Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Digitale Systeem Engineering 1

Week 4 – Schuifregisters, tellers

Jesse op den Brouw

DIGSE1/2018-2019

DE HAAGSE HOGESCHOOL

Schuifregisters

- In de digitale techniek en met name in de digitale communicatie wordt veel gebruik gemaakt van *seriële transmissie*.

- Het voordeel ten opzichte van *parallelle transmissie* is overduidelijk: er is slechts een beperkt aantal verbindingen nodig om informatie (bits) te verzenden of te ontvangen.

- Het nadeel is dat het versturen van een byte of word veel langer duurt dan bij parallelle transmissie.

Schuifregisters

- Voorbeelden van seriële transmissiesystemen*:*

  RS232 (a.k.a. COM-poort in de PC)
  Ethernet
  I^2C (a.k.a. Two Wire Interface)
  SPI
  Infrarood afstandsbediening
  USB
  SATA
  CD/DVD

Schuifregisters

- Bij het gebruik van seriële transmissie worden schuifregisters gebruikt.
- Bij *verzenden* van data gaat het omzetten van parallel naar serieel als volgt:

  Eerst wordt de (parallelle) data *geladen*.
  Daarna wordt de data serieel *naar buiten geschoven*.

- Dit heet PISO: Parallel In Serial Out

DE HAAGSE HOGESCHOOL

Schuifregisters

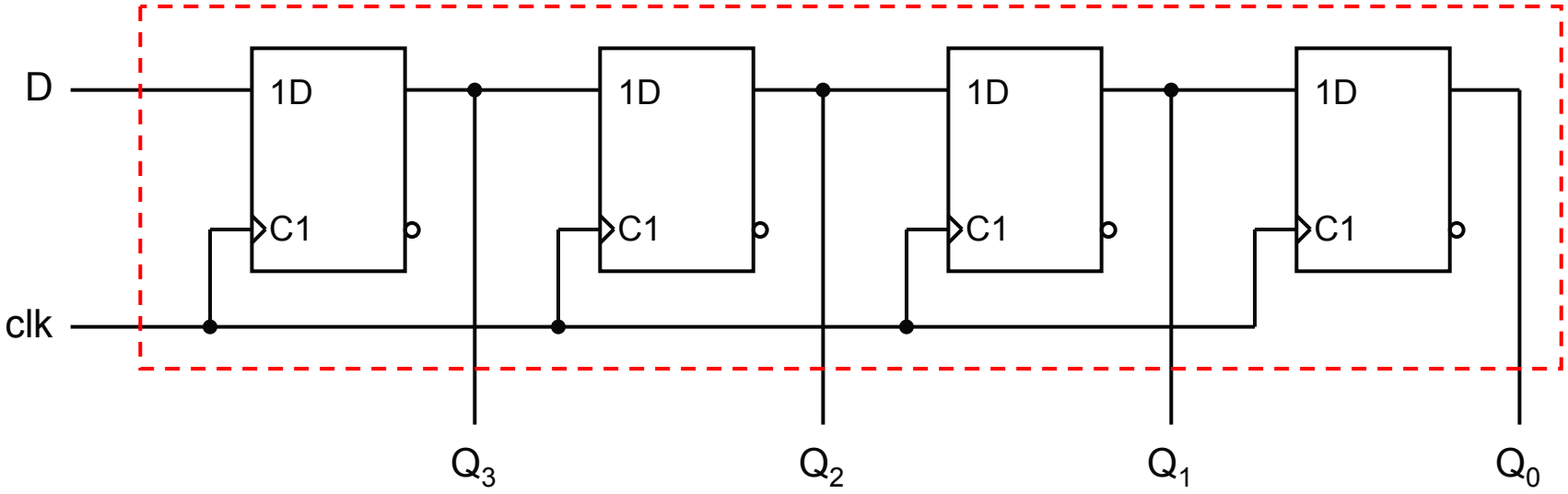
- Bij *ontvangen* van data gaat het omzetten van serieel naar parallel als volgt:

  Eerst wordt de data serieel *naar binnen geschoven.*
  Daarna wordt de (parallelle) data *beschikbaar gesteld*.

- Dit heet SIPO: Serial In Parallel Out

Schuifregister

- Een schuifregister is een groep (D-)flipflops waarvan de data-uitgang van een flipflop is verbonden met de data-ingang van de naastgelegen flipflop. Schuifregisters spelen een belangrijke rol in seriële communicatie. Hieronder een Serial-In-Parallel-Out (SIPO)-schuifregister.



Schuifregisters

- De grote vraag is altijd:

  Welk bit wordt als eerste naar buiten (of binnen) geschoven?

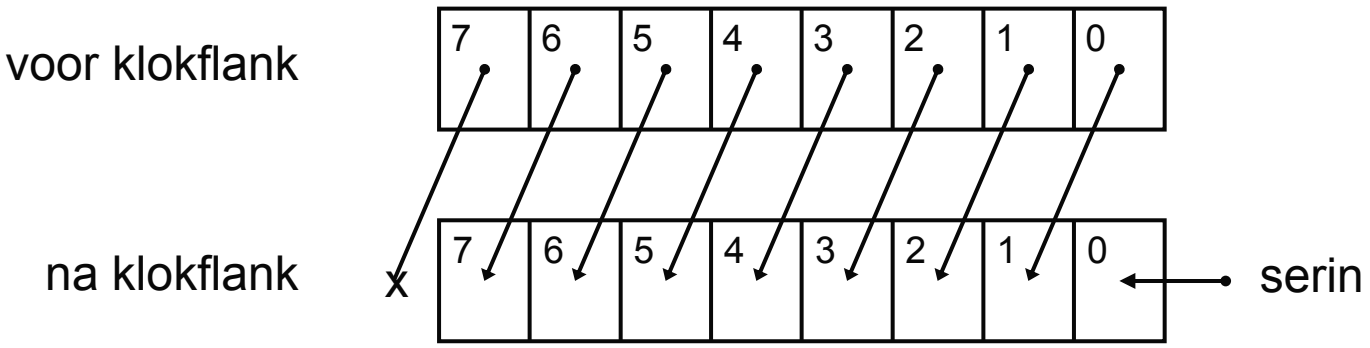
- Dat is systeemafhankelijk:

  RS232 – minst significante bit eerst
  Ethernet – meest significante bit eerst.

- Goed opletten dus.

Schuifregister in VHDL

- Een schuifregister is in VHDL eenvoudig te beschrijven middels een vector slice en een concatenation. Bits 6 t/m 0 worden één plek naar links geschoven en aangevuld met een extern databit. Bit 7 wordt “eruit geschoven”. Er wordt naar links geschoven.



Schuifregister in VHDL

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

entity shiftreg_simple is
    port (clk : in std_logic;
          areset : in std_logic;
          serin : in std_logic;
          q : out std_logic_vector (7 downto 0)
         );
end entity shiftreg_simple;

architecture rtl of shiftreg_simple is
signal q_int : std_logic_vector (7 downto 0);
```

DE HAAGSE HOGESCHOOL

Schuifregister in VHDL

```
begin

    process (clk, areset) is
    begin
        if areset = '1' then                        -- reset
            q_int <= (others => '0');
        elsif rising_edge(clk) then                 -- positive edge
            q_int <= q_int(6 downto 0) & serin;     -- shift left
        end if;
    end process;

    q <= q_int;                                     -- copy data

end architecture rtl;
```

Schuifregisters

- Daarnaast bestaan er ook *bi-directionele schuifregisters*. Dat zijn schuifregister die twee kanten kunnen opschuiven.

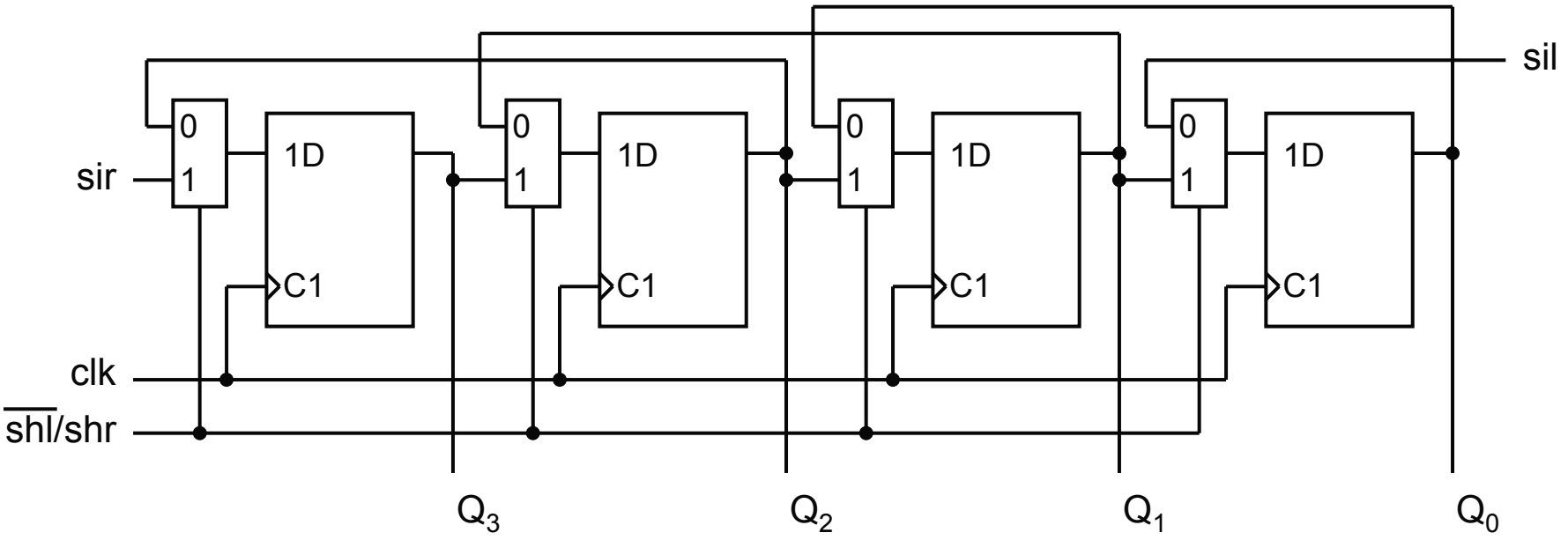
- Bij het bespreken van schuifregisters wordt gesproken van *linksom schuiven* (left shift) en *rechtsom schuiven* (right shift).

- Linksom is letterlijk zoals op papier zou worden getekend van rechts naar links.

- Rechtsom is letterlijk zoals op papier zou worden getekend van links naar rechts.

Bi-directioneel schuifregister

- De schuifrichting wordt aangegeven met een stuursignaal. Met behulp van multiplexers wordt de juiste ingang geselecteerd.



Schuifregisters

- Het schuifregister kan nu onthouden en schuiven, dus deze is geschikt als ontvanger voor seriële data.
- Als er data uit geschoven moet worden, heeft het schuifregister ook een laadmogelijkheid nodig.
- Hiervoor zijn nu twee stuursignalen nodig en 3x1 multiplexers.
- Als voorbeeld de zender-schuifregister voor RS232-communicatie.

Schuifregisters

- RS232-communicatie gaat als volgt*):

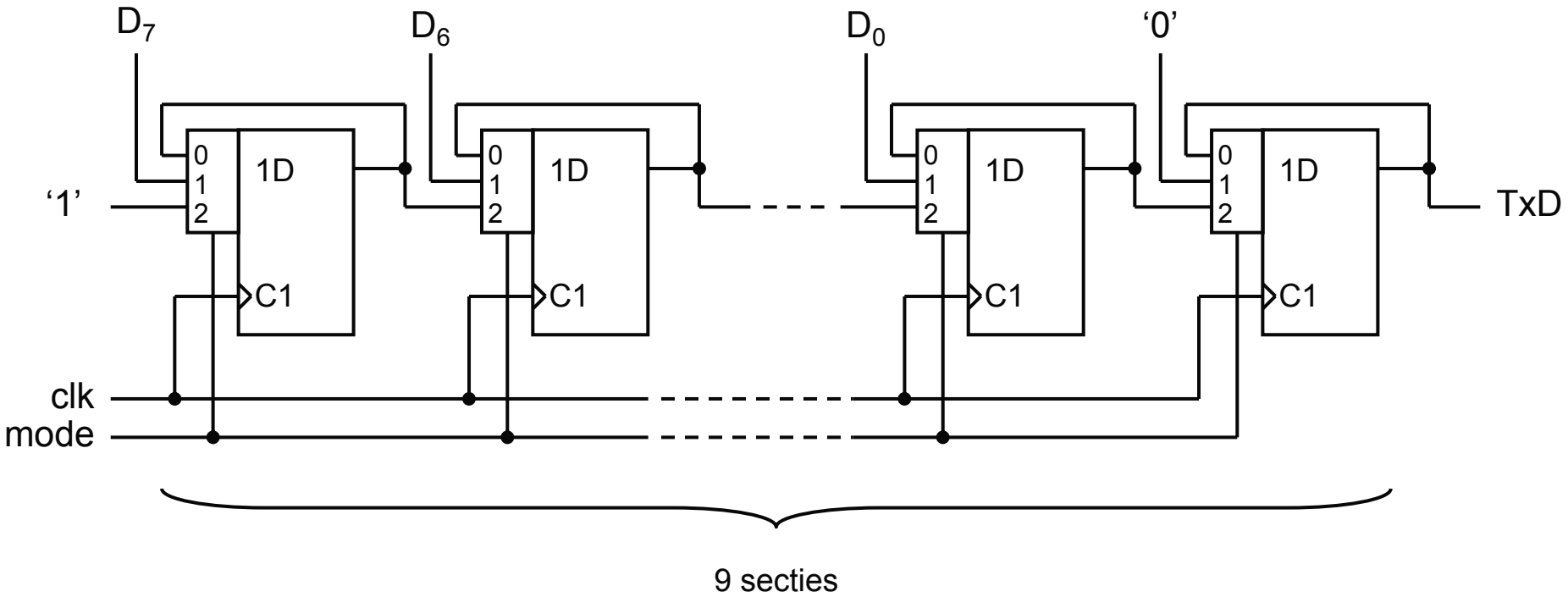
  De zendlijn is 1 in rust.
  Eerst wordt een *startbit* naar buiten geschoven. Deze is 0.
  Daarna worden de 8 databits naar buiten geschoven.
  Vervolgens volgt er één stopbit (1).

- Dit wordt 8N1 genoemd, er is geen *parity check*.

- Er worden 9 schuifsecties gemaakt, de eerste wordt geladen met een 0 (startbit), de laatste schuift een 1 in (stopbit).

nb: er zijn ook andere mogelijkheden

RS232-zenderschuifregister



mode = 00: hold

mode = 01: load

mode = 10: shift

Schuifregisters

- Schuiven heeft ook nog een rekenkundige functie.

- Naar links schuiven en aanvullen met 0 is hetzelfde als vermenigvuldigen met 2.

1011 - het getal (+11 unsigned)
10110 - eerste keer schuiven (*2)
101100 - tweede keer schuiven (*4)
1011000 - derde keer schuiven (*8 = +88 unsigned)

Schuifregisters

- Schuiven heeft ook nog een rekenkundige functie
- Naar rechts schuiven en aanvullen met 0 is hetzelfde als delen door 2.

  1011011 - het getal (+91 unsigned)
  0101101 - eerste keer schuiven (/2)
  0010110 - tweede keer schuiven (/4)
  0001011 - derde keer schuiven (/8 = +11 unsigned)

- De minstwaardige bits gaan wel verloren!

Schuifregisters

- Signed naar links schuiven gaat ook.

  1111011 - het getal (-5 signed)
  1110110 - eerste keer schuiven (*2)
  1101100 - tweede keer schuiven (*4)
  1011000 - derde keer schuiven (*8 = -40 signed)

- Het resultaat moet wel passen!

Schuifregisters

- Signed naar rechts schuiven gaat ook, maar er is wel een voorziening nodig om het tekenbit te behouden.

  1011011 - het getal (-37 signed)
  1101101 - eerste keer schuiven (/2)
  1110110 - tweede keer schuiven (/4)
  1111011 - derde keer schuiven (/8 = -5 signed)

- De minstwaardige bits gaan wel verloren!

- Negatieve getallen worden naar beneden (dus nog negatiever) afgerond.

Tellers

- Tellers worden in digitale systemen voor veel doeleinden gebruikt.
- Een teller kan bijvoorbeeld bepaalde gebeurtenissen tellen.
- Een teller kan ook tijd ‘tellen’ en tijdintervallen genereren. Zo’n teller wordt een *timer* genoemd.
- Tellers hebben de eigenschap een getelde hoeveelheid te *onthouden*. Dat betekent dat tellers *geheugen* bezitten.
- Indien een teller alleen maar een *vast aantal gebeurtenissen* moet (af-)tellen kan de *interne telstand vrij gekozen worden*.

Tellers

- Specifieke eigenschappen van tellers:

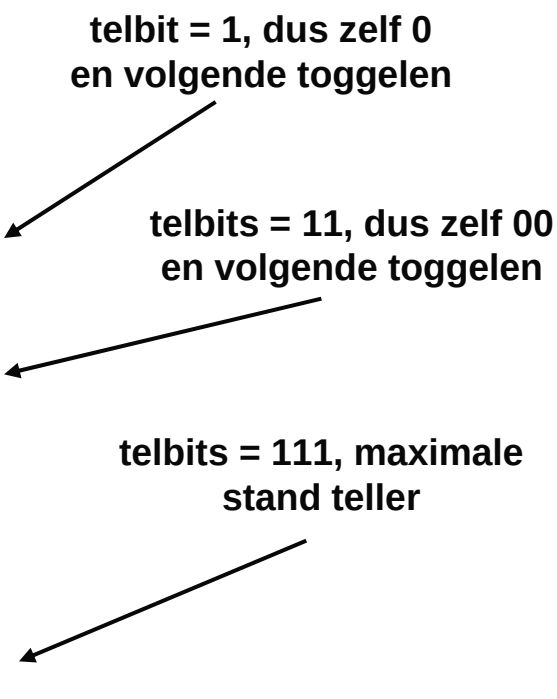
  Omhoog tellen (optellen, up counter)
  Omlaag tellen (aftellen, down counter)
  Omhoog/omlaag tellen (optellen/aftellen, up-down counter)
  Laden (load)
  Tellen actief (enable)
  Wissen (clear, zowel asynchroon als synchroon)
  Telbereik/telcodering (binair, BCD, Gray, One-hot, Johnson, ...)

- Tellers moeten kloksynchroon te worden ontworpen.

Basis tellen

- Een 1-bit teller, 2-bit teller, 3-teller

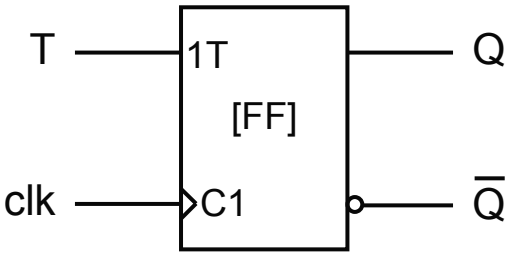
0	00	000
1	01	001
0	10	010
1	11	011
0	00	100
1	01	101
0	10	110
1	11	111



- Indien telbits stand 1, 11 of 111 hebben bereikt moeten deze zelf 0, 00 of 000 worden en het volgende telbit moet toggelen.

T-flipflop

- Het toggelen kan worden gedaan door een T-flipflop.



$$Q^{n+1} = \begin{cases} Q^n \text{ als } T^n = 0 \\ \overline{Q^n} \text{ als } T^n = 1 \end{cases}$$

$$Q^{n+1} = T^n \cdot \overline{Q^n} + \overline{T^n} \cdot Q^n$$
$$= T^n \oplus Q^n$$

Basis tellen

- Het tellen gebeurt heel systematisch.
- Om een telbit te laten toggelen moeten alle voorgaande telbits 1 zijn:

$$T_0 = 1$$
$$T_1 = Q_0$$
$$T_2 = Q_1 \cdot Q_0$$
$$T_3 = Q_2 \cdot Q_1 \cdot Q_0$$
$$\ldots$$
$$T_n = Q_{n-1} \cdot Q_{n-2} \cdots Q_1 \cdot Q_0$$

herschrijven

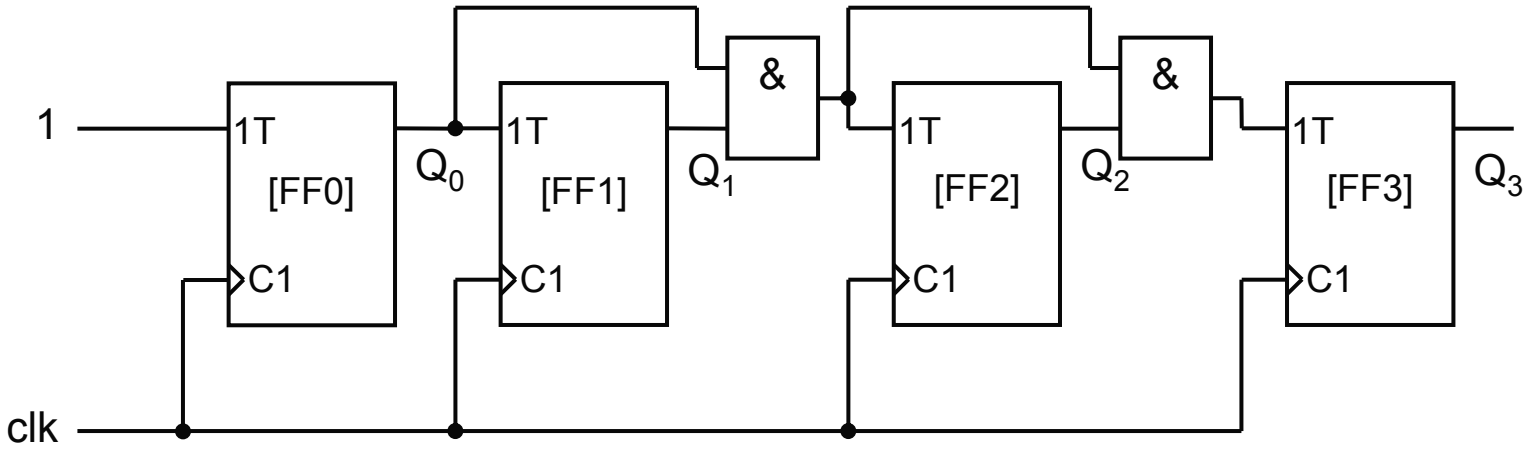
$$T_0 = 1$$
$$T_1 = Q_0$$
$$T_2 = Q_1 \cdot Q_0$$
$$T_3 = Q_2 \cdot Q_1 \cdot Q_0 = Q_2 \cdot (Q_1 \cdot Q_0)$$
$$\ldots$$
$$T_n = Q_{n-1} \cdot (Q_{n-2} \cdots (Q_1 \cdot Q_0)$$

AND met veel ingangen

cascaseschakeling van 2-input ANDs

4-bit teller met T-flipflops

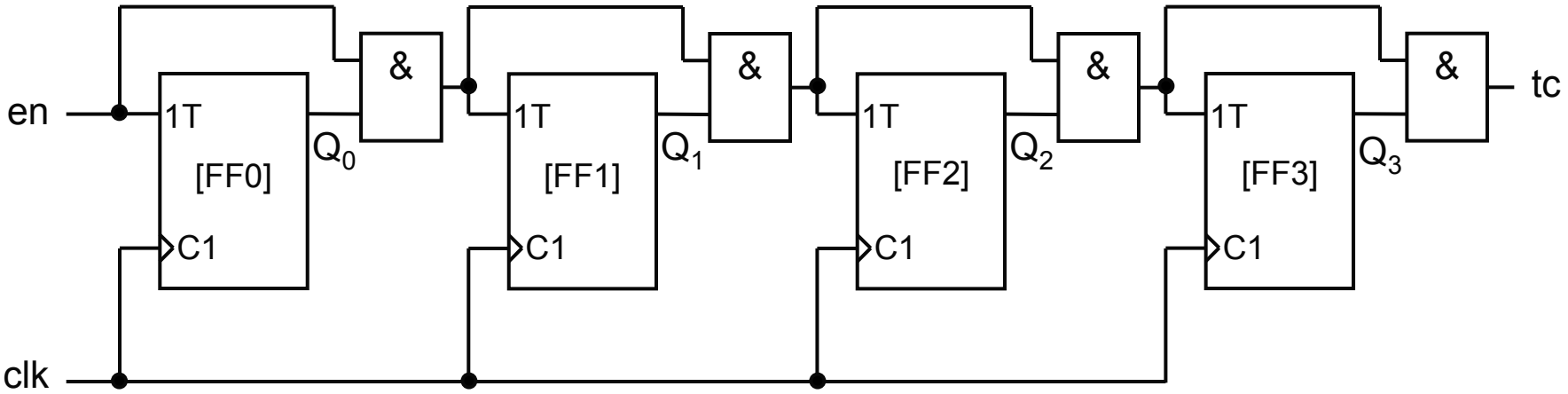
- Hieronder een 4-bit teller op basis van T-flipflops en cascadeschakeling van ANDs.



- Nadeel: maximale frequentie wordt lager naar mate er meer telbits zijn.

Teller met enable en terminal count

- We breiden de teller uit met een enable (en) en een terminal count (tc)
  - Indien en = 1 dan telt de teller op een klokflank
  - Indien de teller in de hoogste stand en enable actief, dan tc = 1 anders 0

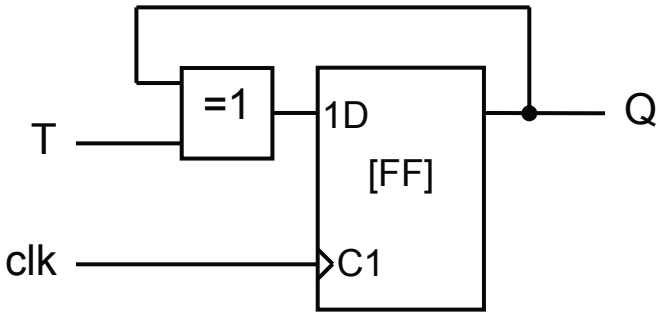


- De teller is nu te cascaderen tot een grotere teller (structural!)

Toggelen met D-flipflop

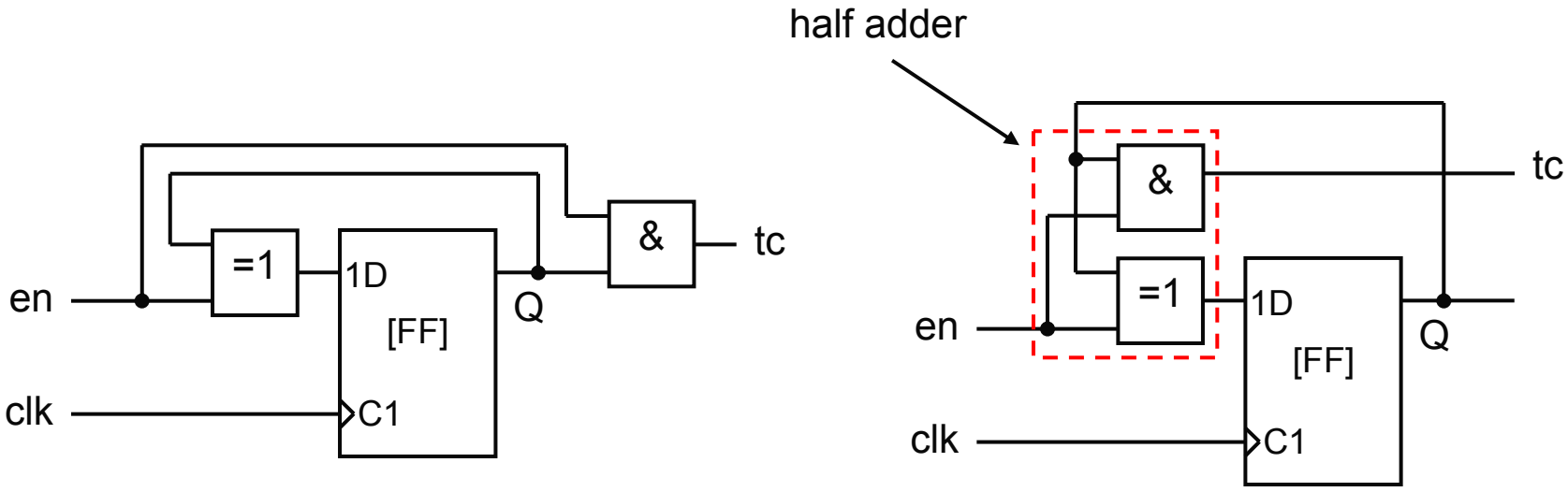
- Een nadeel van een T-flipflop is dat deze niet makkelijk synchroon te clearen is en ook niet makkelijk te laden met een ‘0’ of ‘1’.
- Een D-flipflop kan dit wel (er is wel voorzetlogica nodig).
- Een T-flipflop kan eenvoudig worden ontworpen uit een D-flipflop

$$\left.\begin{aligned} Q^{n+1} &= T^n \oplus Q^n \\ Q^{n+1} &= D^n \end{aligned}\right\} D^n = T^n \oplus Q^n$$



Telsectie D-flipflop

- Een telsectie kan eenvoudig worden ontworpen uit een D-flipflop, een EXOR en een AND.

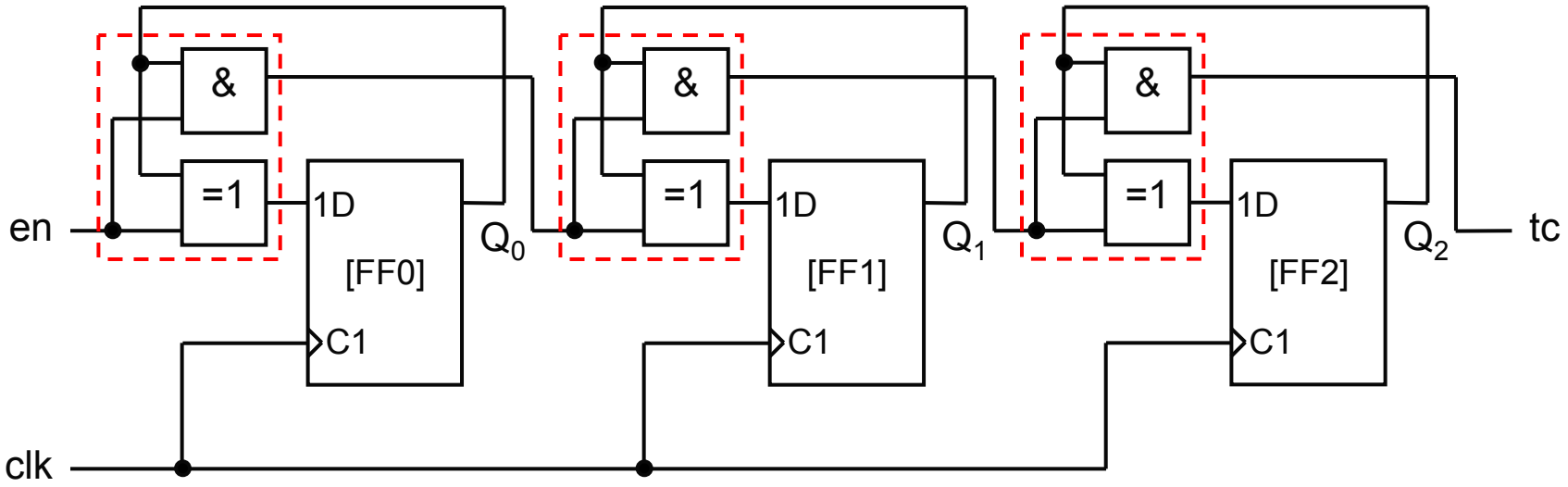


$$Q^{n+1} = en^n \oplus Q^n$$

$$tc^n = en^n \cdot Q^n$$

3-bit teller

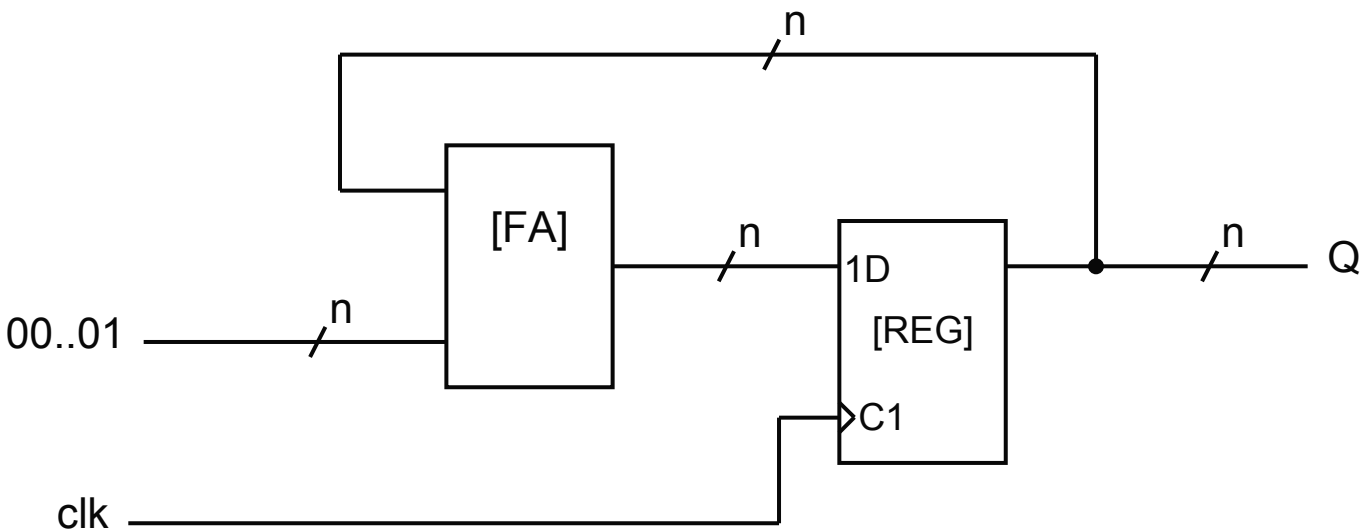
- Hieronder een voorbeeld van een 3-bit teller.



- Een teller is dus te maken op basis van D-flipflops en half adders.

Teller met full adder

- In feite is een teller te realiseren met een *full adder* en een *register*. De huidige stand van het register (de tellerwaarde) wordt teruggevoerd naar de opteller. Op de andere ingang van de teller wordt de constante 00..01 aangeboden. De nieuwe berekende waarde wordt (op een klokflank) in het register geklokt.



Optellen in VHDL

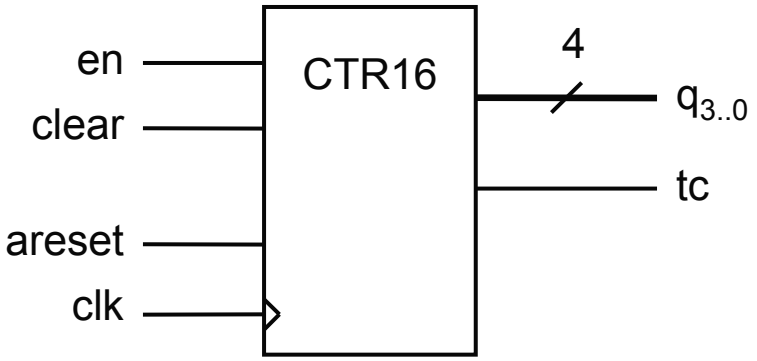
- Een teller is op meerdere manieren te beschrijven met VHDL.
- Natuurlijk is de teller structural te beschrijven met poorten, T-flipflops of D-flipflops.
- VHDL kent echter de rekenkundige optelling (+).
- Hiermee zijn eenvoudig optellers te beschrijven.
- Het voordeel van de rekenkundige optelling is dat de synthesizer slimme en snelle optellers kan realiseren.

4-bit counter

- Hieronder is een blokschema getekend van een 4-bit teller.

- De teller heeft de volgende in- en uitgangen:

- clk: de klok-ingang
- areset: asynchrone reset
- en: enable
- clear: synchrone clear
- q: de (huidige) telstand
- tc: terminal count



- Uitgang `tc` geeft aan wanneer de teller de hoogste stand heeft bereikt.

Entity 4-bit counter

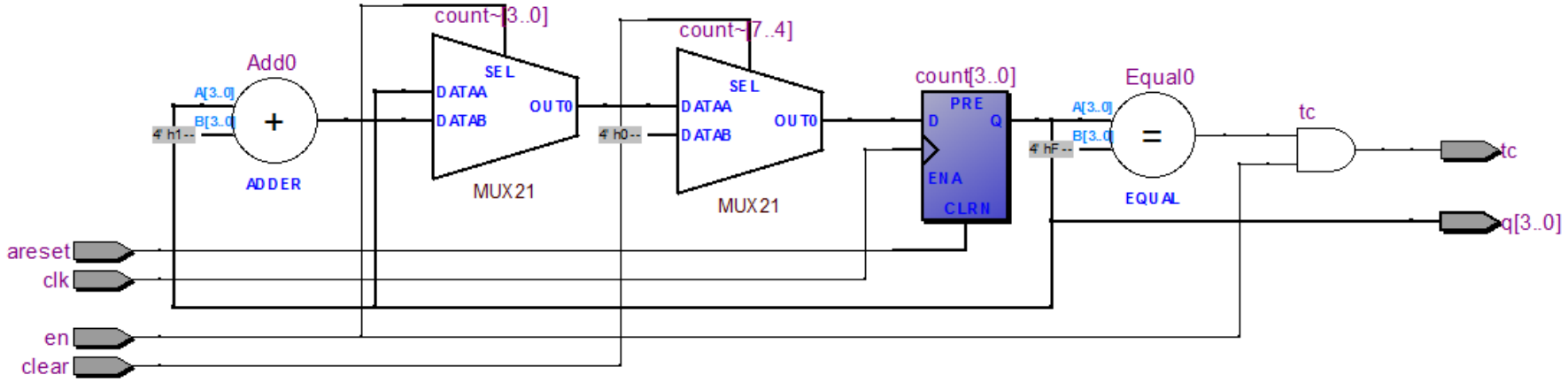
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;                          -- nodig voor '+'

entity counter_4bit is
  port (clk    : in std_logic;                     -- klok
        areset : in std_logic;                     -- asynchrone reset
        en     : in std_logic;                     -- enable
        clear  : in std_logic;                     -- reset naar 0
        q      : out unsigned(3 downto 0);         -- tellerstand
        tc     : out std_logic                     -- termininal count
       );
end entity counter_4bit;
```

Architecture 4-bit counter

```
architecture rtl of counter_4bit is
signal count : unsigned(3 downto 0);
begin
  process (clk, areset) is
  begin
    if areset = '1' then        -- asynchrone reset
      count <= "0000";
    elsif rising_edge(clk) then
      if clear = '1' then       -- synchrone clear
        count <= "0000";
      elsif en = '1' then       -- enable
        count <= count + 1;
      end if;
    end if;
  end process;
  q <= count;                   -- kopieer waarde
  tc <= '1' when count = "1111" and en = '1' else '0';
end architecture rtl;
```

Synthese 4-bit counter

- Bij synthese worden flipflops gerealiseerd voor de interne telstand `count` en `count` wordt aan `q` toegekend).
- De optelling wordt gesynthetiseerd met een full adder, de test op de hoogste stand met een comparator.
- De `if .. then`'s worden gesynthetiseerd met multiplexers.



Opgaven

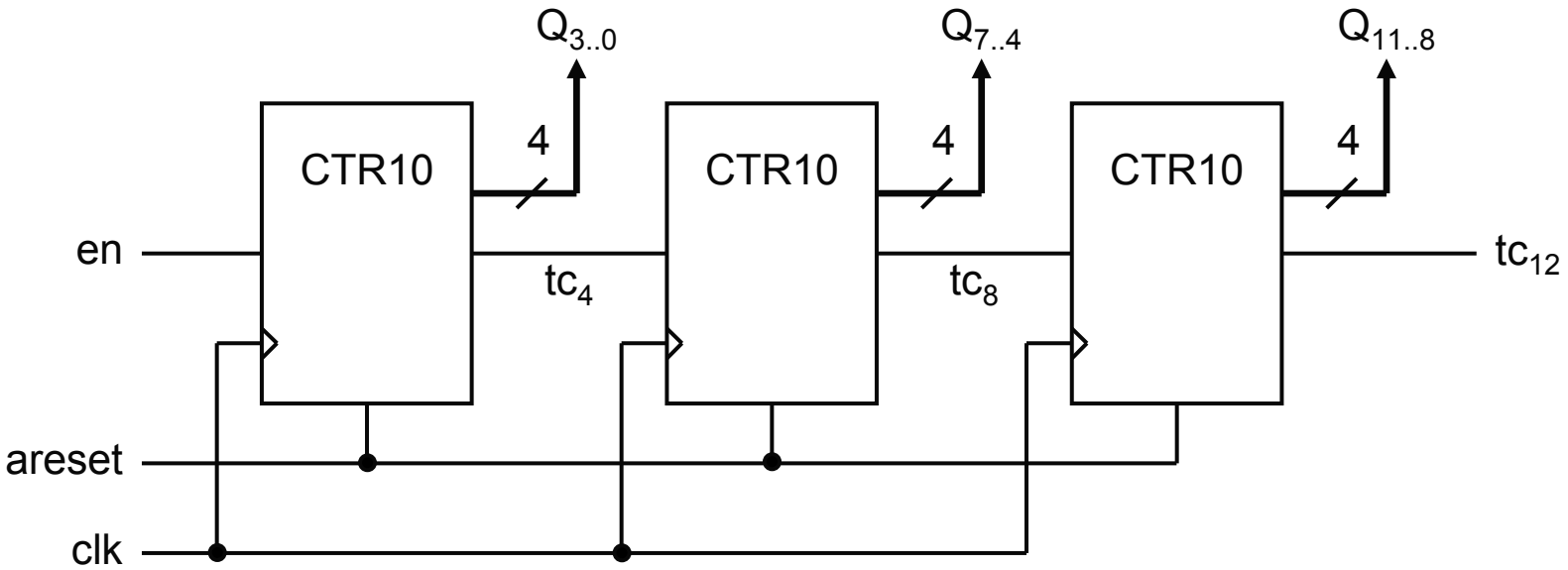
- Ontwerp een VHDL-beschrijving van een 4-bit teller die ook een begintelstand kan laden.
- Ontwerp een 4-bit up-counter met JK-flipflops en poorten.
- Ontwerp een 4-bit down-counter.
- Ontwerp een 6-teller, een teller die telt van 0 t/m 5, met losse poorten en T-flipflops.

BCD teller

- Tellen in het binaire systeem is erg simpel. Het afbeelden van het binaire getal in decimale vorm echter niet.
- Het is dan handiger om Binary Coded Decimal tellers (BCD-tellers) te gebruiken. De telstand is dan eenvoudig af te beelden op bv. 7-segment decoders.
- De telstanden van een BCD-teller lopen van 0 t/m 9. Daarna wordt de telstand weer 0.
- De telstand 9 is dus de hoogste stand van een BCD-teller en moet dus uitgecodeerd worden als terminal count zodat gecascadeerde BCD-tellers te ontwerpen zijn.

BCD teller

- Hieronder een voorbeeld van een BCD 3-decaden teller.



Uitgaande terminal count verbonden met ingaande enable: cascade-schakeling.

Entity BCD counter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;                       -- nodig voor '+'

entity counter_bcd is
  port (clk    : in std_logic;                  -- clock
        areset : in std_logic;                  -- reset naar 0
        en     : in std_logic;                  -- enable
        q      : out unsigned(3 downto 0);      -- tellerstand
        tc     : out std_logic                  -- terminal count
       );
end counter_bcd;

architecture rtl of counter_bcd is
signal count : unsigned (3 downto 0);
```

Architecture BCD counter

```
begin
  process (clk, areset) is
  begin
    if areset = '1' then          -- asynchrone reset
      count <= "0000";
    elsif rising_edge(clk) then
      if en = '1' then            -- enable actief
        if count = "1001" then    -- teller in hoogste stand?
          count <= "0000";        -- ... dan teller weer 0
        else
          count <= count + 1;     -- anders gewoon tellen
        end if;
      end if;
    end if;
  end process;
  tc <= '1' when count = "1001" and en = '1' else '0';
  q <= count;
end architecture rtl;
```

Count before test & others...

- Let op met de volgende toekenning.

```
...
elsif rising_edge(clk) then
   count <= count + 1;                -- tellen!
   if count = "1001" then             -- teller in hoogste stand?
       count <= (others => '0');      -- ... dan teller weer 0
   end if;
end if;
...
```

- De teller wordt eerst verhoogd, maar is niet gelijk beschikbaar, want de toekenning gebeurt onder flankbesturing.

- Met `others` kan een vector op “0...0” gezet worden.

Opgaven

- Gegeven de VHDL-beschrijving van de BCD-teller. Wat doet de teller als deze per ongeluk in stand 12 terecht is gekomen?

- Pas de VHDL-code zo aan dat de teller alleen (verder) telt als de teller zich in telstand 0 t/m 9 bevindt. Denk ook aan de terminal count.

- Pas de code aan zodat de teller een 13-teller wordt.

- Voor het detecteren van telstand 9 zijn in principe alleen telbit 3 en 0 nodig (Q_3 en Q_0). De andere twee zijn don't care. Helaas werkt de constructie `when count = "1--1"` niet. Pas de code aan zodat voor de test alleen naar de waarden Q_3 en Q_0 gekeken wordt.

Tellers met integers

- Het gebruik van integers sluit heel natuurlijk aan bij tellen.
- In VHDL zijn integers standaard 32 bits breed (denk aan synthese!)
- Door het gebruik van de *range* constructie kan heel makkelijk een bepaald telbereik gerealiseerd worden:

```
constant n : integer := 8;
signal count_i : integer range 0 to 2**n-1;
```

- De ** constructie staat voor machtsverheffen. Dit kan alleen met het grondtal 2.

Tellers met integers

- Conversie tussen unsigned en integer gebeurt door middel van conversieroutines die door de fabrikant geleverd worden.

- De lengte (= het aantal bits) van een unsigned kan worden opgevraagd door de *length attribute*:

```
constant n : integer := 8;
signal count_i : integer range 0 to 2**n-1;  -- 2**n-1 == 255
signal count_u : unsigned(n-1 downto 0);

count_i <= to_integer(count_u);
count_u <= to_unsigned(count_i, count_u'length);
-- or: count_u <= to_unsigned(count_i, n)
```

Tellers met integers

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

entity counter_integer is
   generic (n : positive := 8);
   port (clk    : in std_logic;
         areset : in std_logic;
         load   : in std_logic;
         data   : in unsigned(n-1 downto 0);
         q      : out unsigned(n-1 downto 0);
         tc     : out std_logic);
end entity;

architecture rtl of counter_integer is
-- Range is 0 to 2 to-the-power-of-n minus 1
signal count_val : integer range 0 to 2**n-1;
...
```

DE HAAGSE HOGESCHOOL

Tellers met integers

```
...
begin
   process (clk, areset) is
   begin
      if areset = '1' then
         count_val <= 0;
      elsif rising_edge(clk) then
         if load = '1' then
            -- conversion from unsigned to integer
            count_val <= to_integer(data);
         else
            if count_val = 2**n-1 then    -- explicit test!
               count_val <= '0';
            else
               count_val <= count_val + 1;
            end if;
         end if;
      end if;
   end process;
...
```

Tellers met integers

```
    ...

    -- Conversion from integer to unsigned
    -- Note the use of the length attribute
    q <= to_unsigned(count_val, q'length);

    -- Concurrent statement for terminal count
    tc <= '1' when count_val = 2**n-1 else '0';

end architecture rtl;
```

Tellen met bepaald bereik

- Door het gebruik van integers kan heel makkelijk een teller worden beschreven met een bepaald telbereik d.m.v. de *range* constructie.

```
signal count_val : integer range 0 to 199;
```

- Het testen op de maximale waarde moet expliciet beschreven worden.

```
if count_val = 199 then    -- of count_val = count_val'high
   count_val <= 0;
else
   count_val <= count_val + 1;
end if;
```

DE HAAGSE HOGESCHOOL

Tellen met bepaald bereik

- Bij synthese wordt het minimale aantal telbits uitgerekend, volgens de bekende formule: $n = \lceil \log_2 200 \rceil$ (199+1)

- Als de teller buiten het bereik komt, volgt (uiteraard) geen foutmelding; de teller telt gewoon door tot alle telbits op ‘1’ staan.

- Bij simulatie volgt een foutmelding als de telstand buiten het bereik komt en de simulatie stopt, ook als het telbereik exact een macht van 2 is.

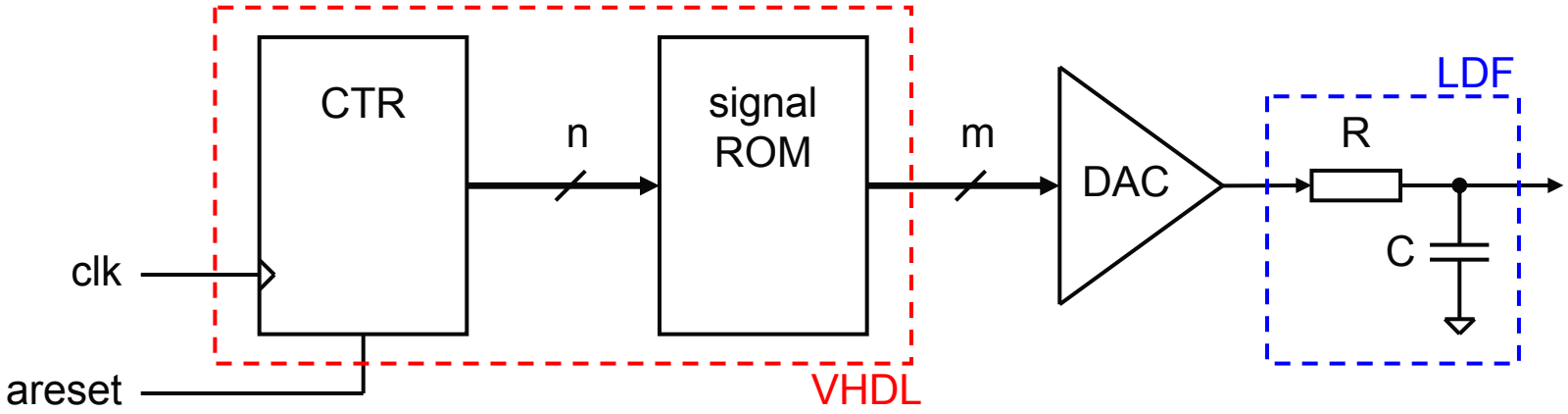
- Bij synthese wordt altijd het getal 0 meegenomen, ook als deze niet in het bereik ligt.

Digitale signaalvorm generator

- Een mooi voorbeeld waar een teller wordt gebruikt is een digitale signaalvorm generator.
- Het te genereren signaal is sinusvormig (als voorbeeld).
- De teller telt cyclisch (begint weer bij 0 als hoogste telstand is bereikt).
- De teller stuurt een ROM-tabel aan. De tabel bevat een gecodeerde sinus. Een ROM is in feite een elektronische waarheidstabel.
- De ROM-tabel stuurt een digitaal-analoog converter aan. Deze genereert het analoge signaal.
- Aan de uitgang van de digitaal-analoog converter is een laagdoorlaat-filter (LDF) gekoppeld die het signaal reconstrueert.

Digitale signaalvorm generator

- Hieronder is het blokschema gegeveven.

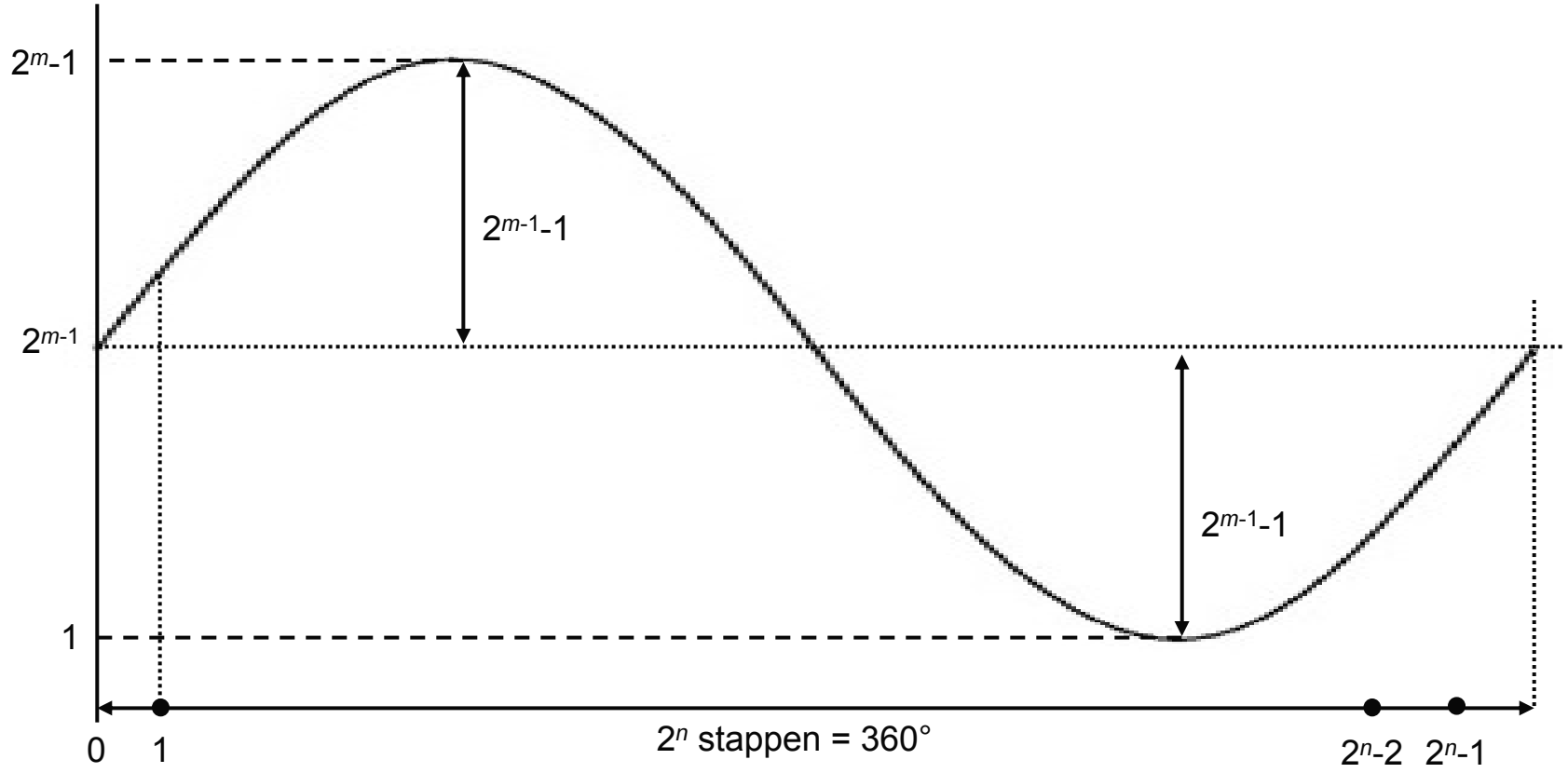


- De teller loopt van 0 t/m 2^n-1. De signal ROM heeft n *adresbits* en m *databits*. Op een adres kan één van 2^m verschillende waarden geplaatst worden.

Opmerking: $0 \leq U_{DAC} \leq U_{voed}$ [V]

Digitale signaalvorm generator

- Omzetten van analoge sinus naar digitale representatie.



Digitale signaalvorm generator

- Het uitrekenen van de tabel gaat als volgt

$$\text{tabelwaarde}_i = \sin\left(i \cdot \frac{360°}{2^n}\right) \cdot \left(2^{m-1} - 1\right) + 2^{m-1} \qquad i = 0 \ldots 2^n - 1$$

- In één rondgang langs een cirkel worden 2^n waarden berekend.
- Elke sinuswaarde wordt geschaald naar helft-1 van het bereik (2^{m-1}-1) met een offset van de helft van het bereik (2^{m-1}).
- De minimale waarde is 1 en de maximale waarde is 2^m-1.

Digitale signaalvorm generator

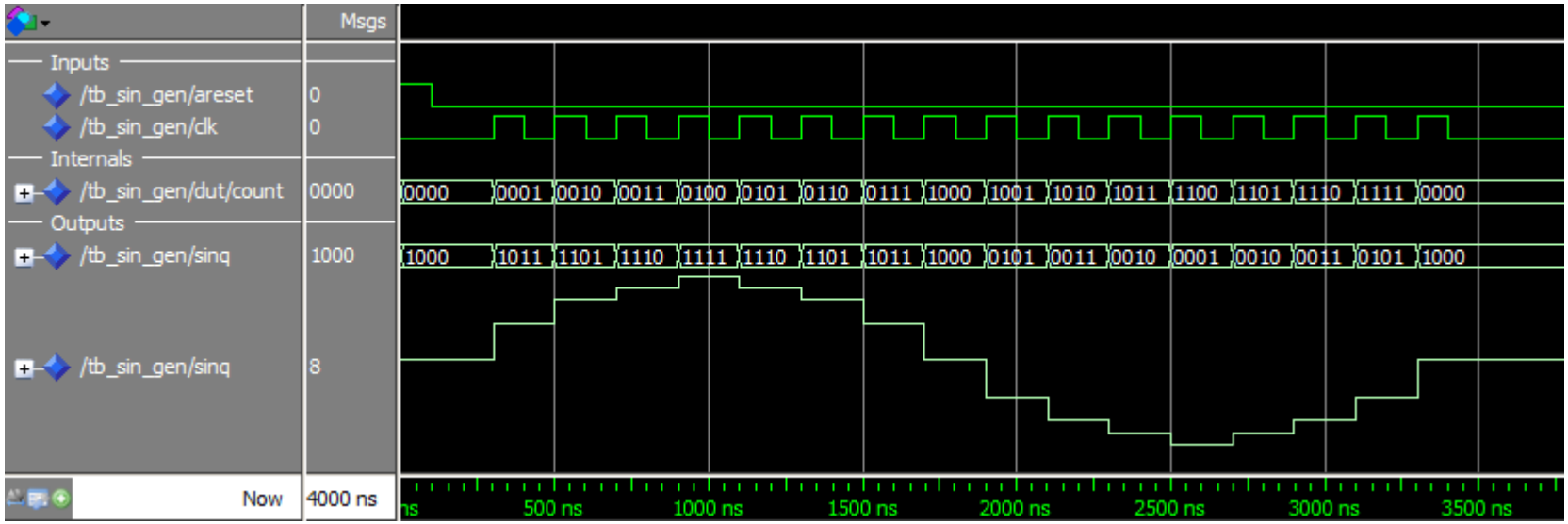
- De beschrijving in de vorm van twee processen:

```
architecture rtl of sin_gen is
signal count : unsigned
                  (3 downto 0);
begin

cntr: process (clk, areset) is
  begin
    if areset = '1' then
      count <= "0000";
    elsif rising_edge(clk) then
      count <= count + 1;
    end if;
  end process;
```

```
  sintbl: process (count) is
  begin
    case count is
      when "0000" => sinq <= "1000";
      when "0001" => sinq <= "1011";
      when "0010" => sinq <= "1101";
      ...
      when "1101" => sinq <= "0010";
      when "1110" => sinq <= "0011";
      when "1111" => sinq <= "0101";
      when others => sinq <= "XXXX";
    end case;
  end process;
end rtl;
```

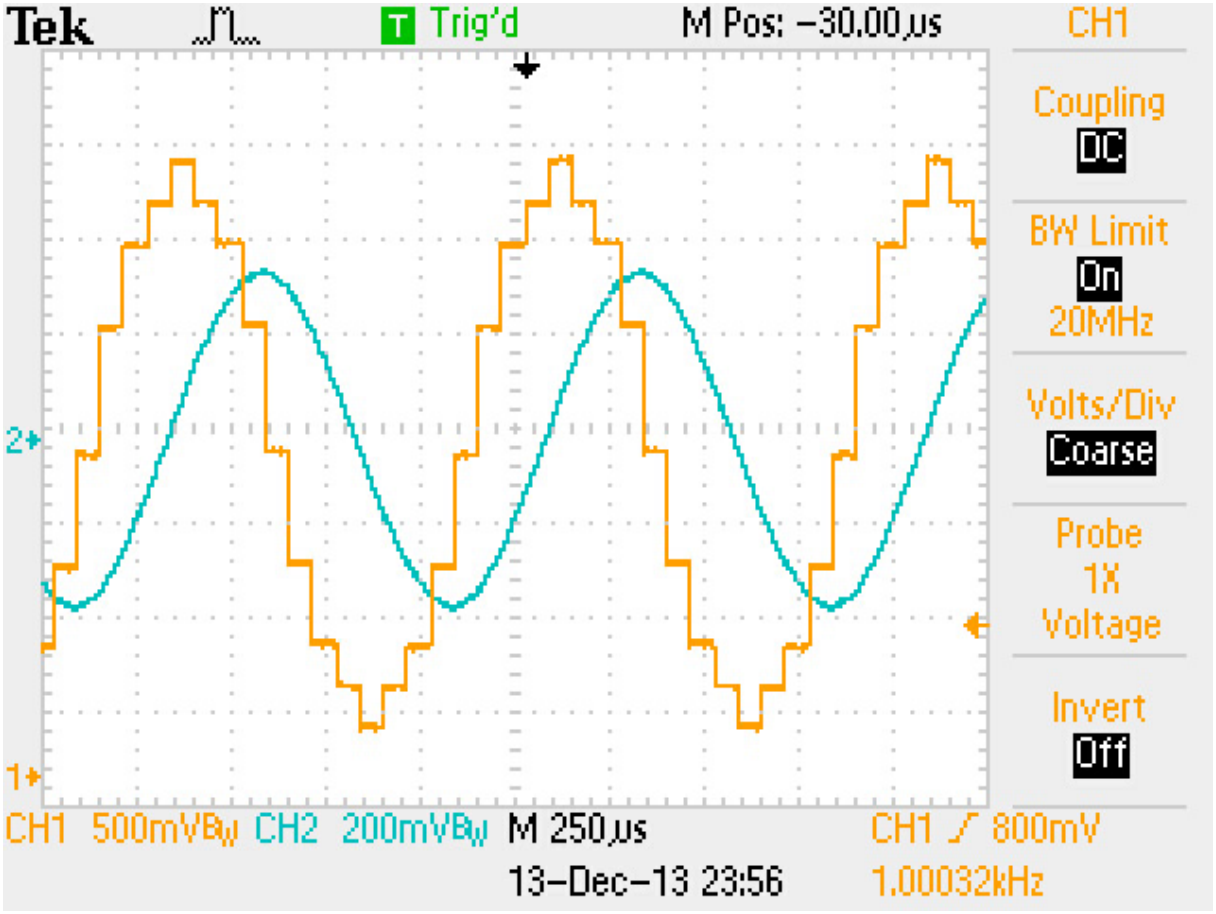
Simulatie

- Hieronder het simulatieresultaat.



- Noot: interne tellerwaarde is afgebeeld als `dut/count`. Signaal `sinq` is ook als “analoog” signaal afgebeeld.

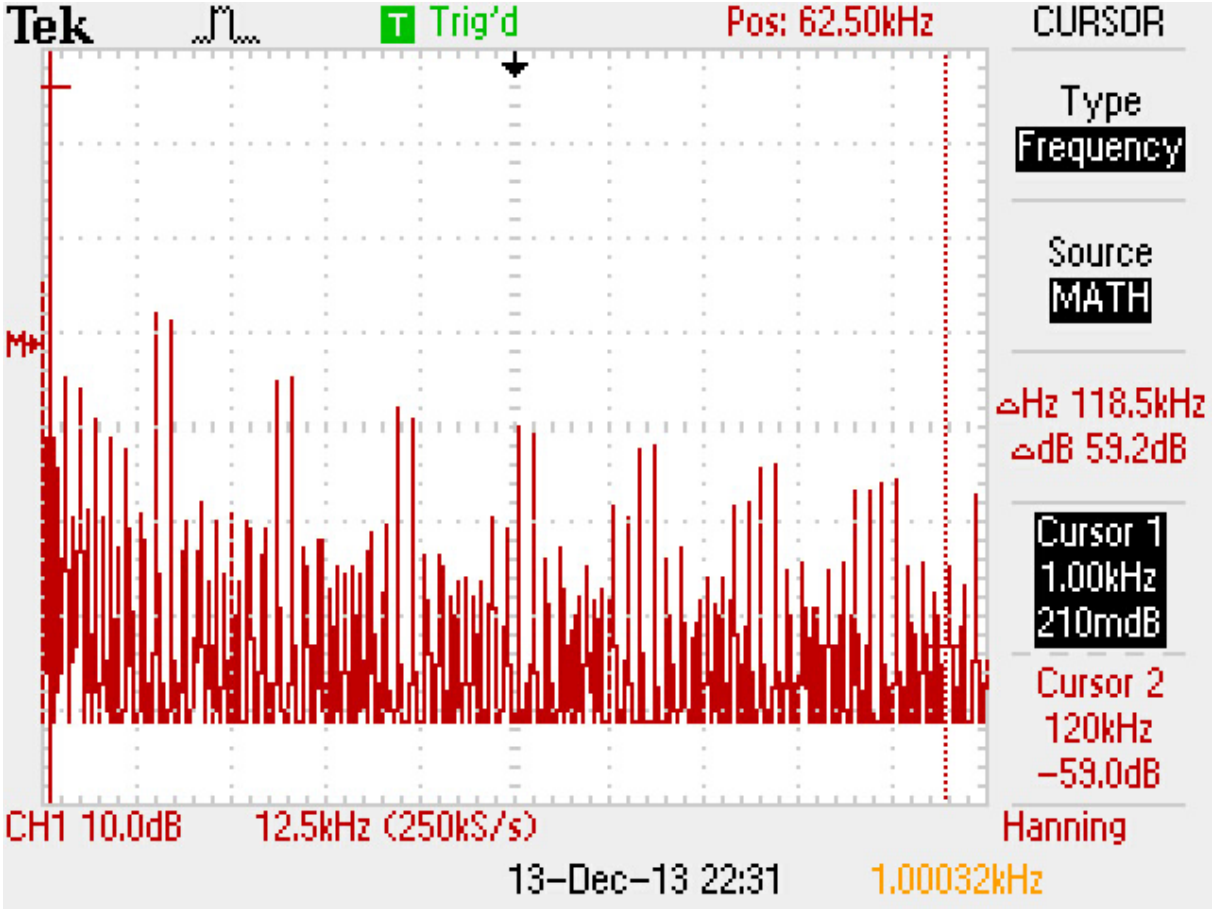
Oscilloscoopbeeld



- Het gele, geblokte signaal komt direct uit de ADC.
- Het blauwe signaal is na LD-filtering.
- Merk op dat de Volts/div niet gelijk staan.
- De fasedraaiing komt door het LD-filter.

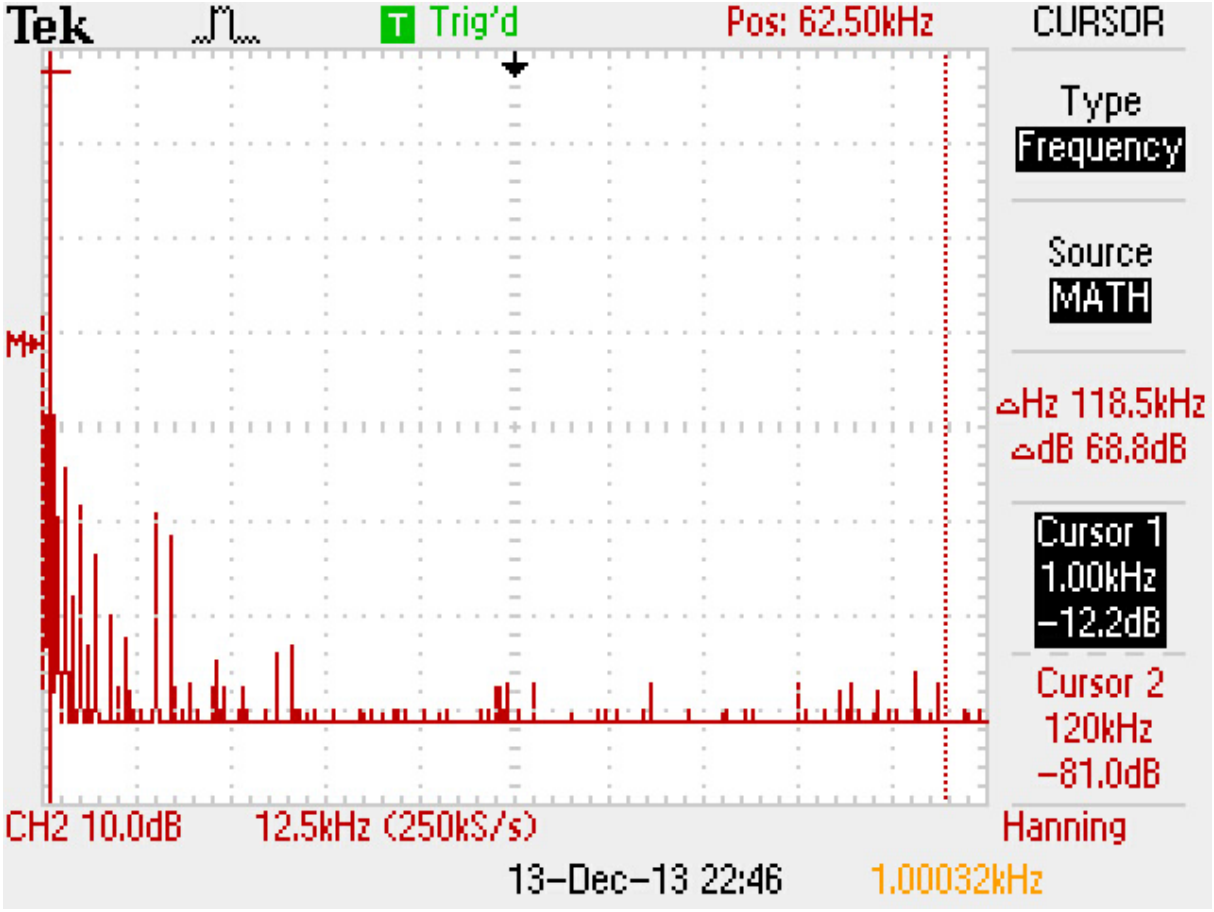
DE HAAGSE HOGESCHOOL

Frequentiespectrum voor LD-filtering



- Het frequentie-spectrum laat een paal zien op 1 kHz.
- Maar ook op 15, 17 kHz, 31, 33 kHz, 47, 49 kHz

Frequentiespectrum na LD-filtering



- Het gefilterde signaal is al beter, er zijn nog maar een paar palen over.

Sinusgenerator revisited

```
architecture rtl_integers of sin_gen is
-- Create array with the sinus tabulated
type sin_array_type is array (0 to 15) of integer;
constant sin_array : sin_array_type :=
                                    (8,11,13,14,15,14,13,11,8,5,3,2,1,2,3,5);
-- Signal count is of type integer
signal count : integer range 0 to 15;
begin
  cntr: process (clk, areset) is
  begin
    if areset = '1' then
      count <= 0;
    elsif rising_edge(clk) then
      if count = 15 then count <= 0; else count <= count + 1; end if;
    end if;
  end process;

  sinq <= std_logic_vector(to_unsigned(sin_array(count),4));
end architecture rtl_integers;
```

DE HAAGSE HOGESCHOOL

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