Module-opdracht: Arithmatic Logic Unit (ALU)

Gemaakt door leden van projectgroep A1
Joran Out, 4331958
Alex Oudsen, 4325494

In opdracht van
Roy Blokker, 4148894
Elke Salzmann, 4311450

26 November 2014
Versie 1.0

Samenvatting

Dit document beschrijft het ontwerp van een Arithmetic Logic Unit (ALU) voor de zogenaamde moduleopdracht bij het EPO-3 project van de opleiding Electrical Engineering aan de TU Delft. Een ALU is een module die op een chip wordt ingezet voor het uitvoeren van rekenkundige en logische bewerkingen. De ALU in dit verslag is ontworpen op basis van een opdracht, welke specificeert dat de ALU in staat dient te zijn tot optellen, aftrekken, naar links- en rechts verschuiven & de logische operaties NOT, AND, OR en XOR. Het ontwerpen van de ALU gebeurt in het IC-ontwerpprogramma *GoWithTheFlow*, waarbinnen nog vele andere programma's worden gebruikt; bijvoorbeeld voor het simuleren van VHDL en het genereren van een layout voor de ontworpen schakeling op een Sea-of-gates chip. De ontworpen ALU en een bijpassende testbench zullen worden beschreven in VHDL, waarbij gebruik gemaakt wordt van diagrammen om de omzetting naar VHDL te vereenvoudigen. Vervolgens wordt het ontwerp op verschillende logische niveau's gesimuleerd in *GoWithTheFlow*, van de ruwe VHDL-code tot op transistorniveau. De benodigde tussenstappen als het genereren van een layout kunnen hierbij geautomatiseerd worden uitgevoerd binnen de ontwerpomgeving. Op deze manier kan worden gecontroleerd of het ontwerp het gespecificeerde gedrag vertoont. Ten slotte worden de verkregen resultaten van de verscheidene simulaties vergeleken met de aangeleverde specificaties, zodat aan de opdrachtgevers kan worden getoond dat de ontworpen module hieraan voldoet.

Inhoudsopgave

1 Inleiding

In dit verslag wordt het ontwerpproces van een Arithmetic Logic Unit (ALU) behandeld. Dit ontwerp is uitgevoerd in het kader van de zogenaamde moduleopdracht behorend bij het EPO-3 project van de opleiding Electrical Engineering aan de TU Delft. Deze moduleopdracht houdt in dat een tweetal leden van de projectgroep opdracht geeft tot het ontwerp van een zogenaamde 'module' voor op een Sea-of-gates chip. Hierbij is een module een schakeling op een chip met een bepaalde functie, welke over het algemeen vaak gebruikt wordt. Twee andere groepsleden dienen deze opdracht uit te voeren volgens de specificaties van de opdrachtgevers. Deze specificaties bevatten in ieder geval informatie over de functionaliteit, in- en uitgangssignalen en testbaarheid van de te ontwerpen module. Ook eventuele randvoorwaarden dienen in de specificaties te zijn opgenomen. Het op te leveren resultaat van de opdracht bestaat uit de ontworpen ALU in behavioural VHDL, een geschikte testbench in VHDL en dit verslag, dat onder andere ook de simulatieresultaten van het ontwerp op de verschillende logische niveau's dient te bevatten. De officiële beschrijving van de Module opdracht is ten slotte te vinden in hoofdstuk 14 van de handleiding behorend bij het EPO-3 project [1], welke ook online via blackboard beschikbaar gesteld is [2].

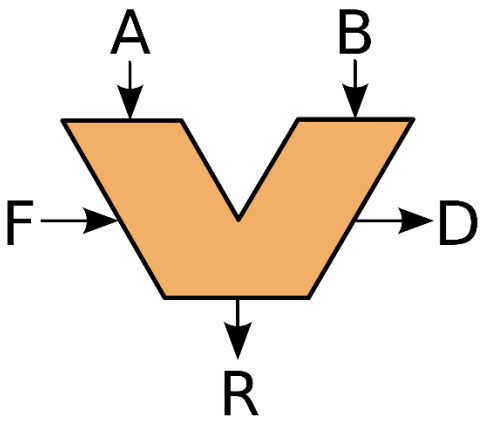
2 Specificaties

Zoals in de inleiding al is aangegeven, zijn de specificaties van de te ontwerpen ALU opgesteld door een ander tweetal leden van de projectgroep. Deze specificaties zijn in overleg met de opdrachtgevers en de groepsbegeleiders van het EPO-3 project zo aangepast dat duidelijk is wat er precies van de ontwerpen ALU verwacht wordt. Bovendien is in de complexiteit van de specificaties rekening gehouden met een beschikbare tijd van drie dagdelen à vier uur voor de uitvoering van de opdracht. Hierop volgend zijn de uiteindelijk vastgestelde specificaties letterlijk in dit verslag overgenomen:

ALU

Een ALU is een arithmetic logic unit. Dit onderdeel voert rekenkundige en logische bewerkingen uit. De inputs A en B zijn binaire vectoren en de functie selector is een vector, die aangeeft wat voor bewerking er uitgevoerd moet worden met de inputs A en B. Het resultaat R kan de som of het product zijn van A en B of de uitkomst van een logische bewerking, zoals NOT A. D is de uitvoer status. In dit geval is het de carry.

In dit geval hoeft de ALU alleen maar op te kunnen tellen, af te kunnen trekken, de inputs een bit te kunnen verschuiven of logische bewerkingen uit te kunnen voeren. Dit kan allemaal binnen één klokperiode.



Specificaties:

- Input A en B zijn 4-bits vectoren
- Functie-selector (F) is 3-bits vector
- D is de carry van 1 bit
- Het resultaat (R) is een 4-bits vector
- De R en D moeten opgeslagen worden in een register
- Inputs A en B en resultaat R moeten 4-bits vectoren zijn in two's complement representation
- Rising edge triggered

Operaties:

Met F mag zelf aangegeven worden welke functie gedaan wordt.

- B bij A optellen
- B van A aftrekken
- A een bit naar links verschuiven (de output van de carry is het MSB, het LSB wordt een 0)
- A een bit naar rechts verschuiven (het MSB wordt een 0, het LSB valt weg)
- A AND B
- A XOR B
- A OR B
- NOT A

3 Ontwerp

De eerste stap in het ontwerpproces van de ALU begint met de interpretatie van de vastgestelde specificaties. Hoewel over de meeste punten in deze specificaties geen onduidelijkheid bestaat, wordt hier voor de duidelijkheid nog wel een overzicht van de specificaties gegeven zoals deze zijn geïnterpreteerd tijdens het ontwerpen van de module:

- De ALU dient te reageren op de rising edge van de ingang klok (clk) waarop deze gaat werken. Dit houdt in dat de uitgangssignalen alleen mogen veranderen bij een opgaande klokflank.
- De ingangen A en B zijn 4-bits vectoren in 2's complement representatie waarop de verschillende bewerkingen worden gedaan die de ALU uit moet gaan kunnen voeren. Afhankelijk van de uit te voeren operatie wordt of alleen signaal A of worden beide signalen gebruikt.
- Het resultaat van de bewerkingen wordt weggezet in een register met de huidige uitgangswaarden van de ALU, welke het resultaat (4 bits) en de carry (1 bit) zal bevatten. Dit betekent praktisch gezien dat de uitgang van de ALU wordt behouden zolang deze niet wordt overschreven door het reslutaat van een volgende operatie. Wat er precies aan welke uitgang terecht komt is afhankelijk van de uit te voeren operatie.
- Het 3-bits ingangssignaal F dient ten slotte te worden geïnplementeerd als een selectiesignaal, waarmee kan worden aangegeven welke van de volgende acht opgegeven functies de ALU moet gaan uitvoeren:

  1. A + B (geeft een 5-bits resultaat, met de MSB in de carry)
  2. A - B (geeft een 5-bits resultaat, met de MSB in de carry)
  3. Left-shift A (MSB in de carry & LSB wordt '0')
  4. Right-shift A (carry en MSB '0' & LSB gaat verloren)
  5. A AND B (carry '0')
  6. A XOR B (carry '0')
  7. A OR B (carry '0')
  8. NOT A (carry '0')

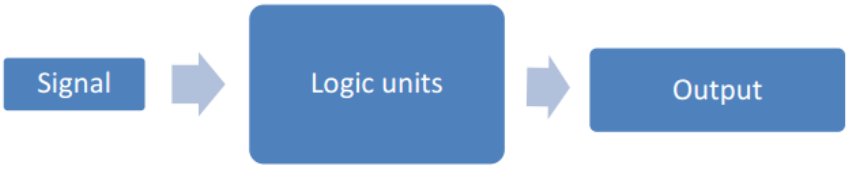
  Welke waarde van F wordt toegekend aan welke van deze functies is overgelaten aan de ontwerpers. Er is uiteindelijk gekozen voor de uitwerking van tabel 1:

Tabel 1: Overzicht gekozen representatie van de functies van de ALU in F

Functie van de ALU	Binaire waarde van F
NOT A	"000"
A OR B	"001"
A AND B	"010"
A XOR B	"011"
A + B	"100"
A - B	"101"
Left-shift A	"110"
Right-shift A	"111"

3.1 Blokrepresentatie van de ALU

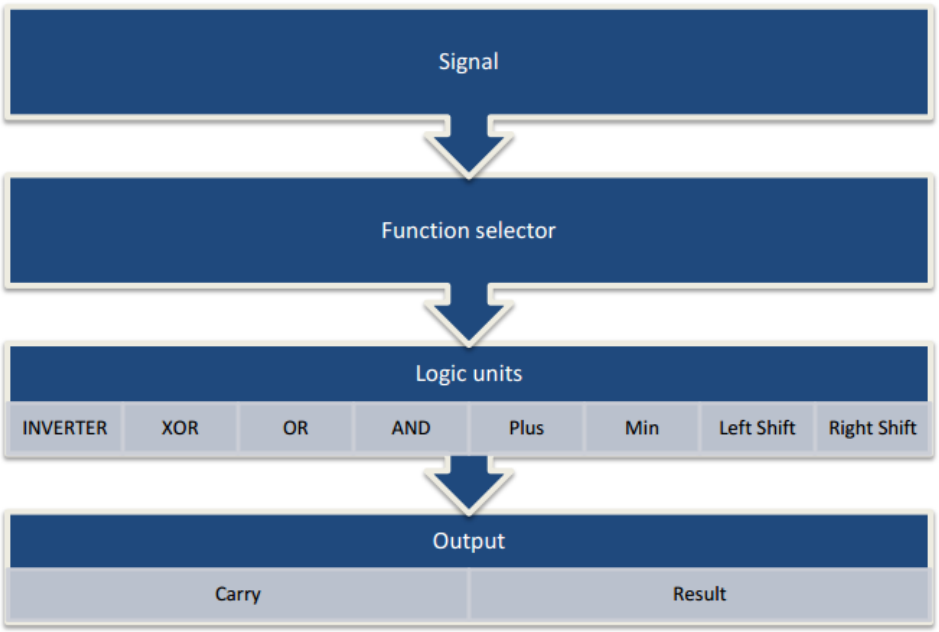
Een eerste stap in de omzetting van specificaties tot VHDL-ontwerp van de ALU, is het opstellen van een black-box-model van de ALU zoals in figuur 1. In dit figuur is de ALU weergegeven als een blok, dat verschillende logische onderdelen (logic units) bevat. Hiermee wordt aangegeven dat het blok ALU een module is die logica implementeert. Verder worden in dit simpele diagram alle ingangen samengenomen tot een blok signal aan de ingang van de ALU en alle uitgangen tot een blok output aan de uitgang van de blokrepresentatie van de ALU.



Figuur 1: Blokrepresentatie van de ALU

3.2 Opdeling in deelschakelingen

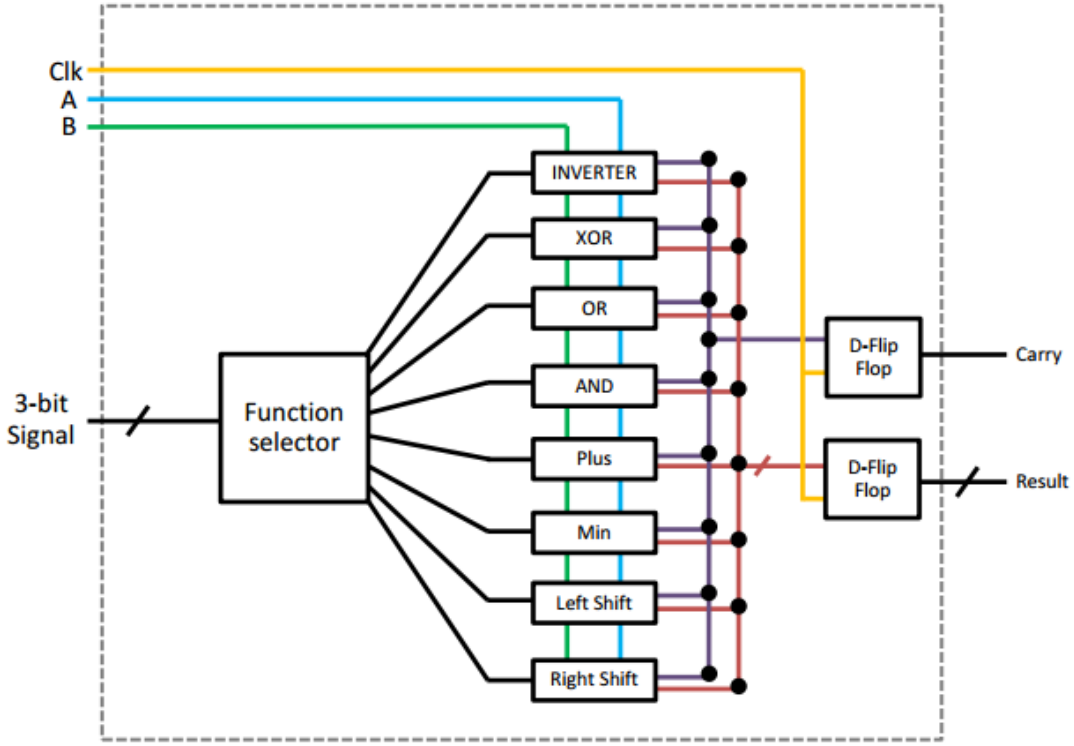
Met behulp van figuur 1 is het echter nog niet mogelijk om de ALU direct in VHDL te implementeren. Daarom worden extra details aan dit diagram toegevoegd, door verschillende deelschakelingen en -signalen te introduceren. Dit geeft een nieuw blokdiagram, welke wordt weergegeven in figuur 2. In dit diagram zijn de verschillende functies van de ALU nu te onderscheiden. Ook zijn de uitgangssignalen nu gesplitst. Dit diagram is echter ook nog niet volledig genoeg voor een VHDL-beschrijving.



Figuur 2: Opdeling van de ALU in verschillende deelblokken

3.3 Omzetting van diagram tot VHDL

De module-opdracht vraagt om een resultaat in behavioural VHDL. Het uiteindelijke doel is dan ook om de ALU-module, zoals deze in de specificaties wordt beschreven, om te zetten in een behavioural VHDL beschrijving. Figuren 1 en 2 bleken hiervoor niet gedetailleerd genoeg. Daarom wordt het blokdiagram nog verder uitgebreid, wat figuur 3 oplevert.



Figuur 3: Overzichtsweergave van de ALU in blokken met bijbehorende signalen

Nu is duidelijk te zien hoe de ALU geïmplementeerd moet gaan worden in VHDL. Alle gespecificeerde in- en uitgangssignalen zijn nu aan te wijzen. Bovendien is aangegeven welk signaal waar naar toe gaat. Ook is duidelijk te zien dat de functieselectie met behulp van het 3-bits signaal F het gemakkelijkste gerealiseerd kan worden met behulp van een case-statement. Ten slotte wordt getoond te zien hoe de registers aan de uitgang worden geïmplementeerd. De implementatie van de arithmetische en logische bewerkingen van de ingangssignalen spreekt voor zich. De resulterende behavioural VHDL beschrijving van de ALU-module is terug te vinden in de appendix in sectie 6. In deze beschrijving is voor de in- en uitgangssignalen de naamgeving van de figuur in de specificaties aangehouden, op het uitgangssignaal carry na, die wordt aangegeven met een c in plaats van een d.
Voor het testen van de geschreven VHDL wordt gebruik gemaakt van een testbench. Deze is ook geschreven in VHDL en terug te vinden in de appendix in sectie 7.

4 Resultaten

Met de beschrijving van de ALU en bijbehorende testbench in behavioural VHDL is het op te leveren product in principe af. Voordat deze files kunnen worden ingeleverd, moet echter wel eerst worden gecontroleerd of deze wel voldoen aan alle specificaties van de opdrachtgevers. Blijkt dit niet het geval, dan dient de VHDL beschrijving van de ALU zo te worden aangepast, dat dit wel het geval wordt. Het testen van de behavioural VHDL beschrijving van de ALU gebeurt in eerste instantie met behulp van de geschreven testbench in de VHDL simulator van *Modelsim*. Het resultaat van deze simulatie is weergegeven in figuur 5 in de appendix in sectie 7 en is veelbelovend: de ALU functioneert zoals verwacht en er lijkt aan alle specificaties van de opdrachtgevers te worden voldaan.

4.1 Synthetiseren

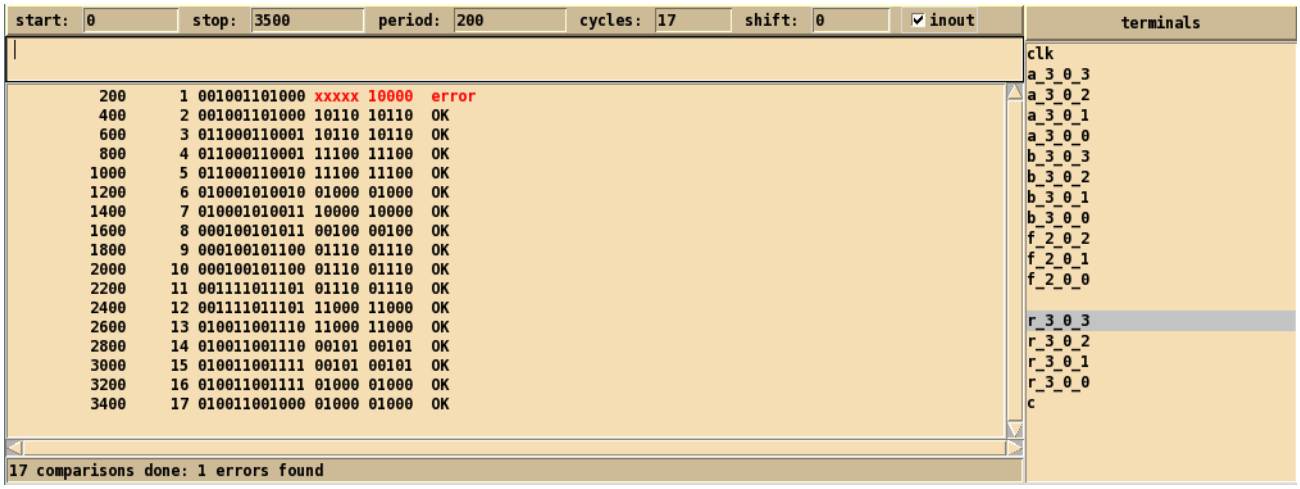
Een geslaagde simulatie op een hoog beschrijvend niveau als VHDL is echter nog geen bewijs dat de ontworpen ALU ook zal gaan werken op een Sea-of-gates chip. Daarom wordt binnen de omgeving van *GoWithTheFlow* een logische synthese uitgevoerd op de VHDL beschrijving van de ALU, wat ervoor zorgt dat deze gedragsbeschrijving wordt omgezet in een netwerkbeschrijving bestaande uit cellen uit de Sea-of-Gates celbibliotheek. Ook deze beschrijving is in VHDL en kan dus worden gesimuleerd met behulp van *Modelsim*. Het resultaat van deze simulatie is te vinden in figuur 6 in de appendix in sectie 7 en is vrijwel identiek aan die van de simulatie van de VHDL behavioural beschrijving in figuur 5 in dezelfde sectie. Het enige verschil is dat de uitgangssignalen nu onderhevig zijn aan enige vertraging. Deze vertraging is echter kort genoeg, zodat de ALU in staat blijft zijn operaties binnen één klokperiode kan voltooien.

4.2 Layout

De gesynthetiseerde VHDL beschrijving van de ALU bevat echter nog geen informatie over de plaatsing van de Sea-of-gates cellen op een chip. Ook met de bedrading van deze chip is dus nog geen rekening gehouden. Daarom wordt binnen *GoWithTheFlow* vanuit de gesynthetiseerde VHDL een layout gegenereerd met behulp van *row placer* voor het plaatsen van de cellen en *trout* in *seadali* voor het automatisch genereren van de bedrading. Alternatief gezien had binnen *seadali* ook het plaatsen en bedraden in één keer gedaan kunnen worden met behulp van place & route, welke *Madonna* aanroept voor het plaatsen en *trout* voor de bedrading. Het probleem is echter dat *Madonna* de cellen zo compact mogelijk probeert te plaatsen, waardoor het voor komt dat *trout* niet genoeg ruimte meer heeft voor het bedraden van alle cellen. Daarom is gebruik gemaakt van *row placer*, zodat het aantal rijen cellen en de onderlinge afstand tussen de cellen handmatig vergroot kan worden. Dit zorgt er voor dat *trout* genoeg ruimte heeft voor het bedraden van alle cellen, maar dit gaat wel ten koste van de efficintie van de schakeling. Om de validiteit van de gegenereerde layout te controleren, kan binnen *GoWithTheFlow* een VHDL beschrijving van de layout worden geëxtraheerd. Deze VHDL beschrijving kan vervolgens weer met behulp van *Modelsim* worden gesimuleerd. Het resultaat van deze simulatie is weergegeven in figuur 7 in de appendix in sectie 7. Deze simulatie geeft hetzelfde resultaat als de simulatie van de gesynthetiseerde VHDL in figuur 6. Dit betekent dat de gegenereerde layout overeenkomt met de VHDL netwerkbeschrijving van de ALU.

4.3 Switch-level simulatie

Naast de zojuist beschreven controlewijze bestaat er nog een tweede manier om te controleren of de gegenereerde layout overeenkomt met de VHDL netwerkbeschrijving. Dit gebeurd door middel van een zogenaamde switch-level simulatie. Deze simulatie wordt uitgevoerd door de *PSPICE* simulator en beschouwt de transistoren in het ontwerp als schakelaars. Deze andere aanpak is over het algemeen een nauwkeurigere manier van verificatie, omdat deze ervoor zorgt dat problemen als spikes of glitches naar voren kunnen komen die in een simulatie met *Modelsim* onopgemerkt blijven. Het resultaat van de switch-level simulatie is weergegeven in figuur 8 in de appendix in sectie 7. Doordat deze simulatie is uitgevoerd door *PSPICE* in plaats van *Matlab*, is het niet zo eenvoudig om het resultaat van de switch-level simulatie te vergelijken met die van de VHDL netwerkbeschrijving. Binnen *GoWithTheFlow* bestaat echter de optie om beide simulaties automatisch met elkaar te laten vergelijken. Hierbij worden vlak voor iedere opgaande klokflank de uitgangssignalen van beide simulaties met elkaar vergeleken. De resultaten van deze vergelijking zijn weergegeven in figuur 4. In de figuur is te zien dat deze vergelijking alleen een foutmelding geeft tijdens de eerste klokperiode. Tijdens de eerste klokperiode is de initialisatie van het systeem echter nog niet helemaal voltooid, waardoor de uitgangssignalen van het systeem nog niet eenduidig gedefinierd zijn. De uitgangssignalen van beide simulaties zijn op dit moment dus nog willekeurig, wat het logisch maakt dat ze niet met elkaar overeenkomen.



Figuur 4: Vergelijking resultaten behavioural VHDL- en Switch-level simulaties

5 Conclusies

Zoals gebleken uit de verschillende simulaties, kan de ontworpen ALU-module op basis van een 3-bits ingangssignaal één van de acht vastgestelde operaties uitvoeren op één of twee 4-bits ingangssignalen, welke zijn gecodeerd in 2's complement. Bijbehorende uitganssignalen worden volgens specificatie bewaard totdat er nieuwe uitganssignalen beschikbaar zijn. De ALU voldoet bovendien aan de eisen om de operaties te starten op de opgaande klokflank en te voltooien binnen één klokperiode. Ten slotte bestaan de uitganssignalen van de ALU, zoals gespecificeerd, uit een 4-bits signaal resultaat en een carry-signaal van 1 bit.

Ondanks het feit dat de ALU voldoet aan de specificaties, zouden er voor een volgende opdracht nog verdere toepassingen of verbeteringen geïmplementeerd kunnen worden. Zo zouden er bijvoorbeeld meer functies geïmplementeerd kunnen worden. Daarnaast zou er een functie toegevoegd kunnen worden waarbij de ALU zijn laatste resultaat kan gebruiken als input. Ten slotte zou ook het toevoegen van een (synchrone) reset, een signaal dat nu niet in de specificaties is opgenomen, een mogelijkheid bieden om de registers aan de uitgang leeg te maken.

Referenties

[1] Bakker, A. et al., *Handleiding practicum Geïntegreerde Schakelingen en project Ontwerp een Chip*, De Module opdracht, TU Delft, September 2014, pp. 68.

[2] Handleiding GS en EPO-3, `https://blackboard.tudelft.nl/bbcswebdav/pid-2389635-dt-content-rid-7944763_2/courses/34308-141502/handleiding_GS_EPO3.pdf`, geraadpleegd op 23 nov. 2014.

6 Appendix A: ALU in VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity alu is
  port (clk : in  std_logic;
        a   : in  std_logic_vector (3 downto 0);
        b   : in  std_logic_vector (3 downto 0);
        f   : in  std_logic_vector (2 downto 0);
        r   : out std_logic_vector (3 downto 0);
        c   : out std_logic);
end entity alu;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

architecture behaviour of alu is

signal d : std_logic;
signal z : std_logic_vector (3 downto 0);

begin

lbl1: process (clk) is
begin

  if (clk'event and clk = '1') then
        r <= z;
        c <= d;
  end if;

end process;
```

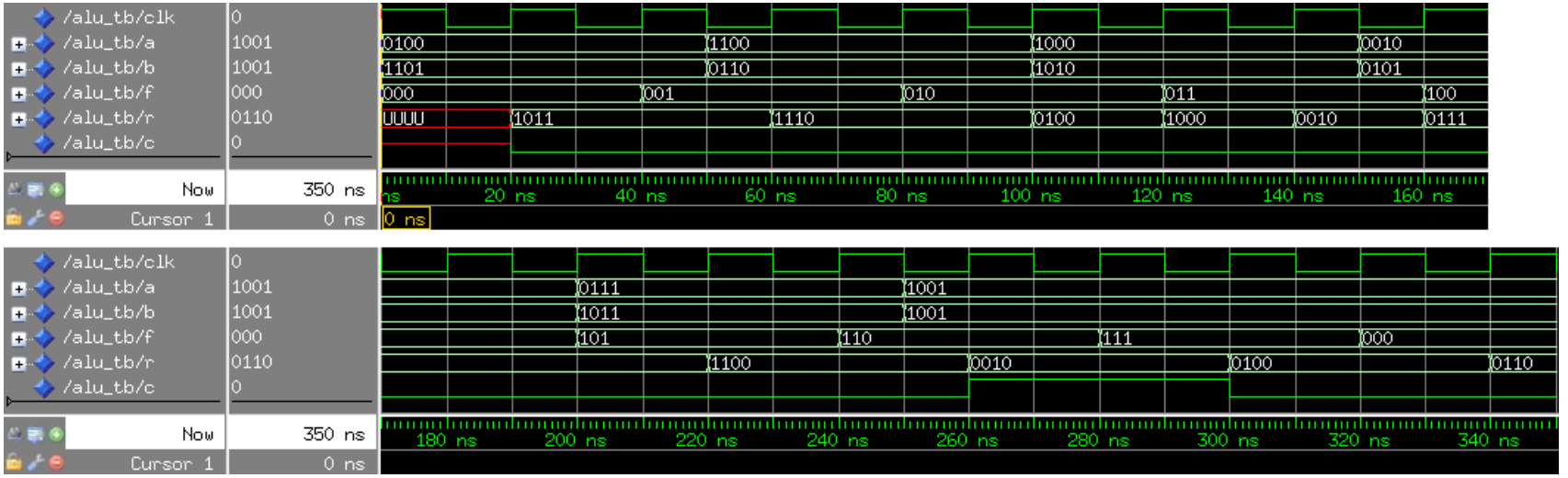
```
lbl2: process (a, b, f) is

variable m  : std_logic_vector (3 downto 0);

begin

  case f is

        when "000" =>
          d <= '0';
          z <= not a;

        when "001" =>
          d <= '0';
          z <= a or b;

        when "010" =>
          d <= '0';
          z <= a and b;

        when "011" =>
          d <= '0';
          z <= a xor b;

        when "100" =>
          m := a + b;
          if ((a(3) = b(3)) and (a(3) = m(3))) or (a(3) = not b(3)) then
                d <= '0';
          else
                d <= a(3);
          end if;
          z <= m;

        when "101" =>
          m := a - b;
          if ((a(3) = not b(3)) and (a(3) = m(3))) or (a(3) = b(3)) then
                d <= '0';
          else
                d <= a(3);
          end if;
          z <= m;
```

```
            when "110" =>
              d    <= a(3);
              z(3) <= a(2);
              z(2) <= a(1);
              z(1) <= a(0);
              z(0) <= '0';

            when "111" =>
              d    <= '0';
              z(3) <= '0';
              z(2) <= a(3);
              z(1) <= a(2);
              z(0) <= a(1);

            when others =>
              d <= d;
              z <= z;

    end case;

end process;

end architecture behaviour;
```

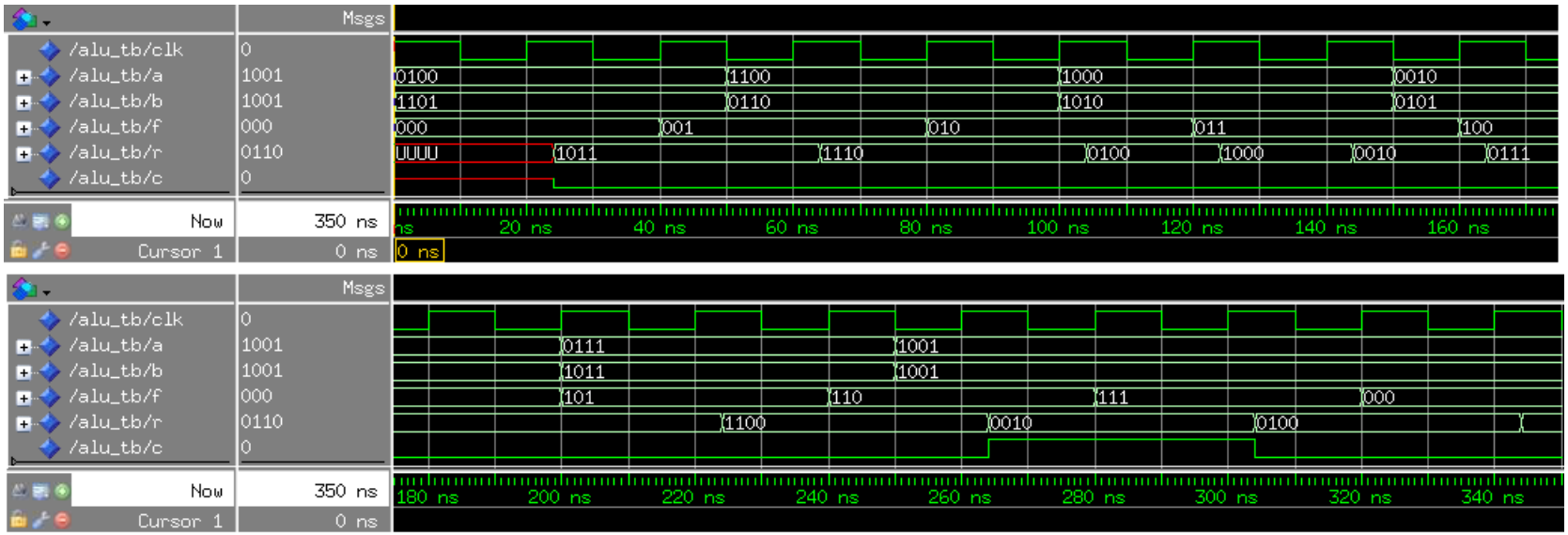
7 Appendix B: Testbench en simulatieresultaten

```
library ieee;
use ieee.std_logic_1164.all;

entity alu_tb is
end entity alu_tb;
```

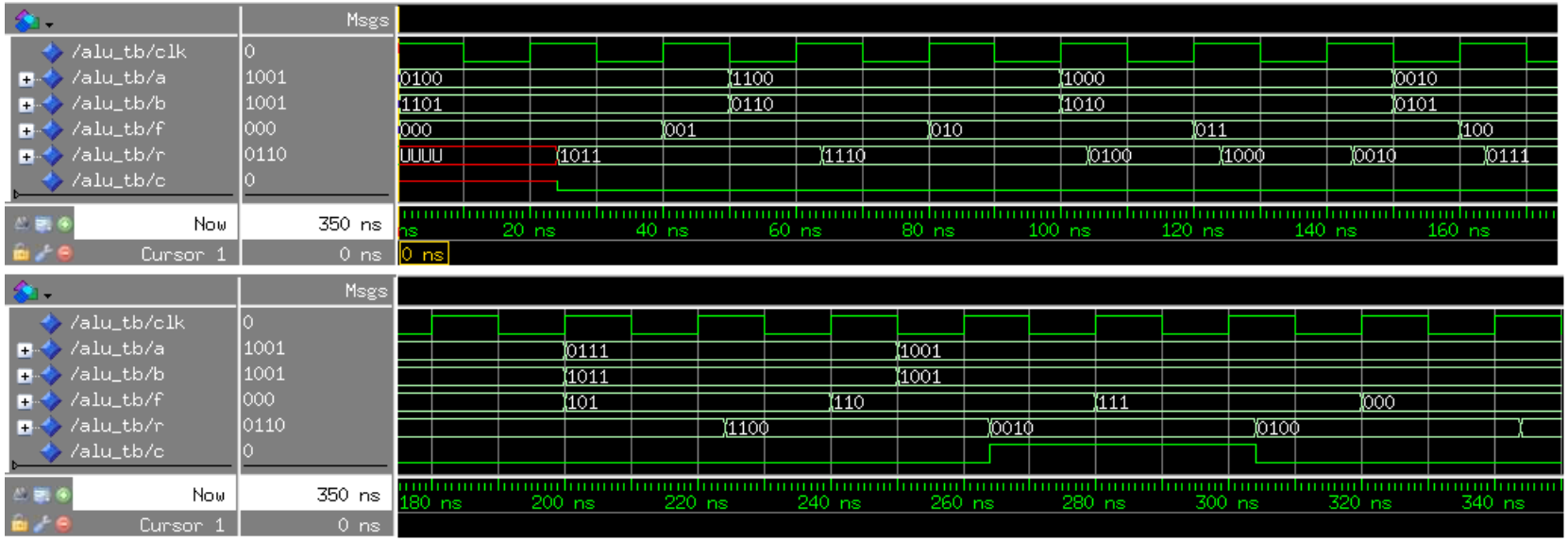
```
library ieee;
use ieee.std_logic_1164.all;

architecture behaviour of alu_tb is
  component alu
    port (clk : in  std_logic;
          a   : in  std_logic_vector(3 downto 0);
          b   : in  std_logic_vector(3 downto 0);
          f   : in  std_logic_vector(2 downto 0);
          r   : out std_logic_vector(3 downto 0);
          c   : out std_logic);
  end component alu;

signal clk : std_logic;
signal a   : std_logic_vector(3 downto 0);
signal b   : std_logic_vector(3 downto 0);
signal f   : std_logic_vector(2 downto 0);
signal r   : std_logic_vector(3 downto 0);
signal c   : std_logic;

begin

clk <=  '1' after  0 ns,
        '0' after 10 ns when clk /= '0' else '1' after 10 ns;
a   <=  "0100" after   0 ns, "1100" after  50 ns, "1000" after 100 ns,
        "0010" after 150 ns, "0111" after 200 ns, "1001" after 250 ns;
b   <=  "1101" after   0 ns, "0110" after  50 ns, "1010" after 100 ns,
        "0101" after 150 ns, "1011" after 200 ns, "1001" after 250 ns;
f   <=  "000"  after   0 ns, "001"  after  40 ns, "010"  after  80 ns,
        "011"  after 120 ns, "100"  after 160 ns, "101"  after 200 ns,
        "110"  after 240 ns, "111"  after 280 ns, "000"  after 320 ns;

lbl1: alu port map (clk, a, b, f, r, c);

end architecture behaviour;
```



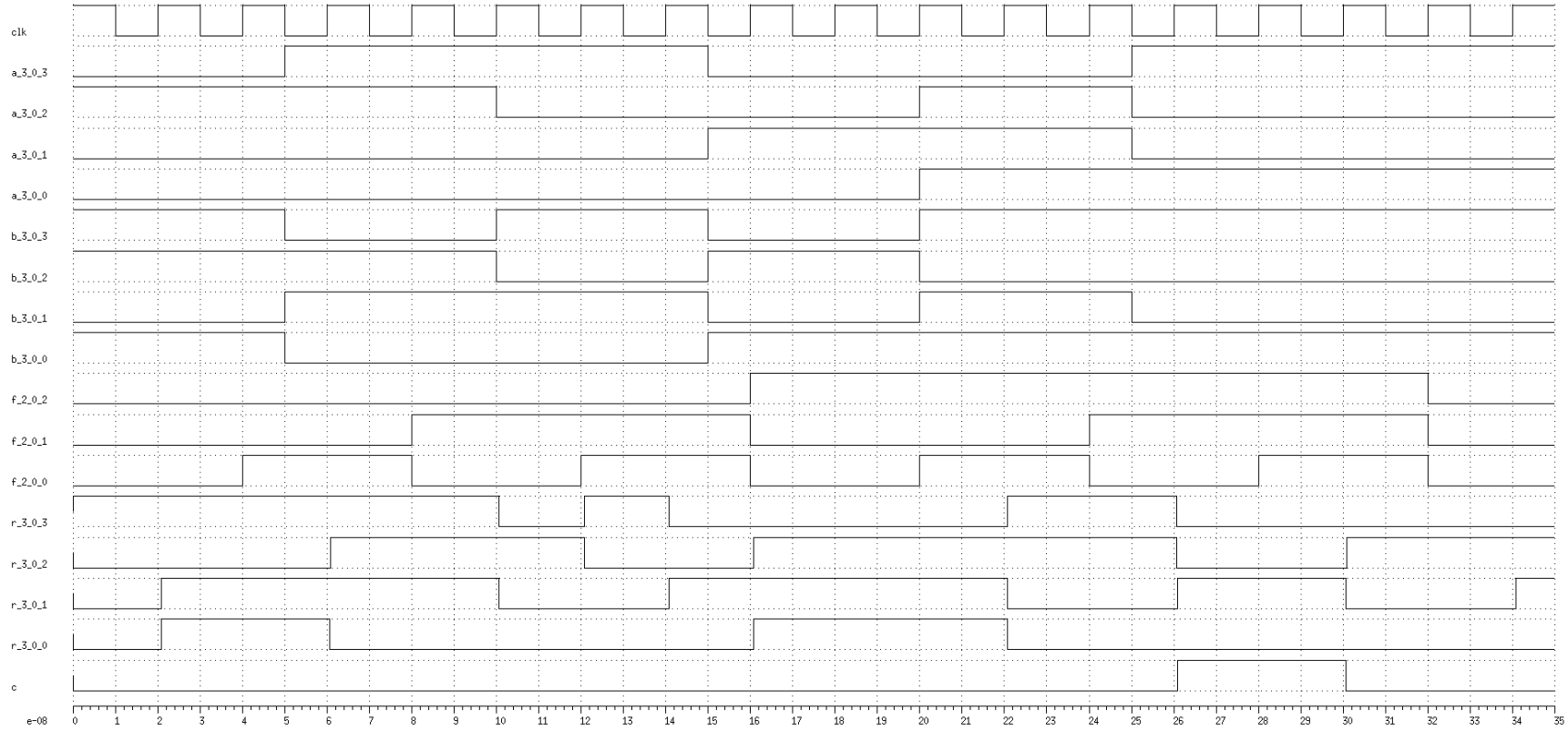
Figuur 5: Simulatieresultaat behavioural VHDL beschrijving



Figuur 6: Simulatieresultaat gesynthetiseerde VHDL beschrijving



Figuur 7: Simulatieresultaat gextraheerde VHDL beschrijving



Figuur 8: Switch-level simulatieresultaat