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

Project Digitale Systemen

Case Study – The Double Dabble algorithme

Jesse op den Brouw

PRODIG/2014-2015

DE HAAGSE HOGESCHOOL

Introductie Double Dabble

- In de digitale techniek wordt veel met decimale getallen gewerkt, simpelweg omdat wij er mee zijn opgegroeid.
- De getallen worden doorgaans binair gecodeerd.
- Het afbeelden van binaire getallen naar decimale getallen is dus noodzakelijk.
- Een efficiente methode om binaire getallen om te zetten naar decimale getallen is het zogenaamde *double dabble* algoritme.
- In feite worden decimale getallen in BCD gecodeerd.

Binair factoriseren

- Een 8-bit binair getal wordt geschreven als

$$a_7a_6a_5a_4a_3a_2a_1a_0$$

- Dit kan worden omgerekend naar decimaal

$$a_7 \cdot 2^7 + a_6 \cdot 2^6 + a_5 \cdot 2^5 + a_4 \cdot 2^4 + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

- Dit kan gefactoriseerd* worden tot

$$((((((a_7 \cdot 2 + a_6) \cdot 2 + a_5) \cdot 2 + a_4) \cdot 2 + a_3) \cdot 2 + a_2) \cdot 2 + a_1) \cdot 2 + a_0$$

* De regel van Horner

DE HAAGSE HOGESCHOOL

Schuiven en tellen

- Werking is dus steeds vermenigvuldigen met 2 en optellen (links beginnen).

- Vermenigvuldigen met 2 is schuiven naar links.

$$((((((a_7 \cdot 2 + a_6) \cdot 2 + a_5) \cdot 2 + a_4) \cdot 2 + a_3) \cdot 2 + a_2) \cdot 2 + a_1) \cdot 2 + a_0$$

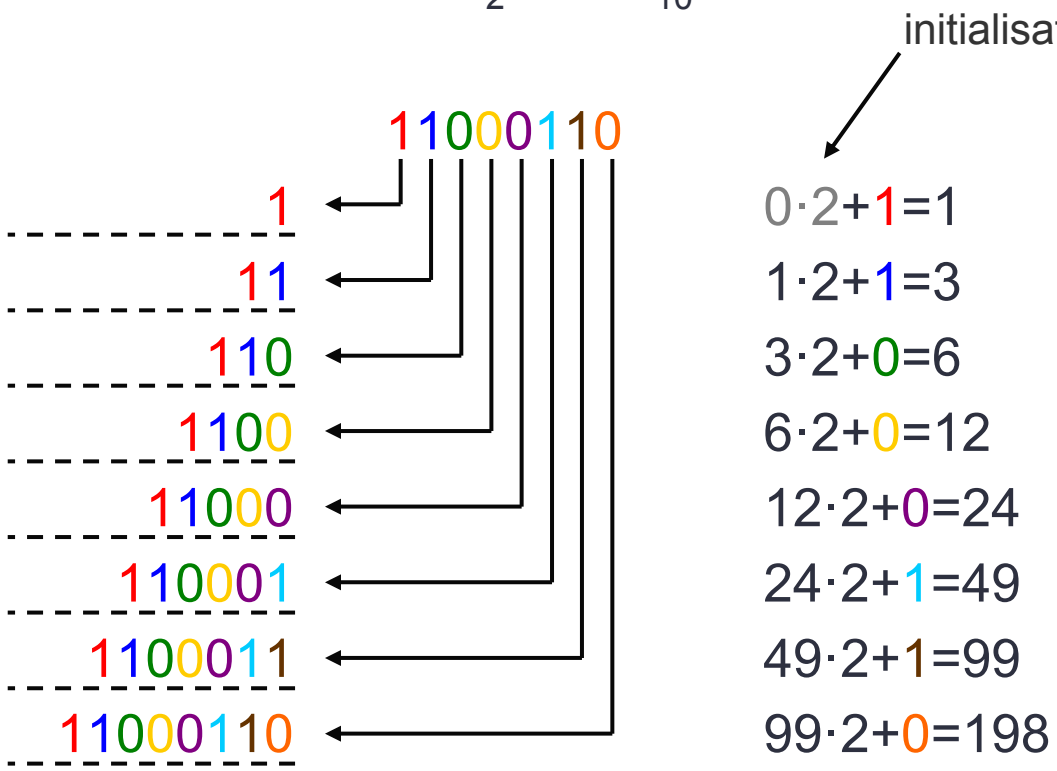
schuiven

optellen

- Het is een iteratief algoritme. *Op elk moment tijdens het itereren is de uitkomst in overeenstemming met de tot dan toe verwerkte bits!*

Voorbeeld

- Een voorbeeld: $11000110_2 = 198_{10}$



- De stippellijnen geven iteratieslagen aan.

Binair naar binair is makkelijk

- Het hiervoor geschetste voorbeeld is echter van binair naar binair.
- Dat heeft als voordeel dat er alleen geschoven hoeft te worden.
- Het optellen gebeurt automatisch door het (naar binnen) schuiven.
- Het algoritme is dus:

  schuiven

Aanpassing voor BCD-cijfer

- Voor afbeelden van decimale getallen moet er omzetting plaatsvinden van binair naar BCD.

- Bekend is dat de codering voor decimale getallen groter dan 9 is:

10_{10}	=	1010_2	=	10000_{BCD}
11_{10}	=	1011_2	=	10001_{BCD}
12_{10}	=	1100_2	=	10010_{BCD}
13_{10}	=	1101_2	=	10011_{BCD}
14_{10}	=	1110_2	=	10100_{BCD}
15_{10}	=	1111_2	=	10101_{BCD}

DE HAAGSE HOGESCHOOL

Schuiven en corrigeren

- Indien het (tussen-)resultaat groter is dan 9, dan moet er 6 bij worden opgeteld.

- Het algoritme wordt dus uitgebreid:

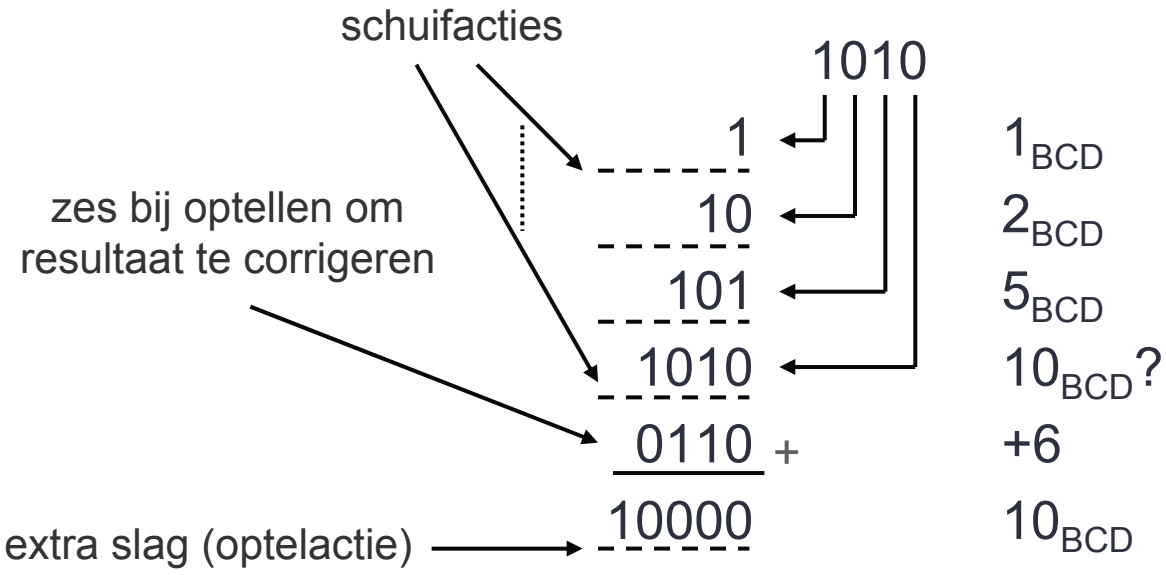
  schuif naar links

  indien antwoord groter dan 9, pas correctie toe

- Merk op dat het gecorrigeerde antwoord tussen 10000_{BCD} en 10101_{BCD} ligt, dat is dus 5 bits.

Voorbeeld

- Als voorbeeld de omzetting van 1010_2 naar 10000_{BCD}.



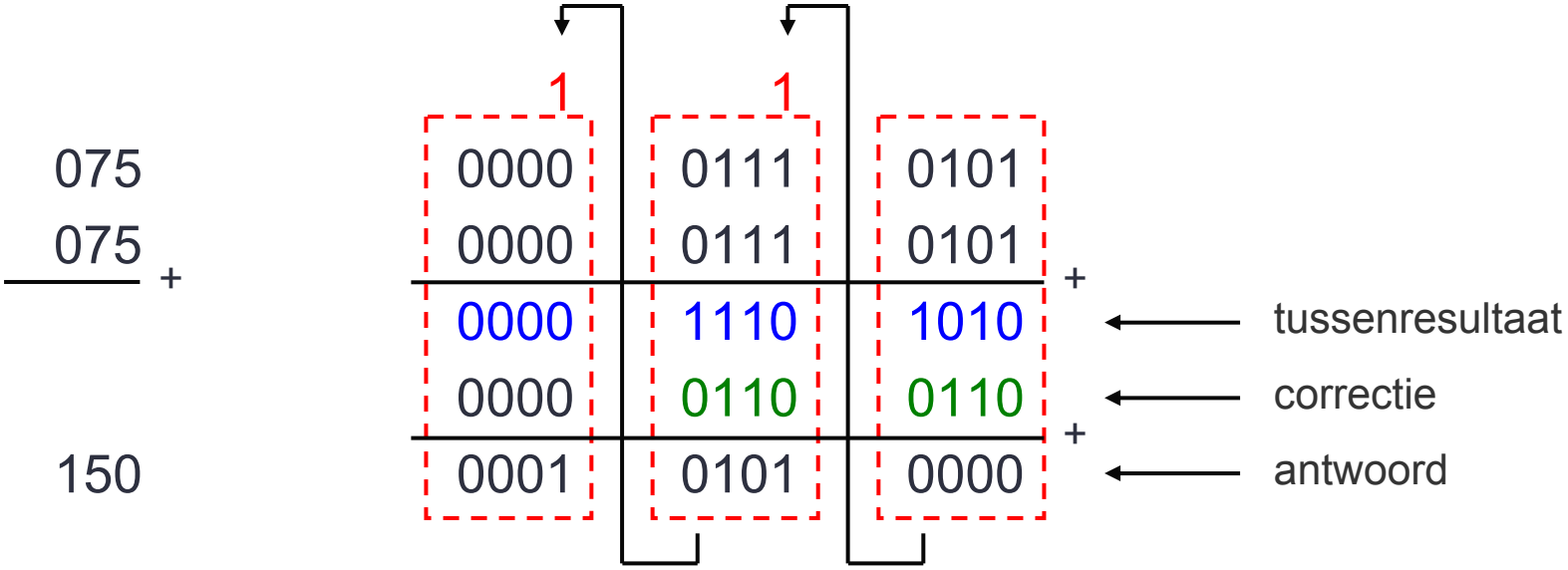
- Merk op: één extra slag om tot het juiste resultaat te komen.

BCD-correctie

- Dit voorbeeld is op te schalen waardoor er meerdere BCD-secties nodig zijn om het resultaat weer te geven.
- Een getal naar links schuiven is hetzelfde als vermenigvuldigen met 2.
- Een getal vermenigvuldigen met 2 is hetzelfde als het getal bij zichzelf optellen.
- Bekend is dat bij BCD-optellers voor een secties een correctie nodig is wanneer deze sectie groter is dan 9.
- Dat leidt ertoe dat er een *ripple-carry* opteller moet worden gebruikt met net zoveel bits als het BCD-antwoord.

BCD correctie

- Voorbeeld:

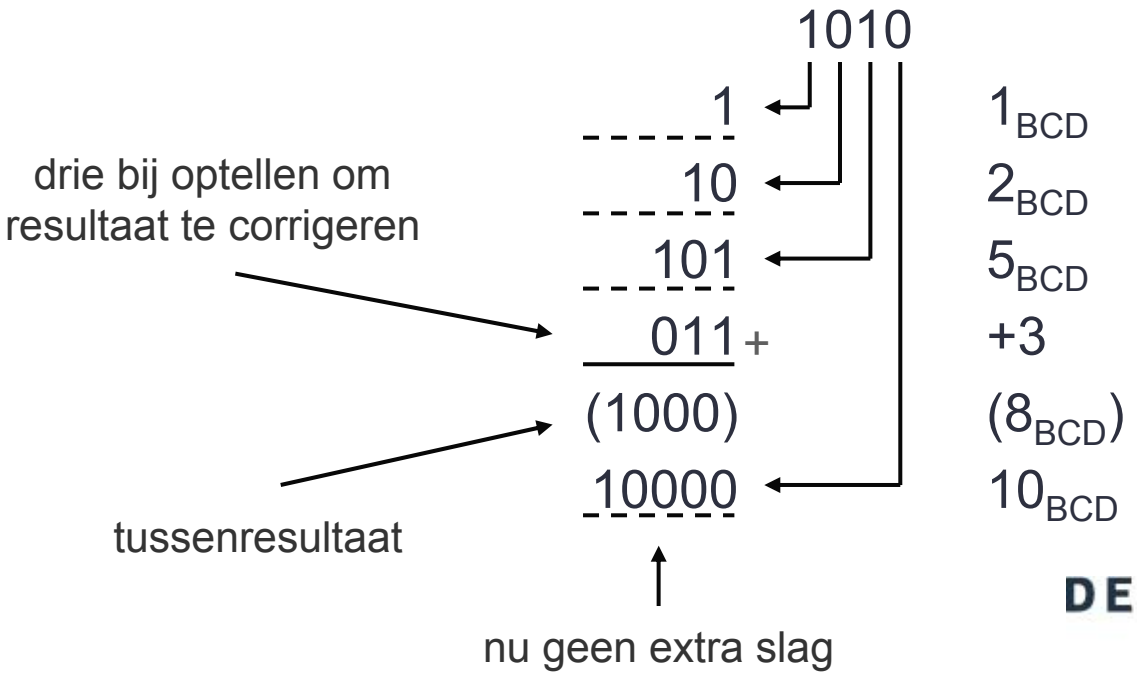


Voorbeeld (slimmer)

- Het optellen van 6 *na* het schuiven kan vervangen worden door het optellen van 3 *voor* het schuiven.

x2 = schuiven

$$1010_2 = 10000_{BCD} = (1010_2+0110_2)_{BCD} = ((101_2+011_2)\cdot 10_2)_{BCD}$$



Correctietabel

- Hieronder een tabel met de waarden 0 t/m 9.

voor schuiven	correctie	na correctie	na schuiven*
0000	-	0000	0000
0001	-	0001	0010
0010	-	0010	0100
0011	-	0011	0110
0100	-	0100	1000
0101	0011	1000	10000
0110	0011	1001	10010
0111	0011	1010	10100
1000	0011	1011	10110
1001	0011	1100	11000

* het ingeschoven bit wordt 0 verondersteld

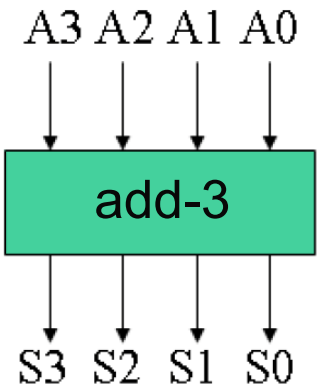
DE HAAGSE HOGESCHOOL

En nu met drie

- Het optellen van 3 heeft nog meer voordelen t.o.v. van het optellen van 6.
- Zo zal het resultaat van de optelling met 3 *nooit* groter worden dan 1100_2 zodat er *geen carry* is naar de volgende BCD-sectie.
- Er is geen extra slag nodig om het resultaat nogmaals te corrigeren.
- Voor het wel of niet optellen met 3 kan een logische schakeling ontworpen worden. Deze is geheel combinatoriek.

Hardware voor de Add-3 module

- De module kan met behulp van bekende technieken uitgewerkt worden tot een combinatorische schakeling.



A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

na correctie

voor correctie

Per BCD-sectie

- Eerder is al verteld dat na iedere iteratieslag de uitkomst in overeenstemming is met de tot dan toe verwerkte bits.
- Bij meerdere BCD-secties moet *per sectie* steeds onderzocht worden of het (deel-)resultaat gecorrigeerd moet worden.
- Niet in alle gevallen is voor de meest significante sectie correctie nodig.

Aantal bits voor BCD-getal

- Een n bits binair getal moet worden omgezet in m decimalen: $2^n = 10^m$
- Daaruit volgt

$$10^m = 2^n$$
$$m = {}^{10}\log 2^n$$
$$m = n \cdot {}^{10}\log 2$$
$$m \approx 0{,}301 \cdot n$$

- Het aantal BCD-bits is dan $k = 4 \cdot m = 4 \cdot n \cdot {}^{10}\log 2 \approx 1{,}204 \cdot n$
- Het antwoord moet naar boven afgerond worden.

Werking 8-bit conversie

- Voor het omzetten kan een algoritme beschreven worden.

- Het algoritme voor een 8-bit binair getal is als volgt:

laad binair getal, zet BCD-antwoord op 0
doe 8 keer
    als de binaire waarde in een BCD-sectie groter is
        dan of gelijk is aan 5, tel dan 3 bij die sectie op
    schuif zowel het binaire getal als het BCD-antwoord één plek naar
        links, waarbij het meestwaardige bit van het binaire getal
        in het minstwaardige bit van het BCD-antwoord geplaatst
        wordt
klaar, BCD-antwoord beschikbaar

Uitwerking 8-bit conversie

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

start

klaar

DE HAAGSE HOGESCHOOL

Opgaven

- Voer een handmatige Double Dabble conversie uit voor het 6-bits getal 101110_2.
- Voer een handmatige Double Dabble conversie uit voor het 8-bits getal 10010000_2.
- Hoeveel BCD-bits zijn nodig om een 16 bit binair getal om te zetten naar BCD?
- Onderzoek of voor een 9-bits, 11-bits en 15-bits binair getal de meest significante sectie correctie nodig is.
- Ontwerp de schakeling voor de correctiehardware (add-3).

Double Dabble hardware

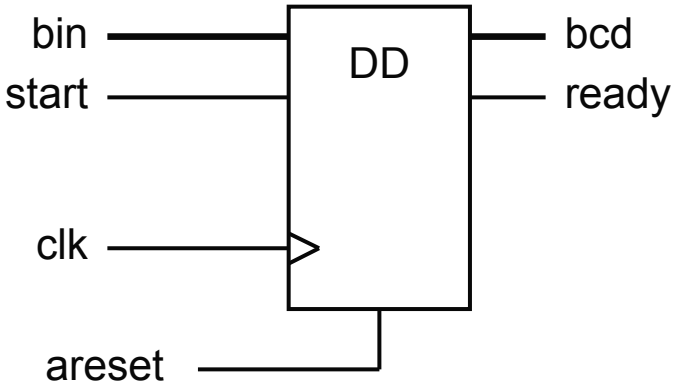
- Voor het ontwerpen van de hardware moet het volgende in acht worden genomen:

- Elke iteratieslag wordt op een klokflank uitgevoerd. Het betreft hier eigenlijk het schuiven omdat het optellen *voor* het schuiven gebeurt.
- Er is een BCD-antwoord van 8·1,204 = 10 bits (afgerond) nodig. Dit wordt tijdens de conversie opgeslagen. Er is een register nodig.
- Het aangeboden binaire getal moet opgeslagen worden omdat het tijdens de conversie aangepast wordt. Er is een register nodig.
- Er is een teller nodig die het aantal iteratieslagen bijhoudt.
- Er is een start-ingang waarmee de conversie gestart wordt.
- Er is een ready-uitgang waarmee wordt aangegeven dat de conversie klaar is.

Double Dabble hardware

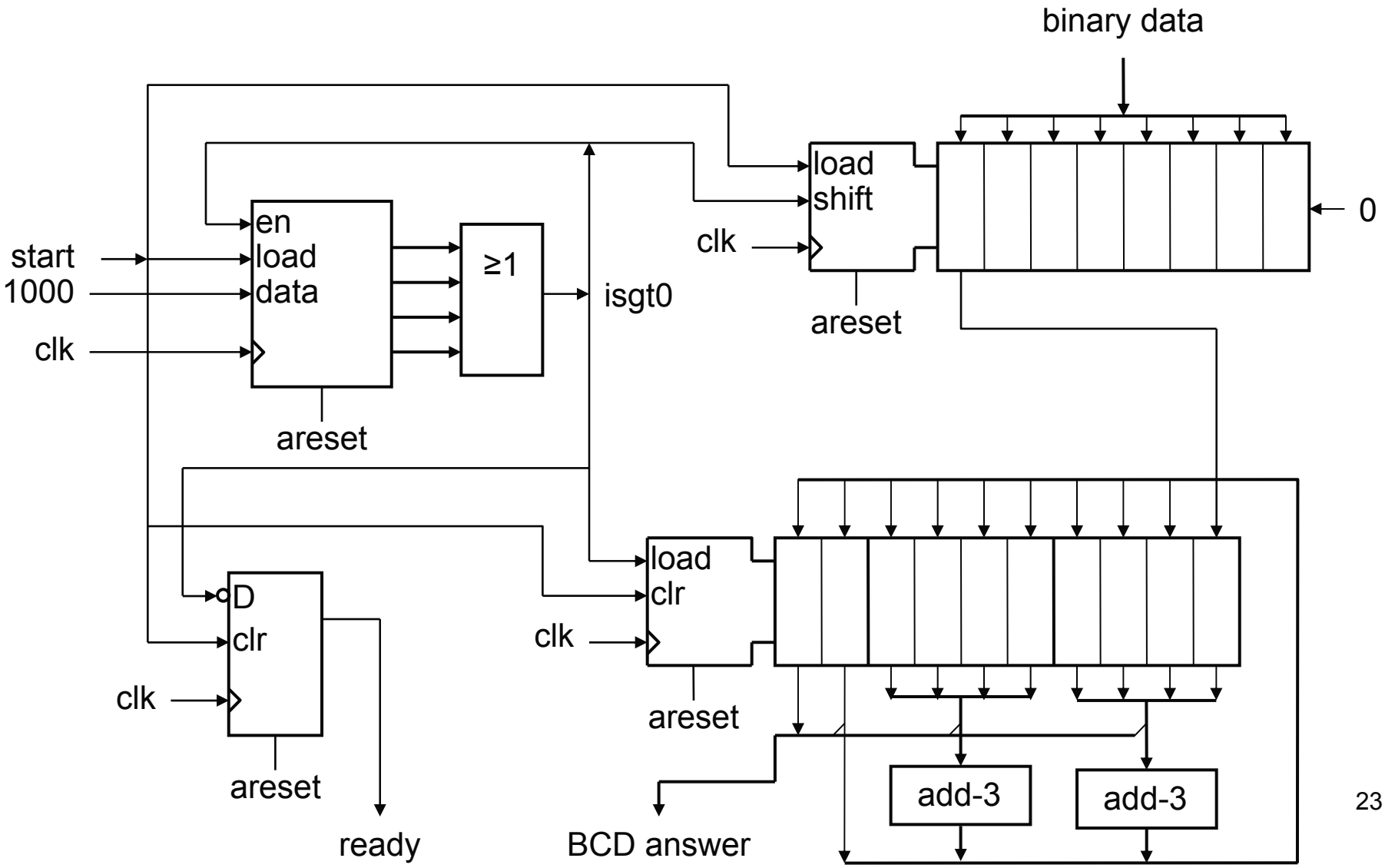
- De betekenis van de signalen:

  clk: het kloksignaal
  areset: asynchrone reset
  start: voor het startsein
  bin : het 8-bit binaire getal
  bcd : het 10-bit BCD-getal
  ready : het antwoord is beschikbaar



- Natuurlijk kan de beschrijving helemaal structural worden opgesteld met op de onderste laag de gedragsbeschrijvingen, maar hier is gekozen voor een meer algoritmische benadering. Het toont goed de kracht van VHDL.

Double Dabble hardware



VHDL-code entity

- De entity-beschrijving.

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

entity double_dabble_8bit is
  port (clk    : in std_logic;
        areset : in std_logic;
        start  : in std_logic;
        bin    : in unsigned (7 downto 0);
        bcd    : out unsigned (9 downto 0);
        ready  : out std_logic
       );
end double_dabble_8bit;
```

8 bits getal

10 bits resultaat

VHDL-code architecture

- De aangeboden binaire waarde moet voor de daadwerkelijke conversie eerst intern opgeslagen worden in een *register*.

- Er is een *teller* nodig die het aantal iteratieslagen bijhoudt.

```
architecture rtl of double_dabble_8bit is
signal counter : integer range 0 to 8;
signal bin_int : unsigned (7 downto 0);
begin
  ...
end rtl;
```

- De **range** constraint zorgt ervoor dat de teller precies genoeg bits krijgt.

VHDL-code process

- Tijdens de conversie moet de tot dan toe berekende BCD-waarde onthouden worden. Het ligt voor de hand om een intern signaal te maken. Dat blijkt echter niet handig omdat tijdens de conversie de interne BCD-waarde tweemaal aangepast wordt (assignments). De interne BCD-waarde wordt als variabele uitgevoerd.

- Het conversieproces is gevoelig voor de klok en de asynchrone reset.

```
process (clk, areset) is
variable bcd_int : unsigned (9 downto 0);
begin
  ...
end process;
```

VHDL-code asynchrone reset

- Als de asynchrone reset actief is moet het systeem in een bekende stand terecht komen.

- Zo moet `ready` op 0 worden gezet om aan te geven dat er geen geldige conversie heeft plaatsgevonden.

```
if areset = '1' then
  counter <= 0;                 -- reset counter
  bcd_int := "0000000000";      -- initialize BCD scratchpad
  bin_int <= "00000000";        -- initialize internal bin
  ready <= '0';                 -- we're not ready
elsif ...
```

VHDL-code startdetectie

- Alle andere acties gaan onder besturing van de opgaande flank. Als `start` actief wordt moet het systeem geladen worden met beginwaarden.

```
elsif rising_edge(clk) then
  if start = '1' then
    counter <= 8;                 -- number of iterations
    bcd_int := "0000000000";      -- BCD scratchpad clear
    bin_int <= bin;               -- copy binary number
    ready <= '0';                 -- we're not ready
  elsif ...
```

VHDL-code correctie op BCD-waarden

- Nu volgt de daadwerkelijke conversie. De teller houdt bij hoeveel iteraties er nog gedaan moeten worden.

```
elsif counter > 0 then
  counter <= counter - 1;
  -- no correction needed for bits 9 down to 8
  if bcd_int(7 downto 4) > "0100" then
    bcd_int(7 downto 4) := bcd_int(7 downto 4) + "0011";
  end if;
  if bcd_int(3 downto 0) > "0100" then
    bcd_int(3 downto 0) := bcd_int(3 downto 0) + "0011";
  end if;
```

bit slice

VHDL-code schuiven

- De nieuw berekende waarden moeten nog wel geschoven worden:

```
bcd_int := bcd_int(8 downto 0) & bin_int(7);
bin_int <= bin_int(6 downto 0) & '0';
ready <= '0';
```

concatenation operator

- Dit werkt als volgt.

- De inhoud van `bcd_int` wordt gevormd uit de 9 minstwaardige bits aangevuld met het meestwaardige bit van `bin_int`.

- De inhoud van `bin_int` wordt gevormd uit de 7 minstwaardige bits aangevuld met een logische ‘0’.

VHDL-code klaar

- Als alle conversieslagen zijn uitgevoerd, wordt `ready` actief.

```
    else                -- counter = 0 and start = 0
      ready <= '1';     -- we're done!!!!!
    end if;             -- end of 'if start = ... '
  end if;               -- end of rising_edge()
```

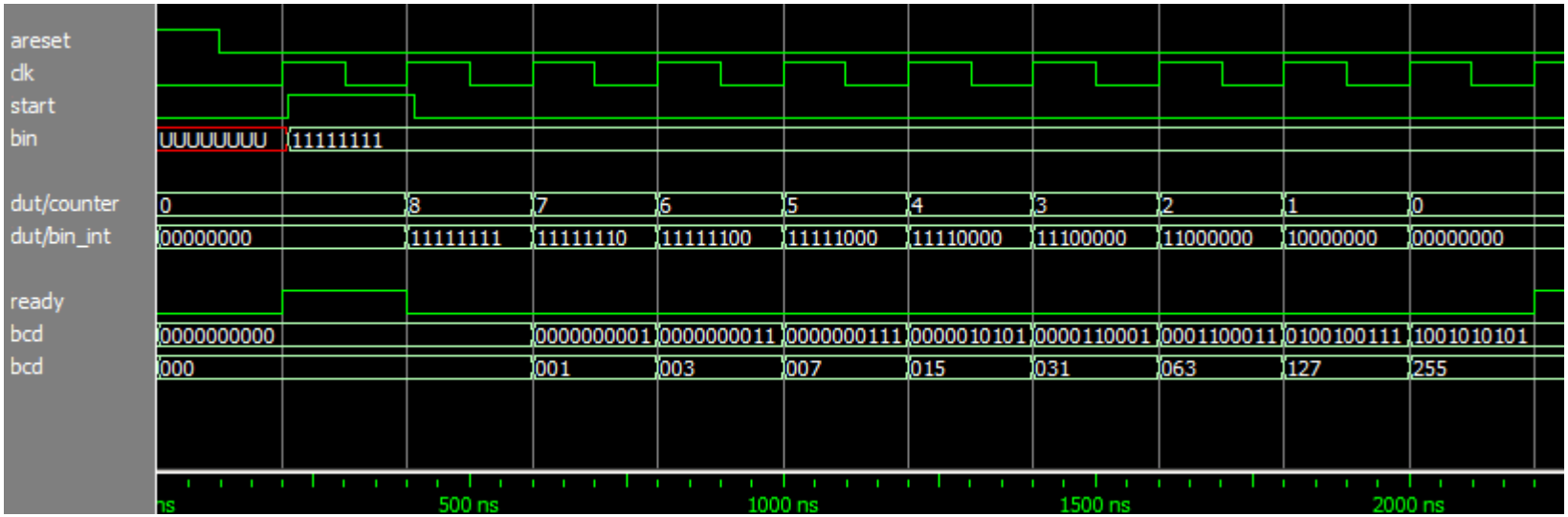
- Nu moet nog de interne BCD-waarde naar buiten gebracht worden.

```
  bcd <= bcd_int;
end process;
```

DE HAAGSE HOGESCHOOL

Simulatieresultaat

- Hieronder het simulatieresultaat



start actief op flank
counter laadt met 8
binair getal geladen

conversie
klaar

Double Dabble voor andere talstelsels

- Het principe van Double Dabble kan ook gebruikt worden voor andere talstelsels *zolang het grondtal maar even is (waarom?)*.
- Als voorbeeld een omzetting naar het grondtal 6.
- De cijfers liggen dan tussen 0 en 5 en kunnen met drie bits worden weergegeven (Binary Coded Hex = BCH).
- De bitcombinaties voor 6 (110) en 7 (111) komen niet voor en moeten overgeslagen worden. Dat kan gedaan worden door *na* het schuiven er 2 bij op te tellen

Double Dabble voor grondtal 6

- Het algoritme wordt dan iets anders:

  schuif naar links
  indien antwoord groter dan 5, pas correctie toe

- Ook nu weer kan het optellen van 2 *na* het schuiven kan vervangen worden door het optellen van 1 *voor* het schuiven.

- Dat moet dan wel gebeuren vanaf 3 (waarom?).

Correctietabel voor grondtal 6

- Hieronder een tabel met de waarden 0 t/m 5.

voor schuiven	correctie	na correctie	na schuiven*
000	-	000	000
001	-	001	010
010	-	010	100
011	001	100	1000
100	001	101	1010
101	001	110	1100

* het ingeschoven bit wordt 0 verondersteld

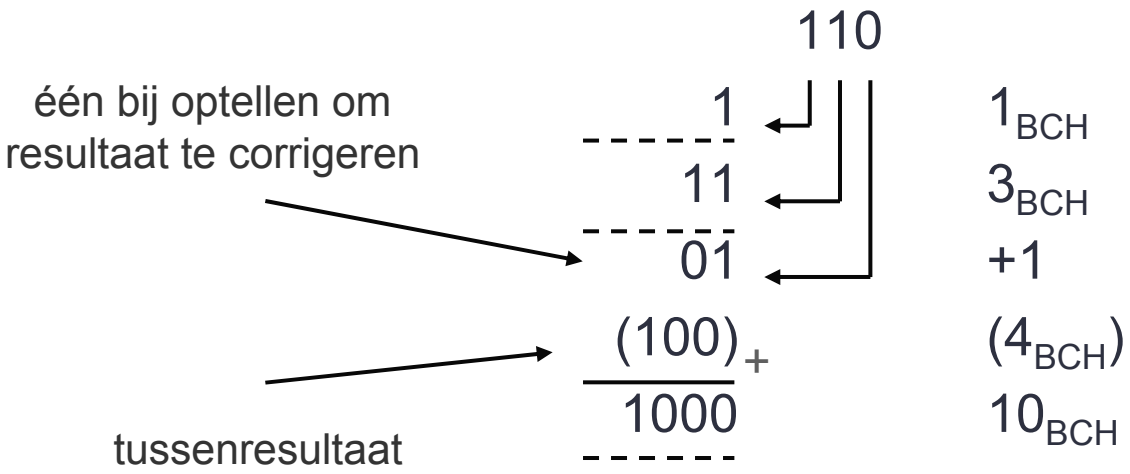
DE HAAGSE HOGESCHOOL

Voorbeeld

- Als voorbeeld het binaire getal 110_2 ($6_{10} = 10_6 = 1000_{BCH}$)

x2 = schuiven ↓

$$110_2 \quad = \quad 1000_{BCH} \quad = \quad (110_2+010_2)_{BCH} \quad = \quad ((11_2+01_2)\cdot 10_2)_{BCH}$$



BCH = binary coded hex (6-tallig)

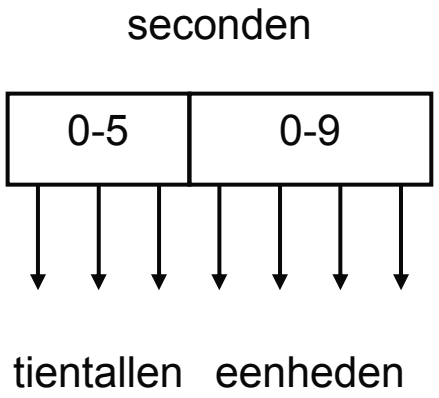
Double Dabble voor tijd

- In veel applicaties wordt de verstreken tijd bijgehouden.
- Een timer is een teller die tijd “telt”. Zo is heel eenvoudig een timer te ontwerpen die elke seconde met 1 opgehoogd wordt.
- De verstreken tijd in seconden wordt als binair getal opgeslagen.
- Het afbeelden van de verstreken tijd als decimaal getal is lastig.
- Zo lopen de bereiken van minuten en seconden van 0 t/m 59.

Double Dabble voor tijd

- Bij het afbeelden op decimalen (7-segment displays) moeten de eenheden en tientallen gesplitst worden.

- Op deze manier kunnen minuten en seconden heel eenvoudig worden weergegeven op bijvoorbeeld 7-segment display.



Double Dabble voor tijd

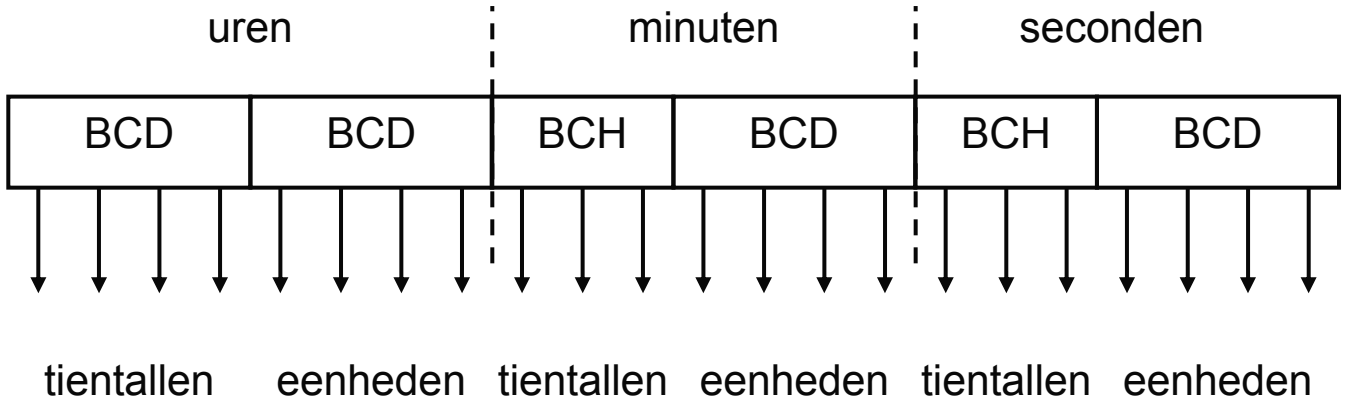
- Double Dabble kan ook gebruikt worden met secties die verschillende grondtal hebben maar waarbij de grondtallen wel even zijn.

- Zo kunnen bijvoorbeeld twee secties worden gemaakt waarbij de meestwaardige secties grondtal 6 heeft (BCH) en de minstwaardige sectie grondtal 10 (BCD).

- Het geheel is uit te breiden zodanig dat verstreken tijd kan worden afgebeeld in uren, minuten en seconden (*Binary Coded Time*).

Double Dabble voor tijd

- De opbouw van een omzetter voor het weergeven van verstreken tijd is als volgt



- Als voorbeeld wordt 1 minuut en 33 seconden (1:33) omgezet. Dit komt overeen met 93 seconden (binair 1011101).

Voorbeeld

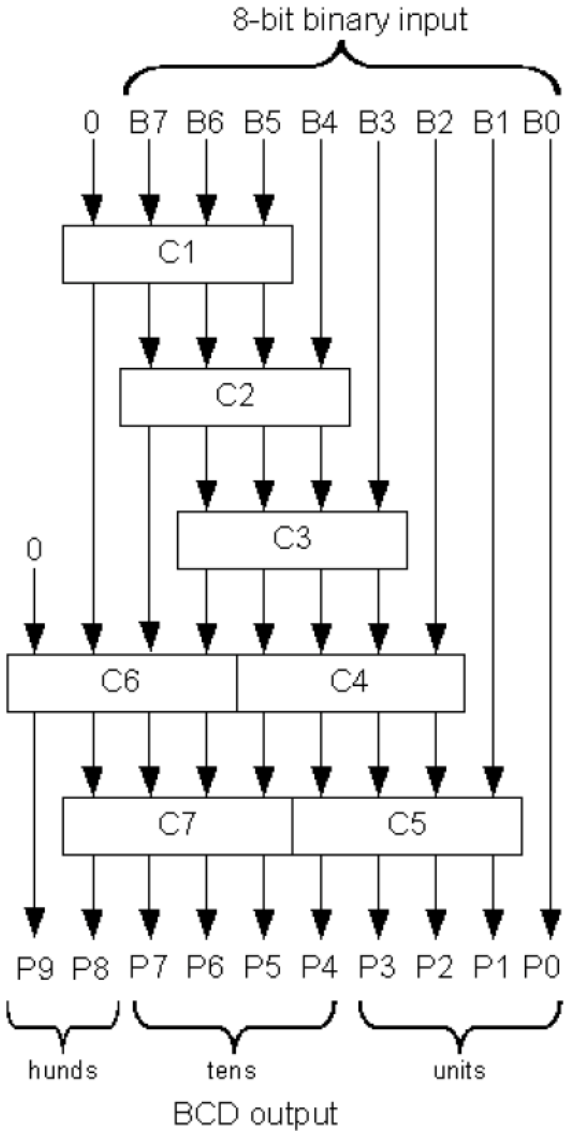
Operatie	Eenh min	Tiental sec	Eenh sec	Timer
Start				1 0 1 1 1 0 1
Shift1			1	0 1 1 1 0 1
Shift2			1 0	1 1 1 0 1
Shift3			1 0 1	1 1 0 1
Add3			1 0 0 0	1 1 0 1
Shift4		1	0 0 0 1	1 0 1
Shift5		1 0	0 0 1 1	0 1
Shift6		1 0 0	0 1 1 0	1
Add1+Add3		1 0 1	1 0 0 1	1
Shift7	1	0 1 1	0 0 1 1	
Time	1	3	3	

BCH met 4 bits

- Om een BCH cijfer te kunnen weergeven zijn drie bits voldoende.
- Natuurlijk kunnen ook vier bits gebruikt worden voor het weergeven.
- Dat levert meer hardware op, maar het makkelijker om het geheel aan te sluiten op een zeven segementen decoder.
- Merk op dat voor correctie nu 5 (4 + 1) opgeteld moet worden, omdat er acht (4 x 2) combinaties meer zijn dan bij drie bits.

Double Dabble combinatorisch

- In principe is voor een omzetter van binair naar decimaal ook een combinatorische schakeling te ontwerpen.
- Dat kan efficient met behulp van de Add-3 module (C1 t/m C7).
- Zonder uitleg wordt hier de oplossing gepresenteerd voor de omzetting van een 8-bit getal.
- Let op de fraaie cascadeschakeling.



Opgaven

- Hoeveel bits zijn er nodig om 99 uur, 59 minuten en 59 seconden (99:59:59) binair te kunnen opslaan als de teleenheid één seconde is?

- Een timer heeft 309 seconden geteld. Voer handmatig de conversie uit van naar 309_{10} naar Binary Coded Time.

- Ontwerp een combinatorische Double Dabble converter voor 9-bits binaire getallen.

- Hoe zit de opbouw van een n-bits combinatorische Double Dabble converter in elkaar?

Referenties

- http://en.wikipedia.org/wiki/Double_dabble
- http://www.embeddedrelated.com/usenet/embedded/show/18060-1.php
- http://www.lothar-miller.de/s9y/categories/44-BCD-Umwandlung
- http://users.utcluj.ro/~tmarita/PMP/Lab/Binary2BCD.pdf
- http://www.uccs.edu/~gtumbush/4211/efficient_BCD_presentation.pdf
- http://vhdlguru.blogspot.com/2010/04/8-bit-binary-to-bcd-converter-double.html

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