

## Opgave 1

### Overzicht opdracht

Ontwerp een schakeling die een slangbeweging nabootst op de 7-segment displays als je ervan uit gaat dat op de ingang van je schakeling binair achtereenvolgens de waarden van 0 tot en met 7 komen te staan. De displays zijn actief laag (een 0 doet een segment oplichten). Beschrijf de schakeling in VHDL en simuleer alle mogelijke ingangswaarden met behulp van een testbench.

#### 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.

- 1.9 en 1.12: std\_logic(\_vector), boolean, enumeration, array, integer, time, real,... [[kanaal Basis VHDL: Datatypes](#)]
- 1.11: package, (un)signed, operators (+-\*>...), conversie [[kanaal Basis VHDL: numeric\\_std](#)]
- 2 en 6.1: concurrent/sequential statements, proces, voorwaardelijk/selectief, multiplexers, ongewenste latches, multiple drivers [[kanaal Basis VHDL: Combinatorische logica](#)]
- 1.10: hiërarchie, component, port map, ongebruikte in/uitgangen [[kanaal Basis VHDL: Hiërarchie](#)]
- 3.1 en 3.2: testbench, proces zonder sensitivity list, stimuli [[kanaal Basis VHDL: Testbench](#)]

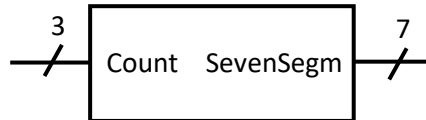
Je hoeft niet zowel de cursus als de videolessen te bekijken, want daar wordt in principe hetzelfde besproken. Het is wel belangrijk dat je minstens een van de twee bekeken en begrepen hebt. Als het dan nog niet helemaal duidelijk is, dan mag je me ook altijd vragen stellen:

- 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)

## Beschrijving: het ontwerp

Als je begint aan een nieuw VHDL-ontwerp, dan start je steeds met het beschrijven van de in- en uitgangen van je ontwerp. Dit doe je aan de hand van een entity.

Voor deze opgave zal je deze component nodig hebben:



Uitzonderlijk is de entity hiervoor gegeven en **je bent verplicht om deze ongewijzigd te gebruiken**. Kopieer dus onderstaande code in je VHDL bestand:

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity CountSnake is
  port ( Count: in unsigned(2 downto 0);
        SevenSegm: out std_logic_vector(6 downto 0));
end CountSnake;
  
```

Merk op dat zowel de ingang als de uitgang zogenaamde bussen zijn, die bestaan uit een verzameling bits. De bus-ingang Count bevat 3 bits en de uitgang SevenSegm bevat er 7. Zoals besproken in de cursus en [videoles over numeric\\_std](#) kan je dit in VHDL beschrijven aan de hand van een std\_logic\_vector, waarvan de uitgang SevenSegm een voorbeeld is (een 7-bit brede bus).

Wanneer de bus een binaire waarde bevat (zoals de Count-ingang), dan werk je best met unsigned als het om positief binaire getallen gaat (zoals hier het geval is) of signed als het getal zowel positief als negatief kan zijn (two's complement).

Als de bus slechts een verzameling bits bevat (zoals bijvoorbeeld 7-segment code), dan gebruik je best een std\_logic\_vector. Merk op dat er geen enkel functioneel verschil is tussen deze drie types. VHDL gaat gewoon op andere manieren om met deze types, bijvoorbeeld bij aritmetische bewerkingen (+, -, >, <, =,...).

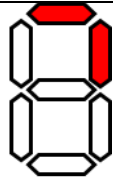
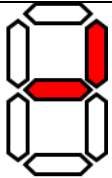
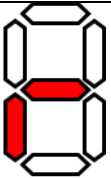
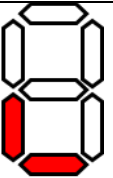
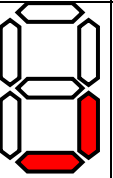
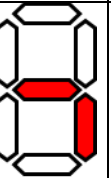
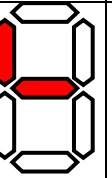

Zoals je kan zien in bovenstaande code van de entity, werd dus voor de Count ingang een 3-bit brede bus gekozen van het type unsigned omdat deze bits als positief binaire getallen beschouwd zullen worden. De SevenSegm uitgang zal de bijhorende 7-segment code bevatten en dus werd hier voor een std\_logic\_vector geopteerd, vermits dit geen binaire getallen betreft.

In elektronica worden bussen meestal zo genummerd, dat ze beginnen bij de "most significant bit" (MSB) en eindigen bij de "least significant bit" (LSB). In het geval van de Count-ingang bijvoorbeeld bit 2 tot en met bit 0, in totaal dus 3 bits.

Als dit gebeurd is, dan kan je beginnen aan de architecture, de inhoud van de component. Je moet hierbij in het achterhoofd houden dat je een schakeling aan het ontwerpen bent, die achteraf door eender welke gebruiker moet kunnen aangestuurd worden. Je kan dus niet op voorhand voorspellen welke ingangswaarden de gebruiker aan de ingang "Count" zal leggen. Je moet dus met alle mogelijkheden rekening houden en ervoor zorgen dat er **steeds** een waarde op de 7 uitgangen komt.

Dit is trouwens een algemeen geldende regel voor digitale schakelingen: **alle** uitgangen moeten **altijd** een waarde toegekend krijgen.

In ons geval gaan we ervan uit dat de gebruiker van onze schakeling achtereenvolgens de waarden 0 tot en met 7 op onze ingang Count zal aanleggen. Dit zou dan moeten resulteren in een soort slangbeweging op de 7-segment displays volgens onderstaande tabel:

Count	0	1	2	3	4	5	6	7
7-segment								

Doet de gebruiker dat niet (en legt hij gewoon willekeurig 3-bit waarden aan), dan zal er natuurlijk geen mooie slangbeweging verschijnen op de 7-segment display, maar dat is de opgave niet. We gaan veronderstellen dat de gebruiker wel degelijk achtereenvolgens 0 tot en met 7 zal aanleggen op de Count ingang.

De bedoeling is dus dat je voor deze functionaliteit een schakeling beschrijft in VHDL. Bekijk zeker de voorbeeldoefening(en) op Blackboard als je geen idee hebt hoe je hieraan moet beginnen.

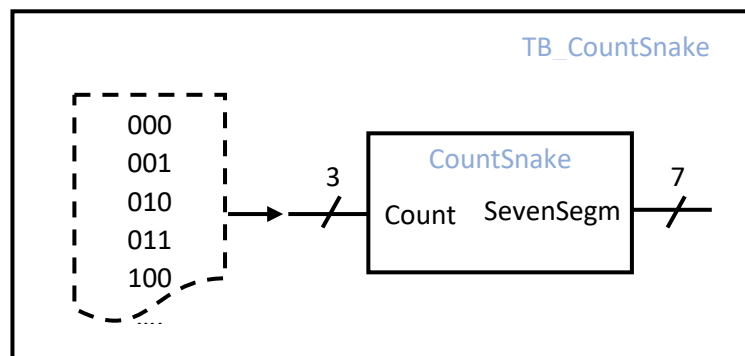
Er werd gezegd dat de displays actief laag zijn, dus de leds van het 7-segment display lichten op als je er een 0 naartoe stuurt. Bij een 1 branden ze niet. Voor deze opgave moet je enkel de 7-segmentcode genereren. In volgende opgaven gaan we meer in detail bekijken hoe we de displays effectief moeten aansturen (met kathodes en anodes, maar dat is dus voor volgende opgave).

Tip: Bekijk [voorbeeldopgave 1 op de Mediasite](#) waar een zeer gelijkaardig ontwerp gedemonstreerd wordt.

### Beschrijving: de testbench

Bij de inleidende opgave heb je alle mogelijkheden getest door ze één voor één manueel aan te leggen op de ingangen. Hoe groter je ontwerp wordt, hoe omslachtiger dit wordt. Daarom wordt er quasi altijd gewerkt met een zogenaamde testbench. Bekijk zeker de cursus en/of [videoles over de VHDL testbench](#).

Een testbench is een omgeving waarin je simuleert wat een gebruiker zoal aan je systeem zou kunnen leggen. In het geval van deze opgave ga je dus alle mogelijke waarden willen simuleren die een gebruiker van jouw ontwerp op de Count ingangen zou kunnen aanleggen. In VHDL zit zo'n testbench altijd "rond" je eigenlijke ontwerp. Het is altijd de toplevel van de hiërarchie en je ontwerp is altijd een lower-level "onder" de testbench. Voor deze opgave ziet dat er dus als volgt uit:



De testbench wordt ook beschreven in VHDL, maar heeft enkele afwijkingen:

- De entity van de testbench is altijd leeg. De simulatieomgeving heeft namelijk geen in- of uitgangen en genereert zijn stimuli zelf.
- De testbench wordt later nooit in een effectieve schakeling omgezet en dient enkel om te simuleren. Dit heeft tot gevolg dat er hier dingen kunnen gebruikt worden die in een effectieve schakeling niet gebruikt kunnen worden, zoals floating point getallen, vierkantswortels, tijdsvertragingen, enz.

Vermits de testbench altijd de toplevel is en je ontwerp zich altijd op een lower-level bevindt, moet je gebruik maken van hiërarchie in je VHDL-code. Dit doe je door zogenaamde components en instances te gebruiken.

Voor deze opgave krijg je de testbench cadeau. De code hiervan kan je hieronder terugvinden. Het is zeer belangrijk dat je de gegeven code helemaal begrijpt, want je zal dit voor de volgende opgaven zelf moeten maken (én je kan er vragen over krijgen tijdens je verdediging). Merk op dat deze testbench enkel stimuli aanlegt aan de ingang Count. Ze test niet automatisch of de uitgangswaarden op SevenSegm correct zijn. Dit zal je dus nog manueel in simulatie moeten nakijken op je waveforms. Bekijk zeker de videolessen over [debugging in VHDL](#) en [simuleren met Vivado](#)!

Er bestaan ook zogenaamde zelftestende testbenches, die zowel stimuli aanleggen aan de ingang(en) alsook de waarden op de uitgangen verifiëren, maar dat is leerstof voor een later vak (3-Digitale elektronica 2).

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.numeric_std.all;
4
5 entity TB_CountSnake is
6 end TB_CountSnake;
7
8 architecture Behavioral of TB_CountSnake is
9     signal Counter: unsigned(2 downto 0) := (others => '0');
10    signal SevenSegm: std_logic_vector(6 downto 0);
11
12    component CountSnake is
13        port ( Count: in unsigned(2 downto 0);
14              SevenSegm: out std_logic_vector(6 downto 0));
15    end component;
16
17 begin
18
19    Testing: CountSnake port map(
20        Count => Counter,
21        SevenSegm => SevenSegm);
22
23    -- zoek in numeric_std alle hieronder gebruikte functies op (*)
24    -- natural = positieve integer
25    p_Stimuli: process -- geen sensitivity list! => "wait" is nodig!
26    begin
27        wait for 100 ns;
28        if Counter < 7 then -- (*) unsigned < natural
29            Counter <= Counter + 1; -- (*) unsigned + natural
30        else
31            Counter <= (others => '0');
32        end if;
33    end process p_Stimuli;
34
35 end Behavioral;
```

Om deze code te kunnen ontleden, zijn er lijnummers aan toegevoegd. Uiteraard mogen deze niet in je VHDL code getypt worden.

Op lijn 3 zie je dat de package `numeric_std` gebruikt wordt. Dit is hier nodig omdat er op lijnen 28 en 29 gebruik wordt gemaakt van de functies `<` en `+` die voor het type `unsigned` gedefinieerd staan in deze package. **Zoek deze op in de package!**<sup>1</sup>

Op lijnen 5 en 6 zie je dat, zoals reeds besproken, de entity van de testbench leeg is. Er zijn geen in- of uitgangen van de simulatieomgeving.

De definitie van het signaal `Counter` op lijn 9 bevat ook een initialisatie `“(others => ‘0’)”`. Dit geeft alle bits van dit signaal de waarde 0 aan het begin van de simulatie. Dit is belangrijk, want anders zal de instructie op lijn 29 niet werken, vermits `Counter` altijd ‘U’ (uninitialized) zal blijven (‘U’ + 1 is nog altijd ‘U’). Over deze “others”-constructie kan je meer informatie vinden in de voorbeeldoefeningen VHDL en in de cursus VHDL (hoofdstuk 1.9 over *array aggregates*).

Van lijnen 12 tot 15 staat de component van `CountSnake` gedefinieerd. Dit is verplicht: in de architecture, maar vóór het “begin” statement moeten alle componenten gedefinieerd zijn die je zal

---

<sup>1</sup> De `numeric_std` package kan je terugvinden op Internet (bv [hier](#)), maar ze staat ook op je harde schijf onder **C:\Xilinx\Vivado\202X.X\data\vhdl\src\ieee\distributable** (het vet gedrukte deel kan verschillen naargelang waar je Vivado hebt geïnstalleerd).

instantiëren (aanroepen, gebruiken) in de testbench. De definitie van de in- en uitgangen moet ook exact hetzelfde zijn zoals beschreven in de entity van het ontwerp (in het andere VHDL bestand).

Op lijnen 19 tot 21 staat dan de instantiatie van het eigenlijke ontwerp dat getest moet worden. Dit gebeurt aan de hand van een zogenaamde port map, waarbij alle in- en uitgangen van het ontwerp gelinkt worden aan de signalen van de testbench zelf. Merk op dat deze signalen niet dezelfde naam hoeven te hebben als de in- en uitgangen van het ontwerp, maar het mag wel. Zo zie je bijvoorbeeld dat de ingang Count gelinkt wordt aan het signaal Counter, maar de uitgang SevenSegm hangt gewoon aan een signaal met dezelfde naam. Het is belangrijk om te weten dat het interne signaal Count van het CountSnake ontwerp niet beschikbaar zal zijn in de testbench, enkel de signalen die in de port map hieraan gelinkt zijn. Het signaal Counter bestaat als het ware op het toplevel niveau, en het signaal Count bestaat enkel op het niveau van het CountSnake ontwerp. Omdat SevenSegm toevallig dezelfde naam heeft in de testbench als binnen het CountSnake ontwerp, valt dit niet op voor deze uitgang maar het is in principe hetzelfde: er bestaat een signaal SevenSegm op het niveau van de testbench en een signaal SevenSegm op niveau van het ontwerp.

Op lijn 19 zie je ook dat je een naam kan (en zelfs moet) geven aan de instantiatie van het CountSnake ontwerp. Gelijkaardig aan het verhaal van de signalen op de verschillende niveaus, krijgt ook CountSnake een naam op het toplevel niveau. Deze naam mag echter niet exact hetzelfde zijn en hiervoor werd bij wijze van voorbeeld de naam Testing gekozen.

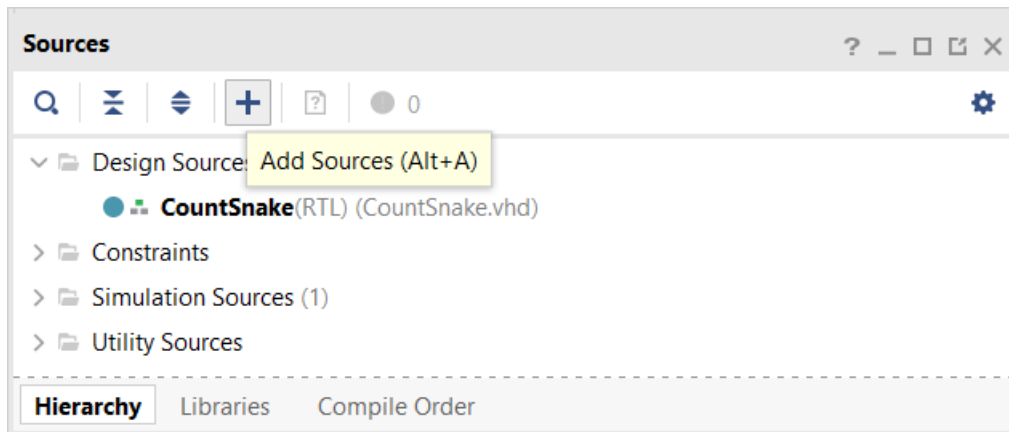
Lijnen 25 tot 33 beschrijven een process zonder sensitivity list. Dit wil zeggen dat het process op ieder simulatietijdstip wordt aangeroepen en als alle instructies afgehandeld zijn, gewoon opnieuw begint. Daarom is het belangrijk dat er een wait instructie gebruikt wordt (lijn 27) om te vermijden dat de simulatie zal “vast” zitten in een oneindige lus op tijdstip 0. Nu wordt het process aangeroepen, het wordt gepauzeerd voor 100 ns en vervolgens zal Counter met 1 verhogen, alvorens opnieuw van bovenaan te beginnen.

**Belangrijk!! Deze code bevat een aantal zeer belangrijke eigenschappen van VHDL (port maps, hiërarchie, numeric\_std,...). Zorg ervoor dat je bovenstaande code helemaal begrijpt: iedere lijn code! Als dit niet het geval is, dan moet je dit vragen aan mij. Je kan dit vak niet succesvol afleggen als je deze dingen niet begrijpt! Je zal hierover vragen krijgen op je verdediging.**

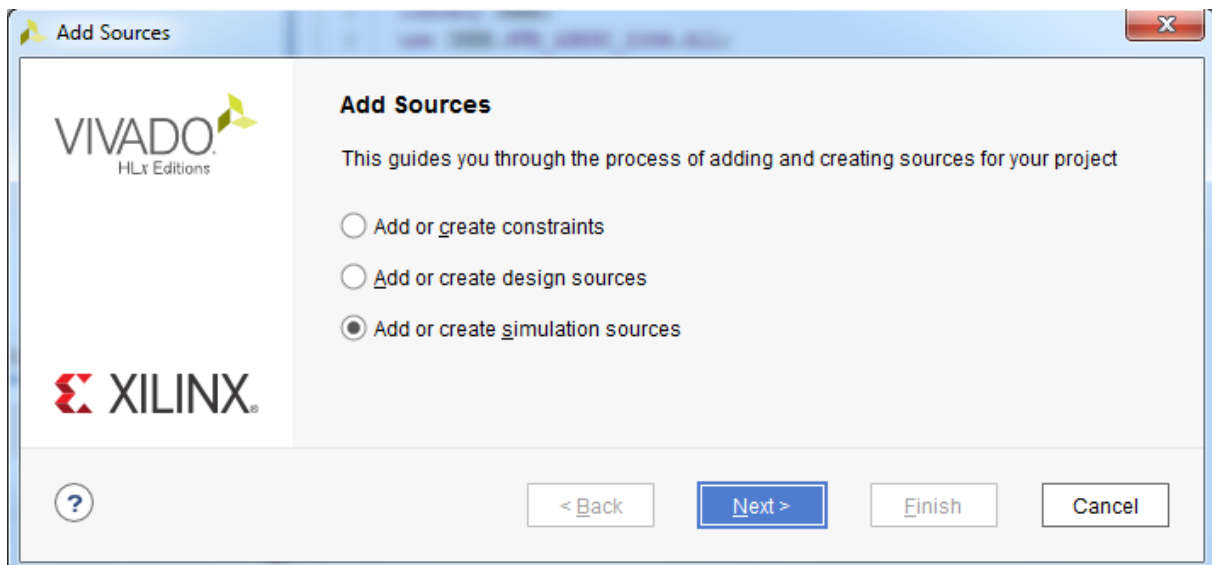
Gebruik nu deze testbench (ongewijzigd) om een simulatie op te starten die alle mogelijke 3-bit waarden aanlegt aan de ingang Count. Kijk vervolgens na op je waveforms of de waarden op de SevenSegm uitgangen overeenkomen met wat je verwacht. Dit vergt nog steeds wat manueel werk (per ingangscombinatie gaan kijken op de waveform of de uitgangen wel kloppen), wat ook geautomatiseerd kan worden (met een “zelftestende testbench”), maar dat is iets voor volgend jaar.

### Beschrijving: de testbench toevoegen aan je project

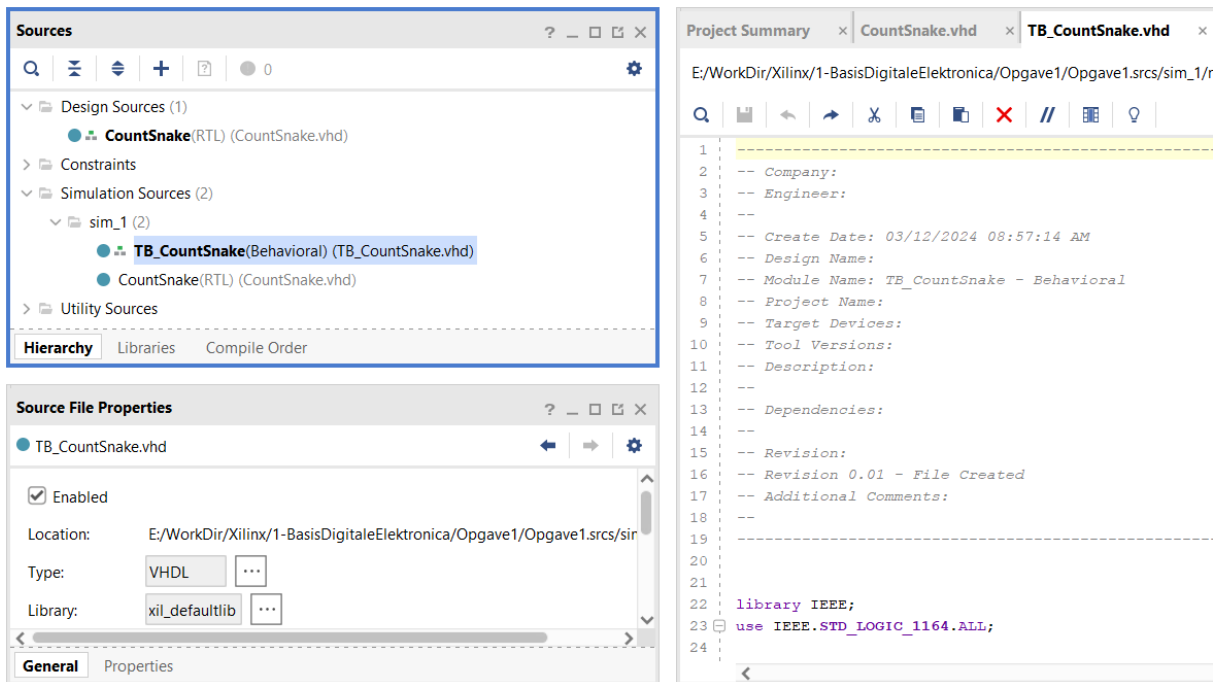
Zoals hierboven beschreven, moet je deze testbench gebruiken om je ontwerp te kunnen simuleren. Je hebt nu dus twee VHDL bestanden nodig: je ontwerp (bv. CountSnake.vhd) en je testbench (bv. TB\_CountSnake.vhd). In de inleidende opgave heb je gezien hoe je een “Design Source” moet toevoegen aan je project. Nu zal je dus ook nog een “Simulation Source” (je testbench) moeten toevoegen. Dit doe je op een gelijkaardige manier, namelijk door op het plusteken te klikken in het “Sources” venster.



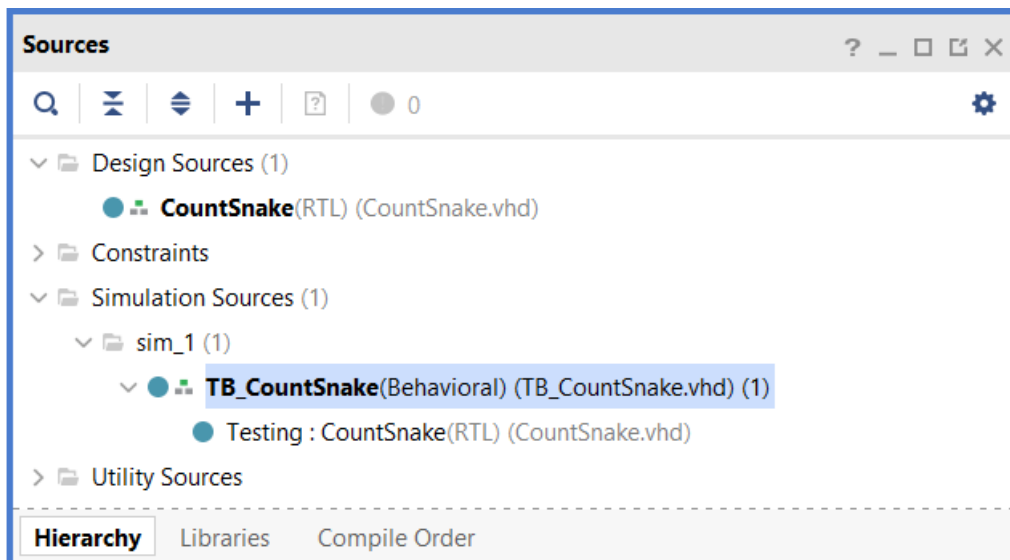
Je krijgt dan opnieuw het “Add Sources” venster, maar ditmaal kies je “Add Simulation Sources...” in plaats van “Add Design Sources...” omdat de testbench niet omgezet moet worden naar een schakeling en enkel dient om het ontwerp te simuleren.



In het “Sources” venster zal je dan ook zien dat de testbench niet onder “Design Sources” komt te staan, maar wel onder “Simulation Sources”. Als je deze openvouwt, dan zie je ook meteen de hiërarchie van het geheel.



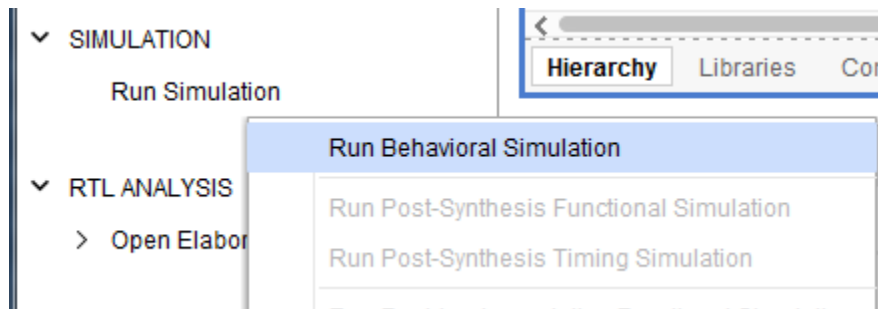
Merk op dat de testbench nog “naast” het eigenlijke ontwerp staat en niet “erboven” in de hiërarchie. Dit komt uiteraard omdat er nog geen component en port map in de VHDL van de testbench staat. Je moet dus nog de VHDL van de testbench aanpassen door de gegeven code van daarnet hierin te kopiëren. Als dat gebeurd is, dan zou je “Sources” venster er als volgt moeten uitzien:



Bekijk de videoles [Vivado projecten](#) op de Mediasite voor een demo over het toevoegen van design sources aan je project, met dat verschil dat je de testbench nu als simulation source toevoegt.

Als zowel je testbench als je ontwerp op de juiste plek staan, dan kan je je simulatie starten op dezelfde manier als bij de inleidende opgave. Je zal ditmaal echter manueel geen waarden meer moeten forceren op je ingangen, vermits je testbench dit automatisch voor jou zal doen. Let op dat je eerst de juiste voorbereidingen treft voor een simulatie. **Bekijk hiervoor zeker de videoles over [simuleren met Vivado](#)!** Je moet eerst 2 instellingen wijzigen bij “Simulation Settings” vooraleer je de simulatie opstart.





Als alles werkt naar behoren en je hebt je waveforms nagekeken, dan moet je je opgave indienen in Blackboard. Dit doe je op de volgende manier:

- Je opent Blackboard en gaat naar de cursus van dit vak.
- Onder "Practicum" > "Indienen" staat er een link "Opgave 1". Klik hierop.
- Dien vervolgens je VHDL bestand met het CountSnake ontwerp in.
- De exacte locatie van je .vhd bestand<sup>2</sup> kan je terugvinden door er in het "Sources" venster op te dubbelklikken. Het bestand zal dan rechts openen en net onder het betreffende tabblad kan je dan de exacte locatie terugvinden.
- Eventueel kan je nog opmerkingen toevoegen in het betreffende tekstvak.
- Klik op "Verzenden" om in te dienen.

Je hebt pas ingediend als je de juiste bestanden vóór de deadline correct hebt ingediend! Als je de verkeerde bestanden hebt ingediend en je merkt dit pas na de deadline, dan kan je alsnog de juiste indienen, maar dan geldt dit niet meer als "tijdig". Als je dus niet zeker bent, dan vraag je het aan mij.

**Merk op dat het niet geldig is om je VHDL code te kopiëren in een Word, PDF, tekst- of eender welk ander bestand. Ik heb je ".vhd" bestand nodig en NIETS ANDERS! Maak dus ook geen zip of dergelijke!**

**Geef ook enkel je ontwerp af, NIET je testbench!!**

Bekijk de [videoles over indienen](#) voor meer informatie over hoe en wat je juist moet indienen en hoe je de feedback moet interpreteren (zie ook volgende pagina's).

<sup>2</sup> Let op: Windows beschouwt .vhd bestanden standaard als een of ander virtueel diskformaat. Als je hier dus vanuit je Windows verkennen op dubbelklikt, dan krijg je een foutmelding van Windows over een "Virtual Hard Disk image" of dergelijke. Ook als je je .vhd bestand als bijlage in een email verzendt naar een UA mailadres, dan zal de firewall dit blokkeren. Bewerk je VHDL code dus in Vivado zelf en kopieer/plak je code rechtstreeks in je mail als je hierover een vraag hebt. Kijk [hier](#) om bestandsextensies zichtbaar te maken in Windows.

### Beschrijving: feedback en quoterig

Op regelmatige basis (meestal 2x per week) wordt er een bijna-automatisch script opgestart dat alle ingeleverde opgaven simuleert aan de hand van een zogenaamde “zelftestende testbench” (hierover meer in de opvolger van dit vak). Deze testbench kijkt na of je ontwerp doet wat het moet doen en zal hierover feedback genereren, die dan weer geüpload wordt naar Blackboard.

Check dus regelmatig op Blackboard of je feedback gekregen hebt over je ingediende opgaven. De feedback bevat een aantal AUTOCHECK boodschappen die je goed moet nalezen. Een voorbeeld:

Note: AUTOCHECKLIST

1: Zijn de 7segment codes van 0 t.e.m. 7 OK?

Note: AUTOCHECK 1: NOK: 7Segment=1111110b Count=0d

Het eerste deel is de AUTOCHECKLIST, waar alle automatische tests opgelijst worden. Voor deze opgave is er maar één test, namelijk “Zijn de 7segment codes van 0 tot en met 7 OK?”. De zelftestende testbench zal dus automatisch alle waarden van 0 tot en met 7 aanleggen op de Count ingang en vervolgens nakijken of de SevenSegm uitgang de juiste waarde bevat. In het bovenstaande voorbeeld was het ontwerp dus blijkbaar niet OK, want als er een 0 (000)<sub>2</sub> aangelegd werd op de Count-ingang, gaf dit “111110” op de SevenSegm uitgang. Dit klopt niet want het zou “001111” moeten zijn (bij een 0 gaat de led aan, bij een 1 gaat hij uit).





Ter info: De “b” achter “111110” geeft aan dat het over een binaire notatie gaat, terwijl een “d” een decimaal getal aanduidt.

Als je fouten hebt in je feedback, dan wil dit zeggen dat je je ontwerp niet (goed) gesimuleerd hebt. Je mag je fouten zeker en vast verbeteren en je opgave opnieuw indienen, maar het spreekt voor zich dat je het meeste punten krijgt voor een opgave die je van de eerste keer juist én op tijd indient.

Let wel op: **De automatische testscripts zijn geen debugmethode!** Iedere keer dat je een opgave fout en/of te laat indient, verlaag je je score. Het is de bedoeling dat je **zelf eerst** je opgave zorgvuldig debugt voor je ze indient. De testscripts zijn ook niet onfeilbaar. Het is bijzonder moeilijk om alle fouten te anticiperen die studenten zouden kunnen maken en ieder jaar worden er weer nieuwe versies van code ingediend die niet (of ten onrechte wél) door de scripts als fout gedetecteerd worden. Kijk dus zeker grondig je code na voor je ze indient!

Je krijgt **geen** punt per opgave, want ik verwacht niet dat je VHDL code van de eerste opdracht fantastisch zal zijn. Je zal op het einde van de rit (na je laatste opgave) één globaal punt krijgen voor al je opgaven, waarin rekening zal gehouden worden met hoe je met deze feedback bent omgegaan, hoeveel keer je per opgave hebt ingediend, hoeveel keer je te laat hebt ingediend, enz. Dezelfde fout in je laatste opgave weegt daarbij veel zwaarder door dan in je eerste opgave.

Kijk regelmatig op Blackboard onder “Cijferlijst” of er nieuwe feedback is binnen gekomen.

Inhoud	Agenda	Mededelingen	Discussies	<b>Cijferlijst</b>	Berichten	Groepen
Itemnaam	Einddatum	Status	Cijfer	Feedback		
 <b>Opgave 1</b> 1 poging ingezonden		Verzonden	Geen cijfer gegeven			
 <b>Feedback opgave 1</b>			0 / 1			
 <b>Opgave 2</b>						

Rechts naast de score kan je op de tekstballon klikken om de feedback te lezen van je ingeleverde opgave. Als je meerdere keren dezelfde opgave hebt ingediend, dan zie je dit ook door op dezelfde tekstballon te klikken. Kijk dus regelmatig naar deze tekstballonnen!

Heel deze uitleg over indienen, feedback en punten wordt ook gegeven in de [videoles over indienen](#).

Er komen ook momenten waarop je je ingeleverde code zal moeten komen bespreken met mij. Ik stel je dan vragen zoals “Waarom staat dit lijntje code hier?” of “Waarom staat hier 00100...?”. Als je hier niet op kan antwoorden, dan trek ik de conclusie dat je deze code gekopieerd hebt, gedownload hebt of met iemand het samengewerkt zonder dat je de code zelf begrijpt. Dat is zeer slecht voor je punten! Vóór je je code indient, is het belangrijk dat je vragen stelt over alles wat je niet (helemaal) begrijpt. Je bent hier om bij te leren en niet om vanaf dag 1 perfecte VHDL code te schrijven. Nadat je je code hebt ingediend, is het mijn beurt om de vragen te stellen. Nogmaals: één slechte opgave is niet het einde van de wereld, zolang je je maar herpakt en de feedback gebruikt die je gegeven wordt.

Nog een laatste tip: Probeer de mentaliteit te vermijden om “te doen wat je moet doen om te slagen voor dit vak en daarna alles te vergeten.” Alles wat je leert in “1-Basis digitale elektronica 1” zal je nodig hebben voor vele vakken in de komende jaren. Leer bijvoorbeeld hoe je moet simuleren en **onthoud** dit ook naar de volgende jaren toe. Je bespaart jezelf er een hoop moeite mee in de toekomst.

Veel succes!