Tutorial

Quartus II

State machine editor

&

State machine wizard

29 april 2014
Pieter van der Star

Inhoudsopgave

State machine editor

Introductie

In deze tutorial gaan we stap voor stap een statemachine beschrijven. Dit gaan we doen met de state machine editor van Quartus. De statemachine die we gaan beschrijven is de volgende:

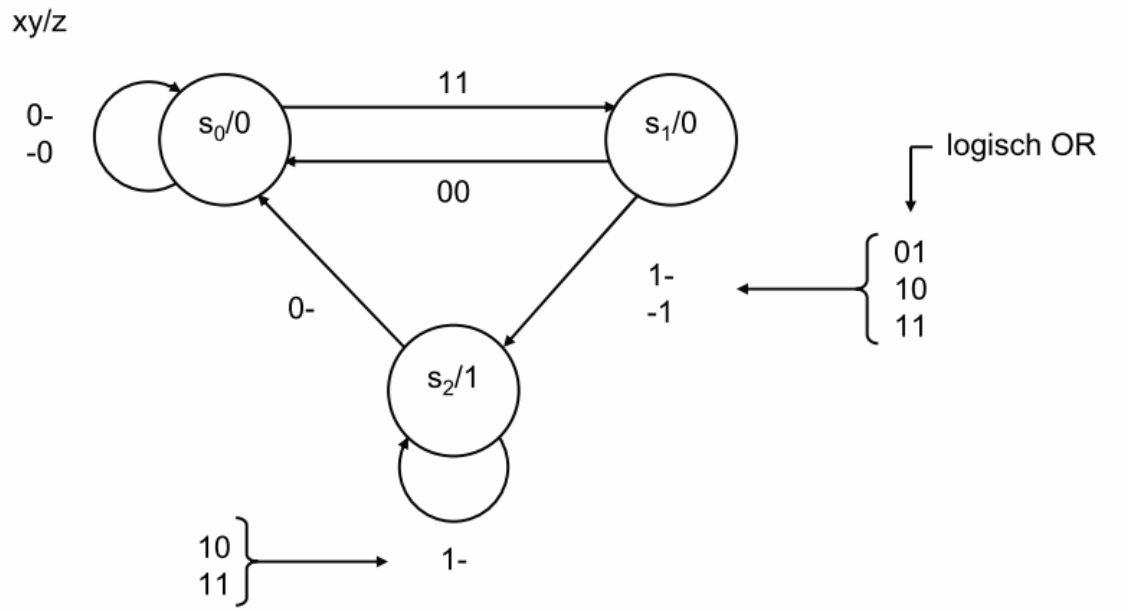


Fig. 1 Moore machine

De kennis die vooraf bekend wordt geacht:

- De taal VHDL
- Aanmaken project in Quartus
- Werking state machine

Aanmaken Project

Als eerste maken we een nieuw project aan genaamd tut_smf.

Aanmaken file

Dan gaan we een state machine file aanmaken. Dit doen we door te klikken op file>New.
Dan krijgen we een keuzescherm waar we kiezen voor "State machine file".
Deze file wordt nu geopend.

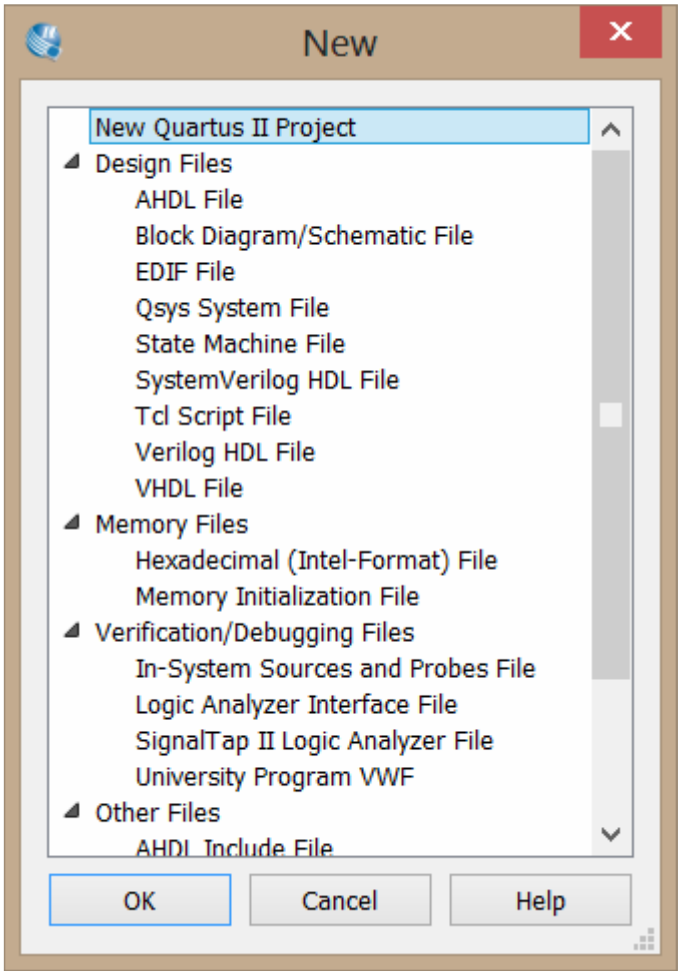


Fig. 2 Open nieuw state machine file

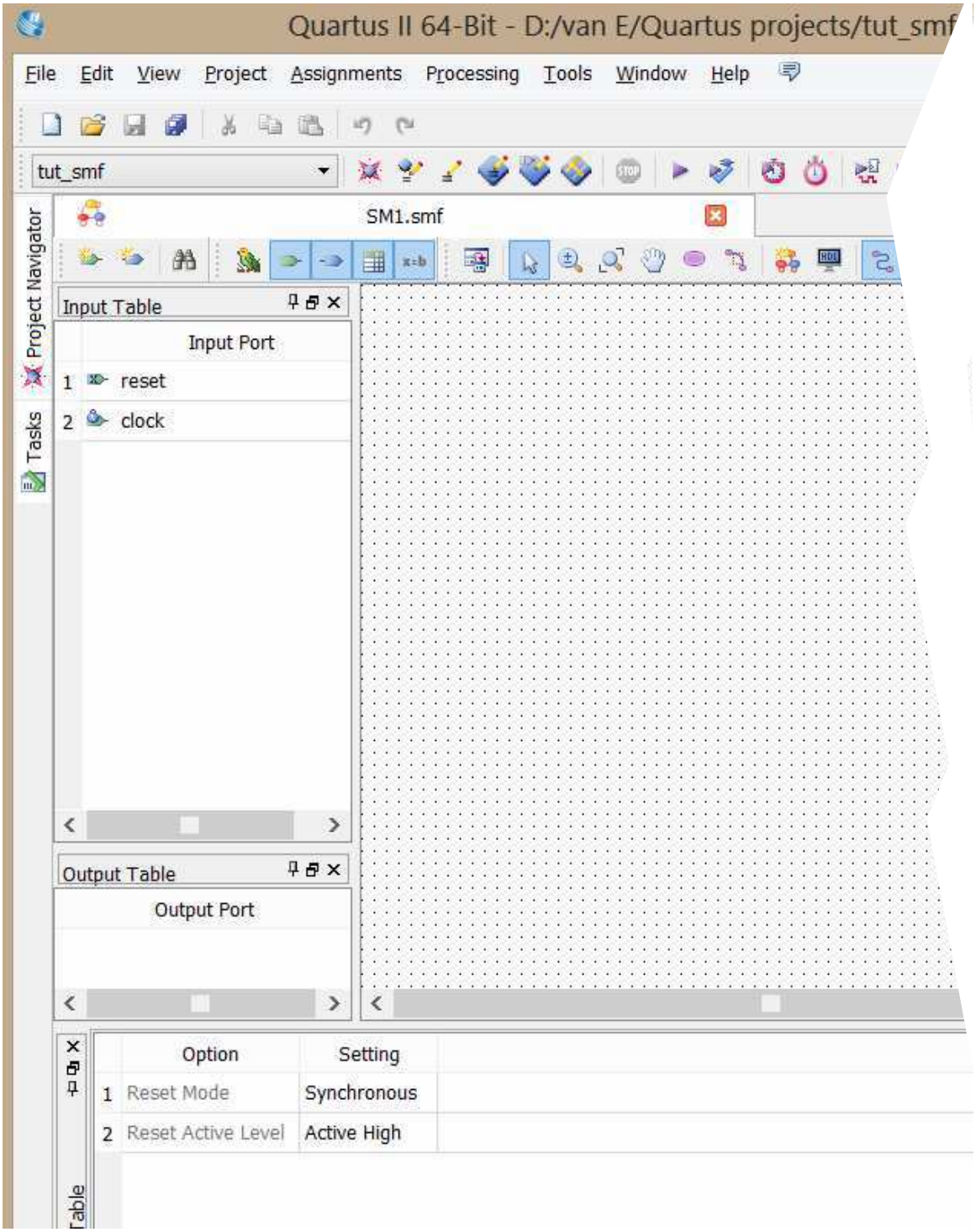


Fig. 3 Quartus na aanmaken state machine file

Voordat we verder gaan slaan we dit bestand eerst op. Dit bestand noemen we "moore_machine". In de bestandsnaam mogen alleen de tekens uit het alfabet, cijfers en het lage streepje mogen gebruikt worden. Andere tekens worden wel toegestaan, maar dit geeft later in het proces problemen[1]. Merk op dat de extensie nu .smf is en geen .vhd of .vhdl.

De gegeven naam is ook meteen de naam van de entity en de naam van het HDL bestand.

[1] We krijgen dan bij het omzetten naar HDL de volgende foutmelding: Error (154013): Component <bestandsnaam> contains an illegal name character combination. Deze foutmelding krijg je ook bij verkeerde state namen. Het kan ook beide zijn.

Gereedschappen

Boven het veld hebben we een gereedschapsbalk (Fig. 4). De knoppen zijn op volgorde van links naar rechts:

- Nieuwe input
- Nieuwe output
- Zoeken en vervangen
- "Bird's eye view" (overzichtsvenster)
- Inputlijst
- Outputlijst
- Statelijst
- Verberg of toon vergelijkingen
- Undock venster
- Selecteer-gereedschap
- Zoomgereedschap
- Vergrootglas
- Schuif zichtbaar veld
- Nieuwe state
- Nieuwe overgang
- State machine wizard
- Maak HDL beschrijving
- Zet elastische pijlen aan of uit

Fig. 4 Gereedschapsbalk state machines

Definiëren in- en uitgangen

We beginnen met het aanmaken van de in- en uitgangen. In dit geval hebben we twee ingangen (x en y) en één uitgang (z). Het aanmaken van ingangen doen we met het gereedschap new input . In de inputlijst verschijnt, onder de standaard inputs (clk en reset) een nieuwe, gele, input. Door dubbel op de naam input 1 te klikken kunnen we deze veranderen. Als we dit doen komt er een waarschuwing. Hierin word gewaarschuwd dat het aanpassen van de inputnamen geen aanpassing van de overgangsbeschrijvingen oplevert. Hoe dit zit komen we later op terug. Deze waarschuwing kunnen we nu negeren door op ok te klikken.

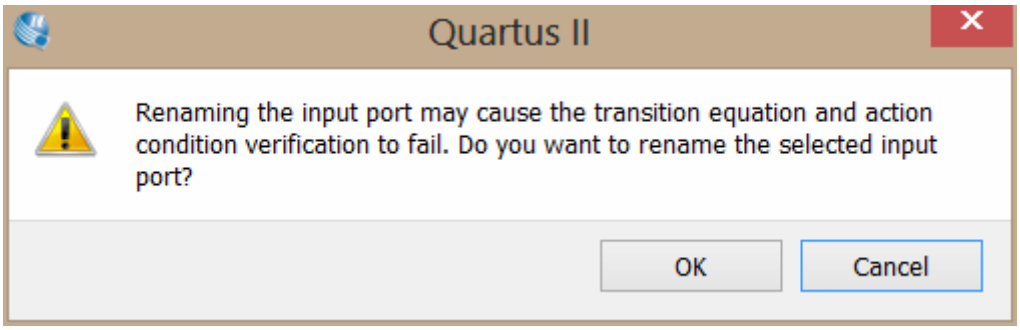


Fig. 5 Waarschuwing bij naamsverandering input

De naam kan nu veranderd worden. In ons geval is dat x.
We maken nu nog een input aan en die hernoemen we naar y.

Als alle ingangen aangemaakt zijn kunnen we de uitgangen aanmaken. Dit doen we op precies dezelfde manier, maar dan klikken we op “New output” i.p.v. op “New input”.

Ook het hernoemen gaat op dezelfde manier.
De uitgang noemen we, net als in de tekening, ‘z’.

Alle poorten; in- en uitgangen, zijn nu aangemaakt en gedefinieerd.
Het scherm ziet er nu zo uit:

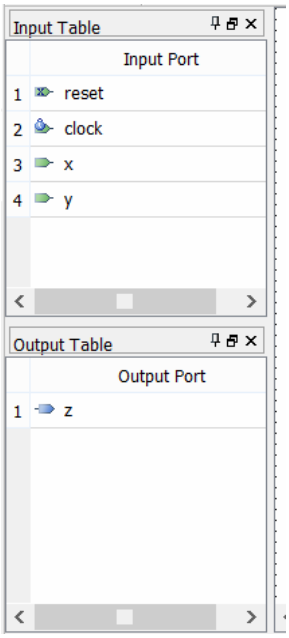


Fig. 6 In- en output table na het aanmaken van de in- en uitgangen

Aanmaken states en overgangen

Nu gaan we states aanmaken. Dit doen we door op de state tool te klikken.
Nu hangt er een doorzichtig bolletje met de naam state1 aan de muis. Door op het veld te klikken wordt deze state aangemaakt. Deze wordt nu ook ondoorzichtig rood en er komt een nieuwe, doorzichtige, state (state2) aan de muis te hangen.

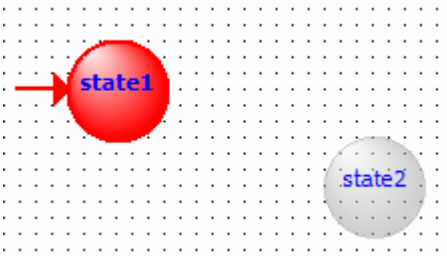


Fig. 7 Gedefinieerde state1 en nog te maken state2

Aan de nieuwe state zit alvast een pijltje. Dit betekent dat er bij een reset naar deze state gesprongen wordt. We zetten er nu op dezelfde manier een nieuwe state bij. Tussen deze state trekken we een lijn.

De lijn trekken we door de line tool te selecteren. We klikken nu op state1 en slepen naar state2. Nu komt er een pijl te staan tussen deze states. Dit doen we nog een keer de andere kant op.

Tot slot moet er nog een pijl van state1 naar zichzelf gezet worden. Dit doen we door op state1 te klikken. Het veld ziet er nu als volgt uit:

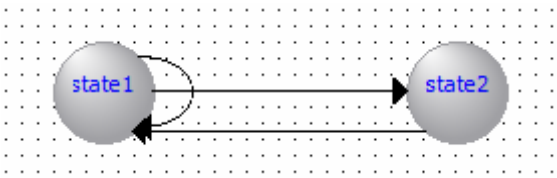


Fig. 8 State machine file na aanmaken twee states met overgangen

Nu gaan we de overgangen toevoegen. Door dubbel op een pijl te klikken kan de overgangsconditie gegeven worden. Dit gebeurt in Verilog met de bitwise operators en de inputs. De bitwise operators zijn:

- '&' And
- '|' Or
- '^' Exor
- '~' Not
- '~&' Nand
- '~|' Nor

De Exnor kan op twee manieren:

- '^~' Exnor
- '~^' Exnor

Er zijn nog een aantal andere mogelijkheden voor overgangscondities. Dit zijn vergelijkingen van een input met een andere input. Een input vergelijken met een constante is niet mogelijk. De vergelijkingsoperatoren zijn:

- '==' Gelijk aan
- '!=' Niet gelijk aan
- '<=' Minder dan
- '<' Minder
- '>=' Groter

Dus voor de pijl van state1 naar state2, bij input 11, vullen we "x&y" in. Merk op dat er geen spaties in zitten. De andere kant op, dat is bij 00, dan moeten we de allebei de ingangen inverteren. Dit levert dus "~x&~y'. Voor de "in de state blijf pijl" mogen we dus alle andere combinaties invoeren. Hier is een handigheidje voor bedacht. Hiervoor wordt de VHDL term "others" gebruikt[2]. We kunnen hier natuurlijk ook "~x|~y" of "x|y"[3] invullen.

Je moet voorkomen dat er meerdere overgangen waar kunnen zijn. Quartus geeft dit netjes voor je aan als je er VHDL van wilt maken, maar het is handig dit zelf in de gaten te houden. Als er ingangsmogelijkheden zijn die meerdere stateovergangen waar maken, dan moet je dit aanpassen.[4]

Mocht je de ingangsnamen willen wijzigen, dan moet je ook alle state-overgangen aanpassen. Deze veranderen niet mee met de ingang.

LET OP!
Als een signaal geen invloed heeft op de state-overgang moet deze niet in de overgangscriteria opgenomen worden, óf het signaal moet voor alle waarden ('1' en '0') in de beschrijving worden opgenomen.

Vul nu zelf het schema aan.

[2] Als je altijd naar een volgende state wilt, maak je maar één overgang aan met "others". Dit levert wel de volgende waarschuwing: Warning (154029): State state3 contains only Others transition

[3] Er zijn nog meer combinaties mogelijk, maar deze laten we buiten beschouwing.

[4] we daar ~(x&y) invullen gaat het fout. We krijgen bij het omzetten naar een HDL dan krijgen we de volgende foutmelding: Error (154039): State s1 contains multiple transitions, but more than one transition equation are TRUE when ~x & y | x & ~y. As a result, the next state cannot be decided.

Als je een pijl geselecteerd heb kun je deze vervormen door de vierkantjes die er bij staan te verslepen. Ook kan je het begin en het einde veranderen. Dit doe je door op de desbetreffende pijl te klikken en dan de vierkantjes aan het begin en het einde te slepen naar de nieuwe positie. Dit kan alleen van state naar state en niet van een plaats op het bolletje naar een andere plaats op hetzelfde bolletje. Als je dit wel wil, dan moet je de functie rubberbanding (elastieken) uitzetten.

Mocht je de "in de state blijf"pijl willen aanpassen, moet je het einde of het begin even naar een andere state slepen. Dan kan je het andere eind op zijn plaats zetten. Vervolgens kan het tijdelijk verplaatste eind weer teruggezet worden. Probeer je toch een begin of einde naar dezelfde state te verslepen krijg je de volgende foutmelding (Fig. 9).

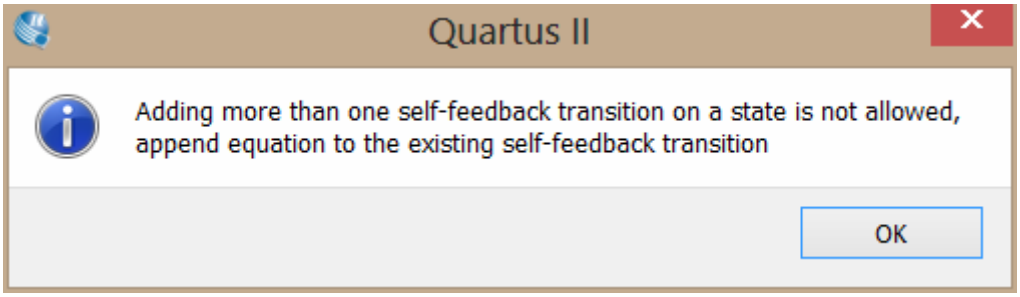


Fig. 9 Foutmelding bij verslepen "in state blijf"-pijl binnen state

Als voorbeeld is de "in de state blijf"-pijl naar onderen verplaatst. Het totaalplaatje ziet er nu uit zoals in Fig. 10.

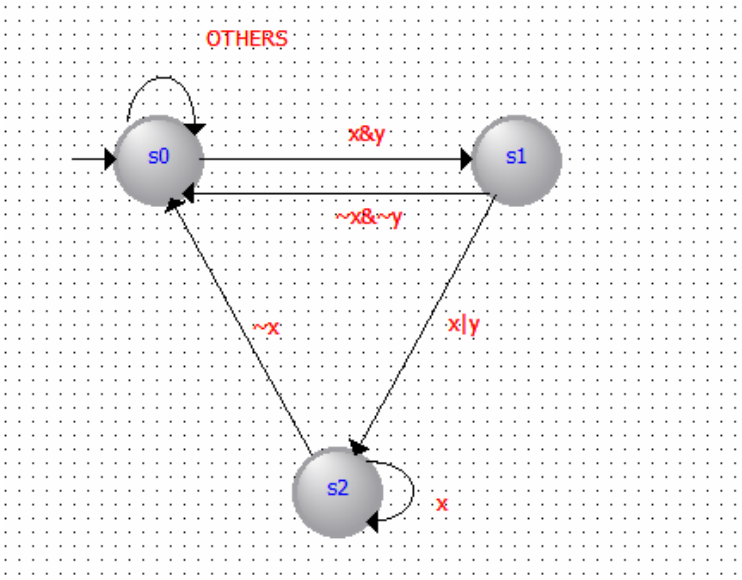


Fig. 10 State machine file na aanmaken states en overgangen

Nu hebben we de states en de inputs gemaakt. Nu moet alleen de output nog gedaan worden. Als we terugkijken naar Fig. 1 Moore machine, zien we dat alleen in state 2 er een 1 op de uitgang komt en dat er in alle andere gevallen het een nul is.

Definiëren uitgangswaarden

We beginnen bij state 1. We klikken dubbel op de state (rechtermuisknop en dan kiezen voor properties kan ook). Als eerste krijgen we een tab met de naam van de state (Fig. 11). Nu we daar toch zijn kunnen we deze naam meteen veranderen in de naam van de opgave (s0). Dit doen we door de huidige naam te selecteren en dan de nieuwe naam in te typen. Als we op apply klikken zien we de naam ook veranderen. Probeer, als dat van toepassing is, duidelijke, logische state-namen te gebruiken. Zo kan je, als je het later terug wil kijken sneller zien wat er gebeurt. Alle VHDL sleutelwoorden (o.a. "begin" en "entity") mogen niet gebruikt worden. Signaaltypen (o.a. std_logic) mogen ook niet. De woorden die je niet mag gebruiken zijn de woorden die Quartus in VHDL code blauw of lichtrood kleurt.

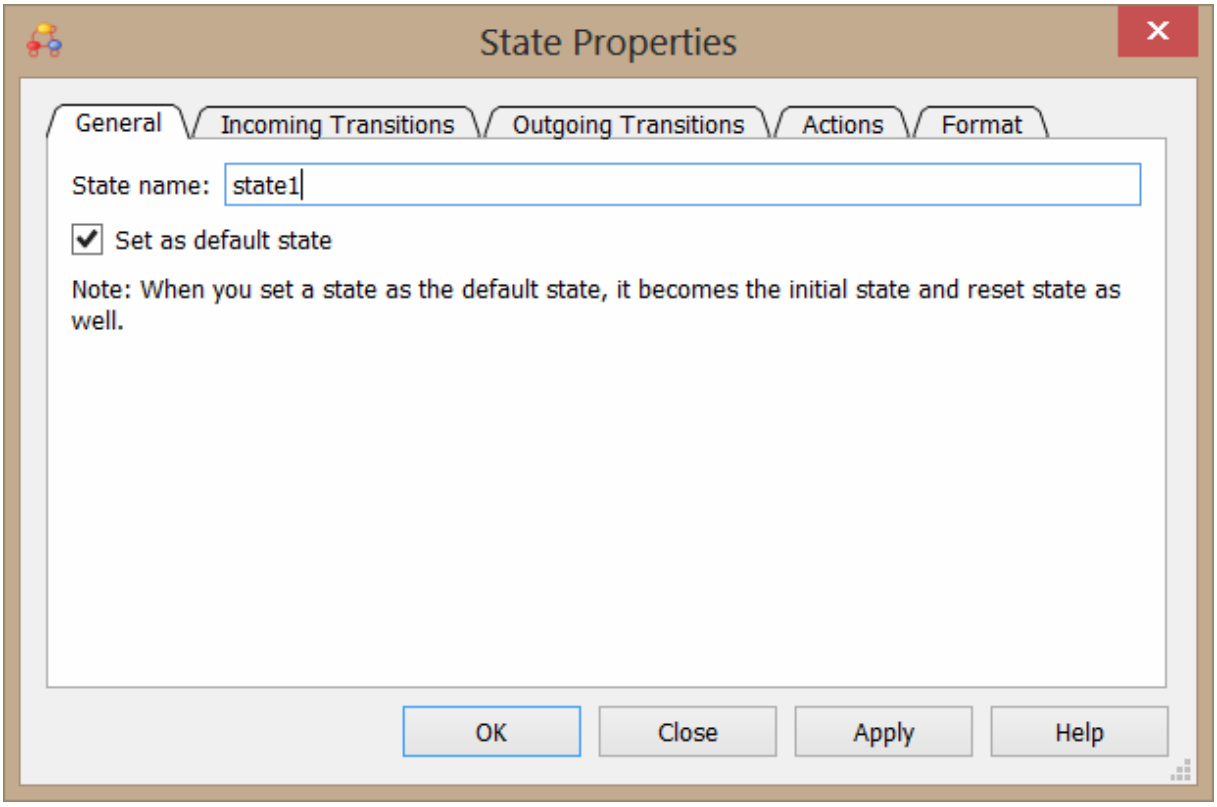


Fig. 11 State properties

Nu gaan we de uitgang definiëren. Klik op het tabblad "Actions". Nu zie je een tabel met "Output port", "Output value" en "Additional Conditions".
Onder "Output port" staat "< New >" hier klikken we dubbel op (Fig. 12). Dan kiezen de uitgang die we willen hebben. In dit geval is het er maar één dus valt er weinig te kiezen. Om de uitgangswaarde te definiëren klikken we dubbel op het vakje naast de gekozen uitgang en onder "Output value". Dan vullen we daar een 0 in en klikken op apply. We hoeven bij outgoing transitions niets in te vullen. Hier komen we later op terug als we een Mealy machine gaan maken.

Nu kunnen we hetzelfde doen met state 2 en 3. Let op dat state 3 een andere uitgangswaarde heeft. Het properties scherm opent nu eerst met het tabblad "outgoing transitions" en niet meer met "General". Dit komt omdat "outgoing transitions" als laatste bekeken is. Om de naam te veranderen moeten we dus weer terug naar "General".

De uitgangen worden altijd als integer gegeven. Hoewel het maar één bit betreft. Als er een waarde wordt toegekend die hoger is dan één bit krijgen we de volgende foutmelding: "Syntax error: Illegal operands size found in 1'b101".

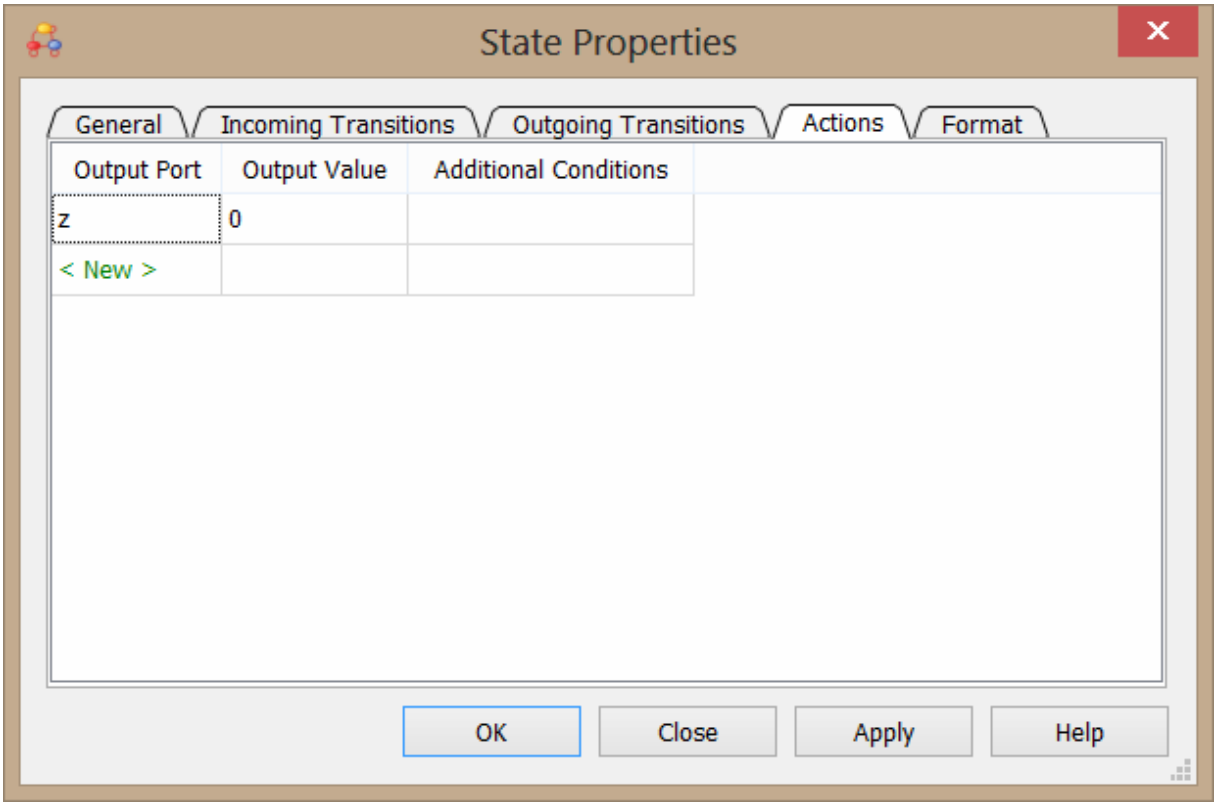


Fig. 12 State properties na definëren uitgang (z)

Machine-eigenschappen

Nu zijn we bijna klaar. We moeten nog een paar zaken instellen. Dit doen we door onder het veld de "State table" aan te vullen.

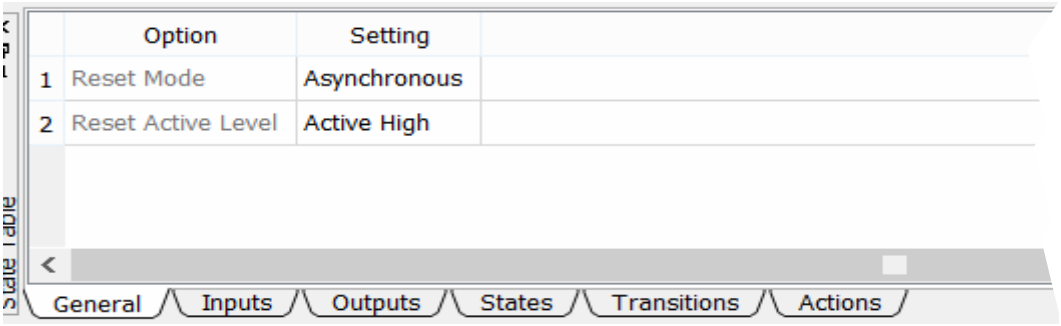


Fig. 13 Tabblad "General" met de juiste opties

Onder het tabblad "General" kan je kiezen of je een synchrone of asynchrone reset wil, en of deze reset active-high (1=reset) of active-low (0=reset) moet zijn. Kiezen doe je door wederom dubbel te klikken en dan uit het menu de gewenste optie te kiezen. Wij kiezen hier voor asynchroon en active-high.
Bij inputs kunnen we alle inputs zien. Hier kan je, afgezien van de namen van de poorten, niets veranderen.

Bij output moeten we wel een paar keuzes maken. Hier kunnen we per output kiezen of het een "registered output" moet zijn of niet. Ook kunnen we ervoor kiezen om de nieuwe waarde pas een klokflank later naar buiten te brengen. Dit doen we allebei niet.

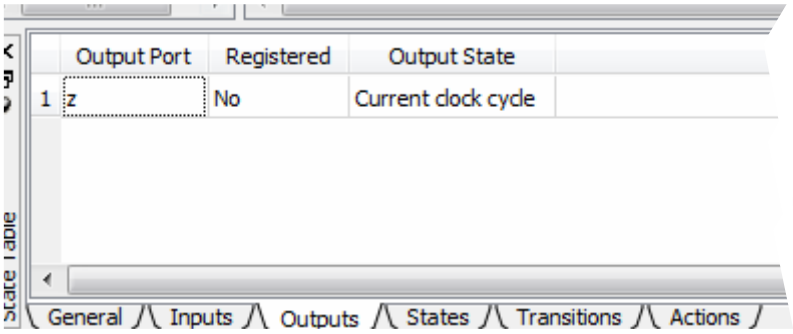


Fig. 14 Tabblad "Outputs" met de juiste opties

Het volgende tabblad is "States". Hier kunnen we de namen van de states veranderen en kiezen naar welke state er gesprongen moet worden bij een reset. Dit doen we door in de kolom reset "yes" of "no" te kiezen. Je kunt uiteraard maar bij één state de optie "yes" kiezen. De laatst gekozen "yes" overschrijft de andere. Wij laten hier de reset bij s0 staan.

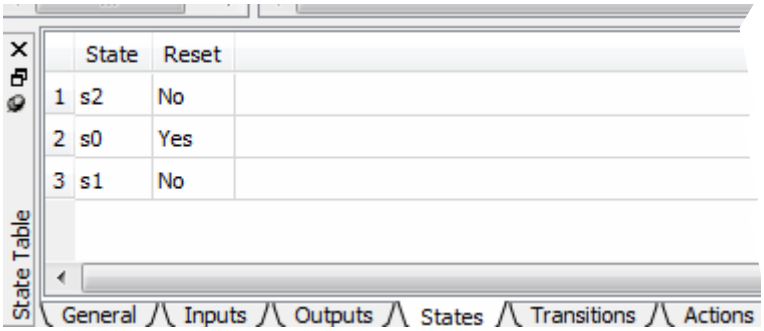


Fig. 15 Tabblad "States" met de juiste opties

Bij "Transitions kan je alle overgangen bekijken. En bij "Actions" zie je alle uitgangen met de uitgangswaarde bij elke state. Uiteraard kun je dit alles ook hier wijzigen. Dit is voor ons niet nodig aangezien we dit allemaal al goed gezet hebben.

Nu zijn we klaar. We slaan het bestand op voordat we verder gaan.

Maken HDL beschrijving

We hebben nu een heel mooie state machine op het scherm staan, maar we willen er uiteindelijk VHDL van hebben. Dit kan met de knop "Maak HDL beschrijving"  . Deze is te vinden in de gereedschapsbalk (Fig. 4). Als we hier op klikken komt het venster "Generate HDL File" (hieronder)naar voren. We kiezen hier voor de optie VHDL en klikken op OK.

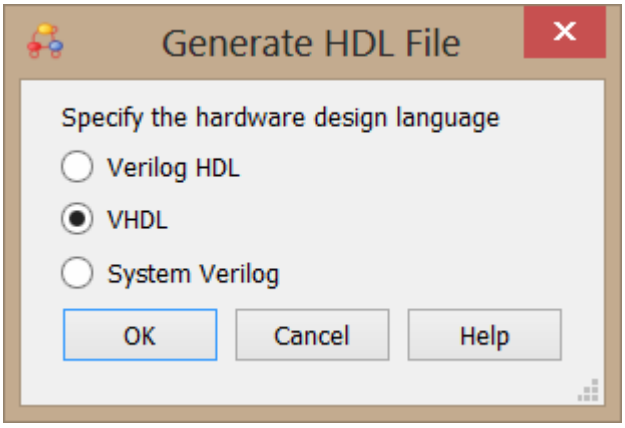


Fig. 16 HDL taalkeuze

We zien nu een nieuw VHDL bestand met de VHDL beschrijving van de state machine. Deze is wel opgeslagen in dezelfde map als de state machine file, maar is nog niet toegevoegd aan het project. Dit doe je door in de menubalk te kiezen voor project>add current file to project.

Als je alles goed gedaan hebt komt er de volgende VHDL code uit[5]:

[5] De gegenereerde code kan afwijken door een andere volgorde van het tekenen van de states.

```
-- Copyright (C) 1991-2013 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.

-- Generated by Quartus II Version 11.1 Build 259 01/25/2012 Service Pack 2
SJ Web Edition
-- Created on Fri May 25 21:33:00 2012

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY moore_machine IS
    PORT (
        reset : IN STD_LOGIC := '0';
        clock : IN STD_LOGIC;
        x : IN STD_LOGIC := '0';
        y : IN STD_LOGIC := '0';
        z : OUT STD_LOGIC
    );
END moore_machine;

ARCHITECTURE BEHAVIOR OF moore_machine IS
    TYPE type_fstate IS (s2,s0,s1);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reset,reg_fstate)
    BEGIN
        IF (reset='1') THEN
            fstate <= s0;
        ELSIF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;

    PROCESS (fstate,x,y)
    BEGIN
        z <= '0';
            CASE fstate IS
                WHEN s2 =>
                    IF (NOT((x = '1'))) THEN
                        reg_fstate <= s0;
                    ELSIF ((x = '1')) THEN
                        reg_fstate <= s2;
                    -- Inserting 'else' block to prevent latch inference[6]
                    ELSE
                        reg_fstate <= s2;
                    END IF;

                    z <= '1';
                WHEN s0 =>
                    IF (((x = '1') AND (y = '1'))) THEN
                        reg_fstate <= s1;
                    ELSE
                        reg_fstate <= s0;
                    END IF;
```

[6] Als er geen voorschrift gegeven is dan wordt er altijd een 'else' ingevoegd. Dit voorkomt latches om de next state te kunnen onthouden.

```
                    z <= '0';
                WHEN s1 =>
                    IF ((NOT((x = '1')) AND NOT((y = '1')))) THEN
                        reg_fstate <= s0;
                    ELSIF (((x = '1') OR (y = '1'))) THEN
                        reg_fstate <= s2;
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        reg_fstate <= s1;
                    END IF;

                    z <= '0';
                WHEN OTHERS =>
                    z <= 'X';
                    report "Reach undefined state";[7]
            END CASE;
        END IF;
    END PROCESS;
END BEHAVIOR
```

[7] Report geeft de opgegeven string weer in modelsim. In dit geval dus als er geen state bepaald kan worden. Dit is in modelsim te zien in het scherm messages. Ook wordt er in de wave viewer een markering geplaatst.

State machine wizard

Introductie

In deze tutorial gaan we stap voor stap een statemachine beschrijven. Dit gaan we doen met de state machine wizard van Quartus. De state machine die we gaan beschrijven is de volgende:

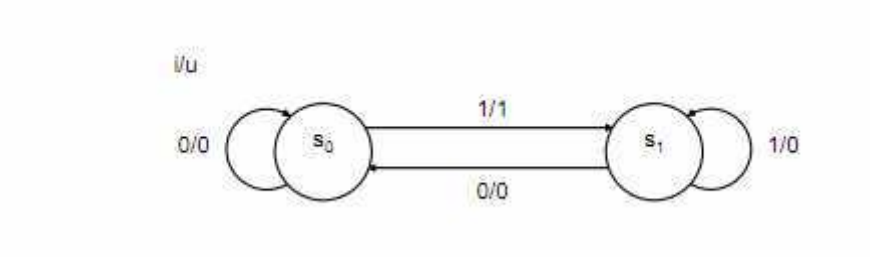


Fig. 17 Mealy machine

Maak een nieuwe state machine file aan en noem deze mealy_machine.

Klik op de state machine wizard knop  om deze te starten.

In het venster dat nu volgt kiezen we voor create new state machine design.

Fig. 18 Startscherm State machine Wizard

Wizard

Dan opent de wizard. We kunnen nu een aantal opties instellen. We kiezen voor een asynchrone, actief-hoge reset. Ook vinken we de optie "Transition to source state if not all conditions are specified" aan. Hiermee word voor elke ingangscombinatie waar geen overgangen voor gespecificeerd zijn automatisch een "blijf in state" overgang aangemaakt.

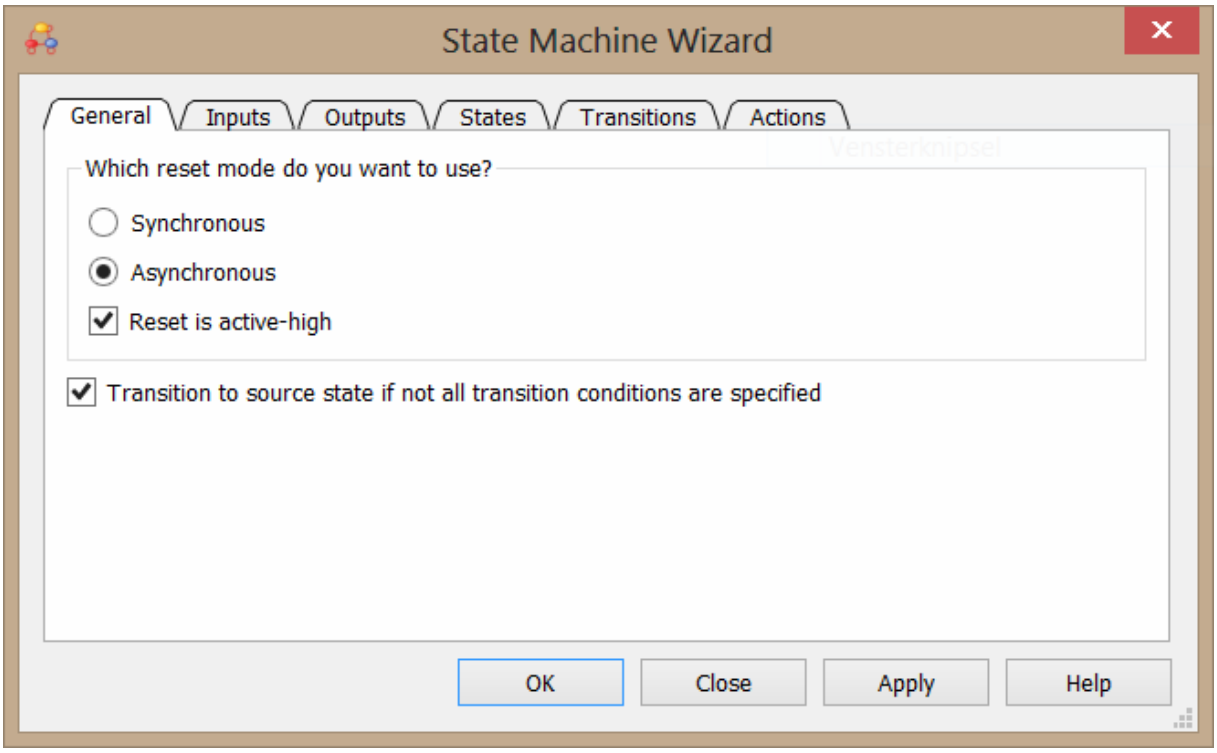


Fig. 19 Tabblad "General" van de State machine wizard

Om verder te gaan klikken we op het tabblad inputs. We zien de klok en de reset al ingevuld staan. Hier willen we één input bij maken. Deze noemen we "i". Dit doen we door te dubbelklikken op "< New >" en dan een i in te vullen. Alle VHDL sleutelwoorden (o.a. "begin" en "entity") mogen niet gebruikt worden. Signaaltypen (o.a. std_logic) mogen ook niet.

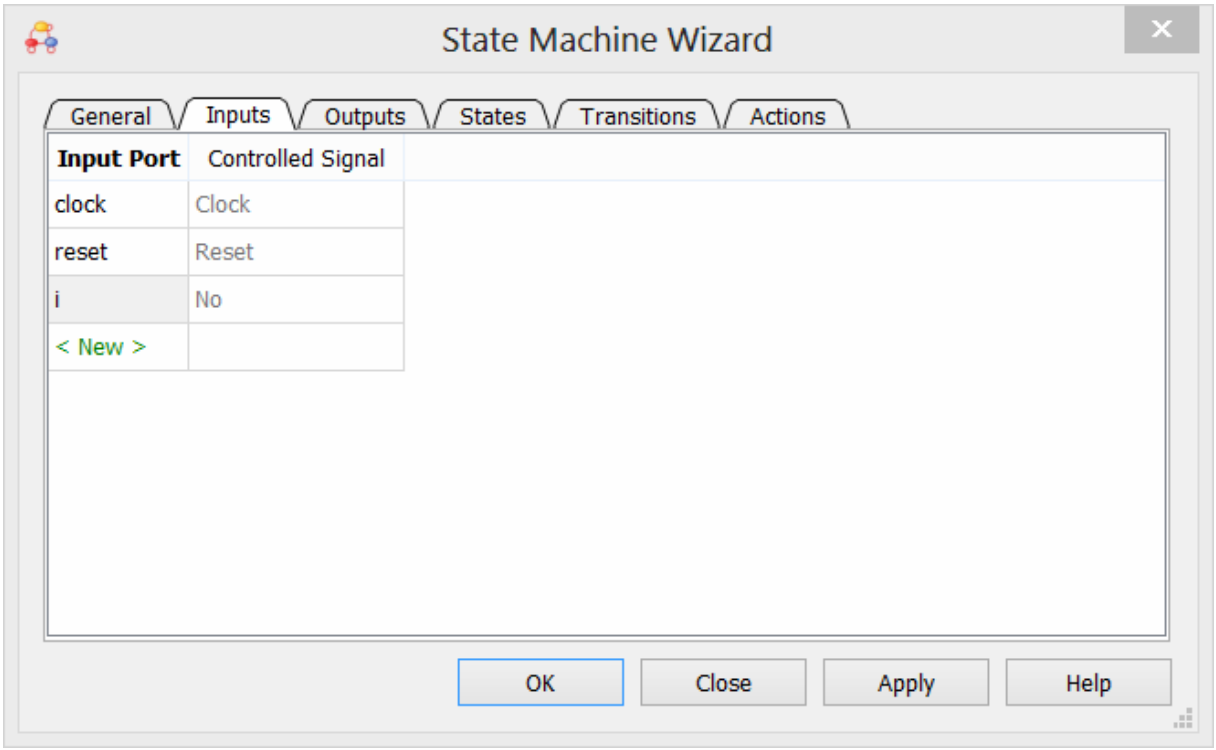


Fig. 20 Tabblad "Inputs" van de State machine wizard

Nu we de ingang gedaan hebben zijn de uitgangen aan de beurt. Om dat te doen klikken we op het tabblad "Outputs". De statemachine die wij gaan maken heeft ook maar één uitgang. Deze uitgang noemen we "u". Deze is niet "registered" en komt op deze klokflank naar buiten[8].

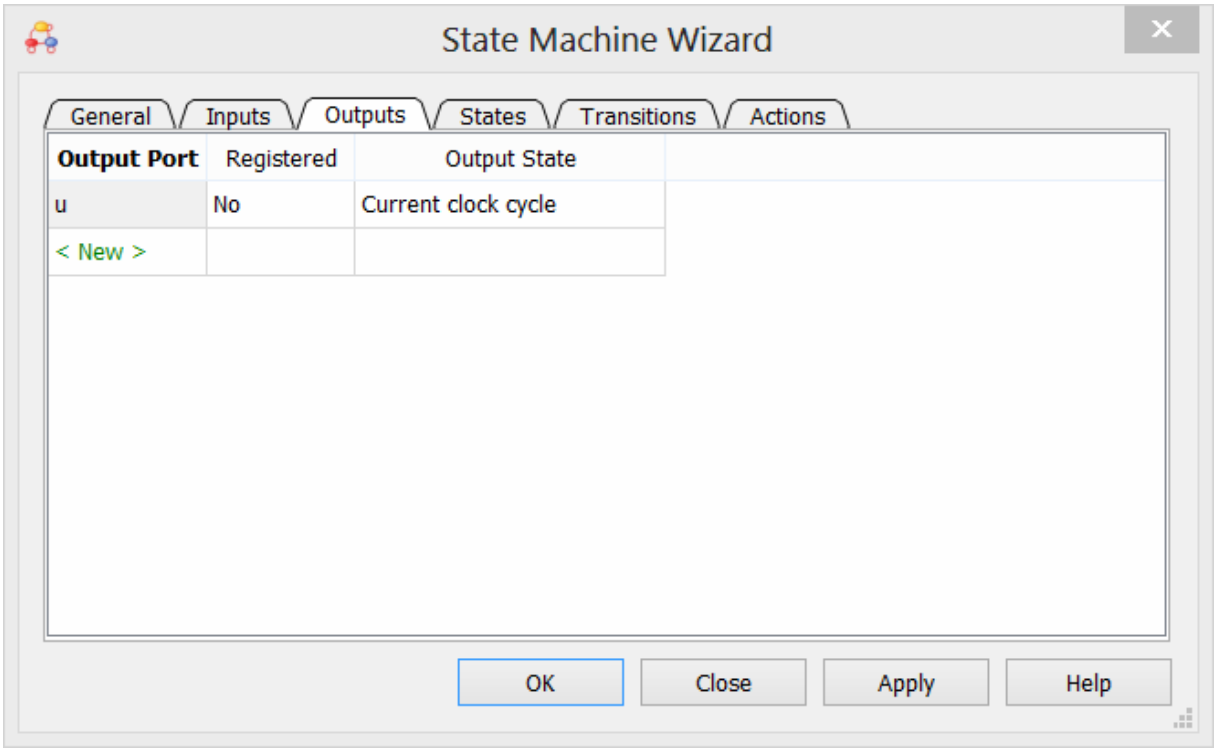


Fig. 21 Tabblad "Outputs" van de State machine wizard

[8] Wat dit precies betekent is te vinden in het bestand Registered output

Nu ook de uitgang aangemaakt is gaan we de states aanmaken. Dit doen we, verrassend genoeg, door op het tabblad "States" te klikken.

De states worden aangemaakt op dezelfde manier als de in- en uitgangen. Onze machine heeft twee states s0 en s1. Deze maken we dan ook aan. Wat onze tekening niet voorschrijft is de state waar bij een reset naar toe gesprongen moet worden. Laten we voor de verandering eens state twee (s1) kiezen.

We moeten nu naast in de kolom reset in de rij van s1 de "no" veranderen in "yes". Dit doen we door te dubbelklikken op de cel, en dan in het menu kiezen voor "yes". Door naast de cel te klikken wordt deze toegewezen. Je ziet nu ook dat de "yes" bij state s0 wordt veranderd in een "no".

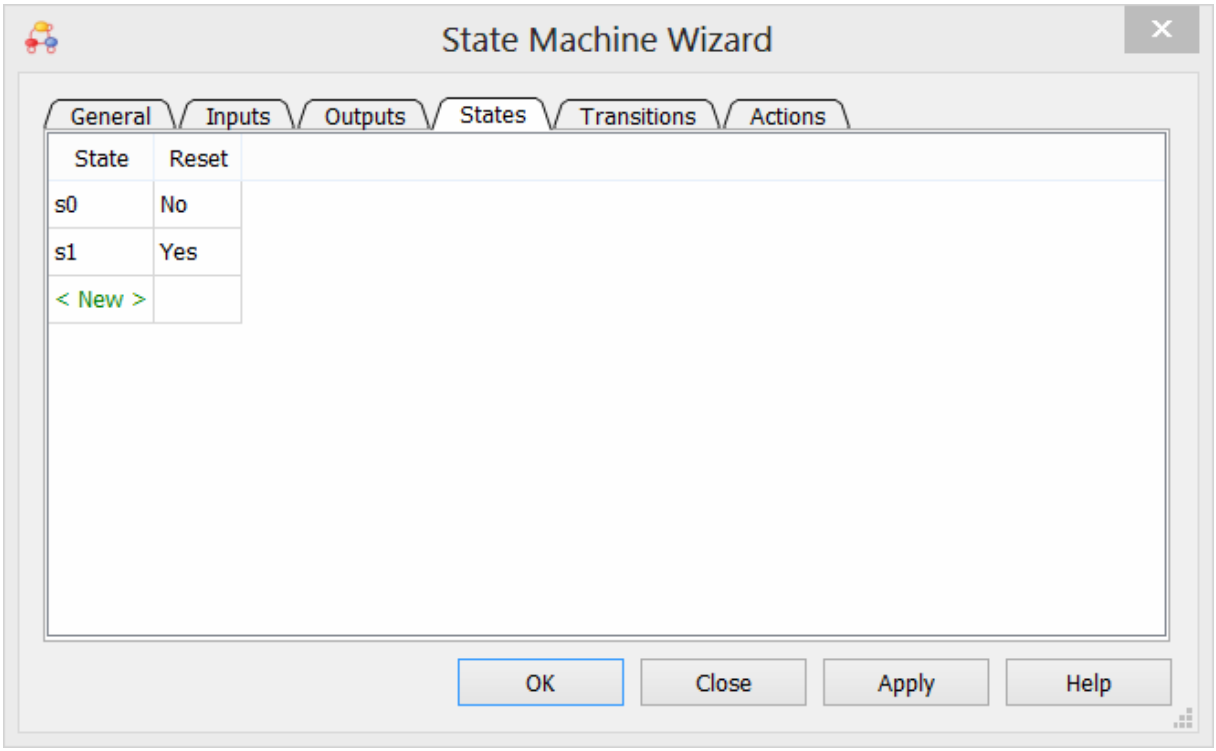


Fig. 22 Tabblad "States" van de State machine wizard

De volgende stap is het invoeren van de overgangen. We klikken daarom op het tabblad "Transitions". Dan gaan we een nieuwe overgang aanmaken. Om dit te doen moeten we weer dubbelklikken op "< New >" en dan kiezen voor s0. De eerste overgang van s0 is die naar s1. We kiezen bij "Destination State" voor s1. Deze overgang wordt gedaan als de ingang een 1 is. Dus vullen we bij de transition, wederom in Verilog de overgangscriteria "i" in. De andere kant op moet ook gebeuren. Alle andere overgangen, dat zijn de "in de state blijf" overgangen, zouden we ook moeten invoeren. Maar omdat we eerder in de tab "General" de optie "Transition to source state if not all conditions are specified" hebben aangevinkt is dit niet nodig.

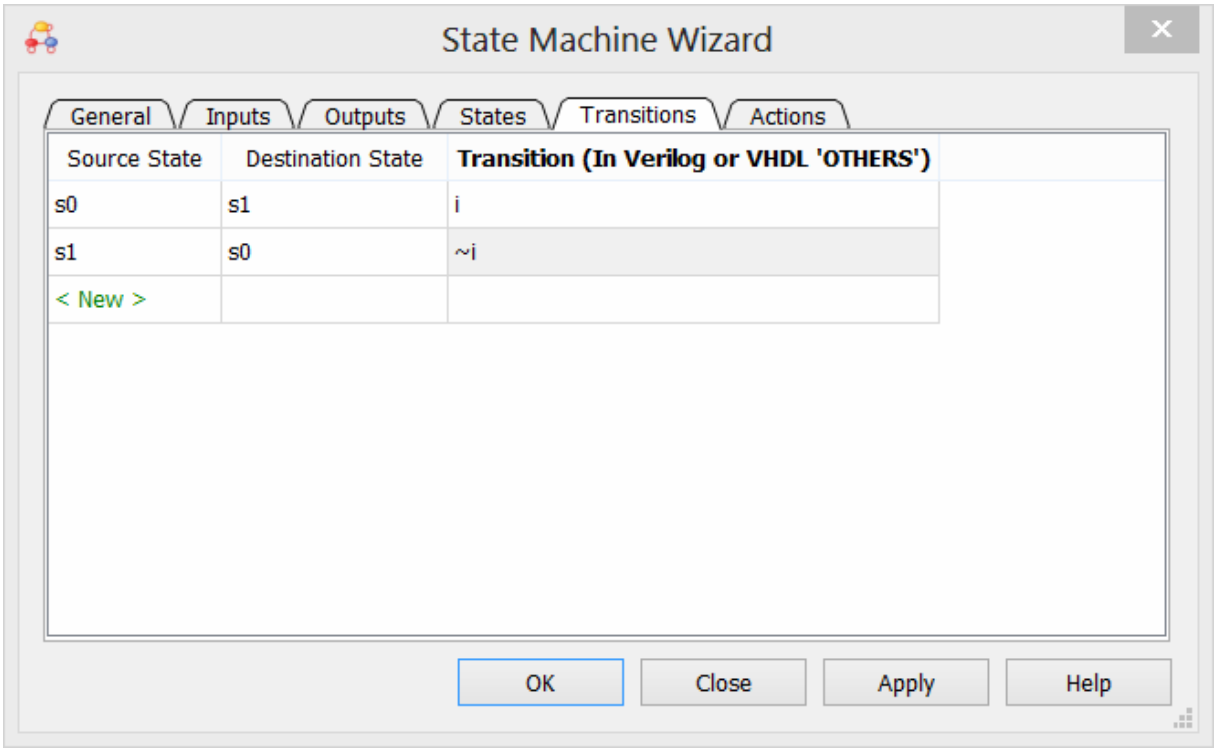


Fig. 23 Tabblad "Transitions" van de State machine wizard

Nu rest alleen het definiëren van de uitgangswaarden nog. Dit doen we door op het laatste tabblad "actions" te klikken. Bij een Mealy-machine is de uitgang niet alleen afhankelijk van de state, maar óók van de ingangswaarden. Eerst kiezen we de uitgang, dan wijzen we die een waarde toe, vervolgens kiezen we een state waarin dit moet gebeuren en als laatste geven we een ingangscombinatie op die ook waar moet zijn voor deze output. In dit geval kiezen we achtereenvolgens voor "u",1,"s0" en "i". Dit betekent dat als de machine in state s0 zit en de ingang is 1, dan uitgang u 1 wordt. Alle andere uitgangen hoeven we niet te specificeren.

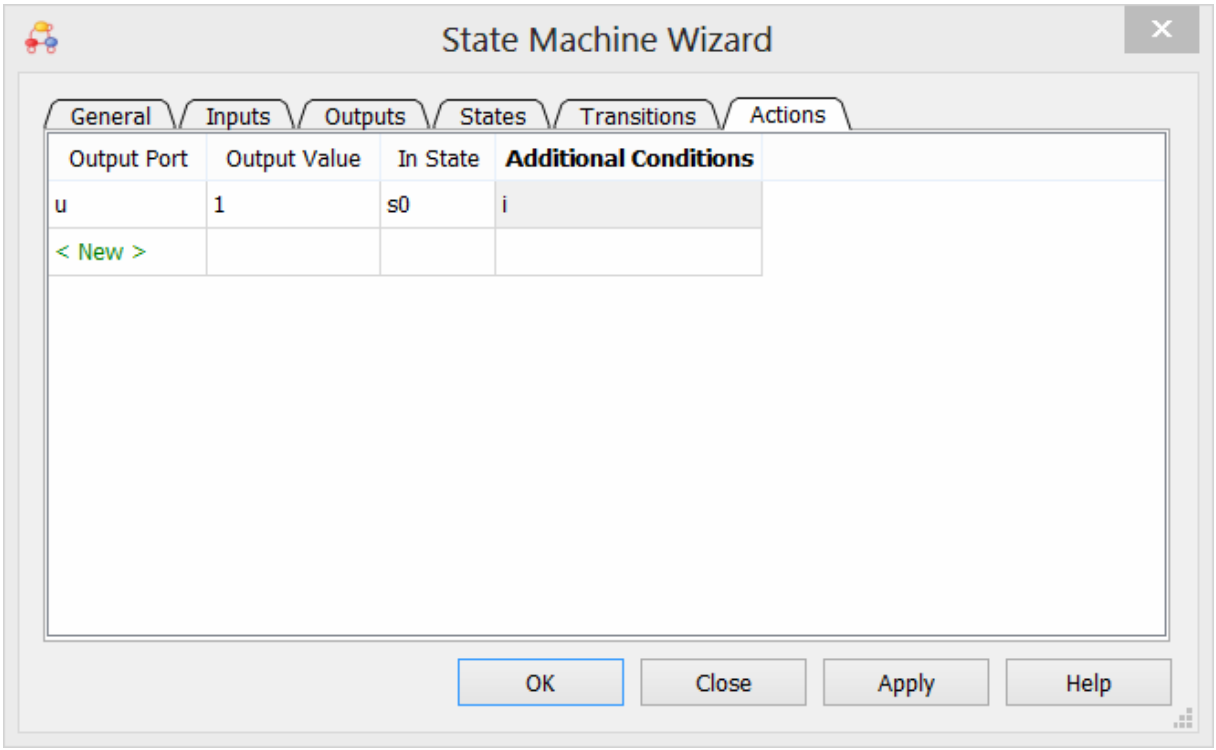


Fig. 24 Tabblad "Actions" van de State machine wizard

Nu klikken we op Apply en daarna op close. We zien de state machine nu in de editor staan. Je kunt deze bewerken maar dat is niet nodig[9]. Alles is al ingesteld via de wizard. Nu kunnen we er VHDL van laten maken.

De beschrijving die er uit komt is als volgt:

```
-- Copyright (C) 1991-2013 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.

-- Generated by Quartus II Version 13.0.1 Build 232 06/12/2013 Service Pack
1 SJ Web Edition
-- Created on Wed Apr 16 11:52:55 2014

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mealy_machine IS
    PORT (
        clock : IN STD_LOGIC;
        reset : IN STD_LOGIC := '0';
        i : IN STD_LOGIC := '0';
        u : OUT STD_LOGIC
```

[9] Bewerken kan via de editor maar óók via de wizard. Door de wizard op te starten en dan te kiezen voor "Edit an existing state machine design."

```
    );
END mealy_machine;

ARCHITECTURE BEHAVIOR OF mealy_machine IS
    TYPE type_fstate IS (s0,s1);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reset,reg_fstate)
    BEGIN
          IF (reset='1') THEN
            fstate <= s1;
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;

    PROCESS (fstate,i)
    BEGIN
        u <= '0';
        CASE fstate IS
            WHEN s0 =>
                IF ((i = '1')) THEN
                    reg_fstate <= s1;
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    reg_fstate <= s0;
                END IF;
                IF ((i = '1')) THEN
                    u <= '1';
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    u <= '0';
                END IF;
            WHEN s1 =>
                IF (NOT((i = '1'))) THEN
                    reg_fstate <= s0;
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    reg_fstate <= s1;
                END IF;
            WHEN OTHERS =>
                u <= 'X';
                report "Reach undefined state";
        END CASE;
    END PROCESS;
END BEHAVIOR;
```

State codering

Je kunt zelf je state codering kiezen. De keuzemogelijkheden zijn:
- auto
- one-hot
- gray
- Johnson
- minimal bits
- sequential
- user-encoded

Auto
Standaard staat dit op automatisch. Quartus kijkt dan zelf wat de beste codering is.

One-hot
Het aantal flipflops is gelijk aan het aantal states. Elke state krijgt een flipflop toegewezen. Als de flipflop van de state die op dat moment actief is, is dan 1 (hot). Alle andere zijn dan nul (cold). Quartus kan niet garanderen dat er altijd maar één bit '1' is. Bij een reset wordt er wél een garantie gegeven dat alle bits '0' zijn. Vaak is het zo dat er twee bits '1' zijn.

Gray
Het aantal flipflops is gelijk aan ^{2}Log (het aantal states). Elke opvolger state is met maar één bitwisseling te bereiken

Johnson
Het aantal bits is gelijk aan de helft van het aantal states. De opvolger state is te maken door alle bits naar rechts te schuiven. Het MSB is het geïnverteerde LSB van de vorige state. Ook is er met de vorige én volgende state maar één bitwisseling.

Minimal bits
Het aantal bits is gelijk aan het aantal bits dat nodig is om het aantal states binair te tellen. Een kleiner aantal bits is niet mogelijk.

Sequential
Het aantal bits is gelijk aan het aantal bits dat nodig is om het aantal states binair te tellen. Waarbij het binaire nummer gelijk is aan het volgordenummer van de state.

User-encoded
Hiermee kan je zelf een state codering opgeven.

Instellen state codering

Je kunt de state codering instellen door in de menubalk te kiezen voor "Assignments" > "settings" (Fig. 25) en dan in het submenu "Analysis & Synthesis Settings" (**Fout! Verwijzingsbron niet gevonden.**).

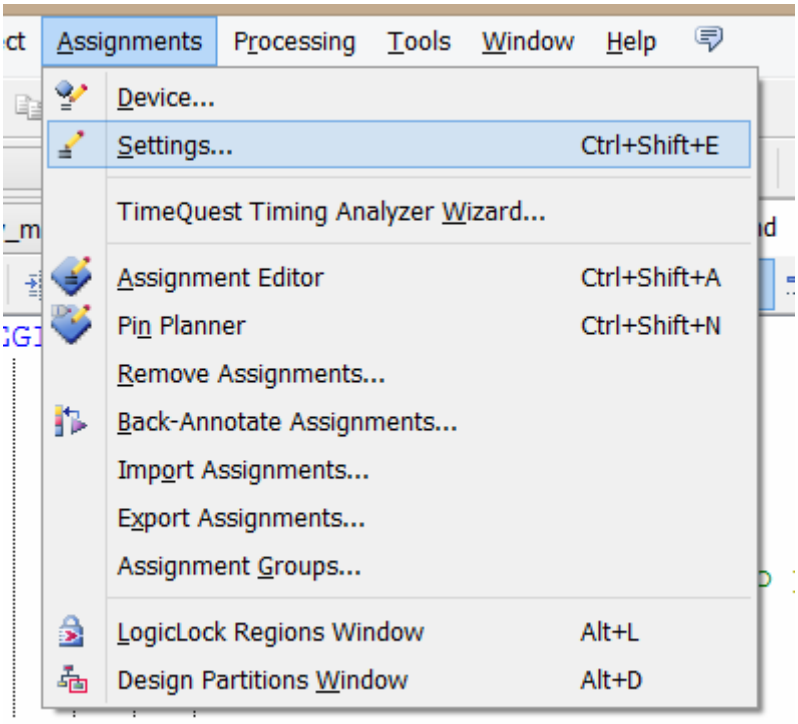


Fig. 25 Menubalk met assignments uitgevouwen

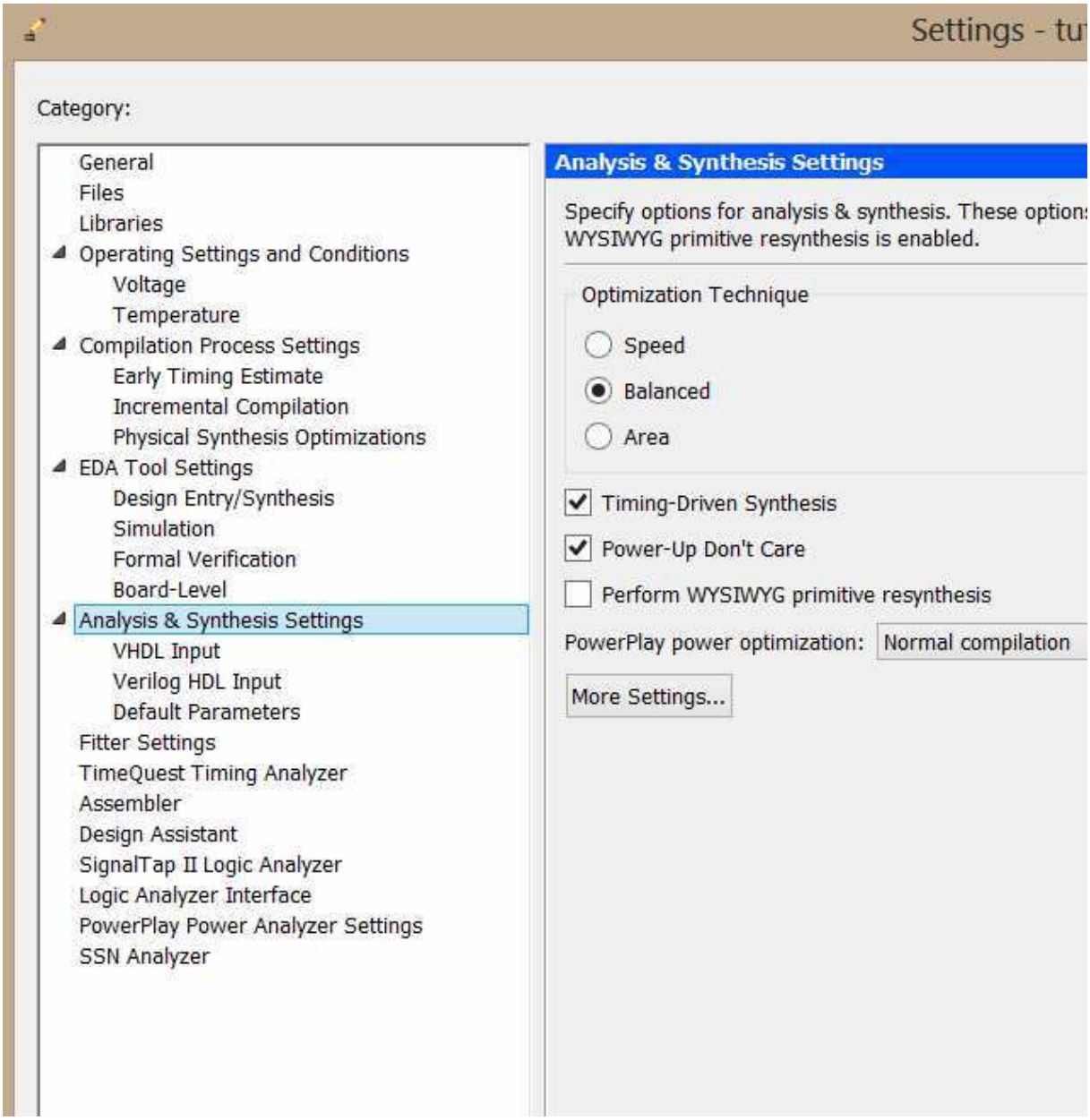


Fig. 26 Het settings venster met menuoptie Analysis & Synthesis Settings

Daar klik je op de knop "more settings…"

Nu komt er een extra scherm te voorschijn (Fig. 27). In dit scherm kan je alle instellingen doen die betrekking hebben op de beschreven logica. In deze lijst gaan we op zoek naar "State machine Processing". Naast deze naam zien we Auto staan. Dit is de standaardinstelling. Door dubbel op de naam te klikken kunnen we een van de bovengenoemde opties kiezen.

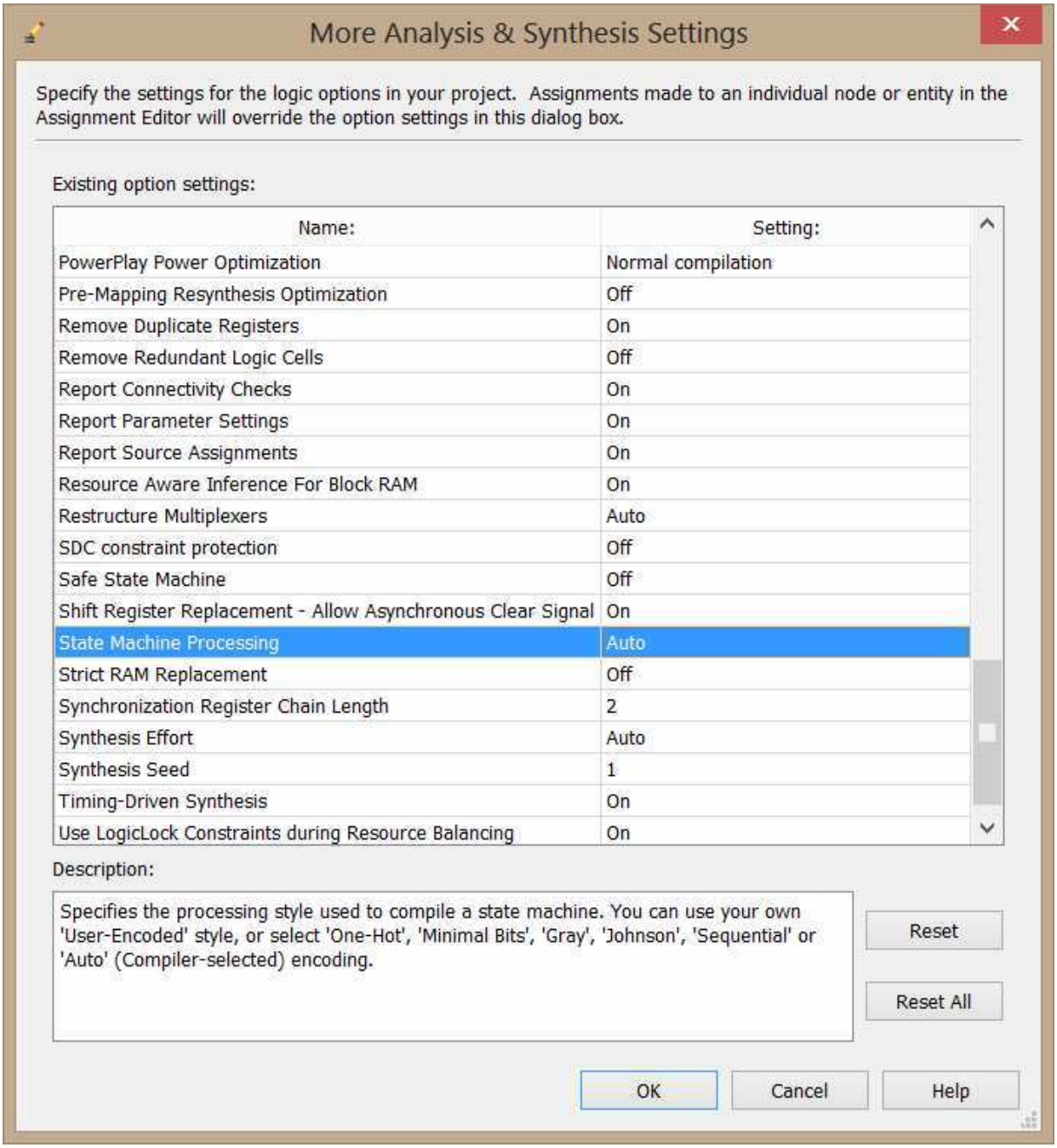


Fig. 27 Het menu More Analysis and Synthesis Settings

Een aantal handige zaken:

- Als je het selectiegereedschap heb geselecteerd en je dubbelklikt op het veld dan wordt er op die plaats een nieuwe state aangemaakt.
- Zie je tijdens het omzetten fouten en heb je ze verholpen klik dan voor het opnieuw proberen me de rechtermuisknop op het meldingenvenster en kies voor "Clear messages from windows". Dit maakt het scherm leeg. Hierdoor zie je eventuele nieuwe fouten.
- Eventuele andere bestanden (geen Verilog of VHDL) kunnen gegenereerd worden door met de rechtermuisknop op het veld te klikken en te kiezen voor generate other files. (Fig. 28 Keuzescherm "Generate other files")
- Bij de state propterties, tabblad "Format" kan je de state een kleurtje geven.

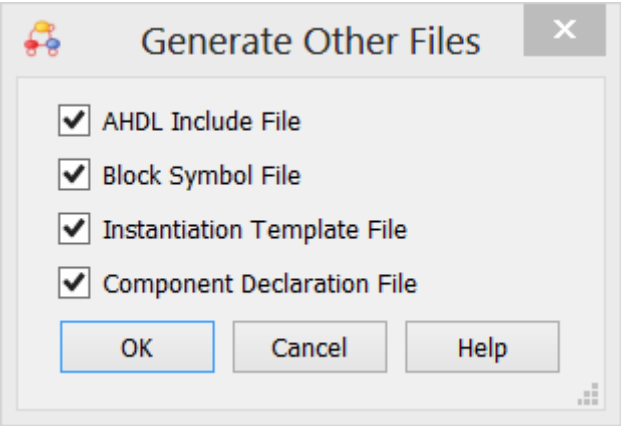


Fig. 28 Keuzescherm "Generate other files"

Quartusversie

Deze tutorial is geschreven door Pieter van der Star.
De gebruikte Quartusversie is:

- Quartus II 64-bit Version 13.0.1 Build 232 06/12/2013 SJ Web Edition (service pack: 1)

Voorgaande edities van deze tutorial zijn geschreven met de versies:

- Quartus II 32-bit Version 11.1 Build 259 01/25/2012 SJ Web Edition (service pack: 2)
- Quartus II 32-bit Version 11.1 Build 216 11/23/2011 SJ Full Version (service pack : 1)

Quartus II

Registered output

&

current/next clock cycle

In de State machine editor & wizard

2 mei 2013
Pieter van der Star

Inhoudsopgave

Bijlagen

Inleiding

De state machine editor geeft, per uitgang, de keuze op welke klokpuls het signaal naar buiten komt, en of het door een register gehaald wordt. In dit document behandelt de verschillen tussen de opties. Er kunnen twee variabelen ingesteld worden. Alle twee de opties hebben twee mogelijkheden. Dit levert in totaal vier mogelijkheden op. Deze zijn:

- Not registered, current clock cycle
- Registered, current clock cycle
- Not registered, next clock cycle
- Registered, next clock cycle

Om er achter te komen wat de gevolgen van deze opties zijn is er eerst een state machine getekend zoals beschreven in het eerste deel van de tutorial. Deze tekening is daarna omgezet naar vhdl. En eventueel verder naar een tekening van de hardware. Ook is er een deel gesimuleerd met modelsim.

State machines

De proeven zijn gedaan met meerdere state machines. Elke state machine heeft, naast de ingangen klok en reset (asyncroon), de ingangen x en y en uitgang z.

De eerste state machine is een Moore machine met twee states. (Fig. 1)
De uitgang is hier alleen afhankelijk van de state van de machine. Alleen in state “state1” is de uitgang 1. In alle andere gevallen is de uitgang 0.

Daarna is deze machine uitgebreid met een extra state. Ook hier is de uitgang alleen afhankelijk van de state van de machine. Alleen in state “state1” is de uitgang 1. In alle andere gevallen is de uitgang 0.

Laatstgenoemde machine is daarna veranderd in een Mealy machine. De machine is niet omgezet, maar er zijn extra voorwaarden gesteld aan de uitgangstoewijzing. De nieuwe voorwaarden die waar moeten zijn om de ingang 1 te maken zijn nu:
De state mahine moet zich in state “state1” bevinden én ingang x moet ook 1 zijn.

Twee state Moore machine

Van deze state machine tekening is voor elke instelling een hardwarebeschrijving van gemaakt. Deze beschrijving is in VHDL. Van deze VHDL code is een hardware beschrijving gemaakt.

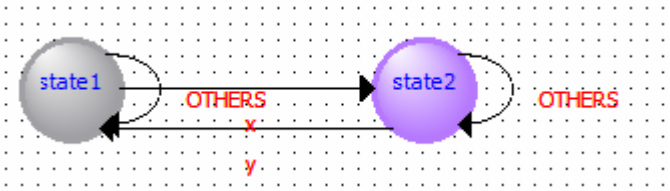


Fig. 1 Moore machine met twee states

De hardware die uit de omgezette VHDL code komt is weinig anders bij de gekozen opties. Bij de instelling registered output, next clock cycle (Fig. 5) wordt er een flipflop aangemaakt. Bij alle andere opties (ook de not registered, next clock cycle) is dit niet het geval. Bij registered, current clock cycle (Fig. 3) wordt er een extra multiplexer aangemaakt.

De resultaten van de synthese zijn op de volgende pagina afgebeeld.

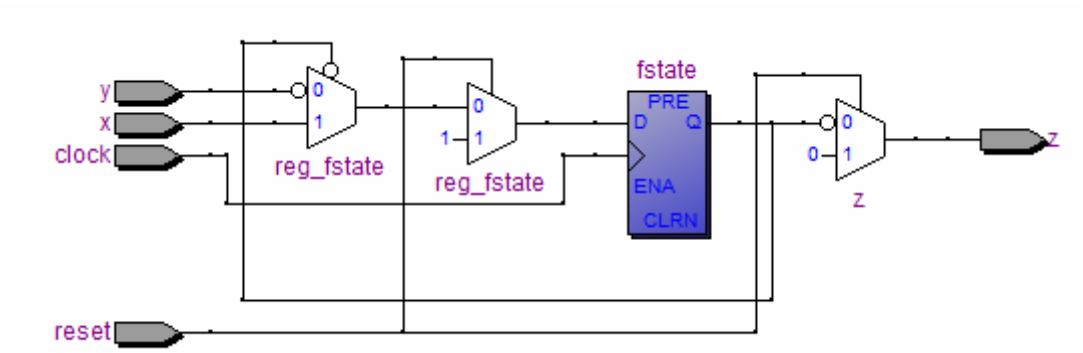


Fig. 2 Not registered, current clock cycle

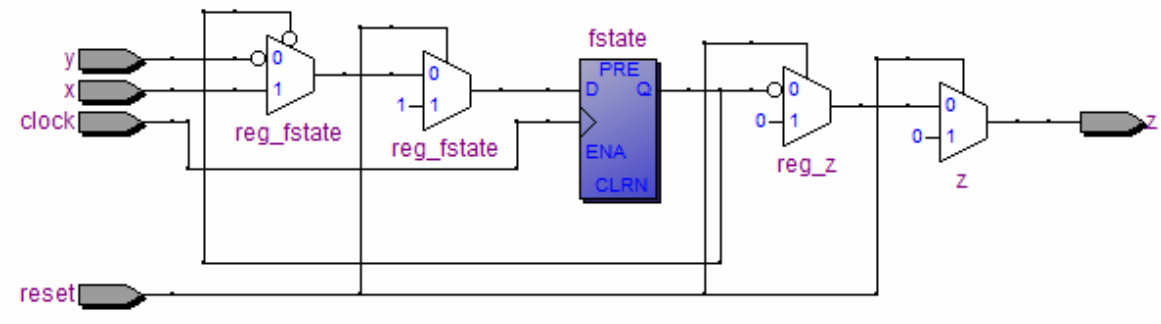


Fig. 3 registered, current clock cycle

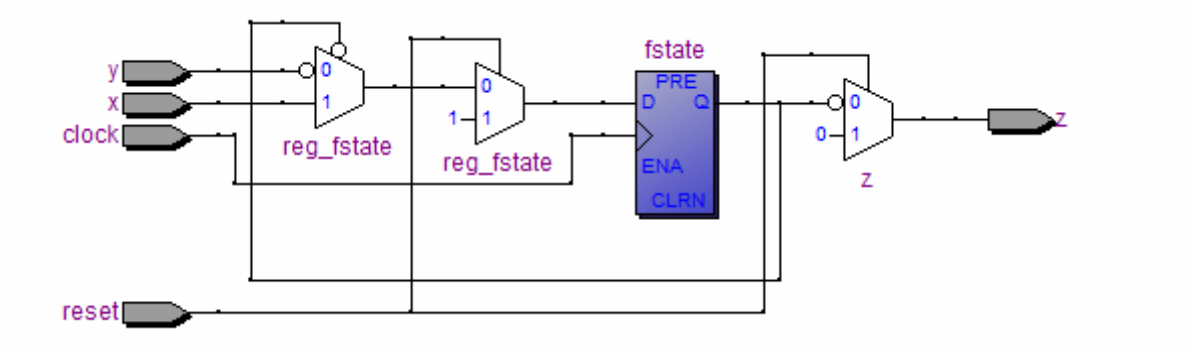


Fig. 4 not registered, next clock cycle

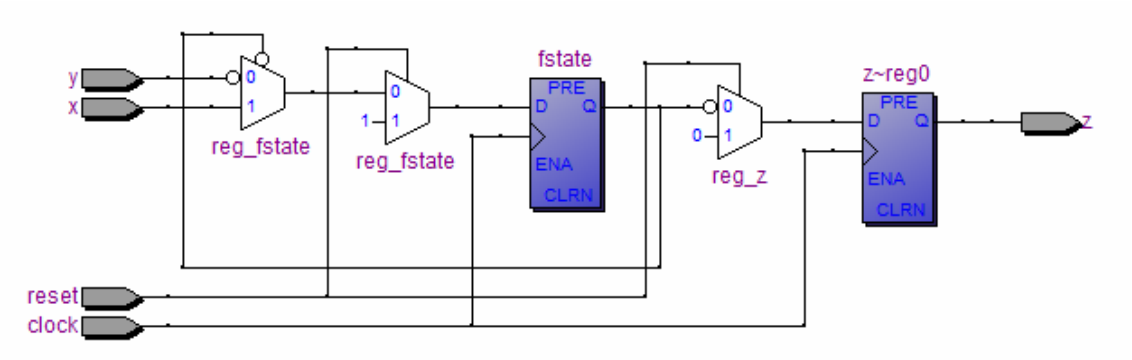


Fig. 5 registered, next clock cycle

Drie state Moore machine

Wat er precies met de uitgang gebeurt bij de twee state Moore machine is niet duidelijk. Daarom is die state machine uitgebreid met één extra state. In Fig. 14 is deze state machine te zien. Van deze state machine is ook een hardware tekening gemaakt. Daarnaast is deze machine ook gesimuleerd met de testbench uit Bijlage 1. De code uit de .do file is te vinden in Do file.

Opvallend is dat de optie next clockcycle alleen effect heeft al er ook voor registered gekozen is. Als er geen uitgangsregister aan gemaakt wordt dan komt het altijd op de huidige klokpuls naar buiten. Wel wordt het signaal meer vertraagd. Dit komt door de extra mux waar het signaal doorheen moet.

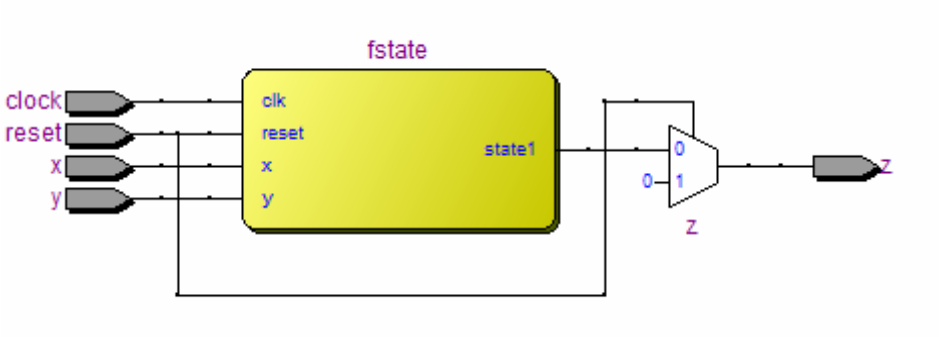


Fig. 6 not registered, current clockcycle

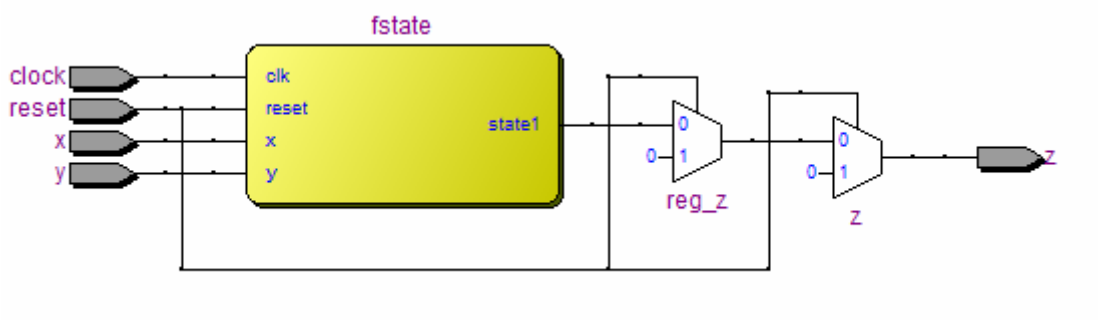


Fig. 7 registered, current clockcycle

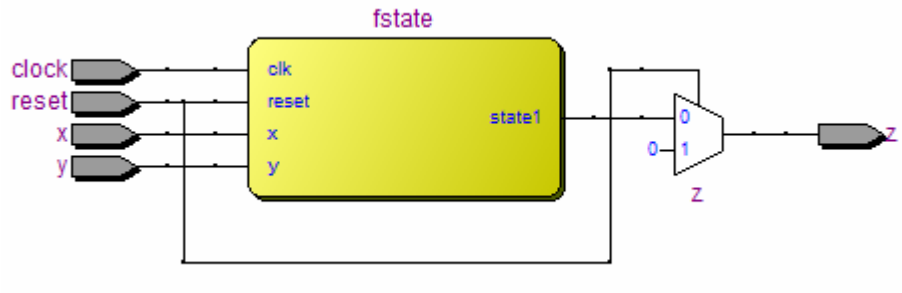


Fig. 8 not registered, next clock cycle

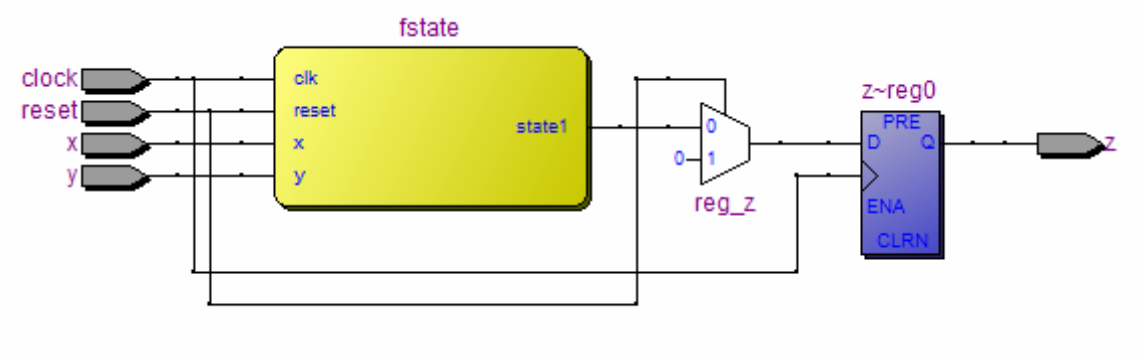


Fig. 9 registered, next clock cycle

De resultaten van de simulatie geven hetzelfde aan. Niet te zien is de vertraging van de extra mux.

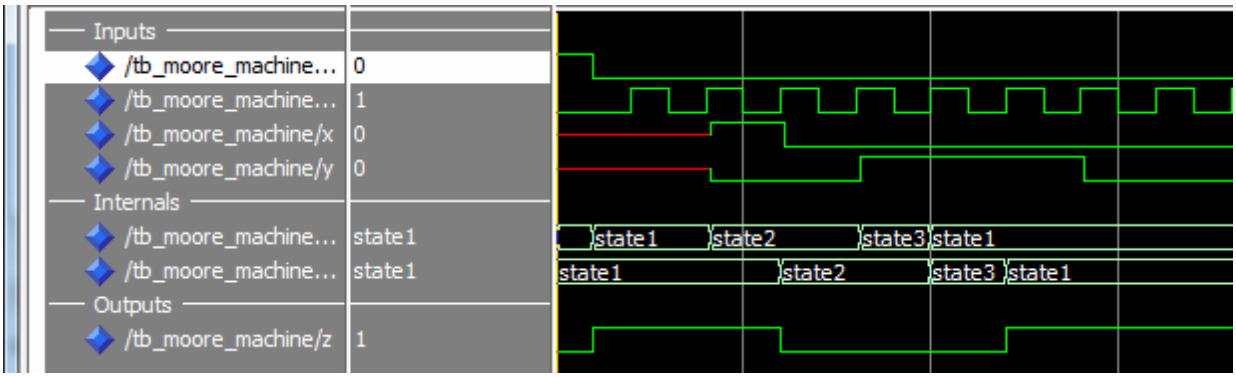


Fig. 10 not registered, current clock cycle

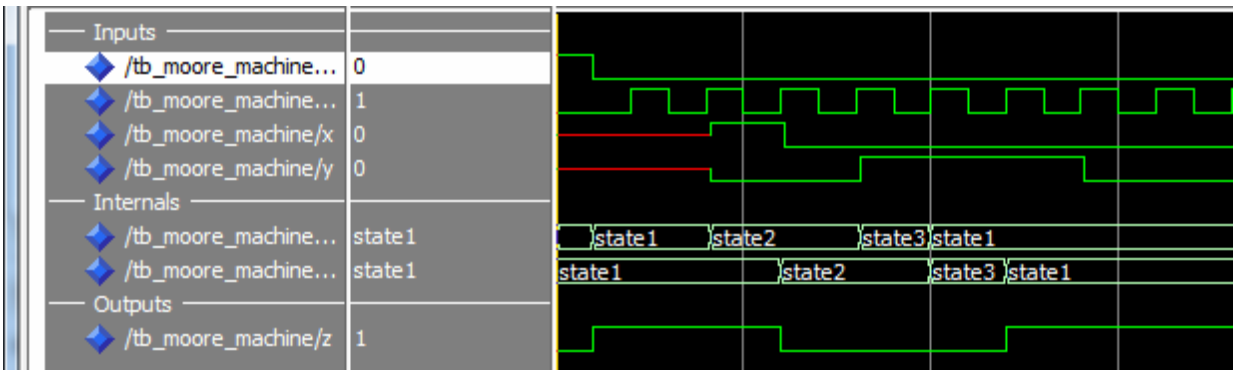


Fig. 11 registered, current clock cycle

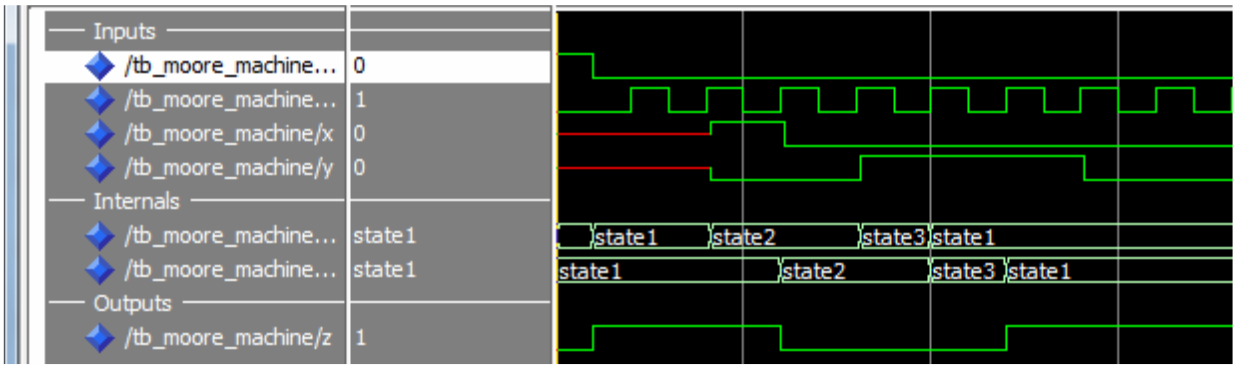


Fig. 12 not registered, next clock cycle

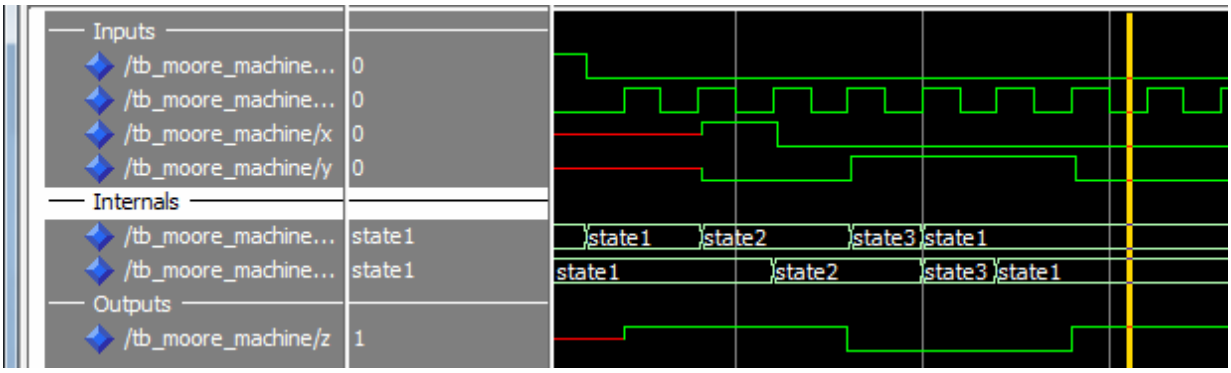


Fig. 13 registered, next clock cycle

Om te kijken of de opties meer effect hebben bij een Mealy machine is de machine veranderd in een Mealy machine. De Moore machine is niet omgezet naar een Mealy machine, maar er zijn extra voorwaarden gesteld aan de waardetoekenning van de uitgang (z). Van deze machine is een beschrijving VHDL gemaakt, en ook is het getest met een testbench. Deze is te vinden in Bijlage 2. Van deze machine is er ook een technology map gemaakt. Deze is te vinden in Bijlage 5

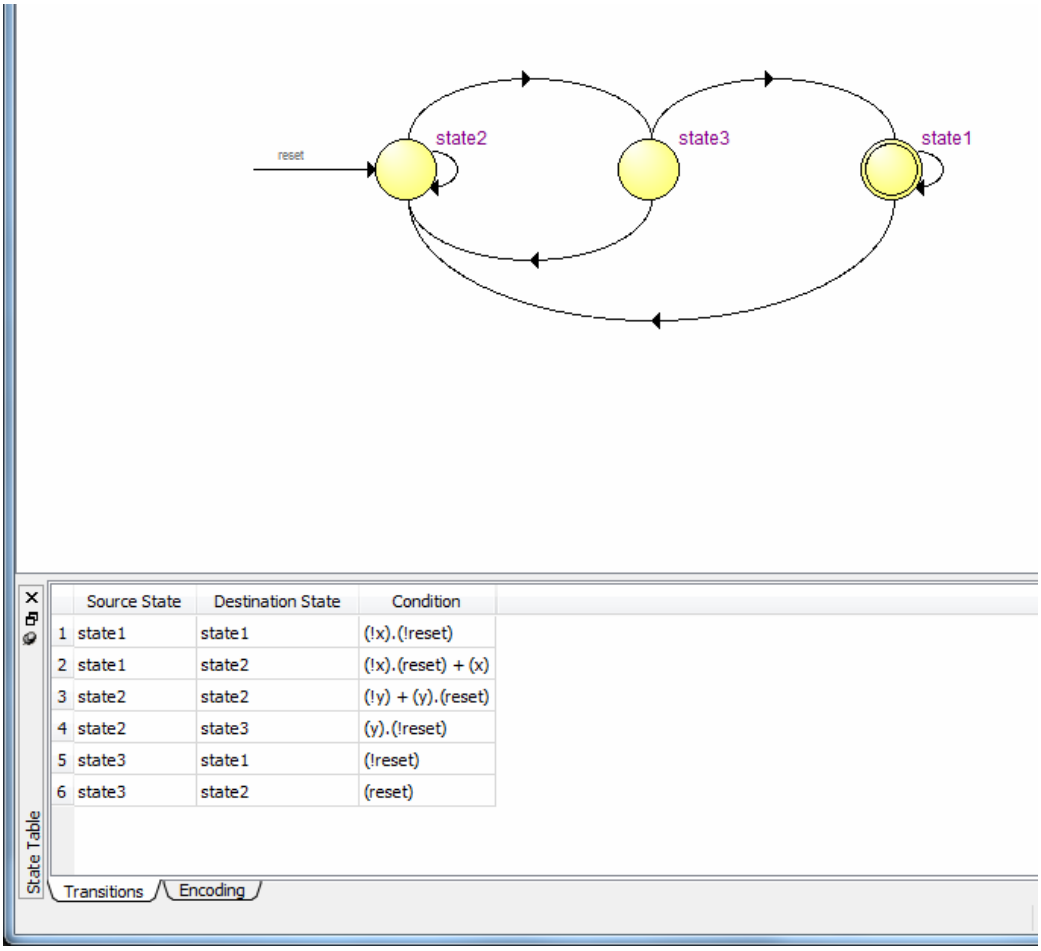


Fig. 14 State machine na synthese zoals weergeven in de state machine viewer

Als de state-machine naar HDL omgezet wordt kan het effect van deze opties bekeken worden. In Bijlage 3 is de code van de mealy machine te vinden. Door de code te vergelijken valt te zien dat, als er voor de optie registered output gekozen wordt, er een extra signaal aangemaakt word. Dit signaal krijgt de naam reg_<outputnaam>. De toewijzing van dit signaal

Hardwarematig betekent dit dat er, bij een Moore-machine, voor de eerste en derde optie niets tussen de statemachine en de output geplaatst wordt. Als de tweede optie gekozen wordt dan levert dit een extra multiplexer op. Voor de laatste optie wordt er nog een flipflop aangemaakt.

Bijlage 1. Testbench voor Moore machine

```
-- Filename : tb_moore_machine.vhd
-- Filetype : VHDL testbench code (behavior)
-- Date     : 20-06-2011


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

entity tb_moore_machine is
end tb_moore_machine;

architecture testbench of tb_moore_machine is

signal reset : STD_LOGIC;
signal clock : STD_LOGIC;
signal x : STD_LOGIC;
signal y : STD_LOGIC;
signal z : STD_LOGIC;

component moore_machine
    port (
            reset : IN STD_LOGIC;
            clock : IN STD_LOGIC;
            x : IN STD_LOGIC;
            y : IN STD_LOGIC;
            z : OUT STD_LOGIC
      );
end component;

begin
      dut : moore_machine
      port map (
            reset => reset,
            clock => clock,
            x => x,
            y => y,
            z => z
      );

      clockgen : process is
      begin
            clock <= '0';
            wait for 200 ns;
            loop
                  clock <= '1';
                  wait for 100 ns;
                  clock <= '0';
                  wait for 100 ns;
            end loop;
      end process clockgen;

      gen : process is
      begin
            reset <= '1';
            wait for 100 ns;
            reset <= '0';
            wait until clock = '1';
            wait for 10 ns;

            wait for 250 ns;
                  x <= '1';
                  y <= '0';
            wait for 200 ns;
                  x <= '0';
                  y <= '0';
```

```
            wait for 200 ns;
                  x <= '0';
                  y <= '1';
            wait for 200 ns;
                  x <= '0';
                  y <= '1';
            wait for 200 ns;
                  x <= '0';
                  y <= '1';
            wait for 200 ns;
                  x <= '0';
                  y <= '0';

            wait;
      end process gen;

end testbench;
```

Bijlage 2. Testbench voor Mealy machine

```
-- Filename : tb_moore_machine.vhd
-- Filetype : VHDL testbench code (behavior)
-- Date     : 20-06-2011


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

entity tb_moore_machine is
end tb_moore_machine;

architecture testbench of tb_moore_machine is

signal reset : STD_LOGIC;
signal clock : STD_LOGIC;
signal x : STD_LOGIC;
signal y : STD_LOGIC;
signal z : STD_LOGIC;

component moore_machine
    port (
            reset : IN STD_LOGIC;
            clock : IN STD_LOGIC;
            x : IN STD_LOGIC;
            y : IN STD_LOGIC;
            z : OUT STD_LOGIC
      );
end component;

begin
      dut : moore_machine
      port map (
            reset => reset,
            clock => clock,
            x => x,
            y => y,
            z => z
      );

      clockgen : process is
      begin
            clock <= '0';
            wait for 200 ns;
            loop
                  clock <= '1';
                  wait for 100 ns;
                  clock <= '0';
                  wait for 100 ns;
            end loop;
      end process clockgen;

      gen : process is
      begin
            reset <= '1';
            wait for 100 ns;
            reset <= '0';
            wait until clock = '1';
            wait for 10 ns;

            wait for 250 ns;
                  x <= '1';
                  y <= '0';
            wait for 200 ns;
```

```
                x <= '0';
                y <= '0';
            wait for 200 ns;
                x <= '0';
                y <= '1';
            wait for 200 ns;
                x <= '0';
                y <= '1';
            wait for 200 ns;
                x <= '0';
                y <= '1';
            wait for 200 ns;
                x <= '0';
                y <= '0';
            wait for 200 ns;
                x <= '1';
                y <= '0';
            wait for 250 ns;
                x <= '0';
                y <= '0';
            wait;
      end process gen;

end testbench;
```

Bijlage 3. Do file

```
# Filename  : tb_moore_machine.do
# Filetype  : Modelsim Script File
# Date      : 20-06-2011


# Set transcript on
transcript on

# Recreate the work directory and map to work
if {[file exists rtl_work]} {
      vdel -lib rtl_work -all
}
vlib rtl_work
vmap work rtl_work

# Compile the Double Dabble VHDL description and testbench
vcom -93 -work work ../../moore_machine.vhd
vcom -93 -work work ../../tb_moore_machine.vhd

# Start the simulator with 1 ns time resolution
#vsim -t 1ns -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L
cycloneii -L rtl_work -L work -voptargs="+acc" tb_moore_machine
vsim -t 1ns -L rtl_work -L work -voptargs="+acc" tb_moore_machine

# Log all signals in the design, good if the number
# of signals is small.
add log -r *

# Add all toplevel signals
# Add a number of signals of the simulated design
add list *

# Add all toplevel signals
# Add a number of signals of the simulated design
add wave -divider "Inputs"
add wave reset
add wave clock
add wave x
add wave y
add wave -divider "Internals"
add wave dut/reg_fstate
add wave dut/fstate
add wave -divider "Outputs"
add wave z

# Open Structure, Signals (waveform) and List window
view structure
view list
view signals
view wave

# Run simulation for 5000 ns
run 5000 ns

# Fill up the waveform in the window
wave zoom full
```

Bijlage 4. VHDL code

```
-- NOT REGISTERED, CURRENT CLOCK CYCLE

-- Copyright (C) 1991-2011 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.

-- Generated by Quartus II Version 11.1 Build 259 01/25/2012 Service Pack 2
SJ Web Edition
-- Created on Sat Jun 23 20:15:43 2012

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY moore_machine IS
    PORT (
        reset : IN STD_LOGIC := '0';
        clock : IN STD_LOGIC;
        x : IN STD_LOGIC := '0';
        y : IN STD_LOGIC := '0';
        z : OUT STD_LOGIC
    );
END moore_machine;

ARCHITECTURE BEHAVIOR OF moore_machine IS
    TYPE type_fstate IS (state1,state2,state3);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;

    PROCESS (fstate,reset,x,y)
    BEGIN
        IF (reset='1') THEN
            reg_fstate <= state2;
            z <= '0';
        ELSE
            z <= '0';
            CASE fstate IS
                WHEN state1 =>
                    IF ((x = '1')) THEN
                        reg_fstate <= state2;
                    ELSE
                        reg_fstate <= state1;
                    END IF;

                    IF ((x = '1')) THEN
                        z <= '1';
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        z <= '0';
                    END IF;
```

```
                WHEN state2 =>
                    IF ((y = '1')) THEN
                        reg_fstate <= state3;
                    ELSE
                        reg_fstate <= state2;
                    END IF;

                    z <= '0';
                WHEN state3 =>
                    reg_fstate <= state1;

                    z <= '0';
                WHEN OTHERS =>
                    z <= 'X';
                    report "Reach undefined state";
            END CASE;
        END IF;
    END PROCESS;
END BEHAVIOR;
```

```
-- REGISTERED, CURRENT CLOCK CYLCE

-- Copyright (C) 1991-2011 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.

-- Generated by Quartus II Version 11.1 Build 259 01/25/2012 Service Pack 2
SJ Web Edition
-- Created on Sat Jun 23 20:29:41 2012

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY moore_machine IS
    PORT (
        reset : IN STD_LOGIC := '0';
        clock : IN STD_LOGIC;
        x : IN STD_LOGIC := '0';
        y : IN STD_LOGIC := '0';
        z : OUT STD_LOGIC
    );
END moore_machine;

ARCHITECTURE BEHAVIOR OF moore_machine IS
    TYPE type_fstate IS (state1,state2,state3);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
    SIGNAL reg_z : STD_LOGIC := '0';
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;

    PROCESS (fstate,reset,x,y,reg_z)
    BEGIN
        IF (reset='1') THEN
            reg_fstate <= state2;
            reg_z <= '0';
            z <= '0';
        ELSE
            reg_z <= '0';
            z <= '0';
            CASE fstate IS
                WHEN state1 =>
                    IF ((x = '1')) THEN
                        reg_fstate <= state2;
                    ELSE
                        reg_fstate <= state1;
                    END IF;

                    IF ((x = '1')) THEN
                        reg_z <= '1';
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        reg_z <= '0';
                    END IF;
```

```
                WHEN state2 =>
                    IF ((y = '1')) THEN
                        reg_fstate <= state3;
                    ELSE
                        reg_fstate <= state2;
                    END IF;

                    reg_z <= '0';
                WHEN state3 =>
                    reg_fstate <= state1;

                    reg_z <= '0';
                WHEN OTHERS =>
                    reg_z <= 'X';
                    report "Reach undefined state";
            END CASE;
            z <= reg_z;
        END IF;
    END PROCESS;
END BEHAVIOR;
```

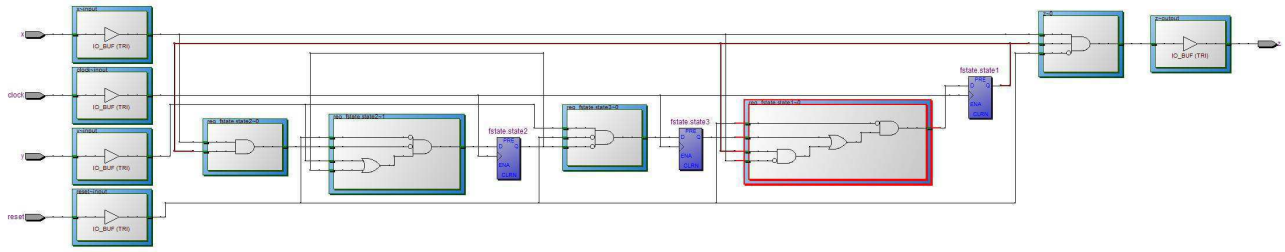
```
--NOT REGISTERED, NEXT CLOCK CYCLE

-- Copyright (C) 1991-2011 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.

-- Generated by Quartus II Version 11.1 Build 259 01/25/2012 Service Pack 2
SJ Web Edition
-- Created on Sat Jun 23 20:31:58 2012

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY moore_machine IS
    PORT (
        reset : IN STD_LOGIC := '0';
        clock : IN STD_LOGIC;
        x : IN STD_LOGIC := '0';
        y : IN STD_LOGIC := '0';
        z : OUT STD_LOGIC
    );
END moore_machine;

ARCHITECTURE BEHAVIOR OF moore_machine IS
    TYPE type_fstate IS (state1,state2,state3);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
        END IF;
    END PROCESS;

    PROCESS (fstate,reset,x,y)
    BEGIN
        IF (reset='1') THEN
            reg_fstate <= state2;
            z <= '0';
        ELSE
            z <= '0';
            CASE fstate IS
                WHEN state1 =>
                    IF ((x = '1')) THEN
                        reg_fstate <= state2;
                    ELSE
                        reg_fstate <= state1;
                    END IF;

                    IF ((x = '1')) THEN
                        z <= '1';
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        z <= '0';
                    END IF;
                WHEN state2 =>
                    IF ((y = '1')) THEN
                        reg_fstate <= state3;
```

```
                    ELSE
                        reg_fstate <= state2;
                    END IF;

                    z <= '0';
                WHEN state3 =>
                    reg_fstate <= state1;

                    z <= '0';
                WHEN OTHERS =>
                    z <= 'X';
                    report "Reach undefined state";
            END CASE;
        END IF;
    END PROCESS;
END BEHAVIOR;
```

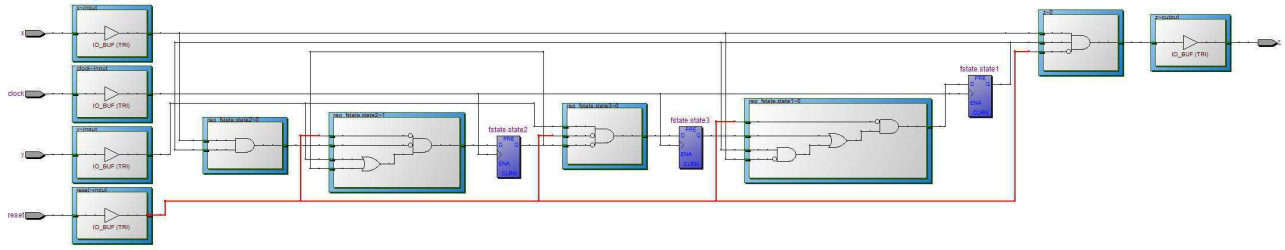
```
-- REGISTERED, NEXT CLOCK CYCLE

-- Copyright (C) 1991-2011 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.

-- Generated by Quartus II Version 11.1 Build 259 01/25/2012 Service Pack 2
SJ Web Edition
-- Created on Sat Jun 23 20:33:10 2012

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY moore_machine IS
    PORT (
        reset : IN STD_LOGIC := '0';
        clock : IN STD_LOGIC;
        x : IN STD_LOGIC := '0';
        y : IN STD_LOGIC := '0';
        z : OUT STD_LOGIC
    );
END moore_machine;

ARCHITECTURE BEHAVIOR OF moore_machine IS
    TYPE type_fstate IS (state1,state2,state3);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
    SIGNAL reg_z : STD_LOGIC := '0';
BEGIN
    PROCESS (clock,reg_fstate,reg_z)
    BEGIN
        IF (clock='1' AND clock'event) THEN
            fstate <= reg_fstate;
            z <= reg_z;
        END IF;
    END PROCESS;

    PROCESS (fstate,reset,x,y)
    BEGIN
        IF (reset='1') THEN
            reg_fstate <= state2;
            reg_z <= '0';
        ELSE
            reg_z <= '0';
            CASE fstate IS
                WHEN state1 =>
                    IF ((x = '1')) THEN
                        reg_fstate <= state2;
                    ELSE
                        reg_fstate <= state1;
                    END IF;

                    IF ((x = '1')) THEN
                        reg_z <= '1';
                    -- Inserting 'else' block to prevent latch inference
                    ELSE
                        reg_z <= '0';
                    END IF;
                WHEN state2 =>
```

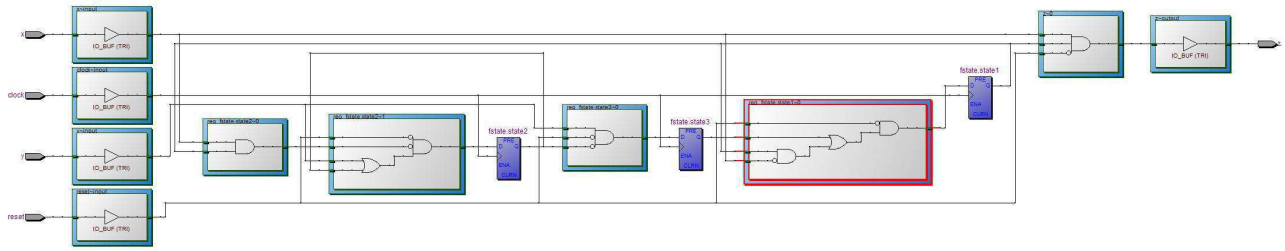
```
                    IF ((y = '1')) THEN
                        reg_fstate <= state3;
                    ELSE
                        reg_fstate <= state2;
                    END IF;

                    reg_z <= '0';
                WHEN state3 =>
                    reg_fstate <= state1;

                    reg_z <= '0';
                WHEN OTHERS =>
                    reg_z <= 'X';
                    report "Reach undefined state";
            END CASE;
        END IF;
    END PROCESS;
END BEHAVIOR;
```

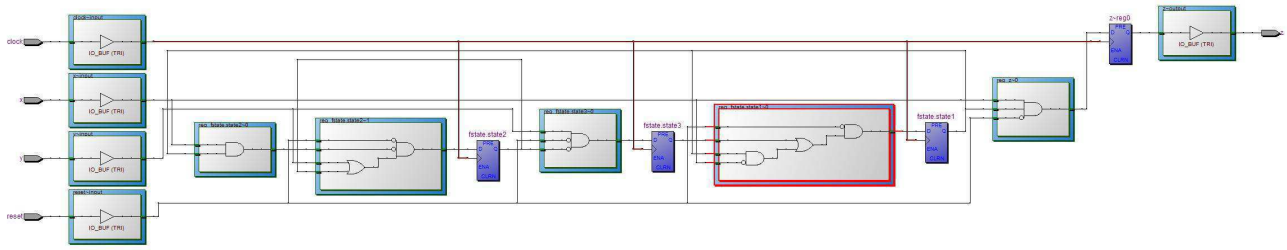
Bijlage 5. Technology map

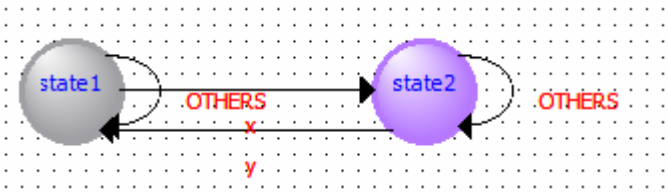


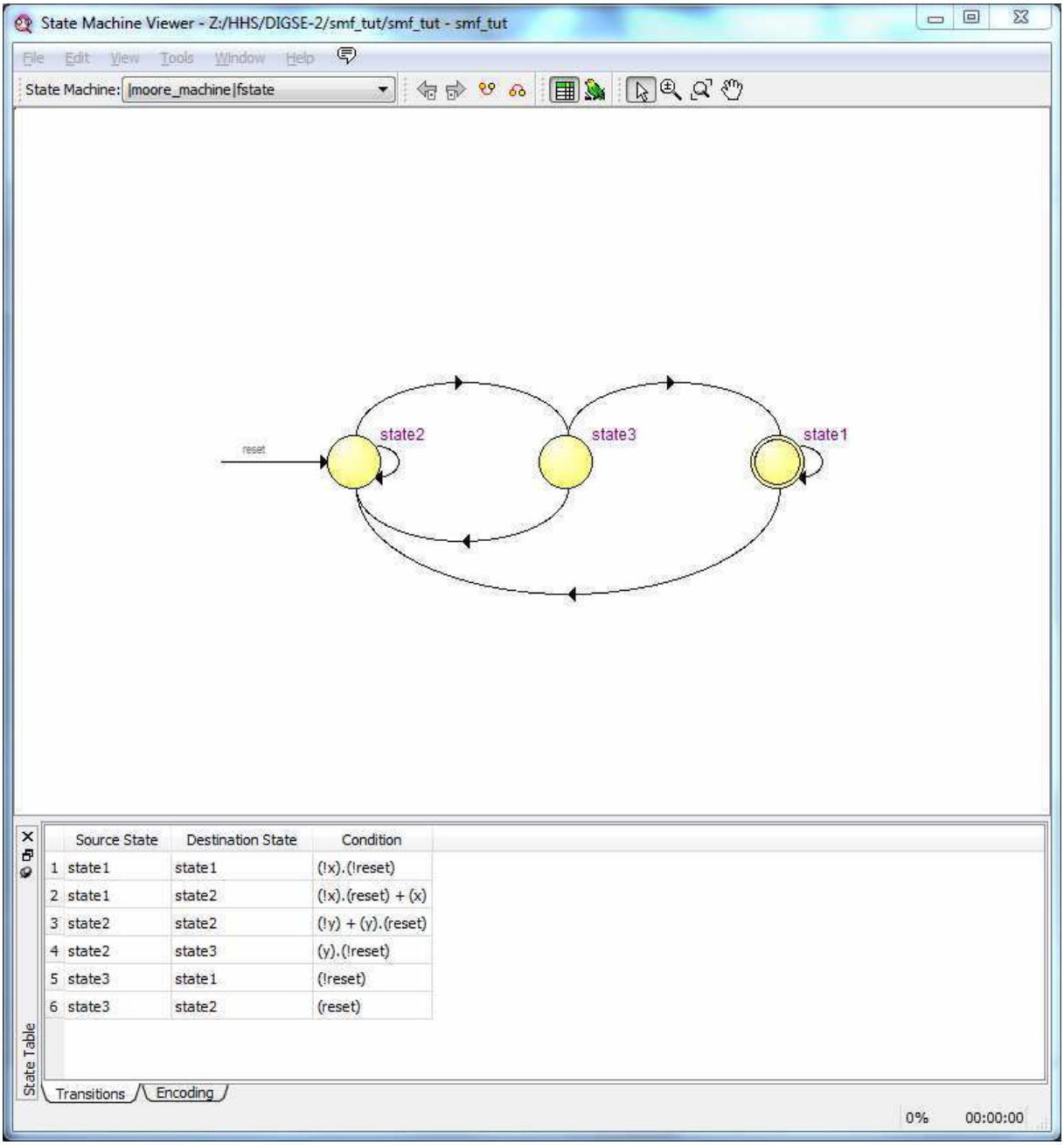
not registered




x
clock
y
reset
IO_BUF (TRI)
fstate state2
fstate state3
fstate state1

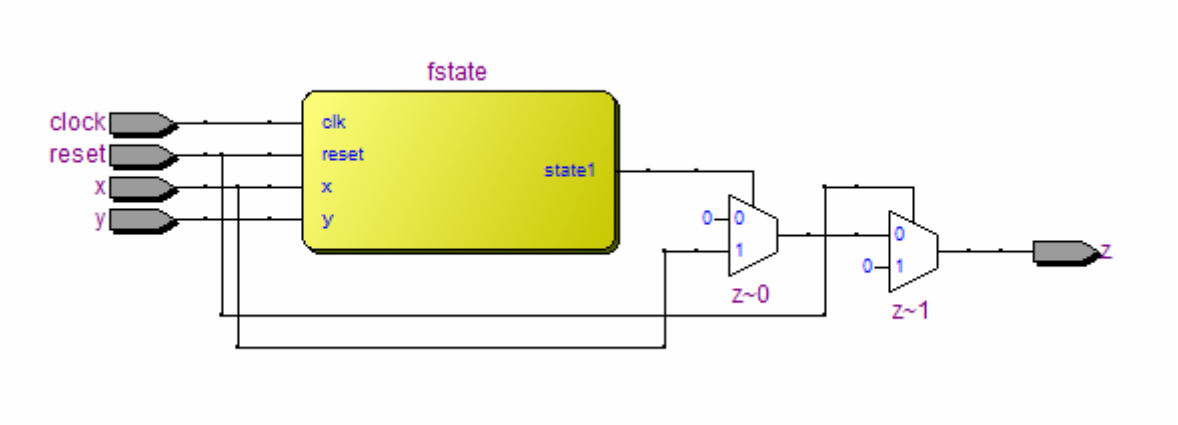

clock
x
y
reset
IO_BUF (TRI)
fstate state2
fstate state3
fstate state1


state1
OTHERS
x
y
state2
OTHERS

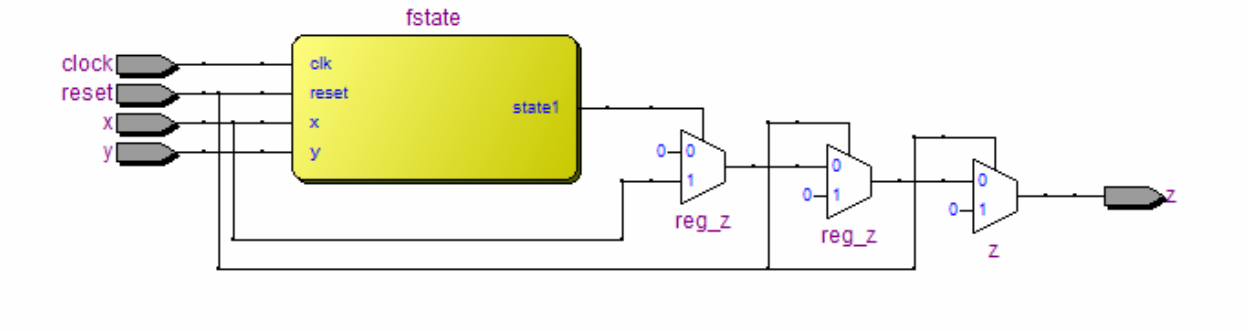


Alleen state1 heeft uitgang 1

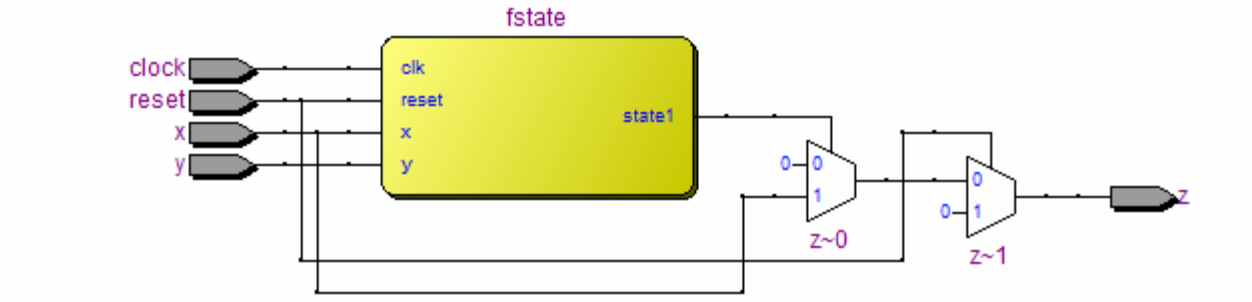
Zelfde machine, maar z nu ook afhankelijk van x
Testbench 1x extra verandering op input x



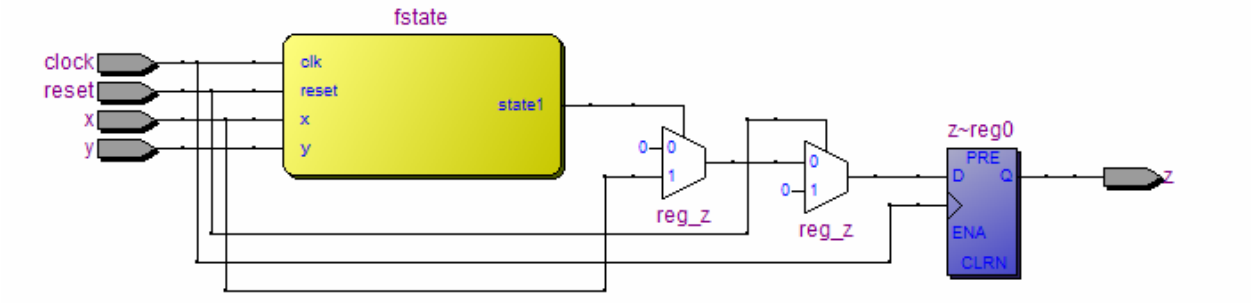
Z; not registered, current clock cycle



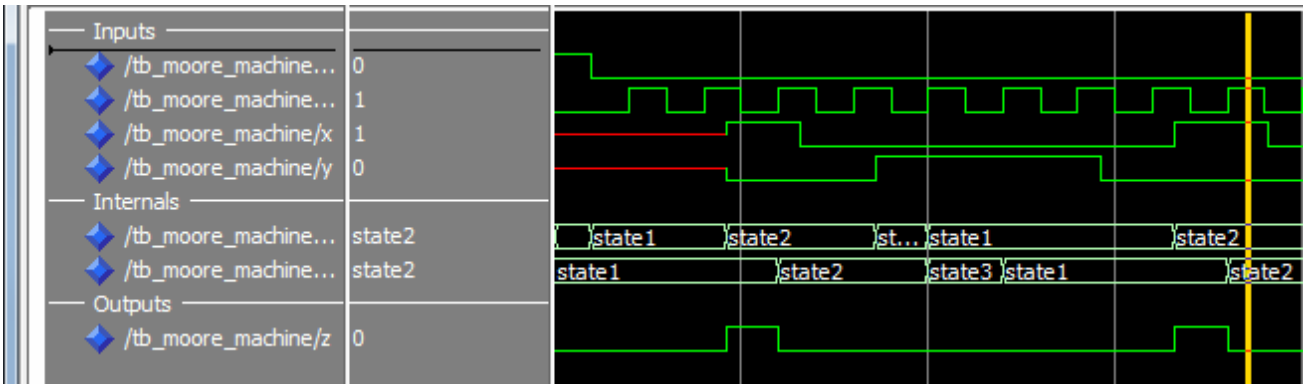
Z; registered, current clock cycle



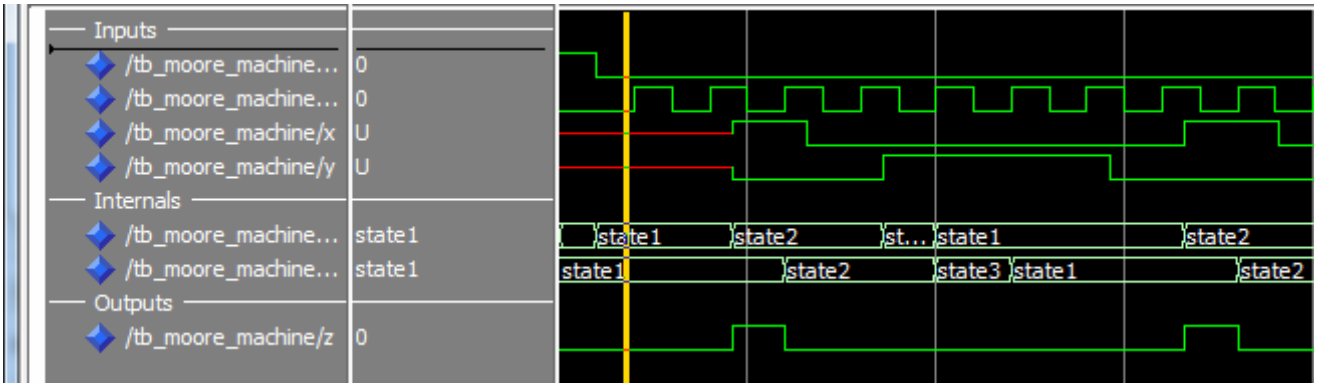
Z; not registered, next clock cycle



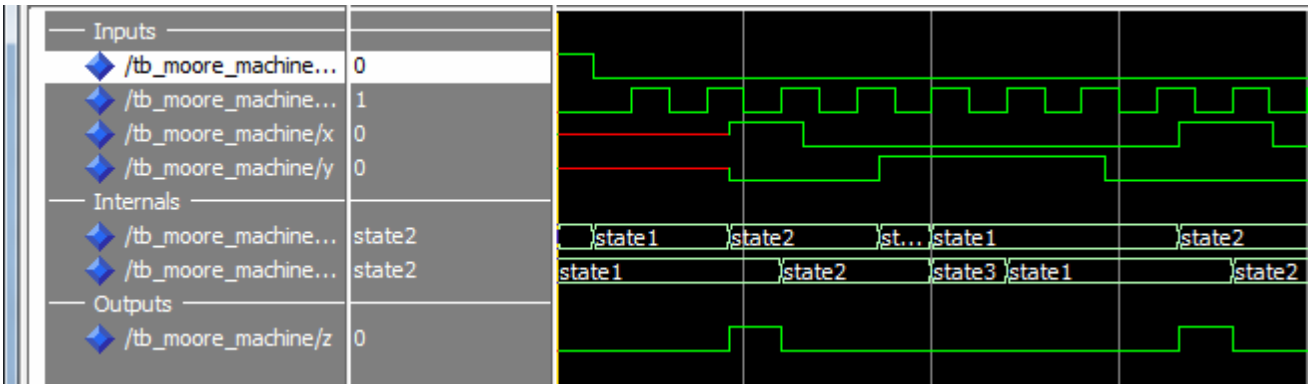
Z; registered, next clock cycle



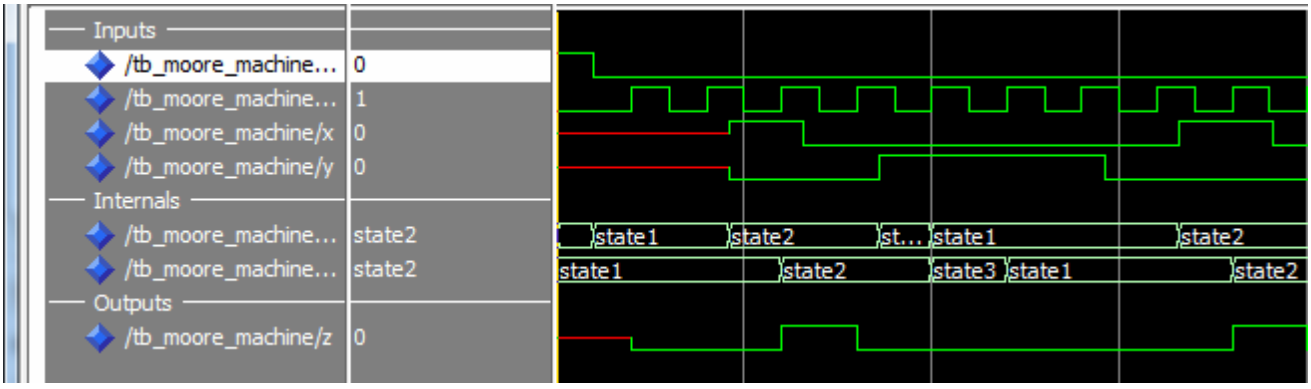
Z; not registered, current clock cycle



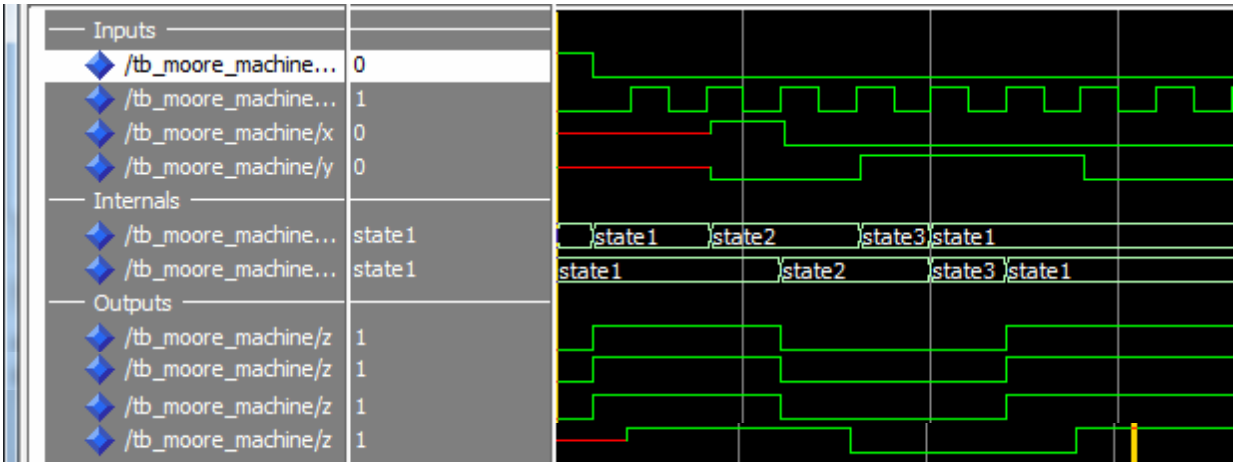
Z; registered, current clock cycle



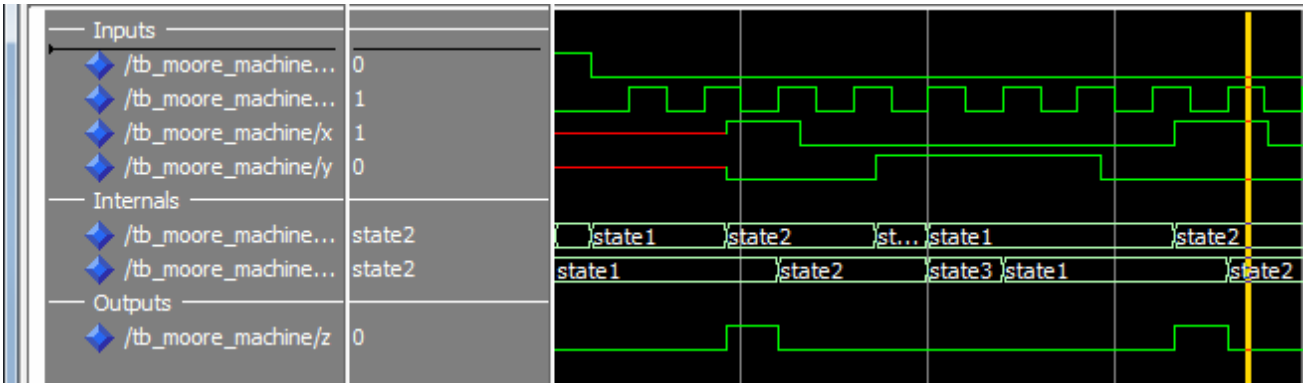
Z; not registered, next clock cycle



Z; registered, next clock cycle



Z niet afhankelijk van x



Z wel afhankelijk van x