Het schrijven van VHDL code voor simulatie en synthese

Inleiding

Tijdens het ontwerpproces is het belangrijk om VHDL code te schrijven die niet alleen de gewenste simulatieresultaten vertoont, maar die ook geschikt om door de logic synthesiser te worden omgezet in een schakeling die realiseerbaar is, en die daarna tijdens simulatie nog steeds het gewenste gedrag vertoont. In dit gedeelte zullen enkele richtlijnen worden gegeven om dit zo succesvol mogelijk te laten verlopen. Belangrijk hierbij is dat men zich tijdens het ontwikkelen van de VHDL code realiseert hoe de structuur van de schakeling zal zijn waarvoor men het gedrag beschrijft. Dit in ogenschouw nemende zullen we de volgende typen van VHDL code onderscheiden en apart behandelen.

- Gedragsbeschrijving combinatorische schakeling
- Gedragsbeschrijving sequentiële schakeling
- Structuurbeschrijving van een schakeling

Daarnaast zal voor sequentiële schakelingen nog worden verduidelijkt hoe alternatieve vormen van VHDL codering invloed hebben op simulatie en synthese.

Gedragsbeschrijving combinatorische schakelingen

Een gedragsbeschrijving van een combinatorische schakeling bestaat uit een architecture body met daarin concurrent statements zoals process statements en dataflow statements. Bij combinatorische schakelingen is het belangrijk om te zorgen dat elk uitgangsignaal een waarde krijgt voor elke combinatie van ingangsignalen. Hieronder volgt een voorbeeld van een combinatorische schakeling met een process statement.

```
library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of combinatorial is
begin
   -- a, b and c are input signals, y is output signal
   lbl1: process (a, b, c)
   -- All input signals must be in the sensitivity list
   begin
      -- Compute output value(s) based on input values
      -- Important: for every input combination, each
      -- output should receive a value !
      if ((a = '1' or b = '1') and c = '0') then
         y <= '1';
      else
         y <= '0';
```

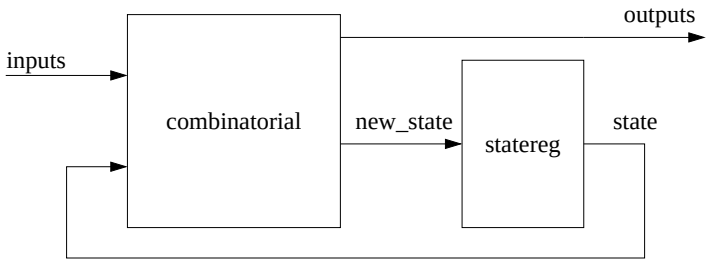
```
      end if;
   end process;
end behaviour;
```

Hier is dezelfde combinatorische schakeling op meer compacte wijze beschreven m.b.v. een dataflow statement:

```
architecture behaviour of combinatorial is
begin
   y <= (a or b) and (not c);
end behaviour;
```

Gedragsbeschrijving sequentiële schakeling

We gaan er van uit dat elke sequentiële schakeling die we ontwerpen beschreven wordt door een Moore machine zoals in figuur 1. Zie ook figuur **??**, waarbij δ en γ zijn samen genomen in een "combinatorial" blok, en Q gelijk is aan het "statereg" blok. Bij het beschrijven van een sequentiële schake-



Figuur 1: Blokschema voor het Moore model.

ling maken we daarom onderscheid tussen een register gedeelte dat de toestand onthoudt ("statereg") en een combinatorisch gedeelte dat de nieuwe toestand en de waarden van de uitgangsignalen berekent aan de hand van de ingangsignalen en de huidige state ("combinatorial"). Hoe de combinatorische schakeling "combinatorial" beschreven kan worden hebben we gezien in de voorafgaande sectie. Bij de beschrijving van het gedeelte dat de toestand onthoudt moeten we ons realiseren dat we in de Sea-of-Gates cellenbibliotheek de beschikking hebben over positive-edge triggered flipflops. Deze flipflops zijn uitgevoerd zonder reset, met een synchrone reset of met een asynchrone reset. Ook op het Altera bord bevinden zich flipflops van deze types. Wanneer "state" de huidige toestand weergeeft, "new_state" de volgende toestand, "RESET_STATE" de reset toestand, "clk" het klok signaal en "res" het reset signaal, en wanneer we uitgaan van een synchrone reset, dan kunnen we het gedeelte voor het opslaan van de toestand beschrijven met

```
   statereg: process (clk)
   begin
      if (clk'event and clk = '1') then
         if res = '1' then
            state <= RESET_STATE;
         else
```

```
            state <= new_state;
         end if;
      end if;
   end process;
```

Een compleet voorbeeld voor een beschrijving voor een sequentiële schakeling wordt dan:

```
library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of fsm is
   type state_type is (STATE1, STATE2);
   signal state, new_state: state_type;
begin
   statereg: process (clk)
   begin
      -- Because we use positive-edge triggered memory elements
      -- (flipflops) with synchronous reset, we assign a new
      -- state when clock goes to '1'
      if (clk'event and clk = '1') then
         if res = '1' then
            state <= STATE1;     -- assign reset state
         else
            state <= new_state;
         end if;
      end if;
   end process;
   combinatorial: process (state, input1)
   begin
      -- Assign output values based on state only (Moore machine)
      -- and compute new_state based on current state and input
      -- signals. Important: for all possible states, every output
      -- should receive a value and new_state should be defined.
      case state is
         when STATE1 =>
            output1 <= '0';
            if (input1 = '0') then
               new_state <= STATE2;
            else
               new_state <= STATE1;
            end if;
         when STATE2 =>
            output1 <= '1';
            new_state <= STATE1;
      end case;
   end process;
end behaviour;
```

Wanneer we zouden kiezen voor een asynchrone reset dan zou het eerste process er als volgt uitzien:

```
   statereg: process (clk, res)
   begin
      -- State register using positive-edge triggered memory elements
      -- and an asynchronous reset (note that res is in sensitity list).
      if res = '1' then
         -- assign reset state
         state <= STATE1;
      elsif (clk'event and clk = '1') then
         state <= new_state;
      end if;
   end process;
```

Een ander voorbeeld van een sequentiële schakeling die op bovenstaande wijze beschreven is, is de hotel schakeling die als voorbeeld wordt gegeven bij de inwerkopdrachten. Nog een ander voorbeeld is een 4 bits teller zoals die hieronder is gegeven. Merk op dat in dit geval de toestand wordt gevormd door de 4 bits van de std_logic_vector count en dat de uitgangen gevormd worden door de 4 bits van de vector a die hier rechtstreeks van worden afgeleid.

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

architecture behaviour of counter is
   signal count, new_count : std_logic_vector(3 downto 0);
begin
   statereg: process (clk)
   begin
      if (clk'event and clk = '1') then
         if reset = '1' then
            count <= (others => '0');
         else
            count <= new_count;
         end if;
      end if;
   end process;
   combinatorial: process (count, enable)
   begin
      a <= std_logic_vector(count);
      if (enable = '1') then
         new_count <= count + 1;
      else
         new_count <= count;
      end if;
```

```
   end process;
end behaviour;
```

Structuurbeschrijving van een schakeling

Een structuurbeschrijving kan gezien worden als een fysiek netwerk/circuit van componenten en de verbindingen daartussen. Afhankelijk van welke componenten er in zijn opgenomen zal de structuurbeschrijving een combinatorische of een sequentiële schakeling beschrijven. De componenten zijn representaties voor entities en architecturen van andere delen uit het ontwerp. Eventueel kunnen kunnen de componenten ook cellen uit de Sea-of-Gates cellenbibliotheek zijn, wanneer de beschrijving alleen voor de Sea-of-Gates chip bedoeld is en niet op het Altera bord wordt afgebeeld. Het is niet nodig om een structuurbeschrijving te synthetiseren. Wanneer ook alle componenten op structuurniveau bekend zijn, dan kan direkt vanuit de structuurbeschrijving een layout aangemaakt worden.

Hieronder volgt een voorbeeld van een structuurbeschrijving. Naast het tekstueel invoeren van de VHDL code kan een VHDL structuurbeschrijving ook op eenvoudige wijze worden aangemaakt met een schematic editor zoals Schentry.

```
library IEEE;
use IEEE.std_logic_1164.ALL;

architecture circuit of network is
   -- First the declaration of the components used
   component comp1
      port (x: in std_logic; y: out std_logic);
   end component;
   component comp2
      port (p, q: in std_logic; r: out std_logic);
   end component;
   -- second the declaration of the nets (internal nodes)
   signal net1, net2, net3 : std_logic;
begin
   -- connections between nets and pins
   output1 <= net2;
   -- instances of components and their connections
   comp1_1: comp1 port map (x=>net1, y=>net3);
   comp1_2: comp1 port map (x=>net2, y=>net1);
   comp2_1: comp2 port map (p=>net3, q=>input1, r=>net2);
end circuit;
```

Verschillende vormen van VHDL beschrijvingen voor sequentiële schakelingen bekeken

Naast de vorm die in de OP handleiding aangeraden wordt voor het beschrijven van sequentiële schakeling zijn er meer vormen mogelijk. Deze andere vormen hebben echter andere eigenschappen tijdens simulatie en/of synthese waarbij rekening gehouden moet worden wanneer ze gebruikt

worden. Hieronder volgen enkele van deze alternatieve vormen voor het beschrijven van sequentiële schakeling, waarbij telkens de afwijkende eigenschappen beschreven worden. Uitgegaan wordt van het voorbeeld van de hotel schakelaar.

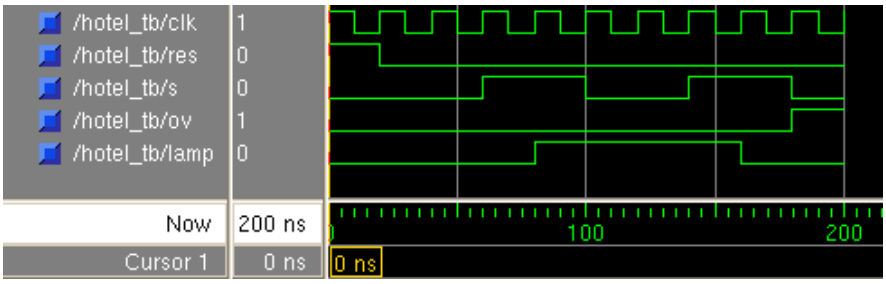
Aangeraden vorm

```
architecture behaviour of hotel is
   type lamp_state is (OFF0, OFF1, ON0, ON1);
   signal state, new_state: lamp_state;
begin
   lbl1: process (clk)
   begin
      if (clk'event and clk = '1') then
         if res = '1' then
            state <= OFF0;
         else
            state <= new_state;
         end if;
      end if;
   end process;
   lbl2: process(state, s, ov)
   begin
      case state is
         when OFF0 =>
            lamp <= '0';
            if (s = '1') and (ov = '0') then
                new_state <= ON1;
            else
                new_state <= OFF0;
            end if;
         when ON1 =>
            lamp <= '1';
            if (s = '0') and (ov = '0') then
                new_state <= ON0;
            else
                new_state <= ON1;
            end if;
         when ON0 =>
            lamp <= '1';
            if (s = '1') and (ov = '0') then
                new_state <= OFF1;
            else
                new_state <= ON0;
            end if;
         when OFF1 =>
            lamp <= '0';
            if (s = '0') and (ov = '0') then
                new_state <= OFF0;
            else
                new_state <= OFF1;
            end if;
```

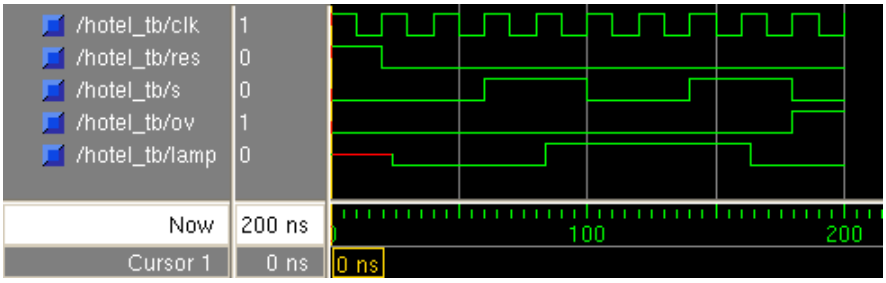
```
      end case;
   end process;
end behaviour;
```

Bij deze vorm wordt er een process blok gebruikt voor het beschrijven van het (synchrone) toestandsregister en een process blok voor het beschrijven van de (asynchrone) schakeling die nodig is om - gebaseerd op de huidige toestand en de ingangssignalen - de nieuwe toestand en de uitgangssignalen te berekenen. Deze vorm wordt aangeraden omdat het een hoog niveau gedragsbeschrijving is die door synthesizers eenvoudig (zonder af te wijken van het oorspronkelijke gedrag) is om te zetten naar een schakeling bestaande uit flipflops en logische poorten. Deze vorm leidt, na synthese voor de Sea-of-Gates bibliotheek, tot een schakeling met 2 flipflops.

De simulatie van bovenstaande gedragsbeschrijving (Figuur 2) en de simulatie resultaten van het gesynthetiseerde circuit (Figuur 3) geven eenzelfde resultaat, afgezien van een klein verschil in vertragingstijden (bij de gesynthesieerde schakeling zijn de poort vetragingstijden meegenomen) en van de waarde van uitgangssignaal in de initialisatie-fase.



Figuur 2: Simulatie behavior beschrijving



Figuur 3: Simulatie gesynthetiseerde schakeling

Berekening van nieuwe toestand binnen het geklokte process blok

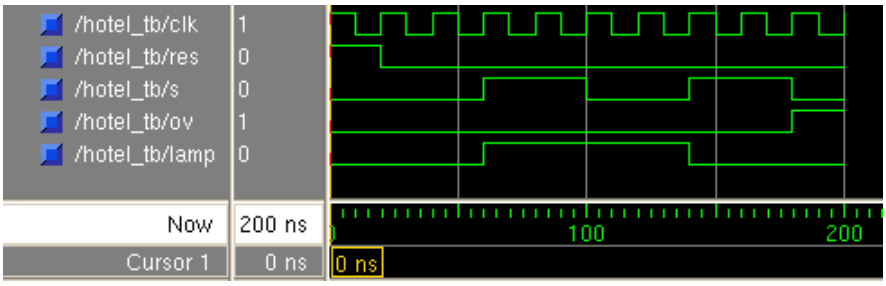
Bij de volgende vorm van VHDL coderen is in het eerste (geklokte) process blok, naast de toekenning van de nieuwe toestand, ook de berekening van de nieuwe toestand opgenomen. In het tweede process is alleen gespecifeerd hoe de uitgang van de schakeling afhangt van de huidige toestand.

```
architecture behaviour of hotel is
```

```
   type lamp_state is (OFF0, OFF1, ON0, ON1);
   signal state: lamp_state;
begin
   lbl1: process (clk)
   begin
      if (clk'event and clk = '1') then
         if (res = '1') then
            state <= OFF0;
         else
            case state is
               when OFF0 =>
                  if (s = '1') and (ov = '0') then
                     state <= ON1;
                  else
                     state <= OFF0;
                  end if;
               when ON1 =>
                  if (s = '0') and (ov = '0') then
                     state <= ON0;
                  else
                     state <= ON1;
                  end if;
               when ON0 =>
                  if (s = '1') and (ov = '0') then
                     state <= OFF1;
                  else
                     state <= ON0;
                  end if;
               when OFF1 =>
                  if (s = '0') and (ov = '0') then
                     state <= OFF0;
                  else
                     state <= OFF1;
                  end if;
            end case;
         end if;
      end if;
   end process;
   lbl2: process (state)
   begin
      if (state = OFF0) or (state = OFF1) then
         lamp <= '0';
      else
         lamp <= '1';
      end if;
   end process;
end behaviour;
```

De simulatie resultaten van de behavior beschrijving zijn in dit geval afwijkend van die voor de aangeraden behavior beschrijving, zie Figuur 4. De verklaring hiervoor is als volgt. Omdat in de test-bench het ingangssignaal s verandert op het moment dat ook de klok omhoog gaat, zal deze “nieuwe” waarde in het eerste process blok van de behavior beschrijving worden meegnomen in de berekening

van de nieuwe toestand op het moment dat de klok omhoog gaat.



Figuur 4: Simulatie behavior beschrijving

In de fysische werkelijkheid zal bij het tegelijkertijd veranderen van klok en ingangssignaal (door de setup tijd van de flipflop) de “oude” waarde van het inganssignaal worden gebruikt voor het berekenen van de nieuwe toestand, en wordt de de “nieuwe” waarde gebruikt tijdens de volgende opgaande klokflank. Het simulatie resultaat van de schakeling die onstaat na synthese voor de vorm die hier beschreven wordt, is dan ook wel gelijk aan het simulatie resultaat voor de aangeraden vorm. Verder zal ook hier tijdens synthese een schakeling worden gegenereerd met 2 flipflops.

Wanneer er voor gezorgd wordt dat de ingangssignalen nooit tegelijkertijd met de opgaande klokflank veranderen zal bovengenoemd simulatie verschil uiteraard niet optreden en zal de hier beschreven gedragsbeschrijving dezelfde resultaten geven (voor simulatie en synthese) als de aangeraden vorm.

Berekening van nieuwe toestand en uitgangssignalen binnen het geklokte process blok

```
architecture behaviour of hotel is
   type lamp_state is (OFF0, OFF1, ON0, ON1);
   signal state: lamp_state;
begin
   lbl1: process (clk)
   begin
      if (clk'event and clk = '1') then
         if (res = '1') then
            state <= OFF0;
            lamp <= '0';
         else
            case state is
               when OFF0 =>
                  if (s = '1') and (ov = '0') then
                     state <= ON1; lamp <= '1';
                  else
                     state <= OFF0; lamp <= '0';
                  end if;
               when ON1 =>
                  if (s = '0') and (ov = '0') then
                     state <= ON0;
                  else
                     state <= ON1;
```

```
                end if;
                lamp <= '1';
             when ON0 =>
                if (s = '1') and (ov = '0') then
                   state <= OFF1; lamp <= '0';
                else
                   state <= ON0; lamp <= '1';
                end if;
             when OFF1 =>
                if (s = '0') and (ov = '0') then
                   state <= OFF0;
                else
                   state <= OFF1;
                end if;
                lamp <= '0';
          end case;
       end if;
    end if;
  end process;
end behaviour;
```

Bij deze vorm vindt zowel de berekening van de nieuwe toestand als de berekening van de uitgangssignalen plaats in het geklokte process blok. Hoewel dit leidt tot een compacte beschrijving met slechts 1 process blok, moet wel rekening worden gehouden met: (1) hetzelfde afwijkende simulatie gedrag van de behavior beschrijving als bij de hiervoor beschreven vorm, en (2) het feit dat de synthesizer nu een schakeling met 3 flipflops zal genereren i.p.v. 2. Dit laatste komt doordat de synthesizer behalve het signal "state" ook het signal "lamp" als een onderdeel van de toestand opvat, en dus naast 2 flipflops voor "state" ook 1 flipflop voor "lamp" gebruikt.

Het lezen van signalen die ook geschreven worden binnen een process blok

```
architecture behaviour of hotel is
  type lamp_state is (OFF0, OFF1, ON0, ON1);
  signal state: lamp_state;
begin
  lbl1: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (res = '1') then
        state <= OFF0;
      else
        case state is
          when OFF0 =>
            if (s = '1') and (ov = '0') then
              state <= ON1;
            else
              state <= OFF0;
            end if;
          when ON1 =>
            if (s = '0') and (ov = '0') then
```

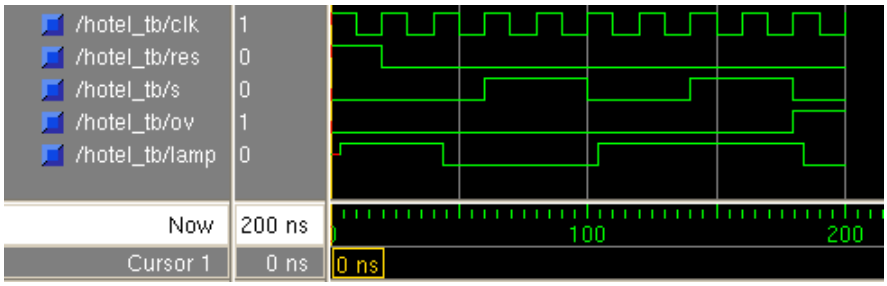
```
                  state <= ON0;
               else
                  state <= ON1;
               end if;
            when ON0 =>
               if (s = '1') and (ov = '0') then
                  state <= OFF1;
               else
                  state <= ON0;
               end if;
            when OFF1 =>
               if (s = '0') and (ov = '0') then
                  state <= OFF0;
               else
                  state <= OFF1;
               end if;
         end case;
      end if;
      if (state = ON0) or (state = ON1) then
          lamp <= '1';
      else
          lamp <= '0';
      end if;
   end if;
end process;
end behaviour;
```

Hierbij is het simulatie resultaat van de gedragsbeschrijving gelijk aan dat van de aangeraden vorm. De verklaring hiervoor is als volgt. Net zoals in de voorafgaande vorm zal de toestand "state" onmiddelijk ge-update wordt met de "nieuwe" waarde van het ingangssignaal en zorgt dit voor 1 klokpuls verschuiving vooruit in de tijd van het het toestandssignaal. Echter, in het if-statement dat de waarde voor lamp berekent, wordt de oude waarde van state gebruikt (dit is een belangrijke algemene eigenschap die geldt voor signals in VHDL beschrijvingen: de nieuwe waarde die voor een signal berekend wordt, wordt pas toegekend tijdens een volgende uitvoering van het process blok !). Daardoor onstaat er 1 klokpuls vertraging voor de waarde van lamp (die ook in dit geval zal worden opgeslagen in een aparte 3e flipflop). Het geheel vertoont daardoor eenzelfde gedrag als de behavior beschrijving van de aangeraden schakeling.

Bij simulatie van het gesynthetiseerde circuit wordt "state" weer - door de setup tijd van de flipflop - ge-update 1 klokpuls nadat het ingangssignaal verandert. De synthesizer zal verder - overeenkomstig de behavior beschrijving - een schakeling genereren waarbij de waarde van de lamp 1 klokpuls achter loopt bij de toestand waar hij eigenlijk bij hoort. De schakeling als geheel vertoont dan een gedrag waarbij de lamp 1 klokpuls later reageert dan bij de aangeraden schakeling, zie Figuur 5

Het gebruik van variabelen i.p.v. signalen binnen een process blok

```
architecture behaviour of hotel is
   type lamp_state is (OFF0, OFF1, ON0, ON1);
begin
```



Figuur 5: Simulatie gesynthetiseerde schakeling

```
lbl1: process (clk)
variable state: lamp_state;
begin
   if (clk'event and clk = '1') then
      if (res = '1') then
         state := OFF0;
      else
         case state is
            when OFF0 =>
               if (s = '1') and (ov = '0') then
                  state := ON1;
               else
                  state := OFF0;
               end if;
            when ON1 =>
               if (s = '0') and (ov = '0') then
                  state := ON0;
               else
                  state := ON1;
               end if;
            when ON0 =>
               if (s = '1') and (ov = '0') then
                  state := OFF1;
               else
                  state := ON0;
               end if;
            when OFF1 =>
               if (s = '0') and (ov = '0') then
                  state := OFF0;
               else
                  state := OFF1;
               end if;
         end case;
      end if;
      if (state = ON0) or (state = ON1) then
          lamp <= '1';
      else
          lamp <= '0';
      end if;
   end if;
```

```
   end process;
end behaviour;
```

Deze beschrijving is gelijk aan de voorafgaande beschrijving behalve dat “state” is gedeclareerd als een variable i.p.v. een signal.

In de voorafgaande vorm zagen we dat de nieuwe waarde van een signal zoals “state” pas beschikbaar is tijdens een volgende uitvoering van een process (op de opgaande klokflank) en zorgt dit voor 1 klokpuls vertraging op het uitgangssignaal. Wanneer we “state” declareren als een variable i.p.v. als een signal, dan treedt dit verschijnsel niet op. Bij een variable wordt een nieuw berekekende waarde in een process onmiddelijk gebruikt in de daaropvolgende statements. Gevolg is dat het simulatie resultaat van de gedragsbeschrijving dan 1 klokpuls voor loopt t.o.v. het simulatie resultaat van de de gedragsbeschrijving van de aangeraden vorm en dat het simulatie resultaat van het gesynthetiseerde circuit gelijk is aan dat van de aangeraden beschrijving.

Ook hier zullen weer 3 flipflops gegenereerd worden tijdens synthese van de schakeling.