

## Inleidende opgave

### Overzicht opdracht

Vereenvoudig manueel (op papier) een gegeven Booleaanse functie met behulp van Karnaugh.

Controleer daarna je oplossing met Vivado door middel van elaboratie en de schematic viewer.

Simuleer vervolgens alle mogelijke ingangswaarden en controleer of dit overeenkomt met de waarheidstabel van deze functie.

#### Leermiddelen voor deze opgave

Voor deze opgave wordt er verondersteld dat je de onderstaande hoofdstukken uit de cursus VHDL kent. Achter ieder topic kan je tussen vierkante haakjes de naam van overeenkomstige videoles vinden die je op de Mediasite van UA kan bekijken.

- 1.1 tot en met 1.4: inleiding en basisprincipes [[kanaal Basis VHDL: Inleiding VHDL](#)]
- 1.5: package, unresolved, std\_(u)logic,... [[kanaal Basis VHDL: std logic 1164](#)]
- 1.6 tot en met 1.8: entity, architecture, signal, combinatorische logica, regels combinatorisch proces, constant, variable [[kanaal Basis VHDL: Basisprincipes](#)]

Je hoeft niet zowel de cursus als de videolessen te bekijken, want daar wordt in principe hetzelfde besproken. Het is wel belangrijk dat je minstens een van de twee bekeken en begrepen hebt. Als het dan nog niet helemaal duidelijk is, dan mag je me ook altijd vragen stellen:

- in het labo, al of niet tijdens je eigen labosessie
- via mail [koen.lostrie@uantwerpen.be](mailto:koen.lostrie@uantwerpen.be) (24/24, 7/7, op feestdagen, vakanties,... Uiteraard kan ik niet altijd onmiddellijk antwoorden, maar meestal is dat toch vrij snel.)
- op afspraak in mijn bureau (U.152)

Op de volgende bladzijden wordt de opgave in meer detail beschreven. Dit zal in alle volgende opgaven ook op deze manier gebeuren.

**Uitzonderlijk** voor deze inleidende opgave maakt het niet heel veel uit als je de nodige cursus en/of videolessen pas achteraf bekijkt, maar voor alle andere opgaven is het wel ten eerste aangeraden dat je **eerst** de cursus/videoles(sen) bekijkt en dan pas aan de opgave begint.

Doe dit thuis op voorhand en niet tijdens de labosessie zelf, want zo zal je veel tijd verliezen. Beschouw de labosessies als contactmomenten waarop je de kans krijgt om zoveel mogelijk vragen te stellen zodat je thuis verder kan werken aan je opdrachten!

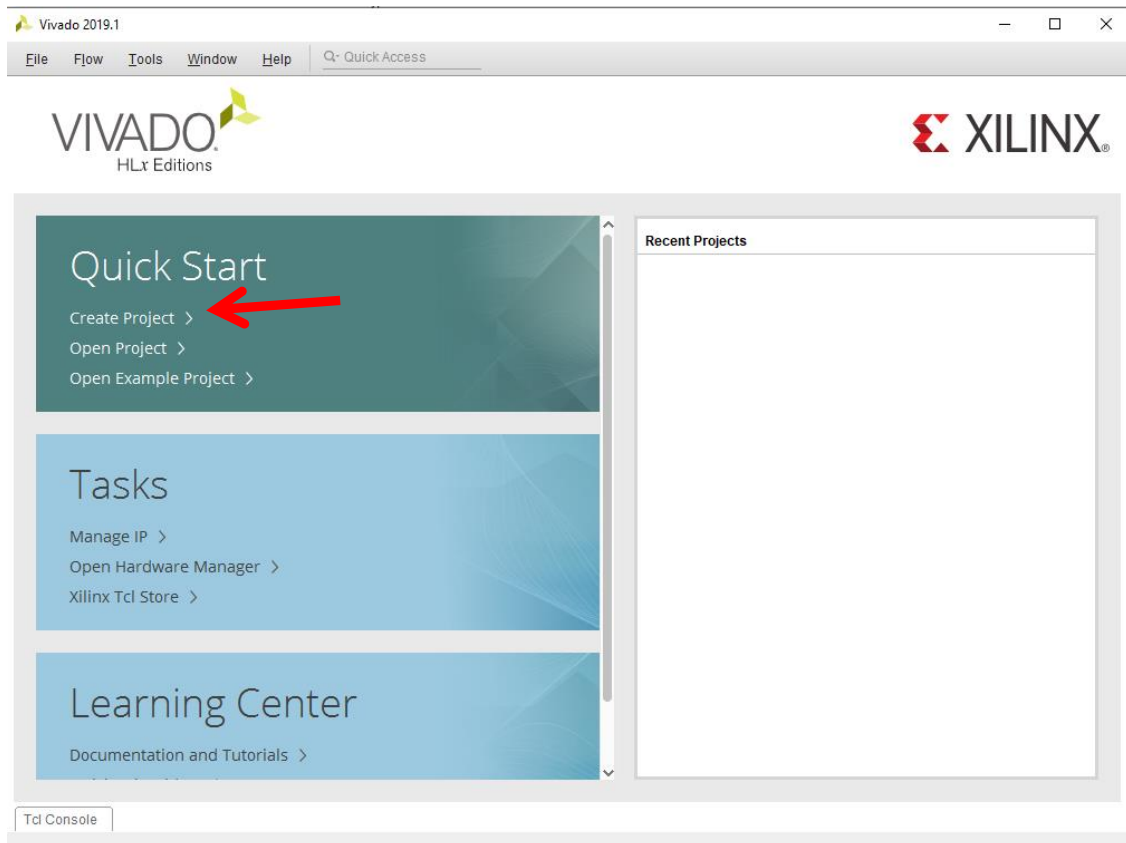
Nog een laatste opmerking voor we starten: Deze opgave staat beschreven in de vorm van een walkthrough: "klik hier", "typ dat", enz. Je kan hier zeer snel doorwerken en niets bijleren, maar daarmee ga je enkel jezelf benadelen, want je zal dit 100% zeker nodig hebben in AL je opgaven van dit jaar én nog enkele andere vakken de komende jaren. Probeer dus goed te begrijpen wat je allemaal aan het doen bent.

Je gaat een Vivado project aanmaken, VHDL bestanden toevoegen, bewerken en simuleren. Al deze dingen worden ook besproken in de videolessen/demo's [Vivado projecten](#) en [Foutmeldingen in Vivado](#) ([kanaal Xilinx Vivado](#)), die je ook zeer veel tijd en zorgen kan besparen in de komende opgaven. Bekijk deze ook **zeker**.

## Beschrijving: project aanmaken

Je krijgt een Booleaanse functie die vereenvoudigd moet worden. Dit zou je ondertussen moeten kunnen aan de hand van Karnaugh. Om je oplossing te kunnen verifiëren, start je Vivado op en beschrijf je de gegeven (niet vereenvoudigde) functie in VHDL.

In het opstartscherm klik je vervolgens op “Create Project”:



Later zal je rechts onder “Recent Projects” je recente projecten kunnen open klikken.

Dit zal een wizard starten waar je de locatie en naam van je project kan invoeren. **Belangrijk: Zorg ervoor dat er nergens in het pad van je project een spatie zit!!** Dit wil zeggen dat je bureaublad meestal ook niet zo’n goed idee is, want dat wordt onder Windows dikwijls opgeslagen onder een folder waarin spaties zit. **Gebruik ook een lokale map, geen cloud- of netwerkmap (OneDrive of dergelijke), om latere problemen te vermijden, bv: “C:\VhdlDesigns\...” of zo. Voeg deze folder ook meteen toe aan de uitzonderingen van je virusscanner zodat je projecten niet geblokkeerd worden.** Virusscanners houden absoluut niet van Xilinx Vivado!

Op een bepaald moment kom je dan aan volgend scherm:

**New Project**

**Project Type**  
Specify the type of project to create.

☒ **RTL Project**  
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☒ **Do not specify sources at this time**

☐ **Post-synthesis Project**  
You will be able to add sources, view device resources, run design analysis, planning and implementation.

☐ **Do not specify design sources at this time**

☐ **I/O Planning Project**  
Do not specify design sources. You will be able to view part/package resources.

☐ **Imported Project**  
Create a Vivado project from a Synplify, XST or ISE Project File.

☐ **Example Project**  
Create a new Vivado project from a predefined template.

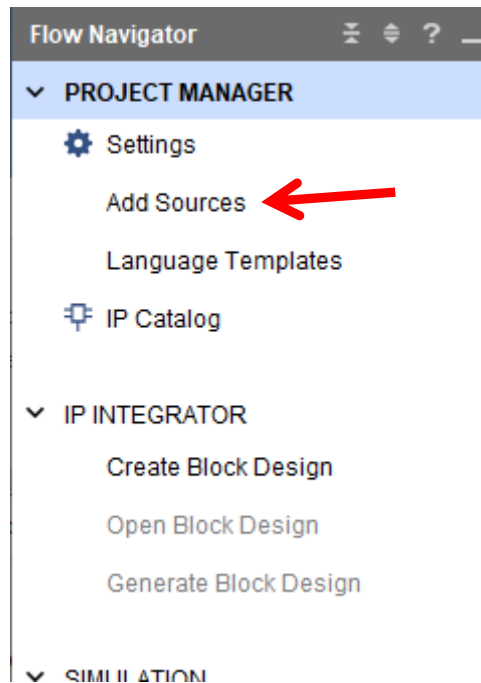
In onze opleiding gebruiken we enkel de bovenste optie “RTL project” en voor deze opgave vink je “Do not specify sources at this time” aan, omdat er nog geen gegeven VHDL bestand bestaat. Dit ga je dadelijk zelf aanmaken.

Bij de keuze van FPGA (“parts or boards”) mag je gewoon op “Next” en “Finish” klikken, vermits je enkel zal simuleren voor deze opgave en nog geen hardware zal configureren in een FPGA.

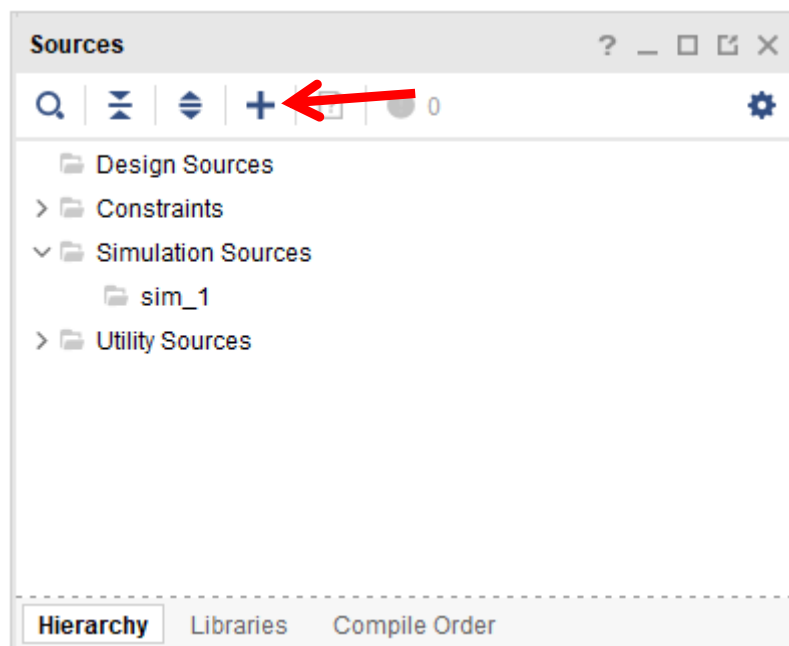
Nu opent het eigenlijke projectvenster waar je gaandeweg alle onderdelen van zal leren kennen. Ondertussen creëert Vivado een aantal folders onder je projectmap. De belangrijkste hiervan zijn:

- <projectnaam.sim>: hieronder komen alle bestanden met betrekking tot de simulatie van je ontwerp (zie verder)
- <projectnaam.srsc>: hieronder komen alle VHDL- (en andere bron-) bestanden van je ontwerp te staan. Het VHDL bestand dat je in de volgende stap zal toevoegen aan je project, zal hier dus ook onder komen te staan.

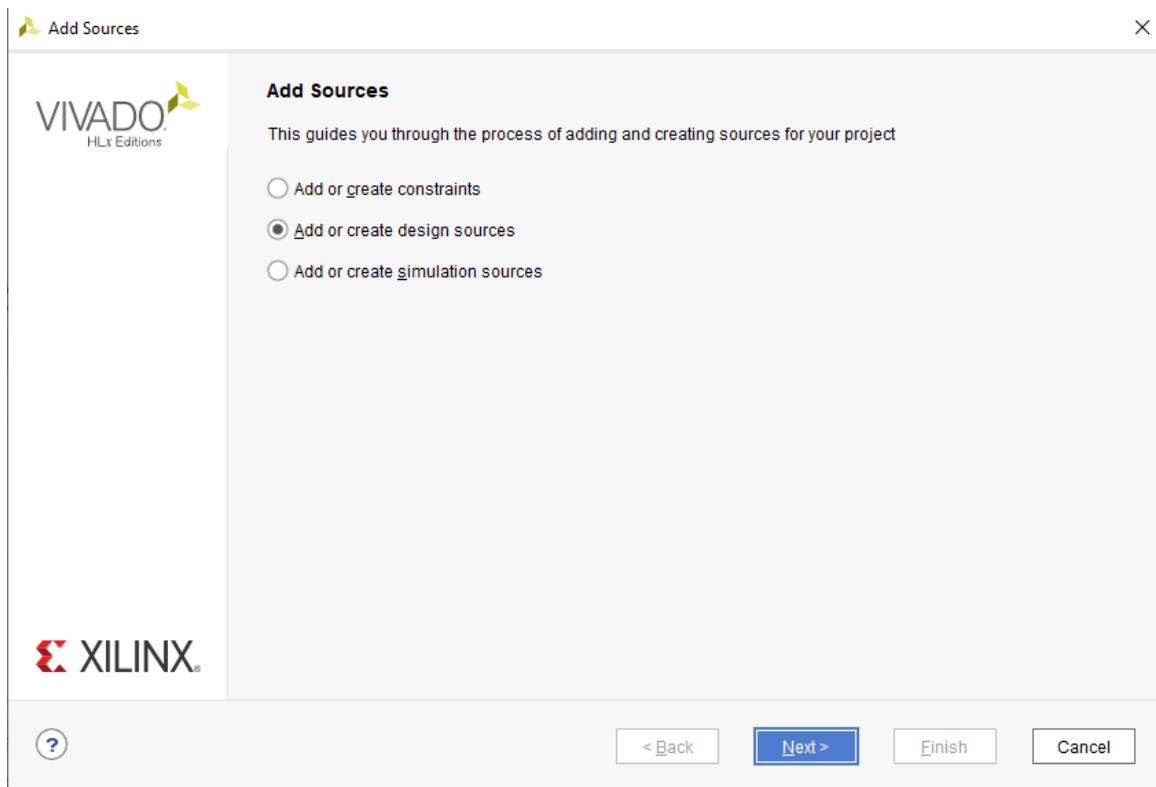
Om te beginnen, moet je een VHDL bestand toevoegen aan het project. Dit kan je doen door ofwel in de “Flow Navigator” op “Add Sources” te klikken:



ofwel door op het plusteken te klikken in het “Sources” venster:

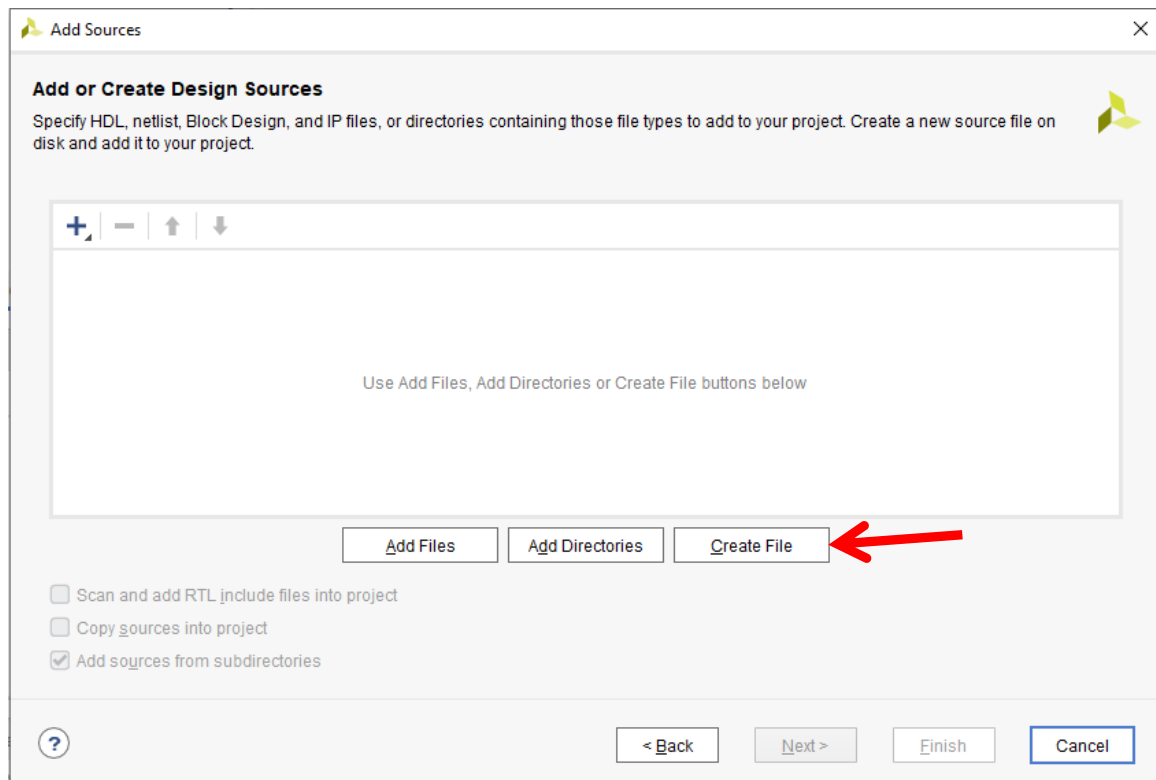


Dit opent in beide gevallen het volgende venster:



Hier ga je wel van alle opties gebruik maken tijdens deze sessies, maar voorlopig kies je “Add or create design sources” omdat je een VHDL ontwerpbestand wil toevoegen (waarin je hardware zal beschrijven). Later zal je zien waar de andere opties voor gebruikt worden.

Vermits er nog geen VHDL bestanden bestaan, klik je vervolgens op “Create File”.



Je kiest een naam van het bestand, **selecteert hier zeker VHDL en niet Verilog** (wat een andere hardware beschrijvende taal is) en klikt vervolgens op "Finish".

Vervolgens krijg je de kans om de naam van je ontwerp ("Entity name") te wijzigen en de in- en uitgangen te definiëren van het ontwerp.

**Let op:** Sommige studenten krijgen onderstaand scherm niet te zien en Vivado blijft oneindig vast zitten. Dit is waarschijnlijk een bug in Vivado die zich niet bij iedereen voordoet. Moest jij een van de gelukkigen zijn, dan kan je dit hoogstwaarschijnlijk oplossen door het volgende te doen:

Wijzig de optie in Vivado "Tools -> Settings -> Tool Settings -> Text Editor -> Syntax Checking -> Syntax checking" van Sigasi naar Vivado en herstart dan Vivado. Dit moet je dus enkel doen als je problemen ondervindt.

Normaal gezien krijg je echter onderstaand scherm, waarop je de nodige in- en uitgangen definieert:

Define a module and specify I/O Ports to add to your source file.  
For each port specified:  
MSB and LSB values will be ignored unless its Bus column is checked.  
Ports with blank names will not be written.

**Module Definition**

Entity name:

Architecture name:

**I/O Port Definitions**

Port Name	Direction	Bus	MSB	LSB
A	in	<input type="checkbox"/>	0	0
B	in	<input type="checkbox"/>	0	0
C	in	<input type="checkbox"/>	0	0
F	out	<input type="checkbox"/>	0	0

OK Cancel

Vivado zal nu je ontwerp toevoegen aan je "Sources" venster en als je hierop dubbelklikt, dan kan je het VHDL bestand bewerken:

PROJECT MANAGER - Opgave1

Sources

- Design Sources (1)
  - KarnaughVoorbeeld(Behavioral) (KarnaughVoorbeeld.vhd)**
- Constraints
- Simulation Sources (1)
  - sim\_1 (1)
- Utility Sources

hierop dubbelklikken opent rechts je bestand

Project Summary x KarnaughVoorbeeld.vhd x

F:\Workdir\1-BasisDigitaleElektronica1\Opgave1\Opgave1.srcs/sources\_1/new/KarnaughVoorbeeld.vhd

hier zie je het volledige pad naar je bestand. Onthoud dit, want je zal het nodig hebben!

```

20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating

```

Vivado heeft nu al een VHDL bestand gemaakt met een entity volgens jouw opgegeven in- en uitgangen en een lege architecture.

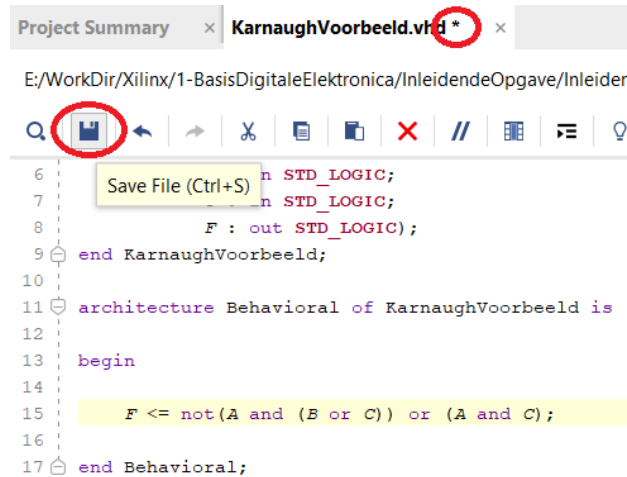
Er staan ook altijd een reser commentaarlijnen bij (beginnend met "--"), die je gerust mag verwijderen voor de leesbaarheid.

Uiteindelijk zal je iets in deze vorm krijgen:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity KarnaughVoorbeeld is
5     Port ( A : in STD_LOGIC;
6           B : in STD_LOGIC;
7           C : in STD_LOGIC;
8           F : out STD_LOGIC);
9 end KarnaughVoorbeeld;
10
11 architecture Behavioral of KarnaughVoorbeeld is
12
13 begin
14
15     F <= not(A and (B or C)) or (A and C);
16
17 end Behavioral;
```

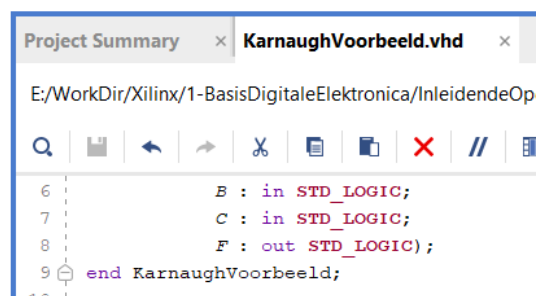
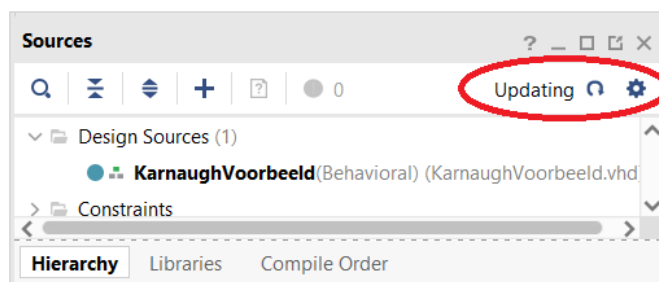
Hierbij moet je natuurlijk **het lijntje met "F <= ..." aanpassen voor je eigen functie**.<sup>1</sup>

**Let op! Zorg ervoor dat je IEDER lijntje code begrijpt! Waarom staat het hier? Wat doet het? Enz. Als dat niet het geval is, dan VRAAG je het!**



Merk op dat er een asterisk verschijnt naast de bestandsnaam bovenaan vanaf het ogenblik dat je iets wijzigt aan je bestand. Vergeet niet van je wijzigingen eerst op te slaan vooraleer je de verdere stappen gaat uitvoeren.

Vanaf je het bestand hebt opgeslagen, zal de asterisk verdwijnen én dan zal je ook links in het "Sources" venster zien dat Vivado je ontwerp aan het "updaten" is:

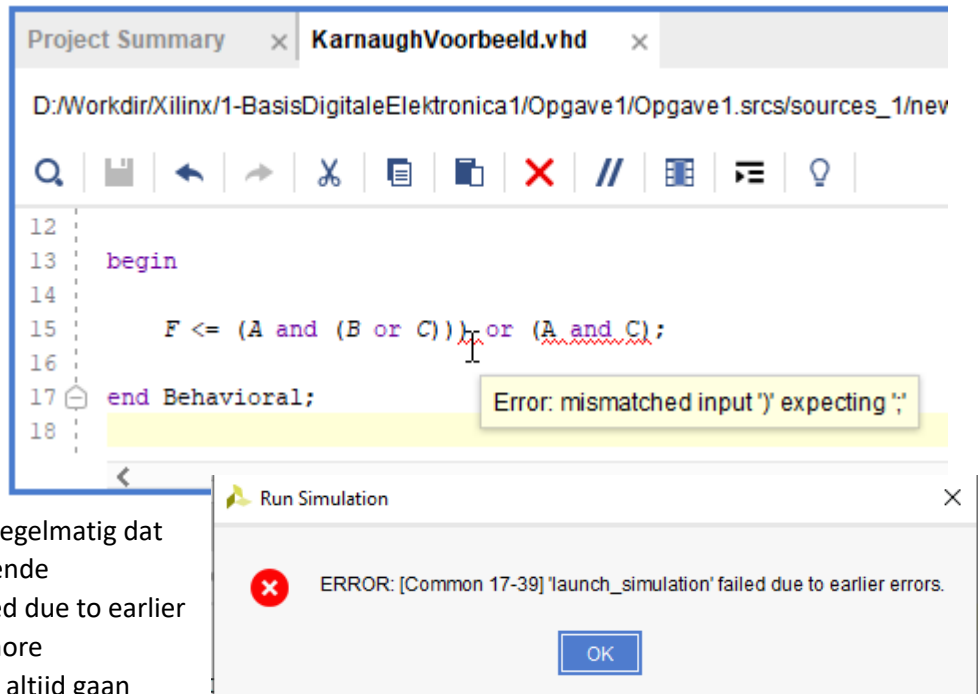


<sup>1</sup> Lees eerst de volgende pagina voor je verder gaat, want er staat nog een belangrijke tip.

## Beschrijving: foutmeldingen

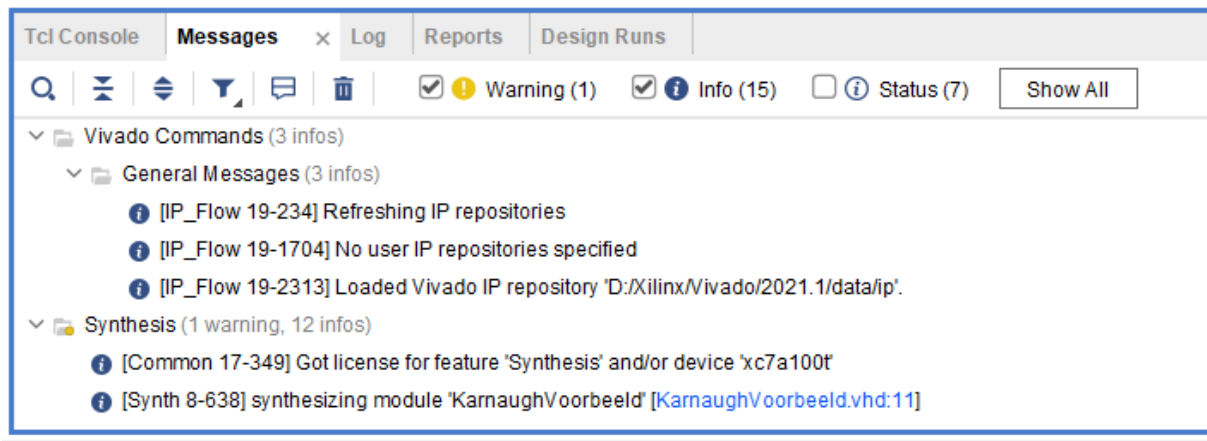
Tijdens het schrijven van je VHDL code zal je ongetwijfeld nog veel fouten maken. Vivado gaat hier niet altijd even duidelijk mee om, vandaar dat we hier toch even tijd aan besteden. Om te beginnen

zijn er verschillende plaatsen en momenten waarop Vivado foutmeldingen geeft. In de eerste plaats krijg je in je VHDL code zelf soms rode rimpellijntjes te zien wanneer er nog een fout zit. Als je hier je muisaanwijzer over plaatst (zonder te klikken), dan krijg je meer informatie over de foutmelding (zoals hier rechts geïllustreerd wordt door een haakje teveel in de code).



Het gebeurt spijtig genoeg ook regelmatig dat Vivado een of andere nietszeggende foutmelding geeft, zoals "... failed due to earlier errors" of "See Tcl console for more information". In dat geval kan je altijd gaan kijken in de volgende twee vensters:

- het "Messages" venster:

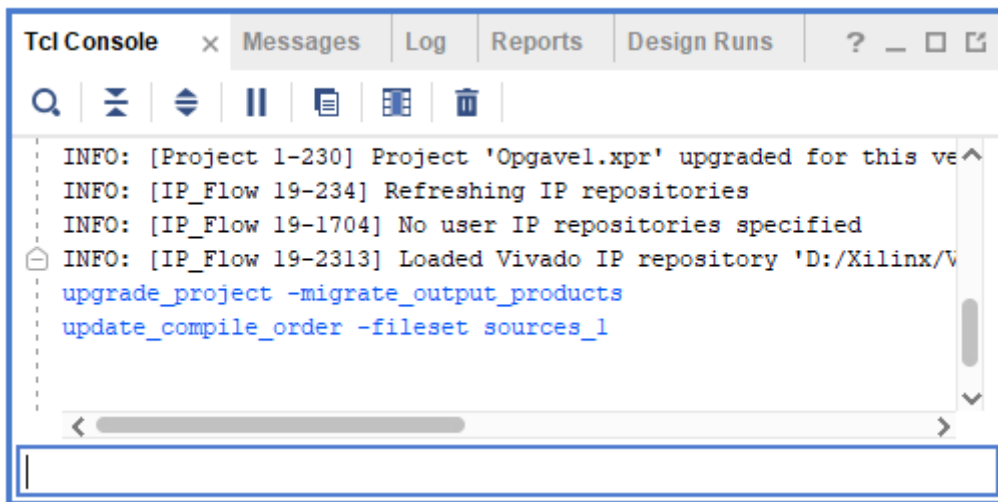


Foutmeldingen worden in verschillende gradaties gegroepeerd: status, info, warning en error. Je kan ze aan- en uitvinken bovenaan het venster. Vink gerust de "Info" boodschappen uit, maar laat zeker alle warnings en errors staan, want hier zal je anders nog belangrijk informatie missen. **Lees en begrijp alle foutmeldingen volledig! Een zin eindigt niet met het woord "error"!**

Een fout die de meeste studenten tegenkomen bij deze opgave, is te weinig haakjes plaatsen. In Booleaanse algebra heeft de vermenigvuldiging ("and") voorrang ten opzichte van de optelling ("or"), maar in VHDL is dat niet! Dat wil zeggen dat de volgende haakjes niet optioneel zijn: `F <= (A and B) or C;` want als je ze zou weglaten, dan weet VHDL niet of hij eerst de "and" of de "or" moet uitvoeren. Let hier op bij het maken van deze opgave!




- de "Tcl Console":



Vergeet hier zeker niet naar boven te scrollen, want meestal staan hier enkel de laatste boodschappen vermeld en is de eigenlijke foutmelding (meestal **NIET** in rood) niet meer zichtbaar.

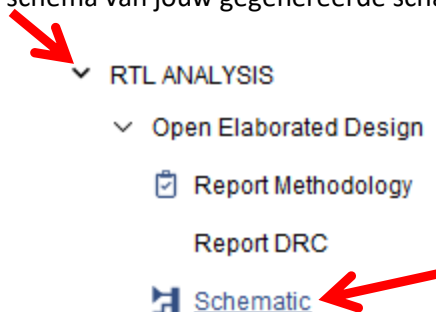
INFO: [VRFC 10-100] Analyzing VHDL file 'D:/Workdir/Xilinx/1-BasisDigitaleElektronica/Opgavel/Opgavel.srcs/sources\_1/new/Kar...'  
INFO: [VRFC 10-3107] analyzing entity 'KarnaughVoorbeeld'  
ERROR: [VRFC 10-4982] syntax error near ')' [D:/Workdir/Xilinx/1-BasisDigitaleElektronica/Opgavel/Opgavel.srcs/sources\_1/new/Kar...]  
ERROR: [VRFC 10-3782] unit 'behavioral' ignored due to previous errors [D:/Workdir/Xilinx/1-BasisDigitaleElektronica/Opgavel/Opgavel.srcs/sources\_1/new/Kar...]  
INFO: [VRFC 10-3070] VHDL file 'D:/Workdir/Xilinx/1-BasisDigitaleElektronica/Opgavel/Opgavel.srcs/sources\_1/new/KarnaughVoor...'  
INFO: [USF-XSim-69] 'compile' step finished in '3' seconds  
INFO: [USF-XSim-99] Step results log file: 'D:/Workdir/Xilinx/1-BasisDigitaleElektronica/Opgavel/Opgavel.sim/sim\_1/behav/xsim...'  
ERROR: [USF-XSim-62] 'compile' step failed with error(s). Please check the Tcl console output or 'D:/Workdir/Xilinx/1-BasisDi...'  
ERROR: [Vivado 12-4473] Detected error while running simulation. Please correct the issue and retry this operation.  
ERROR: [Common 17-39] 'launch\_simulation' failed due to earlier errors.

Besef ook dat deze Tcl Console blijft aanvullen zolang Vivado open staat. Dus eerdere foutmeldingen (die je al verbeterd hebt) blijven hier altijd staan, zolang je Vivado niet hebt afgesloten. Als je een "propere lei" wil hebben, dan kan je op het vuilbakje  klikken om het venster leeg te maken.

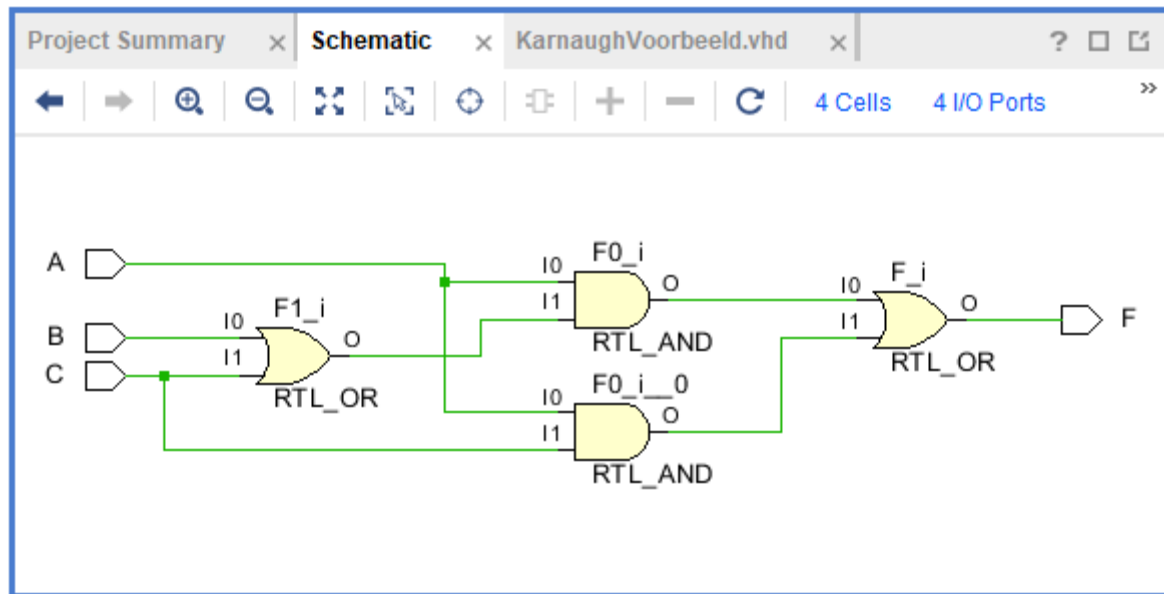
### Beschrijving: elaboratie

Als je geen foutmeldingen meer krijgt, dan laat je Vivado je ontwerp "elaboreren". Dit wil zeggen dat Vivado je VHDL code zal interpreteren en hiervan Booleaanse functies zal maken die dezelfde functionaliteit hebben als jouw code.

In de "Flow Navigator" kan je het vinkje links naast "RTL ANALYSIS" openklikken en vervolgens "Schematic" kiezen voor een schema van jouw gegenereerde schakeling.

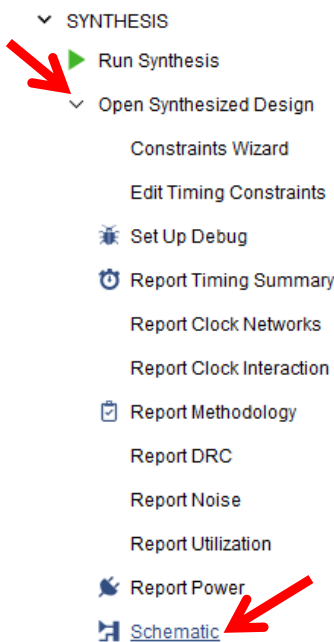
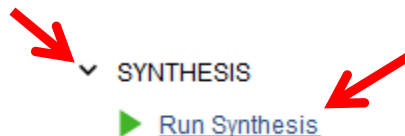


Het schema van de bovenstaande VHDL code ziet er dan als volgt uit:



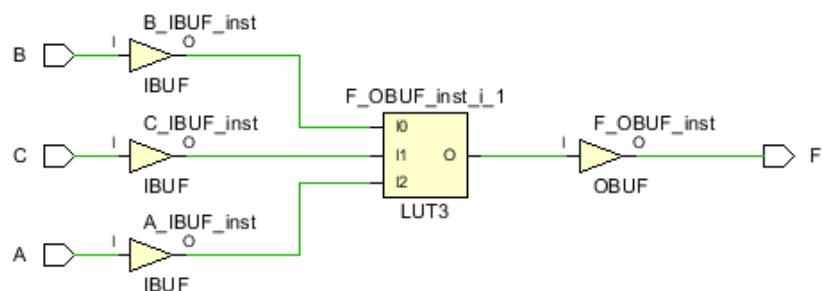
Merk op dat de gebruikte functie  $F = A(B + C) + AC$  niet in zijn meest eenvoudige vorm staat, want  $A(B + C) + AC = AB + AC + AC$ , dus de laatste term is overbodig.

Tijdens elaboratie wordt er dus nog geen optimalisatie of vereenvoudiging toegepast op je VHDL code. Hiervoor moet je eerst nog synthese opstarten. Dit doe je door onder "SYNTHESIS" op "Run Synthesis" te klikken:



Daarna kan je opnieuw een schema bekijken, ditmaal na vereenvoudiging. Klik hiervoor op "Schematic" onder "Open Synthesized Design" (zie links).

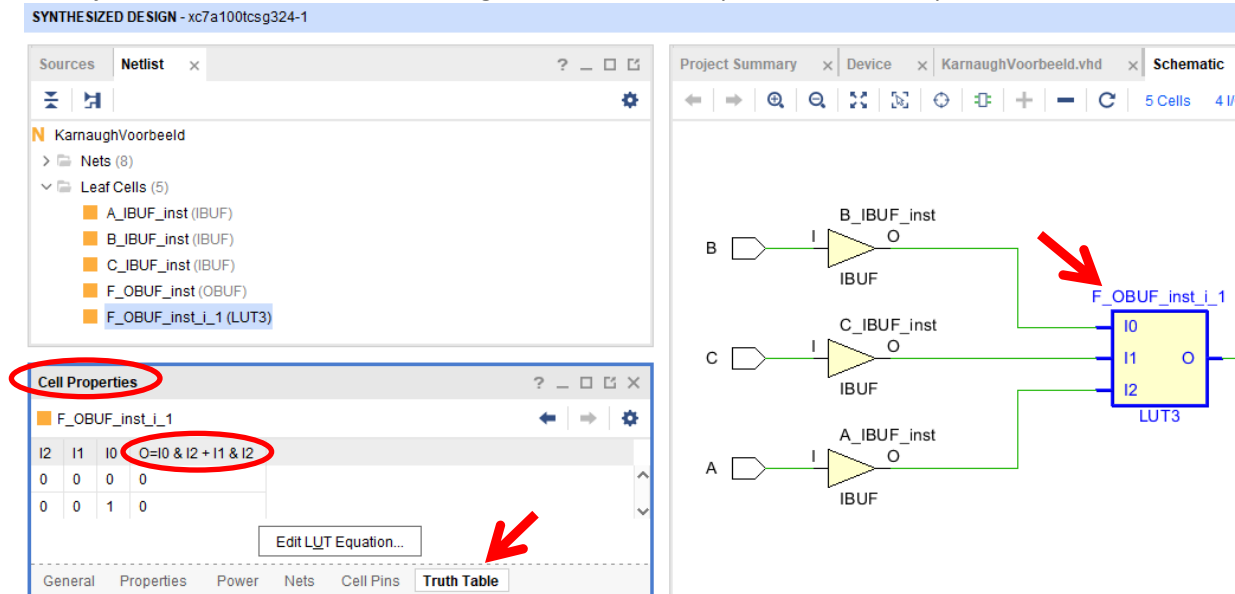
Dit levert een schema op van de volgende aard:



Hier zie je twee dingen op:

Ten eerste: in- en output buffers op iedere in- of uitgang. Dit zijn logische poorten, waarvan de uitgang gewoon de ingang volgt (0 geeft 0 en 1 geeft 1). Het nut hiervan ligt in het verzorgen van de juiste stromen en spanningen wanneer je ontwerp in aanraking komt met de "buitenwereld". Als de uitgang F bijvoorbeeld aan 20 andere logische poorten komt te hangen, dan zal deze OBUF genoeg stroom moeten kunnen leveren voor al die 20 poorten. Vivado doet dit altijd automatisch voor je.

Ten tweede zie je hier een “lookup table” (LUT) staan, die alle logische poorten vervangt en samenvat in één grote waarheidstabel. Om de inhoud van deze waarheidstabel te kunnen bekijken, moet je deze LUT selecteren en vervolgens in het “Cell Properties” venster op “Truth Table” klikken.



In bovenstaand voorbeeld zie je dat de uitgang  $O = I0 \cdot I2 + I1 \cdot I2$  wat overeenkomt met  $F = BA + CA$  als je ziet dat I0 verbonden is met B, I1 met C, I2 met A en O met F. Zoals verwacht, werd de functie vereenvoudigd tot zijn meest eenvoudige vorm.

Waarom de logische poorten vervangen worden door lookup tables, zal je later nog zien wanneer we dieper zullen ingaan op de inwendige opbouw van FPGA's.

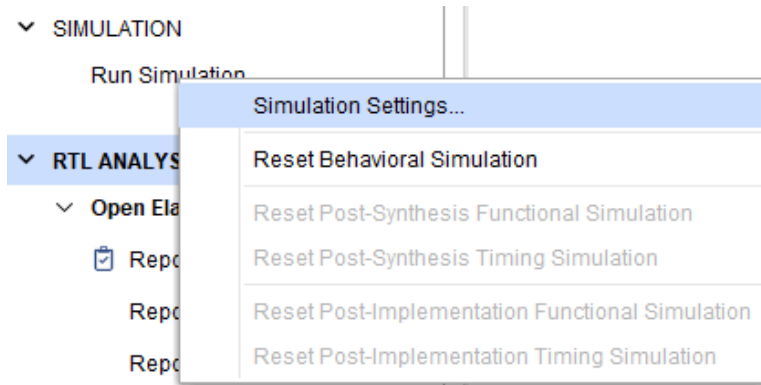
Nu kan je controleren of dit effectief overeenkomt met je op papier gevonden vereenvoudiging van de gegeven functie (in plaats van het hier gebruikte voorbeeld). Als dit niet het geval is, dan heb je een fout gemaakt bij het vereenvoudigen OF bij het ingeven van je VHDL code. Verbeter dit tot het gegenereerde schema overeenkomt met je zelf berekende oplossing. **Laat je oplossing controleren door de docent!**

### Beschrijving: simulatie

Nu heb je een oplossing die je manueel nagekeken hebt met Karnaugh. Dit wil of kan je in toekomstige opgaven uiteraard niet altijd doen, dus voorziet Vivado ook een simulator waar je je ontwerp mee kan simuleren en nakijken op functionele fouten.

Simulatie is HET BELANGRIJKSTE dat je zal leren in dit vak! Je hebt dit in alle komende jaren nodig, vermits dit bij uitstek de meest gebruikte manier is om VHDL te debuggen. Er kan zelfs geopperd worden dat debugging het allerbelangrijkste is van je opleiding tot ingenieur, dus klik hieronder niet te snel door en probeer te onthouden wat je hiervan opsteekt!

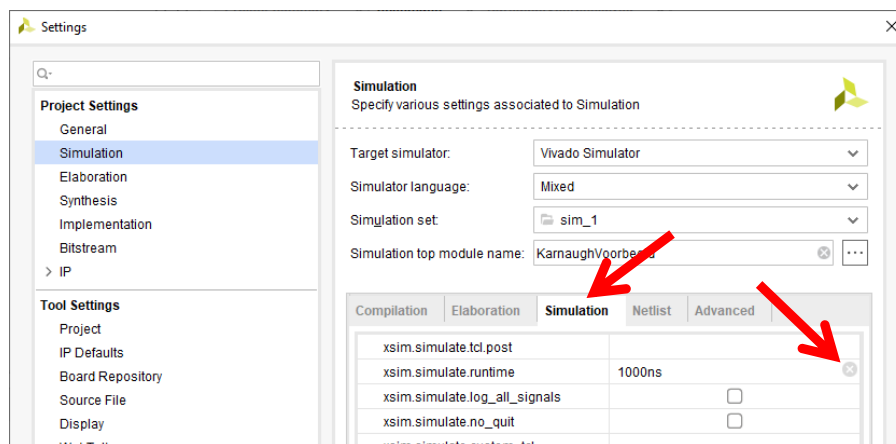
Simuleren in Vivado kan je doen door in de “Flow Navigator” op “Run Simulation” te klikken. Alvorens je dit doet, kan je best even met de rechtermuisknop klikken op “Run Simulation” en vervolgens “Simulation Settings...” kiezen.



Dit opent een nieuw venster waarin je een vervelende standaardinstelling van de simulator wil wijzigen.

Normaal gezien begint Vivado namelijk automatisch 1000 ns (1  $\mu$ s) te simuleren, wat je niet altijd wil. Dit wordt later wel duidelijk. In de simulatie instellingen kan je dit aanpassen, zodat je simulatie niet onmiddellijk start van zodra je ze opent en zodat je eerst tijd krijgt om signalen toe te voegen aan je simulatievenster (zie verder).

Open hiervoor het tabblad "Simulation" in onderstaand scherm en verwijder de "1000 ns" naast de "xsim.simulate.runtime" parameter:



Als je dan op OK klikt, sluit het venster weer en kan je ditmaal met de linkermuisknop op "Run Simulation" en vervolgens op "Run Behavioral Simulation" klikken<sup>2</sup>.

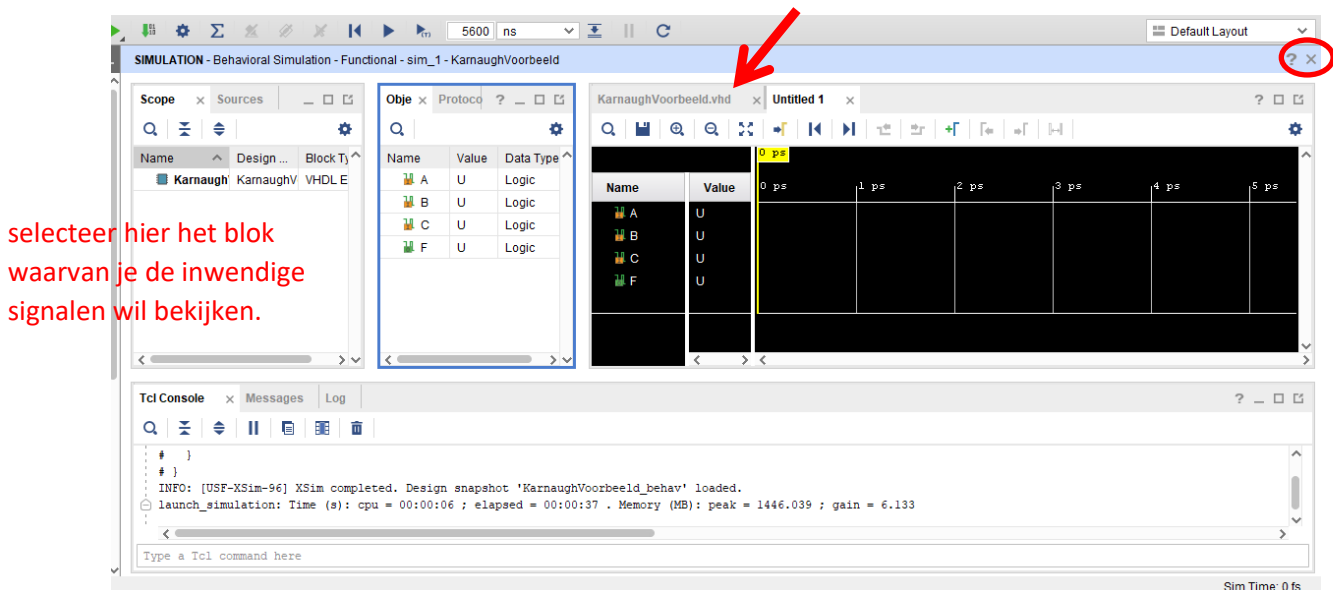
Nu maakt Vivado een executable (.exe) bestand aan, wat dus eigenlijk een apart programma is dat je ontwerp zal simuleren. Dit kan soms problemen opleveren met sommige virus- en malwarescanners. Als dit zo is, dan moet je een **uitzondering toevoegen in je virusscanner** voor deze executable. Om dit niet iedere keer opnieuw te moeten doen, kan je best een map aanmaken waarin je al je VHDL projecten zal bewaren (bv. "C:\VhdlDesigns\...") en deze map in zijn geheel als uitzondering toevoegen aan je virusscanner. Dit werd reeds besproken in het begin van dit document.

Als alles goed gaat, wordt vervolgens de "Project Manager" vervangen door het "Simulation" venster. Als je later de simulatie terug wil sluiten, dan kan je rechtsboven op het kruisje klikken van het "Simulation" venster (in rood omcirkeld op de volgende figuur), NIET van Vivado zelf (want dan

<sup>2</sup> Er is nog een tweede instelling die je best altijd wijzigt, maar dat wordt later besproken in de videoles/demo [Simuleren met Vivado](#) (kanaal Xilinx Vivado). Voor deze inleidende opgave heeft dat nog geen nut.

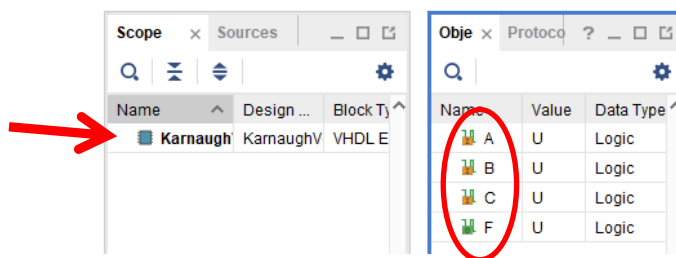
sluit het volledige programma). Merk ook op dat je nog steeds je VHDL code kan bekijken in het andere tabblad naast “Untitled 1” (rode pijl op de volgende figuur).

Hieronder gaan we de werking van de simulator uitleggen, wat je voor alle andere opgaven zeker nog herhaaldelijk zal nodig hebben.



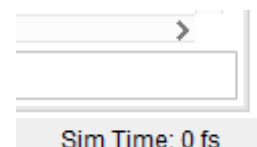
Om te beginnen, kan je in het “Scope” tabblad selecteren waar je je ergens in je ontwerp bevindt. Op het ogenblik heb je slechts één niveau van hiërarchie, maar later zal je hier meerdere niveaus krijgen en dan is het belangrijk dat je hier het juiste niveau selecteert. **Onthoud dit! Zeer belangrijk!!!**

Voor deze opgave kan je het huidige ontwerp “KarnaughVoorbeeld” dus niet verder uitvouwen en kan je enkel het zogenaamde “toplevel” (het hoogste niveau in de hiërarchie van je ontwerp) selecteren. In het “Objects” venster rechts daarvan krijg je dan alle signalen te zien die aan dat niveau gerelateerd zijn. Voor de huidige opgave zijn er nog geen interne signalen, enkel rechtstreeks de in- en uitgangen van het ontwerp: A, B, C en F.



Standaard voegt Vivado alle toplevel in- en uitgangen toe aan het “Waveform” venster rechts, wat in dit geval “Untitled 1” genaamd is. Later zal je hier dus inwendige signalen willen aan toevoegen die zich helemaal binnenin je ontwerp bevinden. Dit zal zeer handig blijken om te kunnen volgen wat er intern in je ontwerp gebeurt.

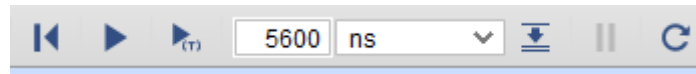
Helemaal rechts onderaan het “Simulation” venster zie je dat de huidige simulatietijd nog op 0 fs staat, wat wil zeggen dat er nog niets gesimuleerd is (als je hierboven tenminste de “xsim.simulate.runtime” parameter verwijderd hebt).



Met de knoppen vlak boven het “Simulation” venster kan je de voortgang van de simulatie besturen, maar vermits je nog geen waarden aan je ingangen hebt aangelegd, wacht je hier best nog even mee.

Nadat hieronder de werking van deze knoppen wordt uitgelegd, moet je dus best eerst nog waarden op A, B en C forceren. Dit wordt verderop nog uitgelegd.

De knoppen die de simulatie besturen, zien er als volgt uit:



De meest linkse knop zet je simulatie terug naar tijdstip 0, wanneer je terug opnieuw wil beginnen simuleren. Merk op dat eventuele wijzigingen in je VHDL code hier niet in zullen opgenomen worden.

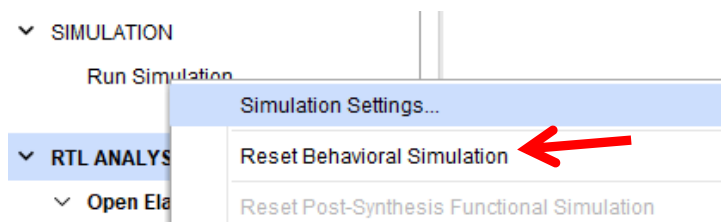
De tweede knop zal de simulatie starten en oneindig laten doorlopen tot je op de voorlaatste “pause” knop drukt.

Meestal wil je de derde knop gebruiken, want die simuleert exact even lang als je aangeeft in het tekstvak rechts daarvan. In dit voorbeeld zou de simulatie dus 5600 ns simuleren en vervolgens automatisch pauzeren.

Met de vierde knop kan je eventueel door je code “steppen”, wat vooral handig is voor het debuggen van eventueel moeilijk te vinden fouten in je code.

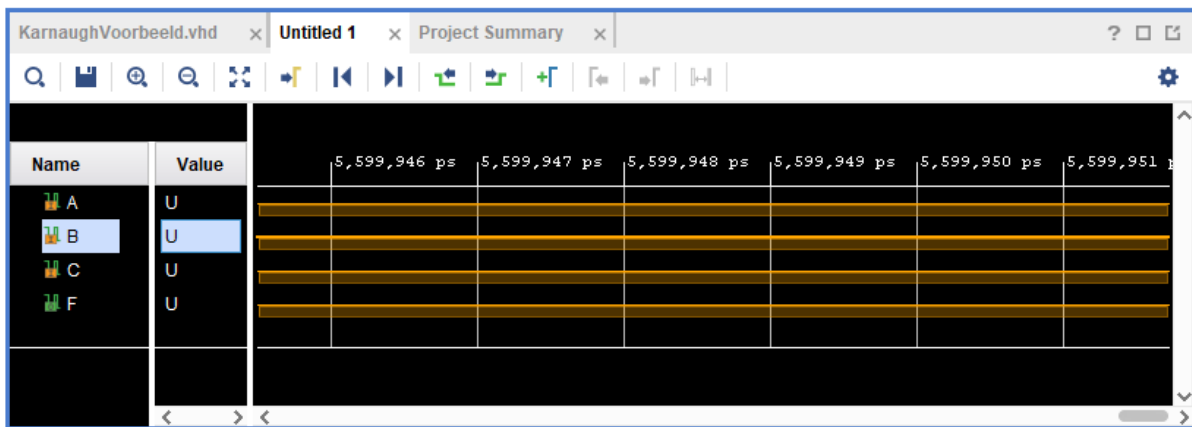
De meest rechtse knop zal heel je simulatie opnieuw opstarten en zal dus ook rekening houden met eventuele wijzigingen in je VHDL code, in tegenstelling tot de meest linkse knop. Dit is hoe deze knop **zou moeten** werken, dus: als je iets wijzigt aan je VHDL terwijl je simulatie open staat, zou Vivado deze wijzigingen in rekening moeten brengen als je op deze knop klikt.

Vivado werkt hier echter niet altijd even vlekkeloos. Als je iets wijzigt aan je VHDL code terwijl de simulatie open staat, dan kan het zijn dat je eerst de volledige simulatie moet sluiten (blauwe kruisje rechtsboven) en terug openen. Soms herkent Vivado dan echter nog steeds de nieuwe VHDL code niet en moet je rechtermuisknop op “Simulation” klikken en “Reset Behavioral Simulation” kiezen. Als het dan nog steeds niet lukt, dan moet je Vivado afsluiten en de volledige <projectnaam.sim> folder verwijderen.



Dit wordt allemaal uitvoerig gedemonstreerd in de videoles [Simuleren met Vivado](#) (hoofdstuk “VHDL wijzigingen tijdens simulatie”).

Als je nu de simulatietijd zou laten lopen, dan krijg je het volgende te zien:

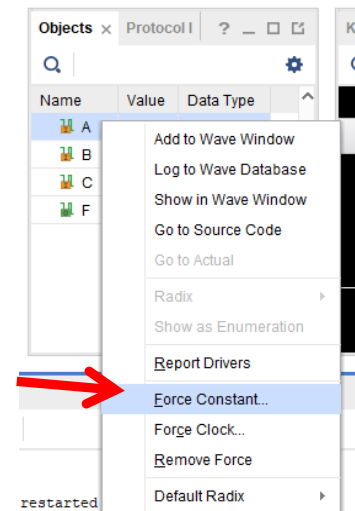


In de loop van de simulatietijd verandert er niets en blijven alle signalen op 'U' staan, wat "uninitialized" betekent in VHDL. Vermits er nog geen enkele waarde aangelegd werd op de ingangen van het ontwerp, is dit natuurlijk niet verwonderlijk. Vivado kan namelijk niet raden wat jij (of later een gebruiker van je schakeling) exact op de ingangen van je ontwerp wil aanleggen.

Er zijn twee manieren om dit te doen, namelijk manueel en via een zogenaamde "testbench". Voor deze opgave zal je dit manueel doen, maar voor de volgende opgaven zal je dit veel liever met een testbench doen, omdat dit het werk aanzienlijk verlicht.

Nu kan je voor ieder signaal in het "Objects" venster een waarde forceren door met de rechtermuisknop op het betreffende signaal te klikken en vervolgens "Force Constant..." te kiezen.

Uiteraard doe je dit enkel voor de ingangen A, B en C want de uitgang F wil je laten berekenen door de simulator aan de hand van jouw opgegeven ingangswaarden.



Bij "Force Value" kan je hier dan de gewenste waarde (0 of 1) invullen voor zowel A, B als C. De andere velden kan je ongemoeid laten, vermits die voor geavanceerdere doeleinden gebruikt worden.

Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal name: /KarnaughVoorbeeld/A

Value radix: Hexadecimal

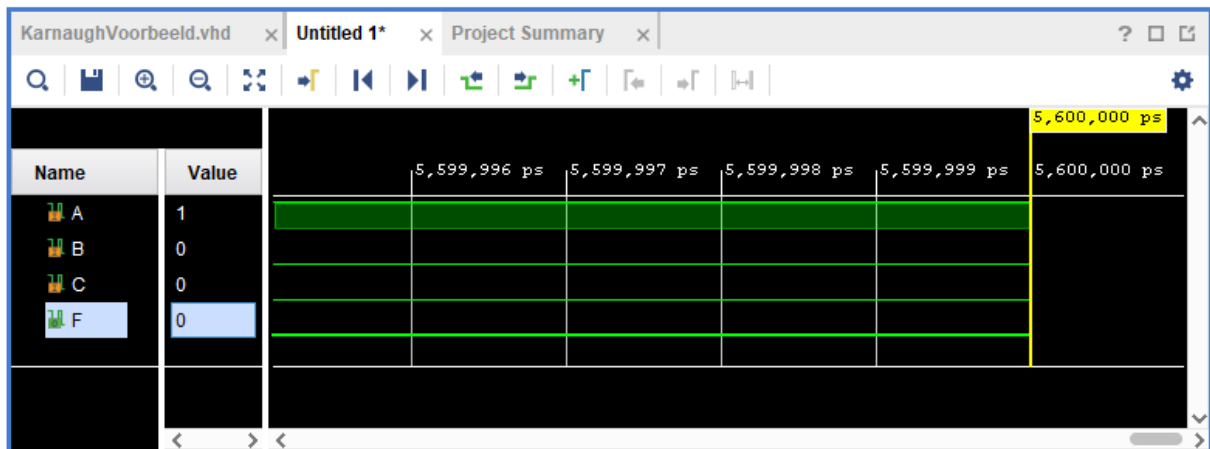
Force value: 1

Starting after time offset: 0ns

Cancel after time offset:

OK Cancel

In dit voorbeeld wordt de ingang A op 1 geforceerd. De “Value radix” is het talstelsel waarmee je de waarde wil opgeven. Als je later bijvoorbeeld decimale waarden wil forceren in plaats van binair of hexadecimaal, dan kan je dat hier aangeven. Forceer op dezelfde manier een waarde voor ingangen B en C en laat de simulator opnieuw een tijdje lopen.



In dit voorbeeld werd A op 1 geforceerd en B en C beiden op 0. De simulator heeft vervolgens berekend dat de uitgang F 0 zal worden bij deze ingangscombinatie. Dit kan je zien in bovenstaande “waveform”.

Nu is het de bedoeling dat je alle mogelijke ingangswaarden simuleert, dus je maakt best eerst een waarheidstabel op papier en geeft vervolgens de eerste ingangscombinatie in (bv. A=0, B=0, C=0) en simuleert vervolgens een bepaalde tijd (bv. 100 ns). Daarna leg je de volgende combinatie aan (bv. A=0, B=0, C=1) en simuleert weer 100 ns, enz. In totaal zal je dus 8 keer de ingangswaarden moeten forceren en telkens een bepaalde tijd simuleren.

Om het geheel een beetje beter zichtbaar te maken in het waveform venster, kan je de knoppen boven dit venster gebruiken. **Experimenteer hier zeker mee, want je zal ze nog nodig hebben!!**

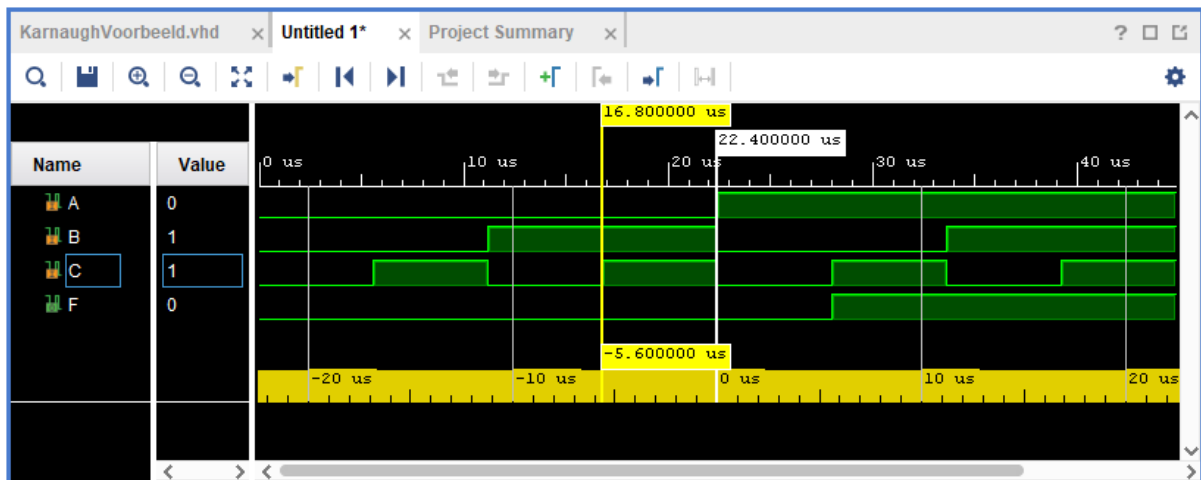


Van links naar rechts:

- 1) Zoeken naar een bepaalde waarde op een bepaald signaal.
- 2) De huidige configuratie bewaren (de signalen in je waveform, de volgorde, de radix,...)
- 3) Inzoomen (kan ook met Ctrl+muiswiel).
- 4) Uitzoomen (kan ook met Ctrl+muiswiel).
- 5) Uitzoomen over de volledige simulatietijd.
- 6) Ga naar de huidige cursor.
- 7) Ga naar tijdstip 0.
- 8) Ga naar het laatst gesimuleerd tijdstip.
- 9) Ga naar de vorige waardeverandering van het huidig geselecteerd signaal.
- 10) Ga naar de volgende waardeverandering van het huidig geselecteerd signaal.
- 11) Voeg een zogenaamde “marker” toe op het huidige tijdstip. Dit is handig om periodes te meten, want onder de cursor zal steeds het verschil in tijd staan tussen de marker en de cursor zelf. In de volgende figuur zie je hiervan een voorbeeld.
- 12) Ga naar de vorige marker.
- 13) Ga naar de volgende marker.
- 14) Verwissel de cursors. Dit is een restant van oudere versies en wordt niet meer gebruikt.



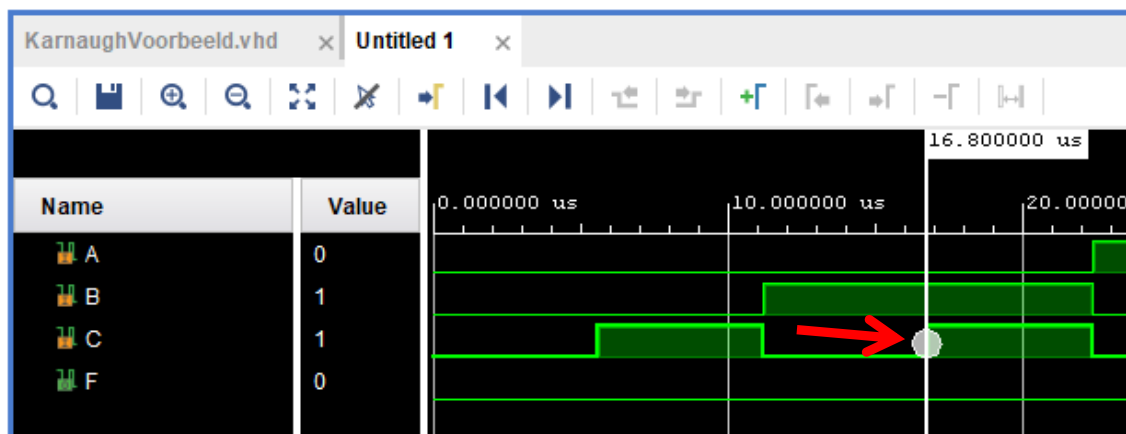
In onderstaande figuur is knop 5 (volledig uitzoomen) gebruikt en werden ook enkele markers toegevoegd om te zien wat het tijdsverloop was tussen twee tijdstippen in de simulatie:



De cursor staat op tijdstip 16,8  $\mu$ s (de transitie van 0 naar 1 op ingang C) en er werd een marker aangemaakt op tijdstip 22,4  $\mu$ s (de volgende transitie van ingang C). Het verschil tussen beide tijdstippen zie je dan onderaan (-5,6  $\mu$ s). Experimenteer hier even mee en **onthoud dit!**

Nog een zeer belangrijke opmerking om zeker te onthouden: De waarden in de kolom "Value" tonen de waarden van de signalen op de plaats van de cursor, **niet** die op het einde van de simulatie! In het bovenstaande voorbeeld kan je zien dat de waarden voor A, B en C 0, 1 en 1 zijn op het tijdstip van de gele cursor, niet op het einde van de simulatie (want daar zijn ze allemaal 1). Dit is blijkbaar zeer verwarrend voor studenten, dus hou hier zeker rekening mee.

Tot slot nog een handige tip: Merk op dat je je cursor kan "slepen" over een overgang van 0 naar 1 of van 1 naar 0 en dat de cursor dan zal "koppelen" aan deze overgang. Je zal zien dat er dan een klein cirkeltje getoond wordt, zoals op onderstaande figuur te zien is. Op die manier staat de cursor exact op die overgang.



Er zit zeer veel informatie in deze inleidende opgave, dus waarschijnlijk zal je een deel ervan vergeten tegen de komende opgaven. Hou ze daarom zeker bij de hand, want je zal ze zeker en vast nodig hebben. Overloop ze desnoods nog eens vlak vóór de volgende zitting(en), zodat je een beetje vertrouwd geraakt met de werking van Vivado. Het zal je nog veel tijd besparen!

De simulatie die je hier nu gemaakt hebt, komt dus overeen met de volgende waarheidstabel:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

In de simulatie werden alle acht ingangscombinaties achtereenvolgens aangelegd en werd vervolgens door Vivado uitgerekend wat de bijhorende uitgangswaarde is.

Deze waarheidstabel was dan ook perfect te verwachten, vermits in de VHDL code de functie  $F = A \cdot (B + C)$  beschreven werd. Hierdoor wordt F enkel gelijk aan 1 als A gelijk is aan 1 én B of C gelijk zijn aan 1 (de onderste 3 gevallen dus).

Op dezelfde manier heb je nu jouw functie beschreven in VHDL en de waarheidstabel ervan gesimuleerd met Vivado.

Je merkt meteen dat deze methode haalbaar is voor een zeer kleine functie, maar vanaf je schakeling ingewikkelder wordt, is het onbegonnen werk om alle mogelijke ingangscombinaties één voor één aan te leggen met "Force Constant...". Daarom wordt er vanaf volgende opgaven gewerkt met een testbench, een simulatieomgeving rond je schakeling die automatisch deze waarden zal aanleggen. Hierover meer in de volgende opgave.

### Videolessen en demo's omtrent Vivado

Voor alle volgende opgaven is het belangrijk dat je goed leert werken met Vivado. Hoe beter je kan werken met de softwaretool, hoe sneller je de opgaven tot een goed einde zal kunnen brengen. Daarom is er een reeks videolessen en demo's gemaakt over Vivado zelf. Je zal hier geen test over krijgen, maar het is in je eigen belang dat je AL deze video's goed bekijkt én herbekijkt wanneer je tegen problemen aanbotst met de tool zelf. Ze staan gegroepeerd onder het [kanaal Xilinx Vivado](#) op de Mediasite van UA.

Vóór opgave 1 raad ik je aan om al zeker de volgende video's/demo's te bekijken:

- [Vivado projecten](#)
- [Simuleren met Vivado](#)
- [Foutmeldingen in Vivado](#)
- [Veel voorkomende warnings in Vivado](#)
- [Vivado bugs](#)

Alle video's zijn ingedeeld in hoofdstukken, dus je zou achteraf vrij snel moeten kunnen doorklikken naar het topic waarover je de uitleg vergeten bent.

Het is een eenmalige tijdsinvestering ( $\pm 1,5u$ ) om deze demo's te bekijken, maar ze zullen je zeer veel tijdswinst opleveren, zeker als je beseft dat je Vivado de komende jaren nog veel zal nodig hebben!