Academie voor Technology, Innovation & Society Delft

Academie voor ICT & Media

Digitale Systeem Engineering 2

Week 1 – Introductie toestandsmachines

Jesse op den Brouw

DIGSE2/2020-2021

DE HAAGSE HOGESCHOOL

Logische circuits

- Logische circuits kunnen verdeeld worden in twee groepen:

- combinatorische logica

  uitgangen zijn alleen afhankelijk van de huidige ingangswaarden.

- sequentiële logica

  uitgangen zijn afhankelijk van de huidige ingangswaarden en van een waarden van flipflops.

Sequentiële systemen

- Dit is alleen mogelijk als de logica geheugenwerking bezit.
- Er zijn twee typen sequentiële schakelingen:
- Synchroon

  Een kloksignaal dient als referentie voor alle acties binnen het systeem.
- Asynchroon

  Er is geen kloksignaal aanwezig. Acties worden veroorzaakt door (eerdere) acties binnen het systeem.
- Synchrone systemen zijn eenvoudiger te ontwerpen dan asynchrone systemen.

Synchrone systemen

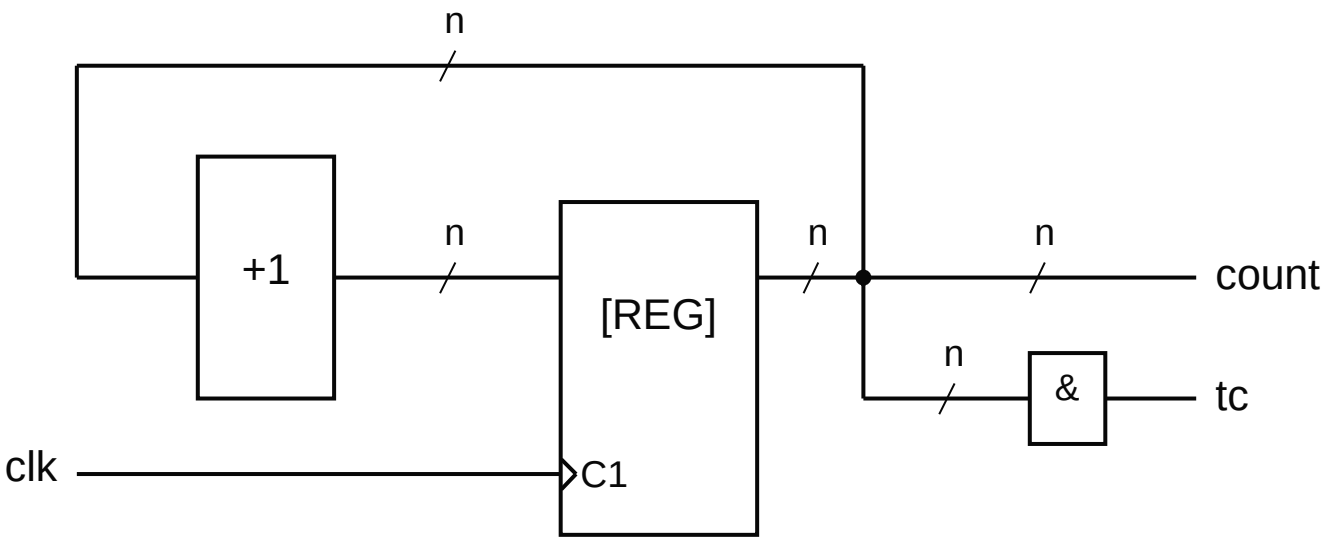
- Waarom kloksynchroon?

- Voordelen:
  - Eenvoudige timingverificatie.
  - Eenvoudige functionele verificatie.
  - Eenvoudige optimalisatie naar de kleinst mogelijke schakeling.
  - Eenvoudig ontwikkeltraject voor het ontwerpen en bouwen van een machine.

- Nadelen:
  - Hulpkloksignaal nodig.
  - Over het algemeen trager dan asynchrone schakelingen (kritieke pad).
  - Verbruikt meer energie dan asynchrone schakelingen

Synchrone systemen

- Een voorbeeld van een synchroon systeem is een teller:



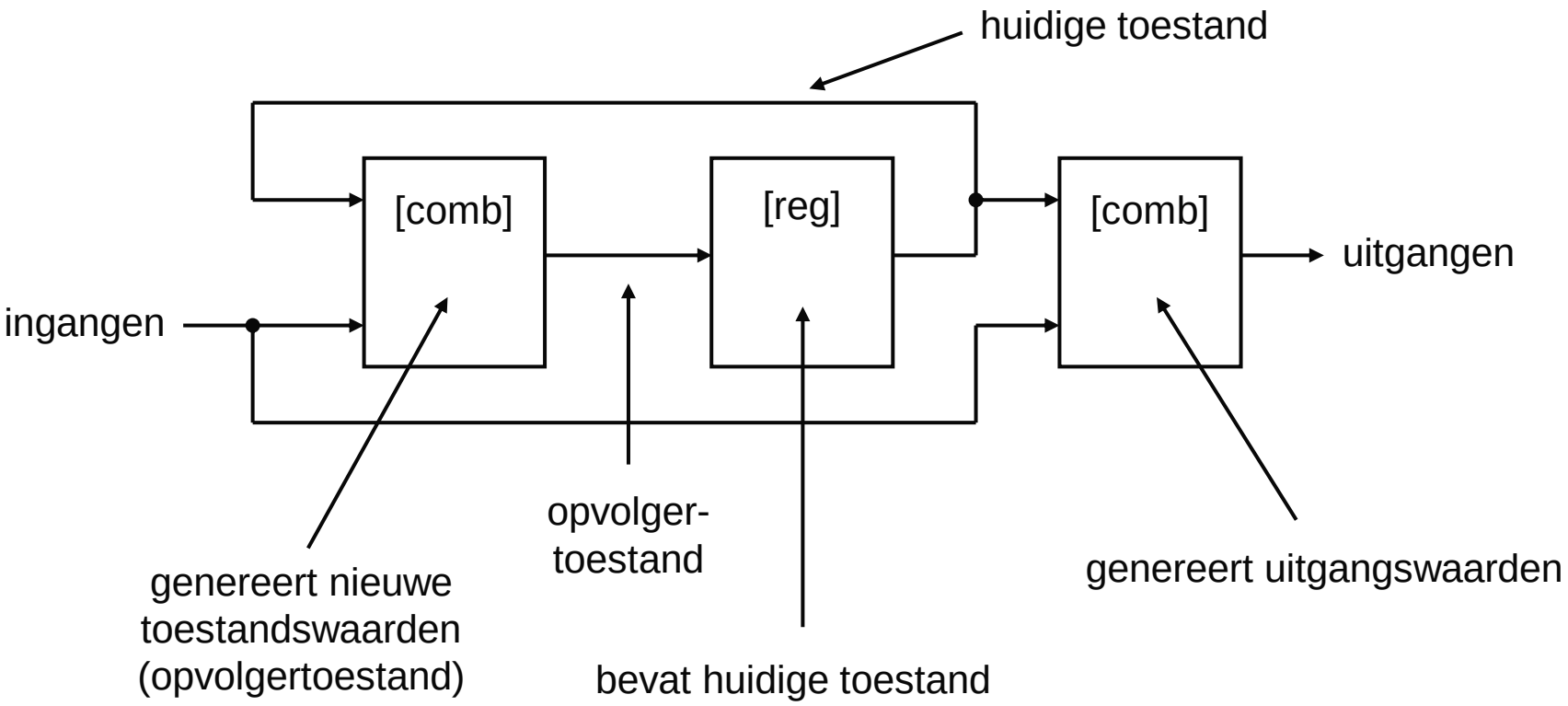
- Het register dient voor opslag van de (huidige) telstand.
- De opteller (+1) genereert de nieuwe telstand.
- tc geeft aan of de teller op de hoogste telstand staat.

Synchrone systemen

- Synchrone systemen worden gerealiseerd door middel van combinatorische logica en één of meer geheugenelementen.
- De geheugenelementen leggen de *huidige toestand* van het synchrone systeem vast.
- De ingangen, samen met de toestand, zorgen ervoor dat de toestand van het systeem verandert.
- De ingangen, samen met de toestand, zorgen ervoor dat de uitgangen een bepaalde waarde krijgen.

Synchrone systemen

- Hieronder de algemene structuur van een synchroon systeem.



Formele definitie

- Toestand

  De toestand van een schakeling legt het verleden van de schakeling vast voor zover dit van belang is voor het huidige gedrag en het gedrag in de toekomst.

- Toestandsgedrag

  Dit zijn de regels die vastleggen hoe de nieuwe toestand afhangt van de huidige toestand en de huidige ingangswaarden.

- Uitgangsgedrag

  Dit zijn de regels die vastleggen hoe de uitgangswaarden afhangen van de huidige toestand en de huidige ingangswaarden.

Formele definitie

- Ad Toestand

  De toestand van een schakeling wordt opgeslagen in *flipflops*. Deze flipflops staan onder directe besturing van het(zelfde) kloksignaal. Dit maakt timingverificatie eenvoudig en overzichtelijk.

- Ad Toestandsgedrag

  Het toestandsgedrag ligt vast in *combinatorische* logica die met de bekende methoden ontwikkeld kan worden. Deze logica heet volgende-toestands-logica (next state logic).

- Ad Uitgangsgedrag

  Ook het uitgangsgedrag ligt vast in *combinatorische* logica die met de bekende methoden ontwikkeld kan worden. Deze logica heet uitgangs-logica (output logic).

Toestandsmachines

- Algemeen bezitten synchrone systemen toestanden en worden ze *toestandsmachines* genoemd.
- Daarnaast is het aantal toestanden *eindig**.
- Praktische synchrone systemen zijn dus eindige toestandsmachines.
- Een veelgebruikte naam in technische literatuur is *Finite State Machine* (FSM). Een andere term is *automaat*.
- Er zijn twee typen: Mealy- en Moore-machines.

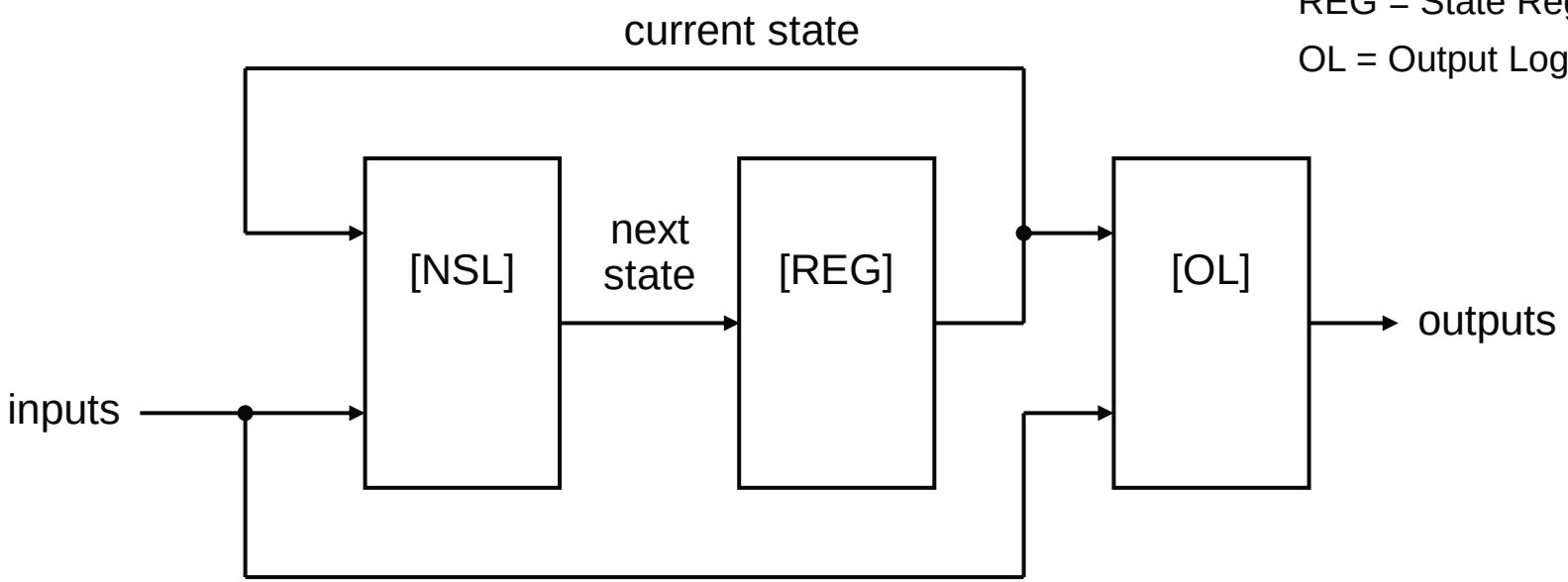
* Bestaan er dan ook machines met een oneindig aantal toestanden?

DE HAAGSE HOGESCHOOL

Mealy-machine

- Hieronder de structuur van een Mealy-machine.

NSL = Next State Logic
REG = State Register
OL = Output Logic

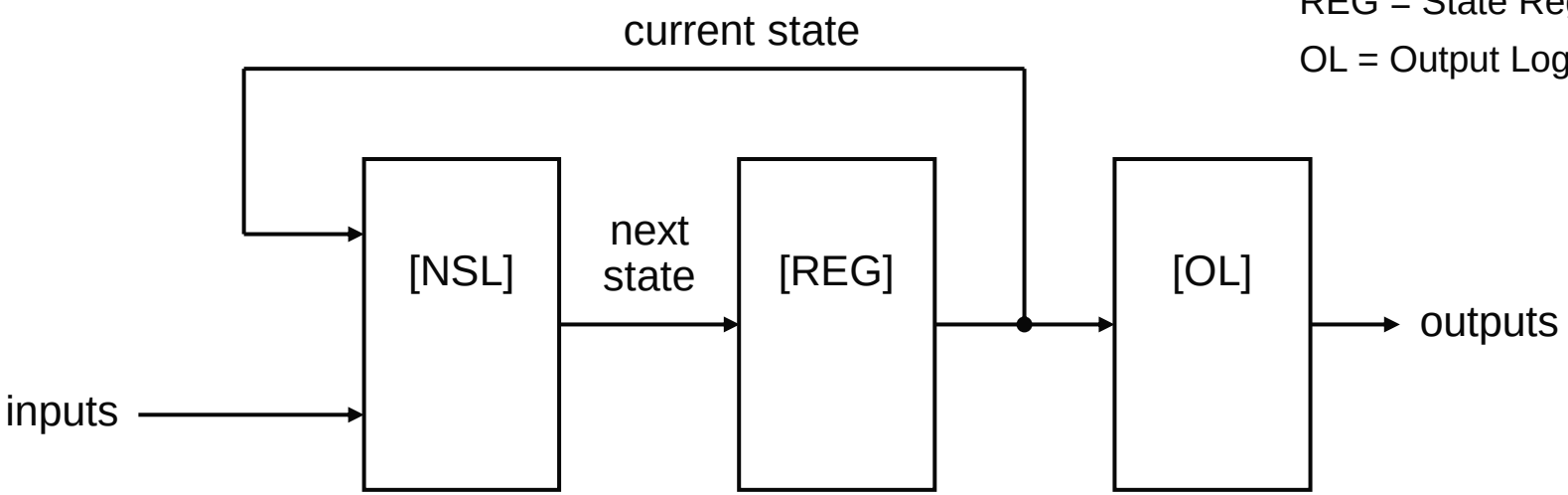


- Bij een Mealy-machine zijn de uitgangen afhankelijk van de toestand van de machine én de ingangen.

Moore-machine

- Hieronder de structuur van een Moore-machine.

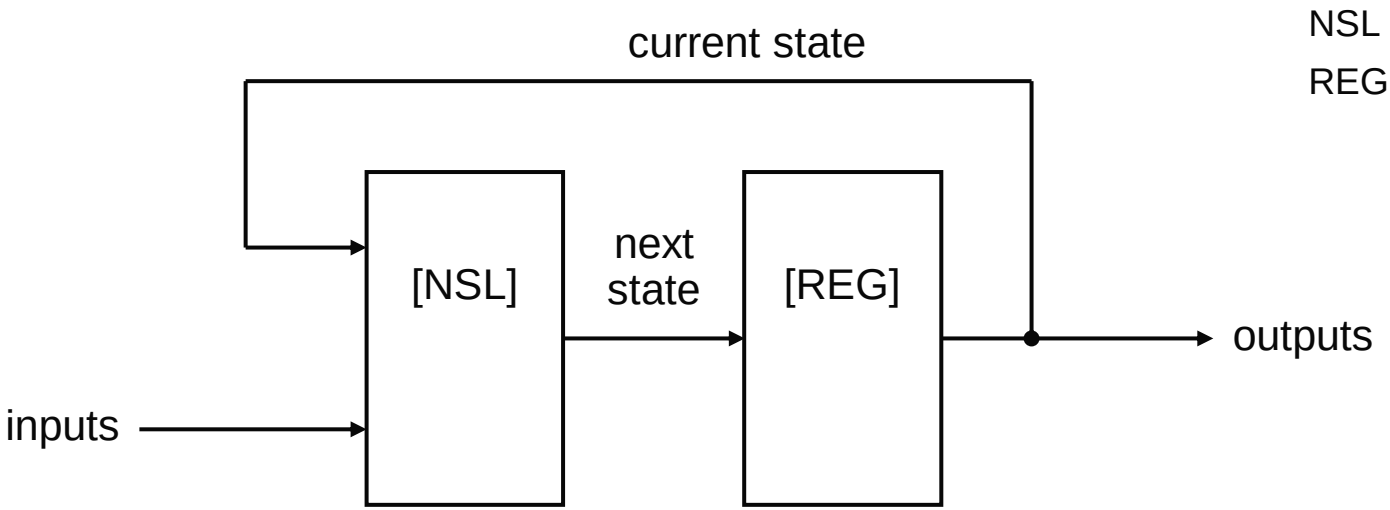
NSL = Next State Logic
REG = State Register
OL = Output Logic



- Bij een Moore-machine zijn de uitgangen alleen afhankelijk van de toestand van de machine. De Moore-machine is een vereenvoudiging van de Mealy-machine.

Medvedev-machine

- Hieronder de structuur van een Medvedev-machine.



- Bij een Medvedev-machine zijn de uitgangen direct gekoppeld aan het register. De Medvedev-machine is een vereenvoudiging van de Moore-machine.

Ontwerpprocedure

- Om tot een toestandsmachine te komen, moeten de volgende stappen genomen te worden:

  Opstellen van het toestandsdiagram

  Opstellen van de toestandstabel*

  Kiezen van toestandscodering

  Opstellen van de opvolgertoestand- en uitgangstabellen

  Functies voor opvolgertoestand en uitgangen opstellen

  Realiseren van de schakeling met logica

* Een geoefende ontwerper kan deze stap overslaan.

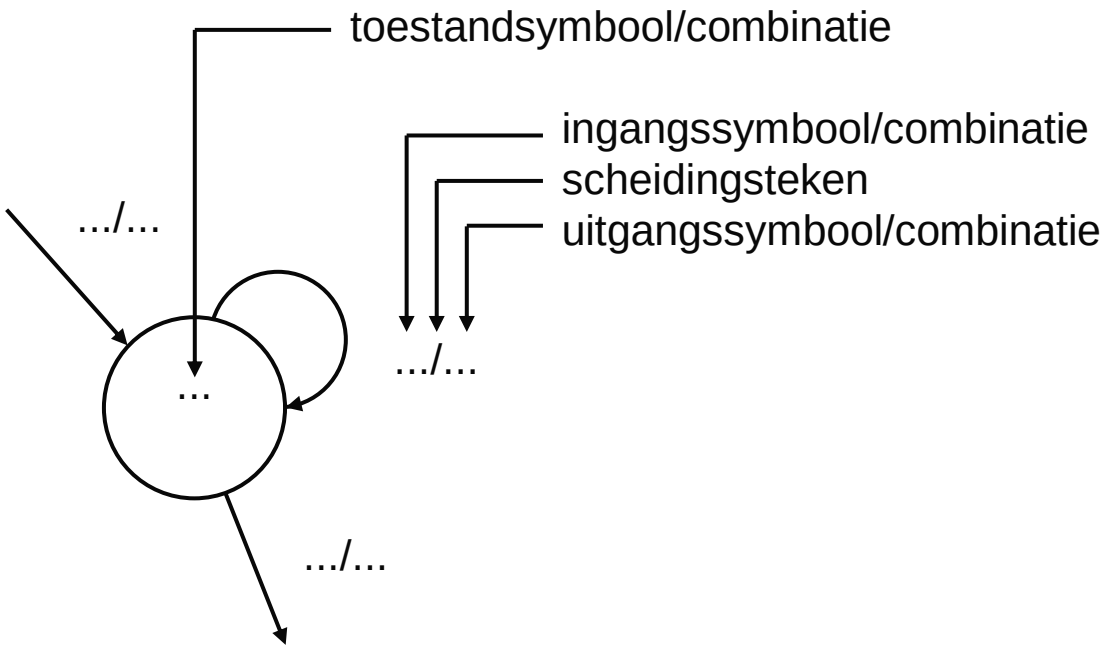
DE HAAGSE HOGESCHOOL

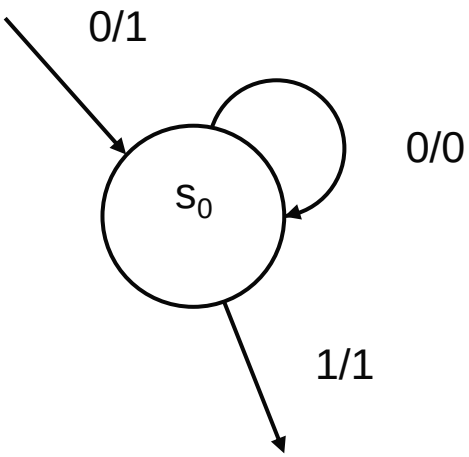
Toestandsdiagrammen

- De werking van een machine kan eenvoudig weergegeven worden met een toestandsdiagram.
- Het geeft aan wat de machine onder welke conditie gaat (moet gaan) doen.
- Een toestandsdiagram bestaat uit een aantal cirkels of bollen die de *toestanden* weergeven en pijlen die de overgangen en de *overgangscondities* (*transities*) van de ene naar de andere toestand aangeven.
- Er zijn twee beschrijvingswijzen, één voor Mealy-machines en één voor Moore-machines.

Toestandsdiagrammen Mealy

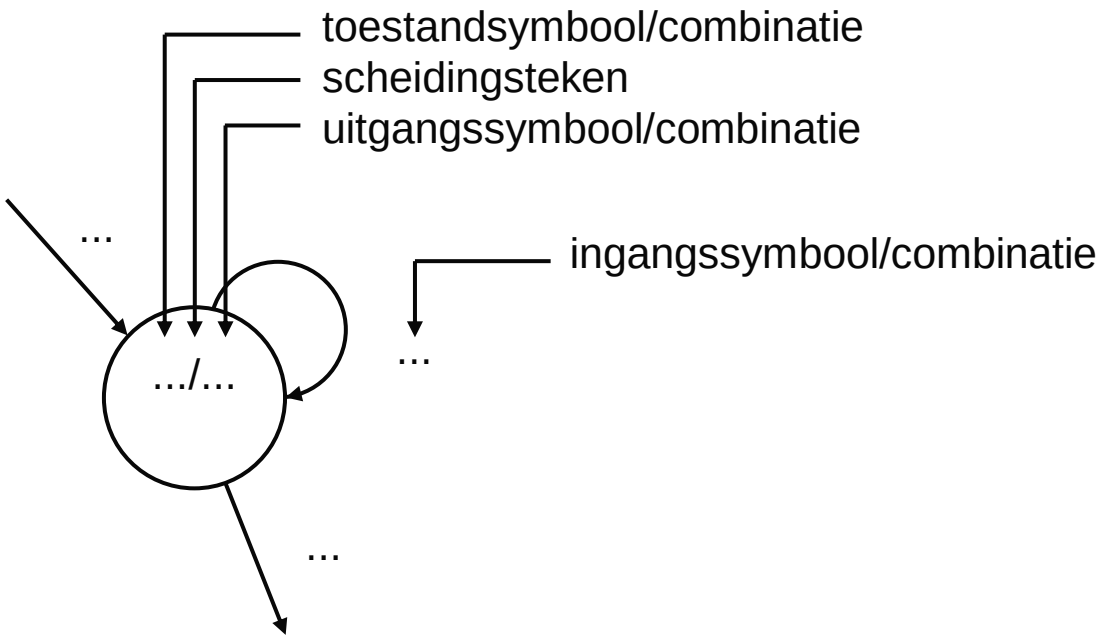
- Bij Mealy worden bij de overgangscondities de uitgangswaarden naast de ingangswaarden geschreven, de toestand in de cirkel.

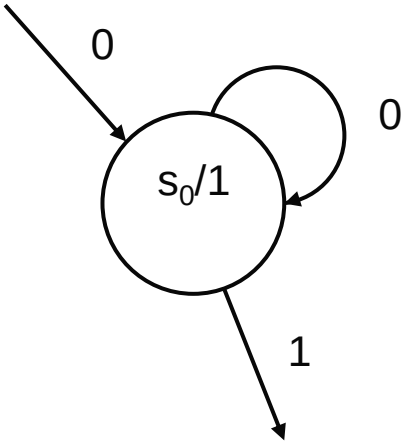




Toestandsdiagrammen Moore

- Bij Moore worden de uitgangswaarden naast de toestand geschreven, de ingangswaarden bij de overgangscondities.





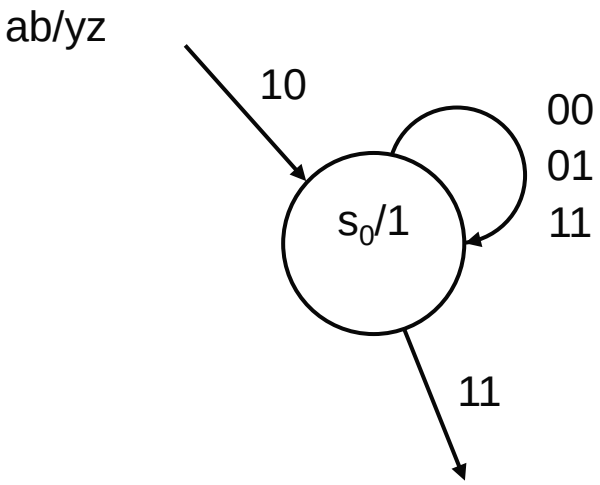
Toestandsdiagrammen

- Het ontwerpen een van toestandsdiagram (en dus de machine) begint bij het bepalen van het aantal toestanden dat nodig is en welke overgangsvoorwaarden nodig zijn.
- Dit volgt uit een (meestal) geschreven beschrijving.
- Hiervoor is geen vaste procedure. De ontwerper moet zelf bedenken wat de machine moet doen op basis van de beschrijving en goed nadenken over eventuele situaties die niet beschreven zijn.
- Er is altijd een begintoestand waar de machine in terecht komt na een reset-opdracht.

Toestandsdiagrammen

- Het ontwikkelde toestandsdiagram moet *volledig* en *niet-tegenstrijdig* zijn.

- Voor elke ingangscombinatie moet een overgang beschreven zijn, voor elke uitgang moet een waarde gedefinieerd worden (don't cares zijn mogelijk).

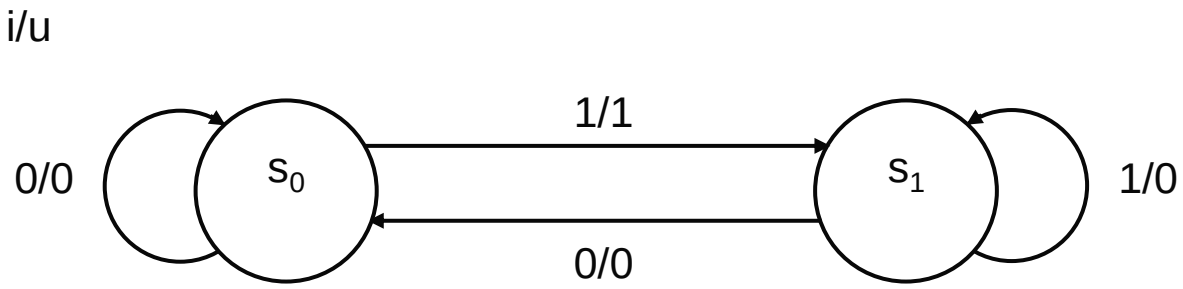


wat moet er gebeuren als
ab = 10 of ab = 11?
wat is de waarde van z?

Voorbeeld Mealy-specificatie

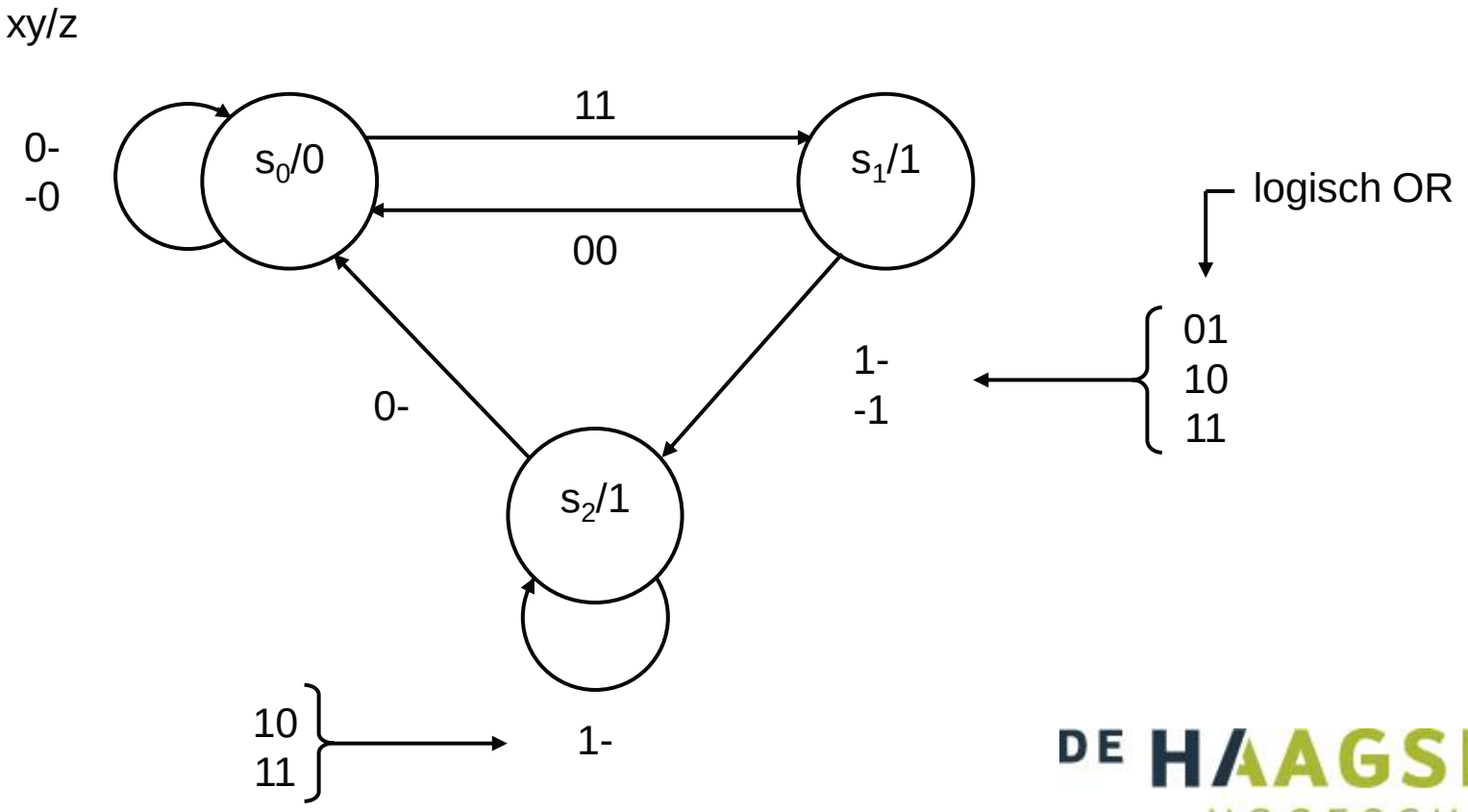
- Als voorbeelden volgen nu een Mealy en een Moore-machine. Deze machines worden gebruikt tot en met de synthese.

- De Mealy-machine heeft één ingang, één uitgang en twee toestanden.



Voorbeeld Moore-specificatie

- De Moore-machine heeft twee ingangen, één uitgang en 3 toestanden. Merk op dat bij sommige overgangscondities don't cares gegeven zijn.



Toestandstabellen

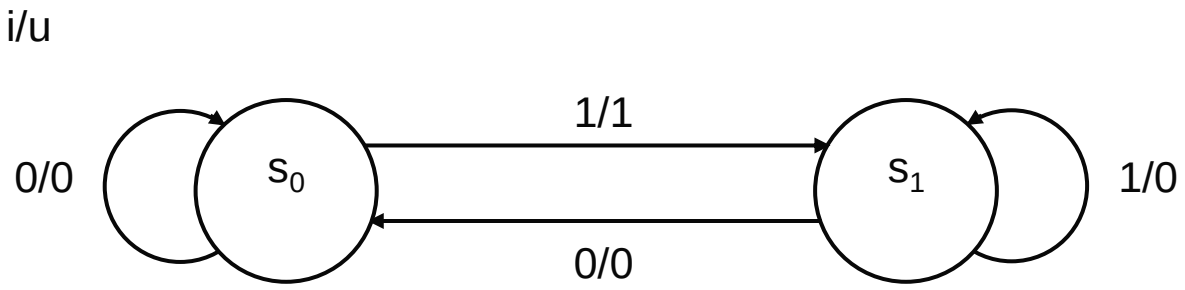
- Een alternatieve vorm van het beschrijven van een FSM kan met behulp van toestandstabellen.

- Hierin worden de huidige toestand, en de opvolgertoestand (next state) en uitgangswaarden als functie van de ingangswaarden beschreven.

- Bedenk dat bij een Mealy-machine de uitgang afhankelijk van de toestand én ingangswaarden is, bij een Moore-machine alleen van de toestand.

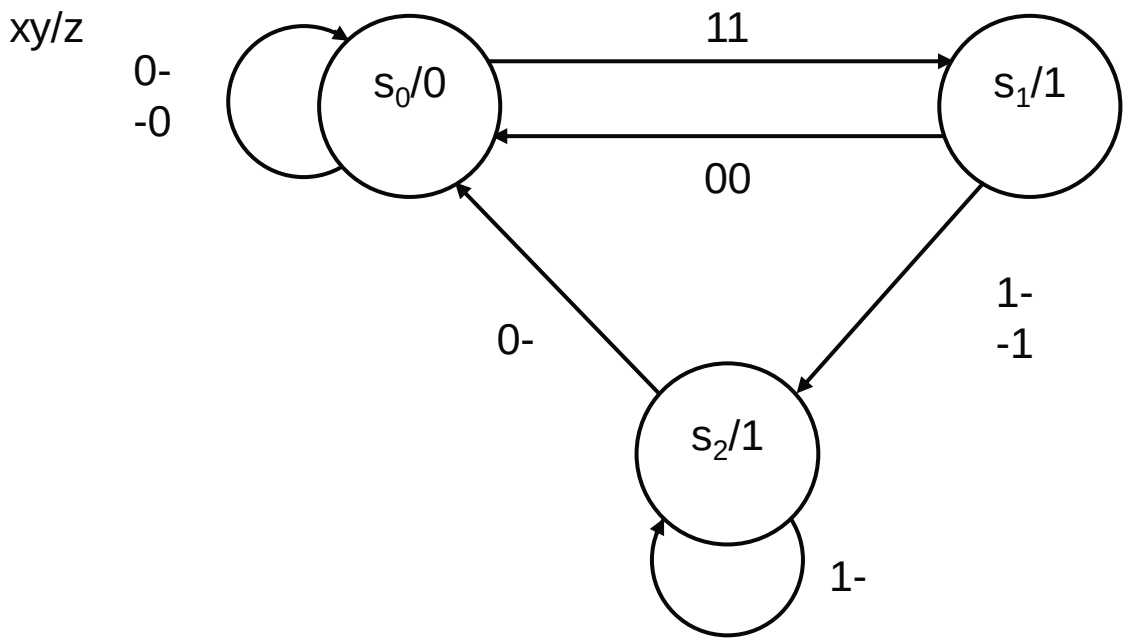
Toestandstabellen

- De Mealy-machine.



toestand	opvolger		uitgang	
	0	1	0	1
s_0	s_0	s_1	0	1
s_1	s_0	s_1	0	0

Toestandstabellen



toestand	opvolger				uitgang
	00	01	10	11	
s_0	s_0	s_0	s_0	s_1	0
s_1	s_0	s_2	s_2	s_2	1
s_2	s_0	s_0	s_2	s_2	1

Toestandscodering

- De volgende stap in het ontwerpproces is het toekennen van codecombinaties aan de toestanden.
- De toestanden moeten uniek gecodeerd worden dus bij n toestanden zijn tenminste $\lceil {}^{2}\log n \rceil$ *toestandsbits* nodig.
- Voor elke toestandsbit is een flipflop nodig.
- De keuze van de codering is vrij te kiezen zolang deze voor elke toestand uniek is.
- Er kunnen meer flipflops gebruikt worden dan strikt noodzakelijk zijn.

Toestandscodering

- Meest gebruikte coderingen zijn:
  - (zuiver) binaire telcode - voor tellers
  - Gray code - voor tellers (extra output logic)
  - One Hot - snelle systemen / systemen met veel flipflops (FPGA)

- Binaire telcode en Gray code gebruiken de minst aantal flipflops.

- One Hot code gebruikt voor elke toestand één flipflop.
  - De achterliggende gedachte is dat de omvang van de producttermen in de next state logic afneemt en hierdoor de maximale propagatietijd van de combinatoriek kleiner wordt. FPGA's hebben veel flipflops aan boord.

- Voordeel: veel don't cares, dus minimale logica.
- Nadeel: veel niet-gebruikte codecombinaties.

Toestands- en uitgangsfuncties

- Nadat de toestandscodering bepaald is, worden de functies voor de opvolgertoestand (*next state equations*) en de uitgangen (*output equations*) afgeleid.

- Uit de toestandstabel wordt een waarheidstabel opgesteld (met van links naar rechts):

  - de toestandscoderingen gecombineerd met de ingangscombinaties
  - de opvolgertoestanden
  - de instelsignalen voor de gebruikte flipflops (D, JK, SR, T)
  - de uitgangscombinaties

- Hier wordt alleen het gebruik van D-flipflops toegelicht.

Toestands- en uitgangsfuncties

- De tabel is vervolgens via de bekende procedures uit te werken tot een logische schakeling.

- De functies hebben de vorm:

$$D_i^n = f(Q_{k-1}^n .. Q_0^n, I_{m-1}^n .. I_0^n) \qquad (Q_i^{n+1} = D_i^n, \text{D - flipflop})$$

$$Z_j^n = f(Q_{k-1}^n .. Q_0^n, I_{m-1}^n .. I_0^n) \qquad \text{(Mealy)}$$

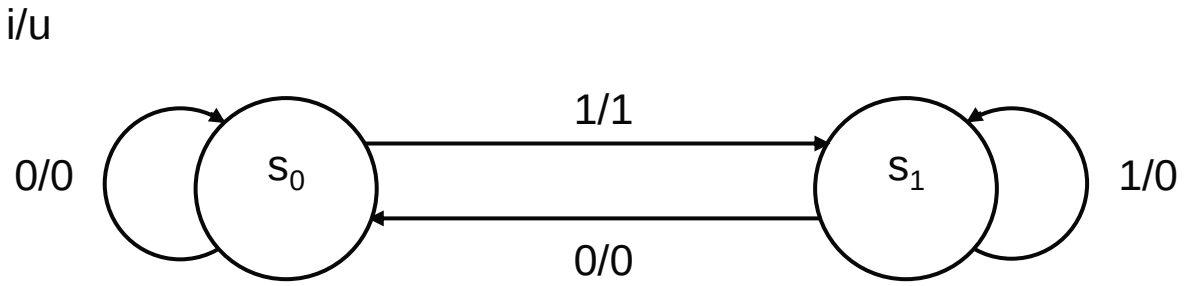
$$Z_j^n = f(Q_{k-1}^n .. Q_0^n) \qquad \text{(Moore)}$$

D_i^n is de ingangswaarde van de i^e flipflop tijdens klokcyclus n
Q_p^n is de waarde van de p^e flipflop tijdens klokcyclus n
I_p^n is de waarde van de p^e ingang tijdens klokcyclus n
Q_i^{n+1} is de waarde van de i^e flipflop na de volgende klokflank.
Z_j^n is de waarde van de j^e uitgang tijdens klokcyclus n
k is het aantal toestandsbits.
m is het aantal ingangen.

DE HAAGSE HOGESCHOOL

Toestands- en uitgangsfuncties

- Er zijn slechts twee toestanden dus één flipflop is genoeg om unieke codes toe te kennen. Toestand s_0 wordt gecodeerd met 0 en s_1 met 1.



Q^n	i^n	Q^{n+1}	D^n	u^n
0	0	0	0	0
0	1	1	1	1
1	0	0	0	0
1	1	1	1	0

Toestands- en uitgangsfuncties / schema

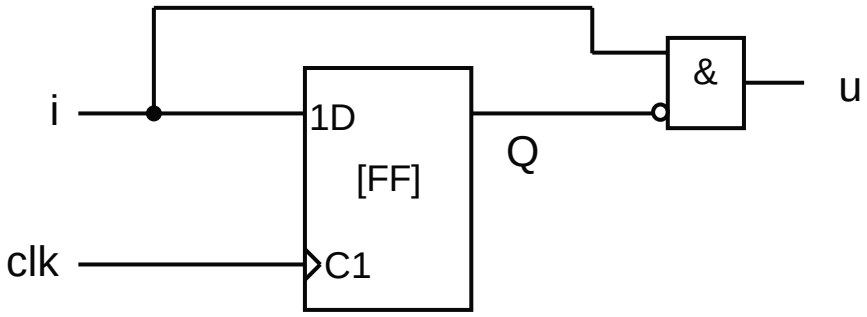
- Uit de waarheidstabel volgt direct de functie voor de toestanden:

$$D^n = i^n$$

- Ook is direct de functie voor de uitgang op te schrijven:

$$u^n = \overline{Q^n} \cdot i^n = [\overline{Q} \cdot i]^n$$

- Het schema is:



Toestands- en uitgangsfuncties

- De Moore-machine heeft drie toestanden. Er zijn twee flipflops nodig om de toestanden te coderen. Een voor de hand liggende code is:

  $s_0 = 00$
  $s_1 = 01$
  $s_2 = 10$

- De combinatie 11 wordt niet gebruikt. *De machine heeft ongebruikte toestanden*.

- Aangezien de machine twee ingangen en twee toestandsbits heeft, zal de waarheidstabel uit 16 regels bestaan.

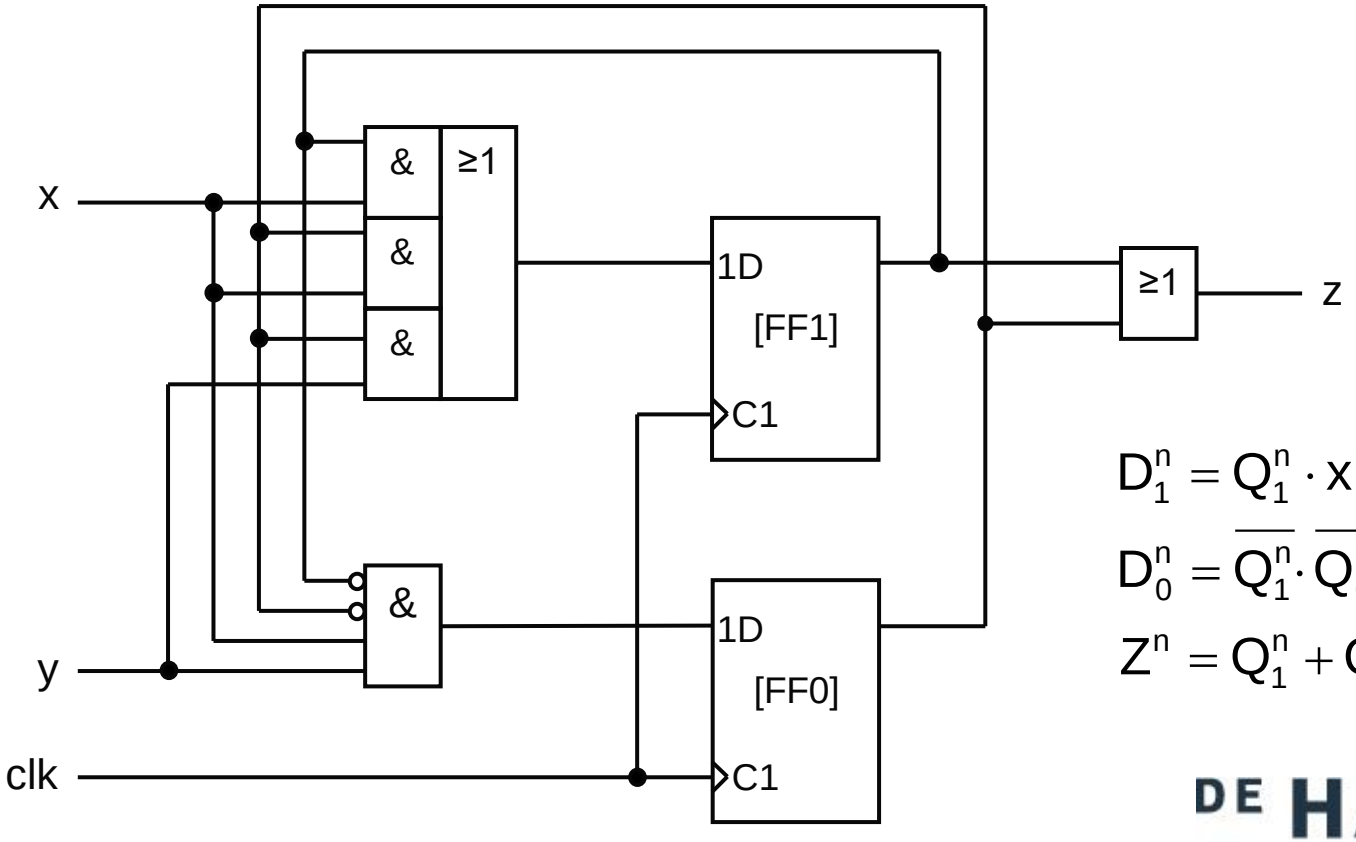
Toestands- en uitgangsfuncties

	Q_1^n	Q_0^n	x^n	y^n	Q_1^{n+1}	Q_0^{n+1}	D_1^n	D_0^n	z^n
s_0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	0	0
	0	0	1	0	0	0	0	0	0
	0	0	1	1	0	1	0	1	0
s_1	0	1	0	0	0	0	0	0	1
	0	1	0	1	1	0	1	0	1
	0	1	1	0	1	0	1	0	1
	0	1	1	1	1	0	1	0	1
s_2	1	0	0	0	0	0	0	0	1
	1	0	0	1	0	0	0	0	1
	1	0	1	0	1	0	1	0	1
	1	0	1	1	1	0	1	0	1
niet gebruikt	1	1	0	0	-	-	-	-	-
	1	1	0	1	-	-	-	-	-
	1	1	1	0	-	-	-	-	-
	1	1	1	1	-	-	-	-	-

DE HAAGSE HOGESCHOOL

Toestands- en uitgangsfuncties / schema

- Met de bekende minimalisatieprocedure worden de volgende functies gevonden:



$$D_1^n = Q_1^n \cdot x^n + Q_0^n \cdot y^n + Q_0^n \cdot x^n$$

$$D_0^n = \overline{Q_1^n} \cdot \overline{Q_0^n} \cdot x^n \cdot y^n$$

$$Z^n = Q_1^n + Q_0^n$$

Toestands- en uitgangsfuncties

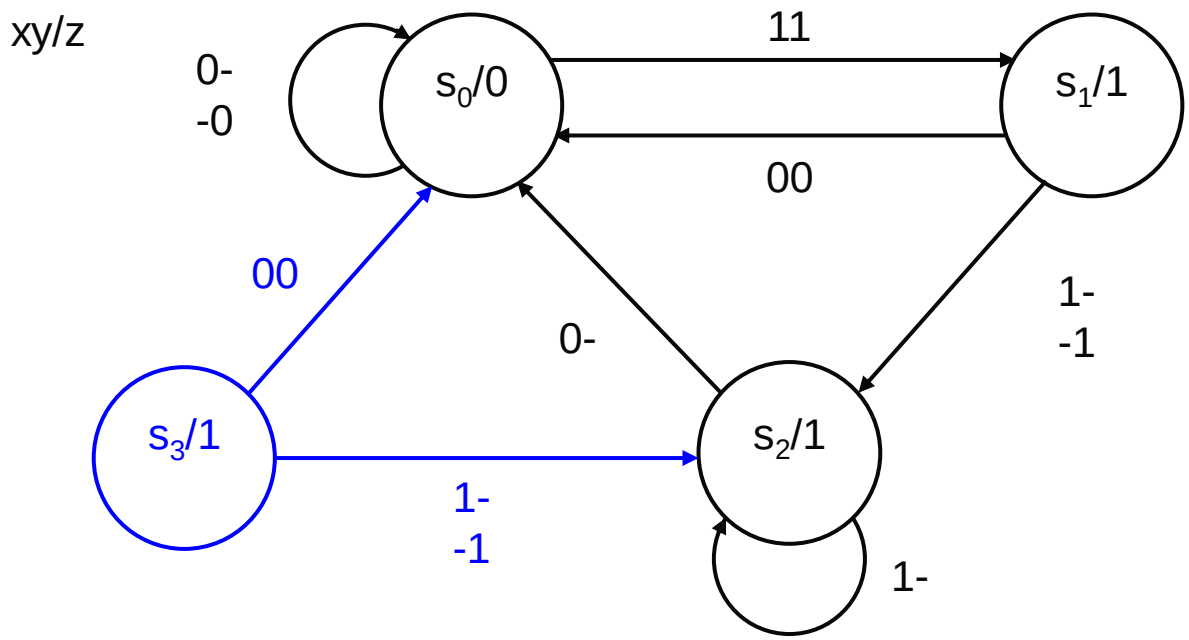
- Eén toestand is niet gebruikt: $Q_1Q_0 = 11$.
- Wat doet de machine als deze per ongeluk in de ongebruikte toestand terecht komt?
- Na onderzoek van de toestandsfuncties komt er de volgende functietabel uit (als $Q_1Q_0 = 11$):

x	y	naar	z
0	0	s_0	1
0	1	s_2	1
1	0	s_2	1
1	1	s_2	1

De uitgang van de schakeling blijft 1!

Toestands- en uitgangsfuncties

- Het volledige toestandsdiagram is (na uitwerking toestandsfuncties):

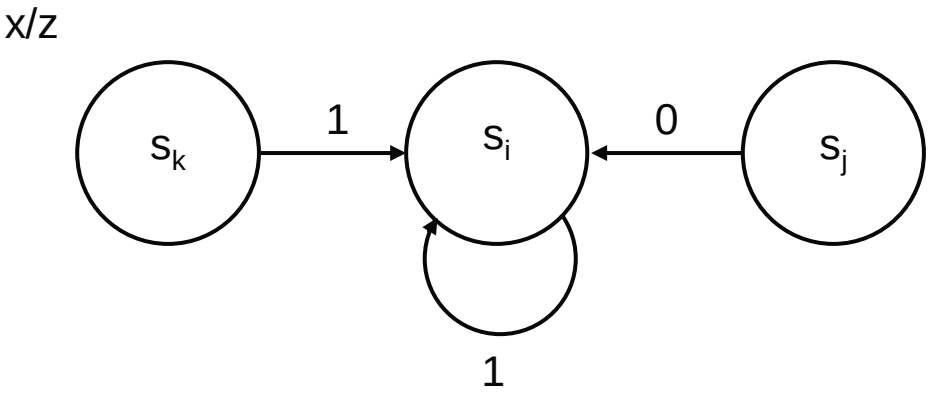


- Het is dus te voorspellen wat de machine gaat doen in ongebruikte toestanden.

One-hot toestandstoekenning

- Het vinden van de toestandsfuncties bij one-hot is eenvoudig.

- In toestand s_i: bekijk alle overgangen van toestanden naar s_i. Elke overgang levert een productterm voor de functie van s_i. De som van deze producttermen is de functie voor s_i.



$$D_i^n = Q_k \cdot x + Q_i \cdot x + Q_j \cdot \overline{x}$$

Opgaven

- Geef de toestand- en uitgangsfuncties van de JK-flipflop. Gebruik de binaire telcode voor toestandscodering.
- Geef de toestand- en uitgangsfuncties van de synchrone 6-teller op. Gebruik de binaire telcode voor toestandscodering.
- Geef de toestand- en uitgangsfuncties van de RG-machine op. Gebruik de binaire telcode voor toestandscodering.
- Geef de toestand- en uitgangsfuncties van de RG-machine op. Gebruik de one-hot-codering voor toestandscodering.

Reset-implementatie

- Een resetingang behoort tot de noodzakelijke faciliteiten van een digitale schakeling.
- Bij het aanzetten is de toestand van een schakeling meestal niet gedefinieerd.
- Een power-on-reset zorgt er dan voor dat de schakeling in een van te voren aangewezen toestand terecht komt: de resettoestand.
- Een reset kan op verschillende manieren ingebouwd worden:

Reset-implementatie

- Via combinatoriek

  Het resetsignaal wordt in feite behandeld als een gewoon datasignaal. Dit heet een synchrone reset of synchrone clear.

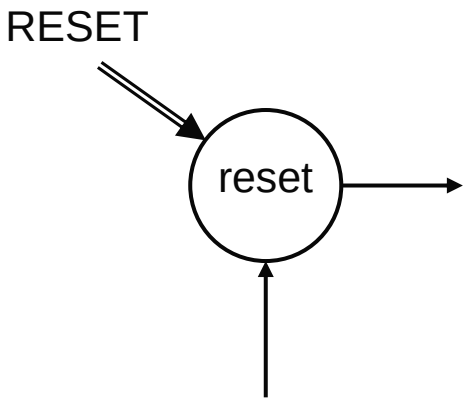
- Rechtstreeks

  Het resetsignaal wordt direct op de flipflops gezet. Dit heet een asynchrone reset.

- Veel (oudere) IC's hebben een asynchrone resetingang. Deze werkt buiten de klok om. Een asynchrone reset is sterk storingsgevoelig en moet met zorg gebruikt worden.

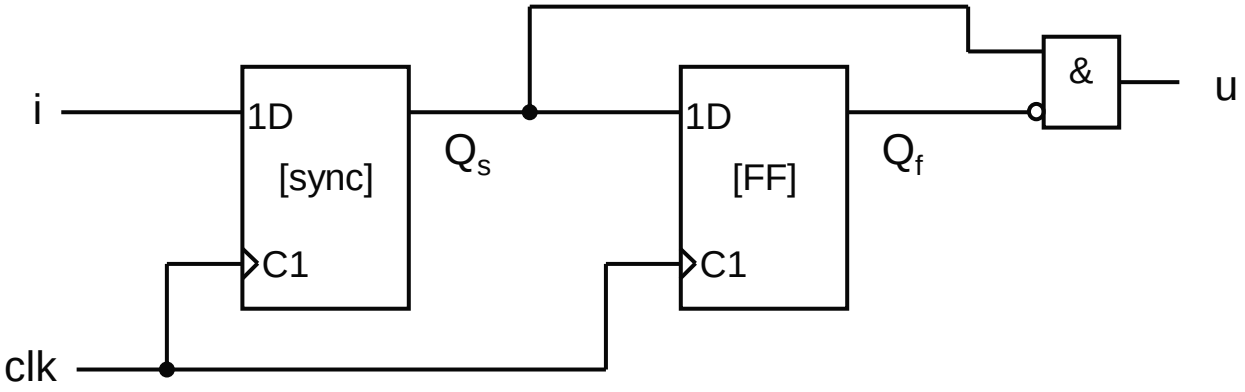
Reset-implementatie

- Bij machines wordt vaak een dominante reset ingebouwd.
- Vanuit elke toestand wordt direct naar de resettoestand gesprongen indien de reset geactiveerd wordt.
- Om het aantal pijlen in het toestandsdiagram te beperken wordt dit meestal aangegeven met een dubbele of dikke pijl.



Synchronisatie ingangen

- Synchronisatie van ingangen levert altijd een Moore-machine op.



- De machine heeft nu één toestandsbit extra.

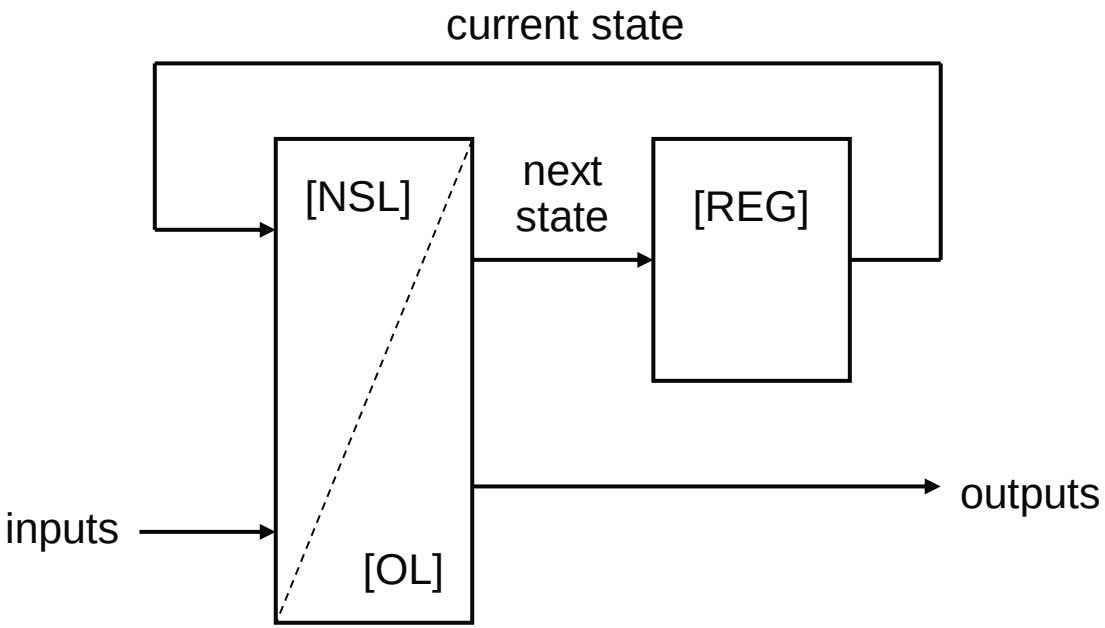
$$D_s^n = i^n$$

$$D_f^n = Q_s^n$$

$$u^n = \overline{Q_f^n} \cdot Q_s^n$$

Alternatieve representatie

- De Mealy-machine is ook als volgt te tekenen. De Next State Logic en Output Logic zijn hierbij gecombineerd tot één blok combinatoriek.

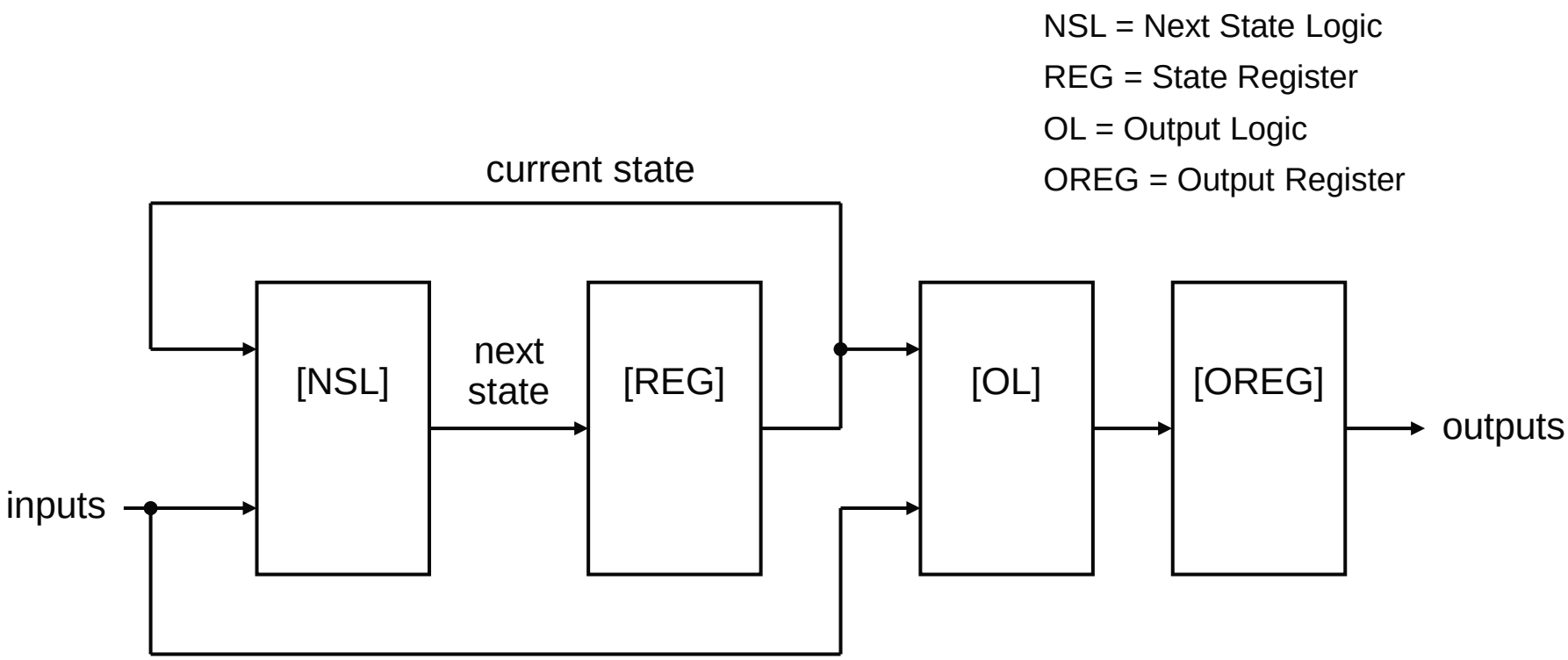


Registered outputs

- Soms is het handig om achter de uitgangen nog een register te plaatsen.
- De uitgangswaarden kunnen namelijk meerdere malen veranderen direct na de actieve klokflank.
- Bij het aanbieden van deze uitgangen aan een systeem met een ander kloksignaal kan dit problemen geven (o.a. metastabiliteit).
- Het register moet dan wel *single transition* zijn.
- Andere namen: synchrone uitgangen, clocked outputs, pipelined outputs.

Registered outputs

- Nadeel: de uitgangswaarden zijn één klokcyclus later beschikbaar.



Toestandsmachines in VHDL

- Voor het beschrijven van toestandsmachines wordt uitgegaan van het drie-proces-model of twee-proces-model:

- Drie-proces-model

  Een proces voor het bepalen van nieuwe toestand (combinatoriek).

  Een proces voor opslag van de (huidige) toestand (sequentiëel)

  Een proces voor het bepalen van de uitgangswaarden (combinatoriek).

- Twee-proces-model

  Een proces voor het bepalen van nieuwe toestand en de uitgangswaarden (combinatoriek).

  Een proces voor opslag van de (huidige) toestand (sequentiëel)

Toestandsmachines in VHDL

- De algemene opzet van het drie-proces-model is als volgt:

```
architecture fsm_3proc of machine is
begin
     combstate: process (current_state, inputs) is
     -- beschrijving opvolgertoestand (NSL)
     end process;

     reg: process (clk, areset) is
     -- beschrijving state register (REG)
     end process;

     combout: process (current_state, inputs) is
     -- beschrijving uitgangen (OL)
     end process;
end fsm_3proc;
```

DE HAAGSE HOGESCHOOL

Toestandsmachines in VHDL

- De algemene opzet van het twee-proces-model is als volgt:

```
architecture fsm_2proc of machine is
begin
     comb: process (current_state, inputs) is
     -- beschrijving opvolgertoestand (NSL)
     -- beschrijving uitgangen (OL)
     end process;

     reg: process (clk, areset) is
     -- beschrijving state register (REG)
     end process;

end fsm_2proc;
```

Toestandsmachines in VHDL

- Eén van de stappen uit het ontwerpproces is het toekennen van codecombinaties aan de toestanden.

- VHDL kent de mogelijkheid een eigen type te definiëren waarmee dit symbolisch kan worden gedaan:

```
type state_type is (s0, s1, s2, got_one, bingo, power_on);
signal current_state, next_state : state_type;
```

- Toekenningen en testen kunnen gedaan worden op het type:

```
if current_state = s2 then
    ...
    next_state <= bingo;
```

Toestandsmachines in VHDL

- Bij (functionele) simulatie worden dan de elementen afgebeeld.

```
type state_type is (s0, s1, s2, gotone, bingo, power_on);
signal current_state, next_state : state_type;
```

- Let erop dat bij het initialiseren van de simulatie-omgeving de signalen `current_state` en `next_state` de meeste linkse element krijgen!

- Bij synthese kan de synthesizer zelf een codering bepalen, bijvoorbeeld zuiver binair of one-hot. Dit is als optie in te stellen. De codering kan ook expliciet door de gebruiker opgegeven worden.

Toestandsmachines in VHDL

- De entity-declaratie van de Mealy-machine.

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

entity mealy_example is
    port (clk:    in std_logic;
          areset: in std_logic;
          i:      in std_logic;
          u:      out std_logic
         );
end mealy_example;

...
```

Toestandsmachines in VHDL

- Type-definitie van de toestanden en declaratie toestanden

```
architecture fsm_3proc of mealy_example is

-- State types
type state_type is (s0, s1);

-- Internal current state and next state
signal current_state, next_state : state_type;

begin
   ...
```

Toestandsmachines in VHDL

- Hieronder de VHDL-beschrijving van de toestandsfuncties:

```
combstate: process (current_state, i) is
begin
   case current_state is
      when s0 => if i = '0' then
                     next_state <= s0;
                 else
                     next_state <= s1;
                 end if;
      when s1 => if i = '0' then
                     next_state <= s0;
                 else
                     next_state <= s1;
                 end if;
      when others => next_state <= s0;
   end case;
end process;
```

Toestandsmachines in VHDL

- Hieronder de VHDL-beschrijving van het toestandsregister:

```
reg: process (clk, areset) is
begin
   if areset = '1' then
      current_state <= s0;
   elsif rising_edge(clk) then
      current_state <= next_state;
   end if;
end process;
```

Toestandsmachines in VHDL

- Hieronder de VHDL-beschrijving van de uitgangsfuncties:

```
combout: process (current_state, i) is
begin
   case current_state is
      when s0 => if i = '0' then
                     u <= '0';
                 else
                     u <= '1';
                 end if;
      when s1 =>     u <= '0';
      when others => u <= 'X';
   end case;
end process;

end architecture;
```

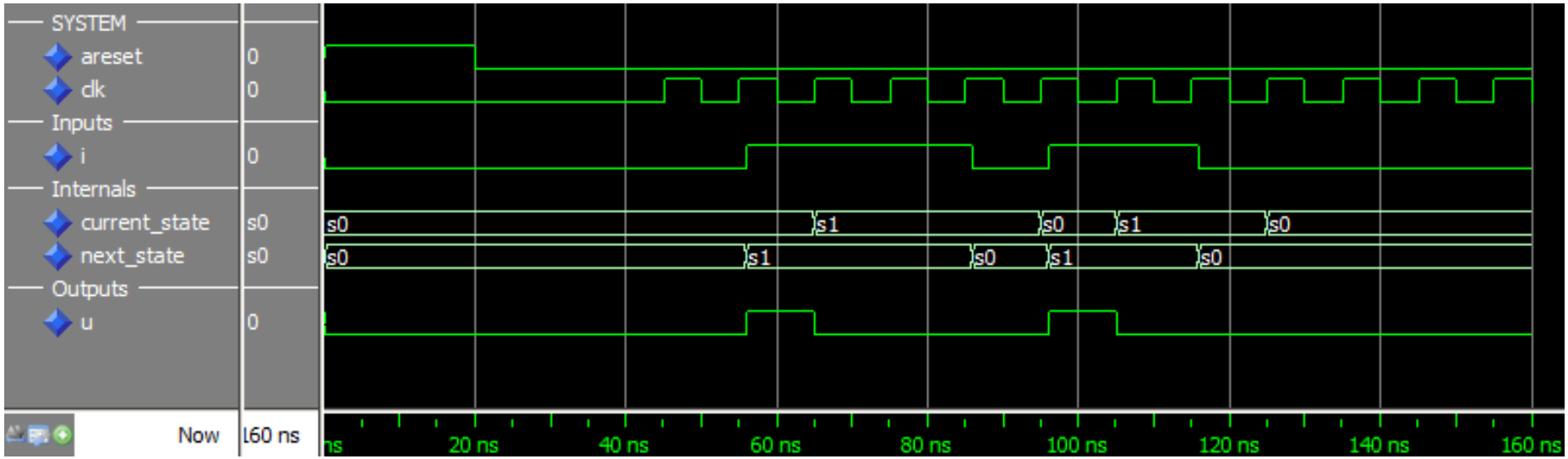
Toestandsmachines in VHDL

- Het proces voor de toestandsfuncties en de uitgangswaarden in het twee-proces-model:

```
comb: process (current_state, i) is
begin
   case current_state is
      when s0 => if i = '0' then
                     next_state <= s0; u <= '0';
                 else
                     next_state <= s1; u <= '1';
                 end if;
      when s1 => if  i = '0' then
                     next_state <= s0; u <= '0';
                 else
                     next_state <= s1; u <= '0';
                 end if;
      when others => next_state <= s0; u <= 'X';
   end case;
end process;
```

Simulatieresultaat Mealy-machine

- Hieronder het simulatieresultaat. Eerst wordt het systeem gereset. Daarna wordt een reeks enen en nullen aangeboden.



Toestandscodering revisited

- Het aantal verschillende toestandscoderingen* wordt gegeven door

$$N_D = \frac{2^k!}{(2^k - n)!k!}$$

N_D: het aantal verschillende coderingen

n: het aantal toestanden

k: het aantal toestandsbits

- Als voorbeeld: voor een machine met drie toestandbits en vijf toestanden zijn 1120 verschillende coderingen mogelijk.

- Om de meest optimale codering te vinden, moeten alle combinaties onderzocht worden.

* Zie [Thijssen], pag. 211.

Toestandscodering revisited

- De keuze van de codering heeft directe consequenties voor de omvang van de combinatorische logica.
- De praktijk leert echter een aantal “optimale” coderingen voor bepaalde klassen van machines.
- Thijssen heeft onderzocht dat wanneer een groot aantal willekeurig gekozen coderingen wordt gebruikt, de verhouding tussen de codering met het minst aantal poorten (*gate equivalent*) en de codering met het meest aantal poorten een factor 1/3 – 1/4 bedraagt.
- Dus: probeer een tiental willekeurige combinaties en selecteer die met de minst aantal poorten.

Toestandscodering Moore-machine

- Eerder zijn de functies uitgewerkt voor de codering $S_0 = 00$, $S_1 = 01$ en $S_2 = 10$. Dit levert de volgende functies op:

$$D_1^n = Q_1^n \cdot x^n + Q_0^n \cdot y^n + Q_0^n \cdot x^n$$

$$D_0^n = \overline{Q_1^n} \cdot \overline{Q_0^n} \cdot x^n \cdot y^n$$

$$Z^n = Q_1^n + Q_0^n$$

- Bij een codering van $S_0 = 11$, $S_1 = 10$ en $S_2 = 00$ worden de functies

$$D_1^n = Q_0^n + \overline{Q_1^n} \cdot \overline{x^n} + \overline{x^n} \cdot \overline{y^n}$$

$$D_0^n = Q_0^n \cdot \overline{x^n} + Q_0^n \cdot \overline{y^n} + \overline{Q_1^n} \cdot \overline{x^n} + \overline{x^n} \cdot \overline{y^n}$$

$$Z^n = \overline{Q_1^n}$$

- De tweede codering levert meer hardware (= transistoren) op.

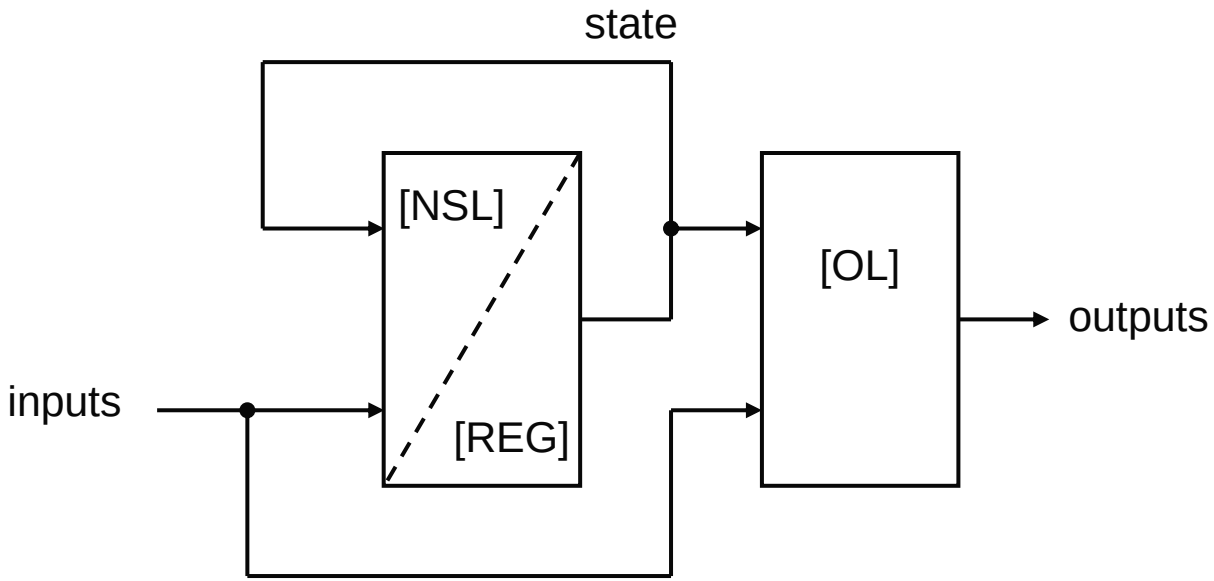
Toestandscodering in VHDL

- Het is mogelijk om in de VHDL-beschrijving de toestandscodering op te nemen. Hiervoor wordt een *pragma* gebruikt.

```
architecture fsm_3proc of moore_example is
-- State types
type state_type is (s0, s1, s2);
-- !!!FORCE STATE ENCODING!!!
attribute ENUM_ENCODING : string;
attribute ENUM_ENCODING of state_type : type is "00 01 10";
--attribute ENUM_ENCODING of state_type : type is "11 10 00";
--attribute ENUM_ENCODING of state_type : type is "100 010 110";
--attribute ENUM_ENCODING of state_type : type is "001 010 100";
-- Internal current state and next state
signal current_state, next_state : state_type;
```

Alternatieve twee-proces-model

- Het is mogelijk om de NSL en het toestandsregister te combineren tot één proces. Het voordeel hiervan is dat er maar één *signal* gebruikt wordt om de toestand bij te houden.



Toestandsmachines in VHDL

- De algemene opzet van het twee-proces-model is als volgt:

```
architecture fsm_2proc_alt of machine is
begin
     combreg: process (clk, areset) is
     -- beschrijving opvolgertoestand onder klokflanksturing (NSL/REG)
     end process;

     comb: process (state, inputs) is
     -- beschrijving uitgangen (OL)
     end process;

end fsm_2proc_alt;
```

Alternatieve VHDL-codering

```
-- only the NSL and REG
combreg: process (clk, areset) is
begin
   if areset = '1' then
      state <= s0;   -- the reset state
   elsif rising_edge(clk) then
      case state is
         when s0 => if i = '0' then state <= s0;
                    else state <= s1; end if;
         when s1 => if i = '0' then state <= s0;
                    else state <= s1; end if;
         when others => state <= s0;
      end case;
   end if;
end process;
```

Opgaven

- De Moore-machine heeft drie toestanden en dus minimaal twee toestandsbit nodig. Bepaal het aantal mogelijke codecombinaties.

- Bepaal het aantal mogelijke codecombinaties voor een machine met 10 toestanden en een minimum aan toestandsbits.

- Bepaal het Next State en Output functies van de Moore-machine met de volgende toestandscodering:

  $S_0 = 11$, $S_1 = 10$ en $S_2 = 00$
  $S_0 = 00$, $S_1 = 11$ en $S_2 = 10$

Referenties

- http://www.cs.cmu.edu/afs/cs/academic/class/15671-f95/www/handouts/sm-basics/node8.html
- http://www.ee.usyd.edu.au/tutorials/digital_tutorial/part3/t-diag.htm
- http://en.wikipedia.org/wiki/Finite-state_machine

- Digitale Techniek, Deel 2 – A.P. Thijssen ea – 5e druk, 2000, ISBN 90-407-1838-5
- Fundamentals of Digital Logic with VHDL Design – Brown, 3rd Ed, 2008, ISBN 9780071268806
- Digital Design: Principles and Practices – John F. Wakerly, 4th Ed, 2006, ISBN 0-13-173349-4
- “A Note on the Number of Internal Variable Assignments for Sequential Switching Circuits”, E.J. McCluskey, S.H. Unger, Electronic Computers, IRE Transactions on (Volume EC-8, Issue: 4), DOI: 10.1109/TEC.1959.5222055

Academie voor Technology, Innovation & Society Delft

Academie voor ICT & Media

De Haagse Hogeschool, Delft
+31-15-2606311
J.E.J.opdenBrouw@hhs.nl
www.dehaagsehogeschool.nl

DE HAAGSE HOGESCHOOL