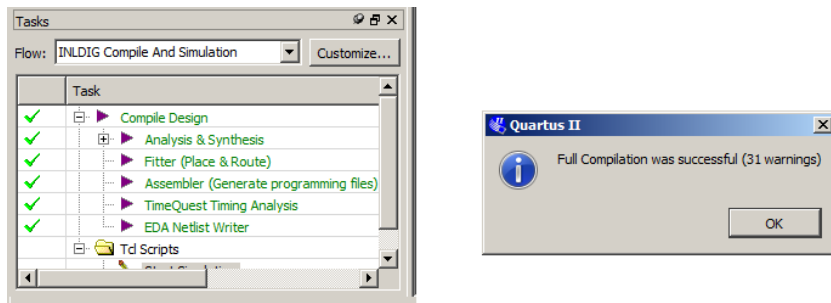
Nu de simulatie uitgevoerd is, wordt de schakeling gecompileerd. Compileren valt uiteen in twee delen: synthese en implementatie. Synthese houdt in dat de schakeling wordt vertaald met als resultaat een netlist; een beschrijving van de digitale logica in primitieven. Denk hierbij aan poorten, flipflops, LUTs (LookUp Table, ROM) of speciale voorzieningen zoals Phase Locked Loops of klokbuffers. Deze primitieven zijn voor elk configureerbaar type weer anders. Implementatie houdt in dat de primitieven worden *gemapped* op de LE's.

Start de compilatie via de menuoptie **Processing**→**Start Compilation** of gebruik de sneltoetscombinatie **Ctrl+L**. Zie figuur 4.32.



Figuur 4.32: 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 4.33. In het algemeen zijn de waarschuwingen niet problematisch, maar kijk de meldingen toch even na.



**Figuur 4.33:** De compilatie is gelukt.

Je krijgt na compilatie een rapport met alle verrichte werkzaamheden. Een interessant onderdeel hiervan is het aantal gebruikte *Logic Elements*. Hieraan kan je zien hoe groot je ontwerp is. Zie figuur 4.34.

Table of Contents	Flow Summary																																
<ul style="list-style-type: none"> <li>Flow Summary</li> <li>Flow Settings</li> <li>Flow Non-Default Global Settings</li> <li>Flow Elapsed Time</li> <li>Flow OS Summary</li> <li>Flow Log</li> <li>Analysis &amp; Synthesis</li> <li>Fitter</li> <li>Assembler</li> <li>TimeQuest Timing Analyzer</li> <li>EDA Netlist Writer</li> </ul>	<table> <tr> <td>Flow Status</td><td>Successful - Fri Aug 31 15:17:08 2012</td></tr> <tr> <td>Quartus II 32-bit Version</td><td>11.1 Build 216 11/23/2011 SP 1 SJ Web Edition</td></tr> <tr> <td>Revision Name</td><td>tutorial</td></tr> <tr> <td>Top-level Entity Name</td><td>tutorial</td></tr> <tr> <td>Family</td><td>Cyclone III</td></tr> <tr> <td>Device</td><td>EP3C16F484C6</td></tr> <tr> <td>Timing Models</td><td>Final</td></tr> <tr> <td>Total logic elements</td><td>1 / 15,408 (&lt; 1 %)</td></tr> <tr> <td>  Total combinational functions</td><td>1 / 15,408 (&lt; 1 %)</td></tr> <tr> <td>  Dedicated logic registers</td><td>0 / 15,408 (0 %)</td></tr> <tr> <td>Total registers</td><td>0</td></tr> <tr> <td>Total pins</td><td>5 / 347 (1 %)</td></tr> <tr> <td>Total virtual pins</td><td>0</td></tr> <tr> <td>Total memory bits</td><td>0 / 516,096 (0 %)</td></tr> <tr> <td>Embedded Multiplier 9-bit elements</td><td>0 / 112 (0 %)</td></tr> <tr> <td>Total PLLs</td><td>0 / 4 (0 %)</td></tr> </table>	Flow Status	Successful - Fri Aug 31 15:17:08 2012	Quartus II 32-bit Version	11.1 Build 216 11/23/2011 SP 1 SJ Web Edition	Revision Name	tutorial	Top-level Entity Name	tutorial	Family	Cyclone III	Device	EP3C16F484C6	Timing Models	Final	Total logic elements	1 / 15,408 (< 1 %)	Total combinational functions	1 / 15,408 (< 1 %)	Dedicated logic registers	0 / 15,408 (0 %)	Total registers	0	Total pins	5 / 347 (1 %)	Total virtual pins	0	Total memory bits	0 / 516,096 (0 %)	Embedded Multiplier 9-bit elements	0 / 112 (0 %)	Total PLLs	0 / 4 (0 %)
Flow Status	Successful - Fri Aug 31 15:17:08 2012																																
Quartus II 32-bit Version	11.1 Build 216 11/23/2011 SP 1 SJ Web Edition																																
Revision Name	tutorial																																
Top-level Entity Name	tutorial																																
Family	Cyclone III																																
Device	EP3C16F484C6																																
Timing Models	Final																																
Total logic elements	1 / 15,408 (< 1 %)																																
Total combinational functions	1 / 15,408 (< 1 %)																																
Dedicated logic registers	0 / 15,408 (0 %)																																
Total registers	0																																
Total pins	5 / 347 (1 %)																																
Total virtual pins	0																																
Total memory bits	0 / 516,096 (0 %)																																
Embedded Multiplier 9-bit elements	0 / 112 (0 %)																																
Total PLLs	0 / 4 (0 %)																																

**Figuur 4.34:** Overzicht van het resultaat van de compilatie.

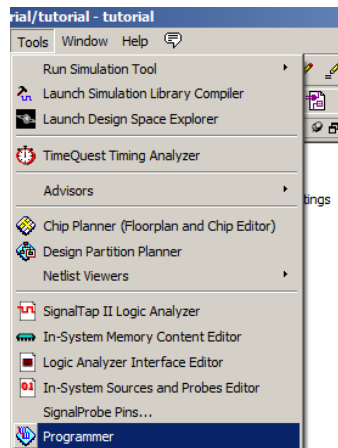
## 4.10 Configureren van de Cyclone III

De compilatiestap is nu afgerond. We gaan verder met het configureren van de Cyclone III. Dat doen we gaan door het configuratiebestand in deze component te 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.

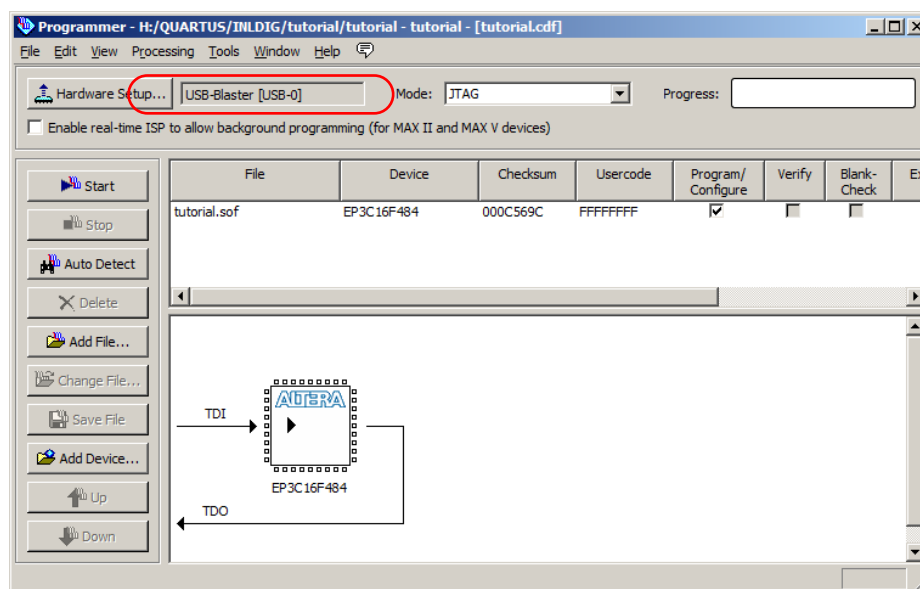
**Raadpleeg de docent voordat je de apparatuur aansluit en aanzet!**

Selecteer in de Project Manager de menuoptie **Tools**→**Programmer** (figuur 4.35).



**Figuur 4.35:** *Starten van de programmer.*

De programmer wordt gestart (figuur 4.36).

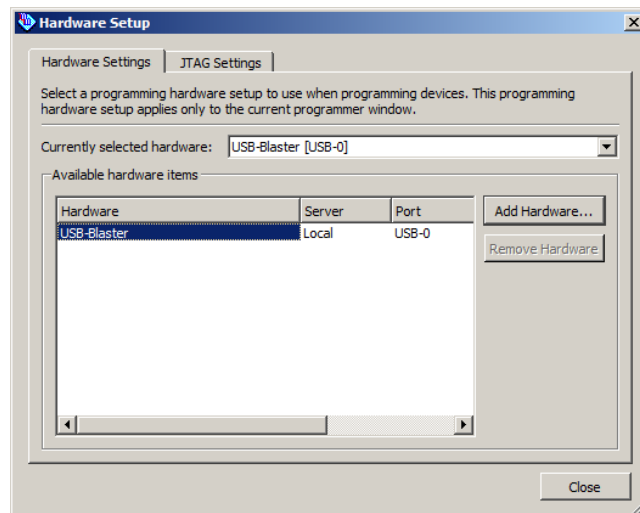


**Figuur 4.36:** *Overzicht van de Programmer IDE.*

Je kan zien of de programmer het ontwikkelbord heeft gevonden als in het veld naast Hardware Setup de regel USB-Blaster [USB-0] staat.

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 4.37.

Je ziet ook dat de programmer een bestand tutorial.sof heeft geselecteerd. Dit is het configuratiebestand dat in de Cyclone III geladen moet worden.



**Figuur 4.37:** *Selecteren van de USB-Blaster download-hardware.*

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

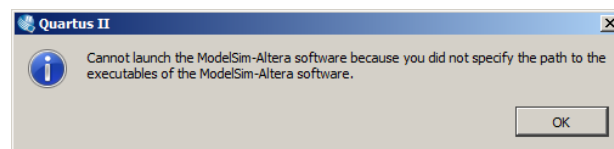
De tutorial is nu ten einde. Veel succes met het practicum.

## 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 Instellen pad naar ModelSim

Als in Quartus het pad naar ModelSim niet correct is ingesteld, krijg je de volgende foutmelding (figuur 5.1).



**Figuur 5.1:** De simulator kan niet worden gestart.

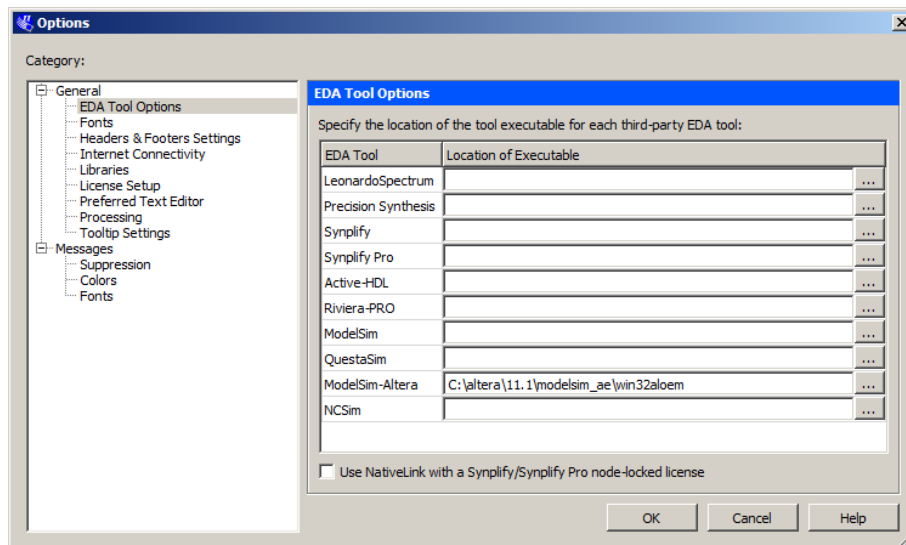
Ga als volgt te werk:

- 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\11.1\modelsim\_ae\win32aloem

Op je eigen PC hangt dat af van het installatie-pad. Bij gebruik van de Web Edition moet je modelsim\_ae vervangen voor modelsim\_ase. Zie figuur 5.2.

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



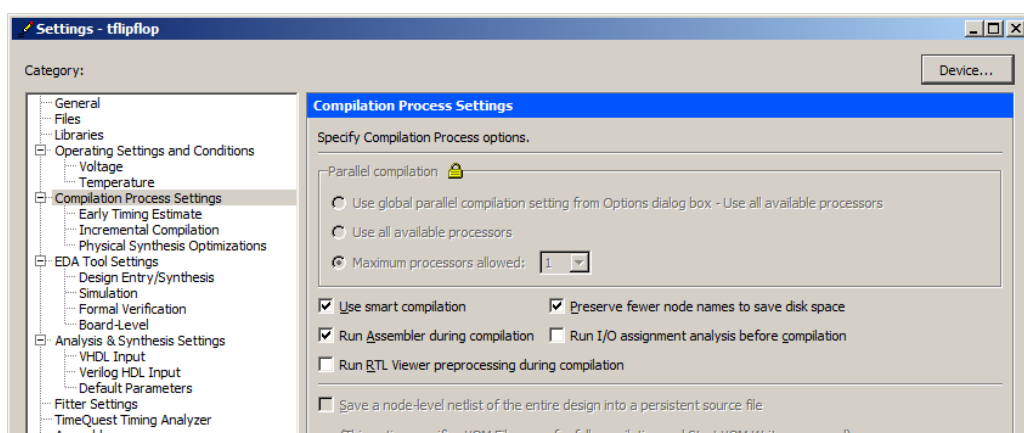


Figuur 5.2: Instellen van pad naar ModelSim.

## 5.2 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 5.3.



Figuur 5.3: Instellen van van de optie Smart compilation.

### 5.3 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.

### 5.4 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.

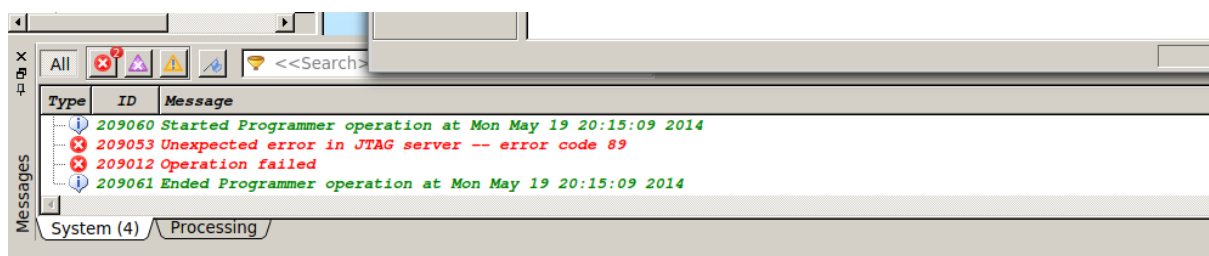
- Open de map waar het gebruikersprofiel is opgeslagen, meestal iets in de trant van C:\Users\<gebruikersnaam>
- Open het initialisatie-bestand quartus2.ini
- Voeg de volgende code toe, uiteraard met de juiste padnaam

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

- Sluit het bestand
- Start Quartus opnieuw op

### 5.5 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 5.4.



**Figuur 5.4:** 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 (let hierbij goed op het afbreken van de regels):

```
# USB-Blaster

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

# USB-Blaster II
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="09fb", ATTRS{
    idProduct}=="6010", MODE="0666", SYMLINK+="usbblaster2/%k"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="09fb", ATTRS{
    idProduct}=="6810", MODE="0666", SYMLINK+="usbblaster2/%k"
```

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

## 5.6 Bestandsnaam en entity-naam

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`).

Het is raadzaam om de bestandsnaam (zonder extensie) hetzelfde te houden als de entity-naam. Daarmee voorkom je problemen.

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.

## 5.7 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.

De mappen `db`, `incremental_db`, `output_files` en `simulation` kan je gewoon verwijderen. Let erop dat je geen eigen aangemaakte bestanden in de map `simulation` moet hebben staan.

Onderstaand script kan je draaien in een Windows-command box en verwijdert bijna elk bestand dat niet nodig is (de bovengenoemde mappen met inhoud moet je eerst zelf verwijderen).



```
1 @echo off
2 rem
3 rem
4 echo.
5 echo This program will delete all unnessecary files from Quartus projects.
6 echo.
7 echo Please be VERY carefull! Press Ctrl-C to break.
8 echo.
9 pause
10 echo.
11
12 del /s *.done
13 del /s *.rpt
14 del /s *.sof
15 del /s *.pof
16 del /s *.summary
17 del /s *.jdi
18 del /s vsim.wlf
19 del /s *.bak
20 del /s transcript
21 del /s *assignment_defaults.qdf
22 del /s *.pin
23 del /s modelsim.ini
24 del /s *.qws
25 del /s *.smsg
26 del /s *.map
27 del /s *.cdf
28 del /s *.dpf
29
30 echo.
31 echo Done.
32 echo.
33 pause
```

**Listing 5.1:** *Windows opruimsript*

# 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.

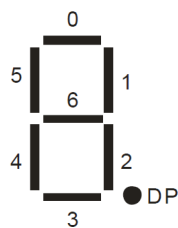
Type	Quartus-naam	Pinnaam	Opmerking
Clock	CLOCK_50	G21	Global Clock 1
Push Buttons	BUTTON[0]	H2	Ontdenderd, actief laag
	BUTTON[1]	G3	
	BUTTON[2]	F1	
Switches	SW[0]	J6	Niet ontdenderd, actief hoog
	SW[1]	H5	
	SW[2]	H6	
	SW[3]	G4	
	SW[4]	G5	
	SW[5]	J7	
	SW[6]	H7	
	SW[7]	E3	
	SW[8]	E4	
	SW[9]	D2	
Leds	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	Quartus-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

## C. INLDIG-FLOW ONDER LINUX

Tijdens het practicum wordt gebruik gemaakt van een eigen *flow*. In een flow staan de handelingen die door Quartus gedaan moeten worden om tot het gewenste resultaat te komen. Deze flow is nodig voor het uitvoeren van de practicumopdrachten.

Er is echter een probleem: ModelSim kan alleen maar VHDL-code (en Verilog) simuleren. Er is een script gemaakt (`start_sim.tcl`) dat alle `.bdf`-bestanden vertaalt naar VHDL-code en vervolgens de simulator start. Het script is zo geschreven dat het ook op Windows draait.

Om de flow op Linux te laten draaien moet je de volgende handelingen verrichten:

- Installeer de Quartus-software op een standaard plaats, bijvoorbeeld `opt/bin/`
- Installeer de Modelsim-software<sup>4</sup> onder de Altera-root (met versienummer), meestal iets van `/opt/bin/altera/13.0sp1/`
- Download het bestand `inldig_common_tutorial.zip` van BlackBoard of de website <http://ds.opdenbrouw.nl/quartus/>
- Pak het bestand uit in een map, bijvoorbeeld `/home/username/QUARTUS/INLDIG`. Je krijgt dan twee mappen genaamd `common` en `tutorial`

Kopieer het bestand `tmwc_INLDIG_Compile_And_Simulation.tmf` in de directory `common` naar de directory `/home/username/.quartus.altera`. Als je die directory niet hebt, moet je één keer Quartus starten of zelf aanmaken. Wijzig het pad onderin het bovengenoemde bestand naar de juiste locatie. **Let op: geen spaties in de padnaam!** Als voorbeeld is een pad van de gebruiker jesse opgenomen.

```
... beginstuk weggelaten ...
<task>
  <id>Start Simulation</id>
  <name>Start Simulation</name>
  <item_bitmap>tcl_command</item_bitmap>
  <status_ok_if>project_is_open</status_ok_if>
  <action type =
    "tcl_command">/home/jesse/QUARTUS/INLDIG/common/start_sim.tcl</action>
  </task>
</tasks>
```

---

<sup>4</sup> Vanaf versie 13.0 wordt ModelSim automatisch mee-geïnstalleerd.



## D. INLDIG-FLOW ONDER WINDOWS

Tijdens het practicum wordt gebruik gemaakt van een eigen *flow*. In een flow staan de handelingen die door Quartus gedaan moeten worden om tot het gewenste resultaat te komen. Deze flow is nodig voor het uitvoeren van de practicumopdrachten.

Er is echter een probleem: ModelSim kan alleen maar VHDL-code (en Verilog) simuleren. Er is een script gemaakt (*start\_sim.tcl*) dat alle .bdf-bestanden vertaalt naar VHDL-code en vervolgens de simulator start. Het script is zo geschreven dat het ook op Linux draait.

Om de flow op Windows te laten draaien moet je de volgende handelingen verrichten:

- Installeer de Quartus-software op een standaard plaats, bijvoorbeeld `c:\altera\`.
- Installeer de Modelsim-software<sup>5</sup> onder de Altera-root (met versienummer), meestal iets van `c:\altera\13.0sp1\`
- Download het bestand `inldig_common_tutorial.zip` van BlackBoard of de website <http://ds.opdenbrouw.nl/quartus/>.
- Pak het bestand uit in een map, bijvoorbeeld `D:\QUARTUS\INLDIG\`. Je krijgt dan twee mappen genaamd `common` en `tutorial`

De flow is opgeslagen onder de naam `tmwc_INLDIG_Compile_And_Simulation.tmf` in de map `common` en moet geïnstalleerd worden in de *profile map* van de gebruiker, meestal iets van `C:\Users\<gebruikersnaam>\` (natuurlijk zonder `<` en `>`).

Om nu de flow onder Windows te gebruiken moet je de padnaam in het bovengenoemde bestand wijzigen in de juiste locatie. **Let op: geen spaties in de padnaam!**

```
... beginstuk weggelaten ...

<task>
  <id>Start Simulation</id>
  <name>Start Simulation</name>
  <item_bitmap>tcl_command</item_bitmap>
  <status_ok_if>project_is_open</status_ok_if>
  <action type =
    "tcl_command">D:/QUARTUS/INLDIG/common/start_sim.tcl</action>
  </task>
</tasks>
```

---

<sup>5</sup> Vanaf versie 13.0 wordt ModelSim automatisch mee-geïnstalleerd.