Eindtentamen Digitale Systemen (ET1405) 18 juni 2008, 9.00 – 12.00 uur

De tentamen is **open boek** en bestaat uit 18 multiple choice (MC) vragen en 4 open vragen. De MC-vragen dienen beantwoord te worden op het uitgereikte **MC-formulier**. Enkele aanwijzingen bij het invullen van de MC-formulieren:

- slechts 1 antwoord is het correcte antwoord (NB: a,b,c,d staan door elkaar op het antwoordvel)
- vul de gekozen vakjes *helemaal* in (liefst met ballpoint, of met potlood)
- vul het formulier pas aan het einde in om fouten te voorkomen
- *geen* veranderingen aanbrengen: haal dan een nieuw formulier
- het onbeantwoord laten van een vraag werkt altijd in uw nadeel
- vergeet niet uw *studienummer* in te vullen (cijfers *en* vakjes!)

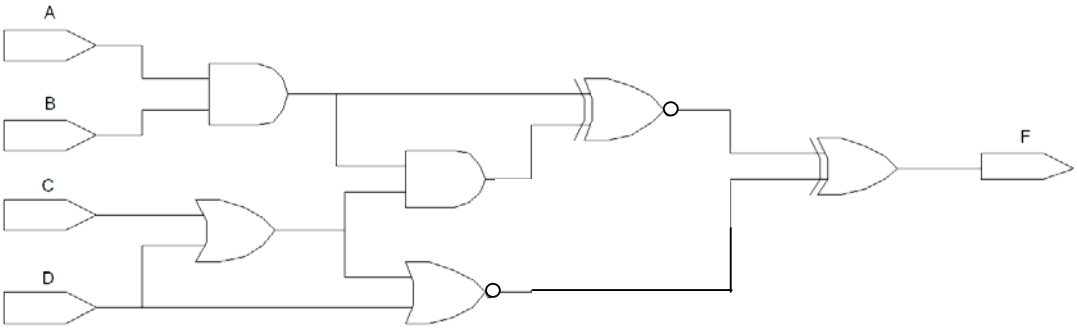
U mag het boek "Contemporary Logic Design", het VHDL boek, en eventuele prints van het **college** slides bij u hebben. Verder dus niets! Wij benadrukken dat u tijdens toetsen het tentamen dus **GEEN** gebruik mag maken van oude examens en toetsen. Gebruikt u deze toch dan zijn de tentamen fraude regels van toepassing.

Als een 5 is behaald, wordt nader onderzocht of er op grond van toets en practicumresultaten aanleiding is om het cijfer om te zetten in een 6.

Succes!

A. MC-vragen (gewicht: 18 × 3.5 % = 63 %)

Vraag 1



Voor bijgaand circuit gelden de volgende vertragingstijden:
NAND = NOR = 3ns, AND = OR = 5ns, XOR = XNOR = 7ns.

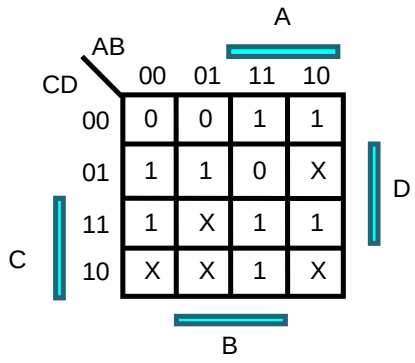
Oorspronkelijk geldt A = B = C = 1 en D = 0. Dan wordt C = 0.
Welke van de volgende uitspraken is correct?

- **a.** Er treedt een 0-1-0 hazard op mbt. F
- **b.** Er treedt een 1-0-1 hazard op mbt. F
- **c.** F wordt 1 na 24 ns en verandert niet meer
- **d.** Er treedt geen hazard op mbt. F

Vraag 2

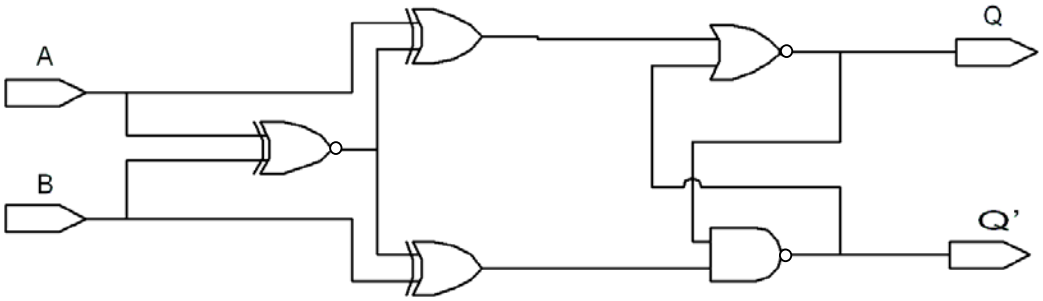
Welke expressie representeert een hazard-free netwerk voor nevenstaande K-map ? (X betekent don't care en we nemen aan dat slechts één ingang van waarde verandert)

- **a.** F = C + A'D + AD'
- **b.** F = A'D + AC + AD'
- **c.** F = CD + A'D + AD'
- **d.** geen van bovenstaande antwoorden.



Vraag 3

Gegeven bijgaand circuit:



Welke van de volgende uitspraken is correct?

- **a.** Dit circuit is geen bruikbaar geheugenelement: de Set-combinatie ontbreekt.
- **b.** Dit circuit is geen bruikbaar geheugenelement: de Reset-combinatie ontbreekt
- **c.** Dit circuit is een latch; AB = 01 is de verboden ingangscombinatie
- **d.** Dit circuit is een latch zonder verboden ingangscombinatie

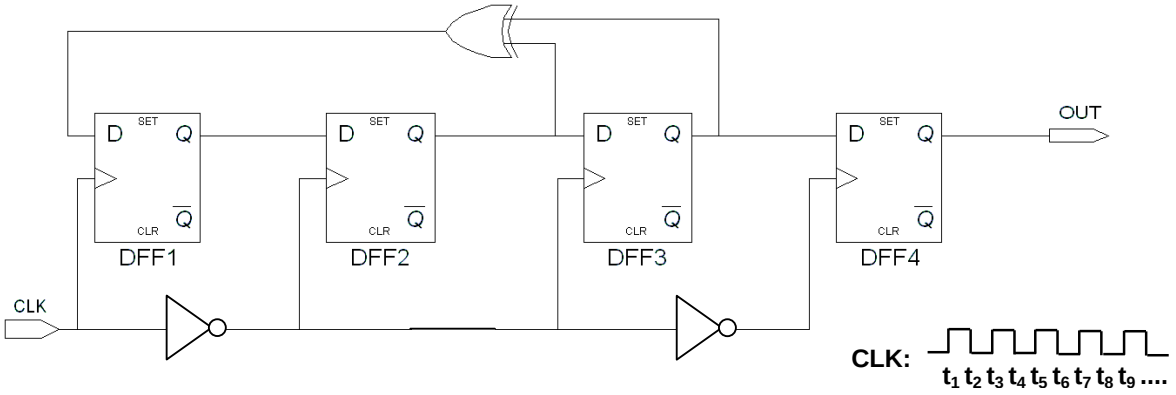
Vraag 4

Gegeven de berekening **C** = **A – B** in 2's complement notatie. Er geldt **A** = 100010 en **B** = 011010.
Wat is de waarde van **C**?

- **a.** 4
- **b.** -28
- **c.** -56
- **d.** **C** kan niet in een 6-bit 2's complement representatie worden berekend wegens overflow.

Vraag 5

Gegeven het volgende circuit met positive edge-triggered D filpflops:

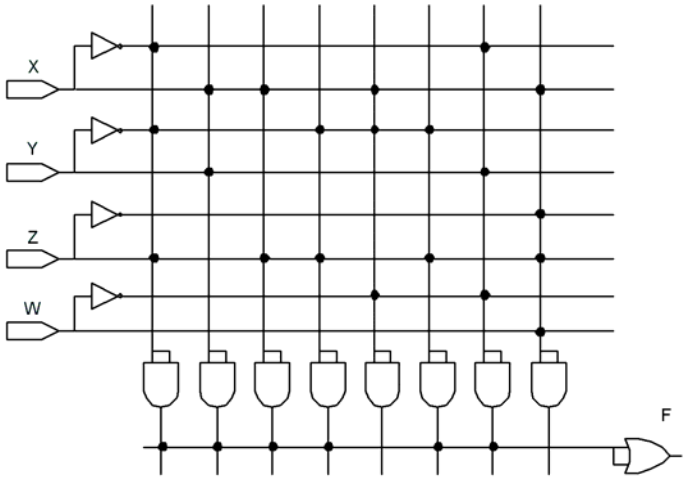


Voor bovenstaand circuit gelden de volgende vertragingstijden: INV = 1ns, XOR = 3ns, uitgang DFF t.o.v. klok = 3ns en CLK periode is 20ns. Indien tijdens de halve klokperiode voor tijdstip t_1 alle Qs **1** zijn, wanneer wordt dan voor het eerst **0** op 'Out' waargenomen? Ongeveer 3ns na:

- **a.** t_3
- **b.** t_4
- **c.** t_5
- **d.** t_6 of later.

Vraag 6

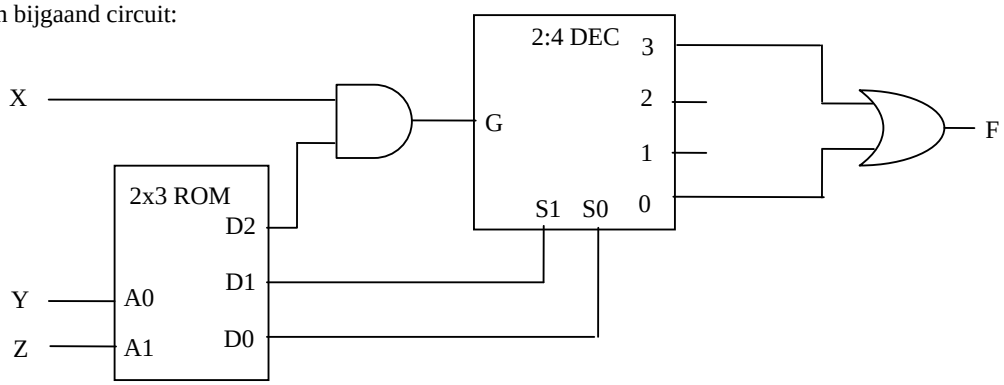
Welke van de volgende vereenvoudigingen is correct?



a. F = x(z + y) + w'(y + z)
b. F = x(z + y) + y(z + w')
c. F = y(x + w') + z(x + y')
d. geen van bovenstaande antwoorden.

Vraag 7

Gegeven bijgaand circuit:



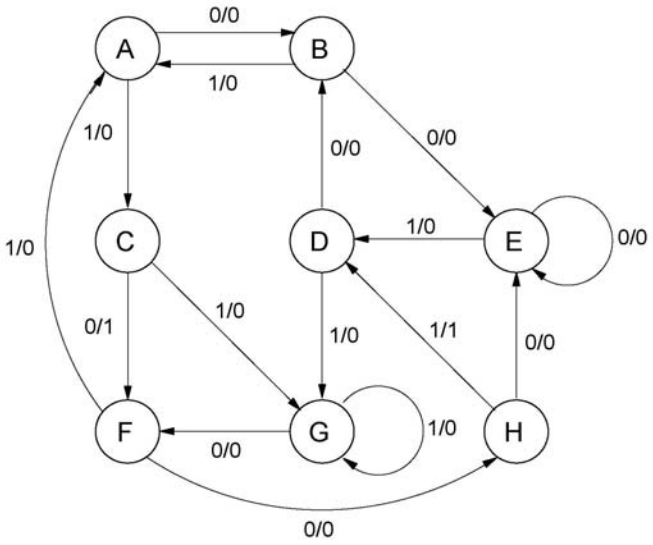
De ROM heeft de volgende inhoud:

A1	A0	D2	D1	D0
0	0	1	0	0
0	1	0	1	0
1	0	1	1	0
1	1	0	1	1

Welke van de volgende vereenvoudigingen is correct?

a. F = x(y' + z')
b. F = xy'z
c. F = x y'z'
d. geen van bovenstaande antwoorden.

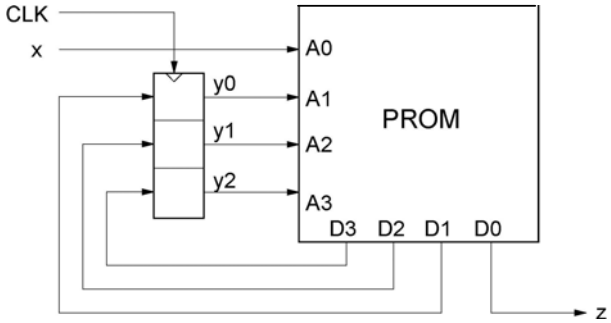
Vraag 8



Bovenstaand toestandsdiagram beschrijft een machine met 1 ingang x en 1 uitgang z. Uit de reeks achtereenvolgens binnenkomende bits x kunnen bitpatronen herkend worden. De machine geeft met z = 1 aan dat een van de 2 bitpatronen herkend is. Hierbij kunnen de patronen overlappend zijn, dat betekent dat een herkend patroon al weer een gedeelte van het volgende te herkennen patroon kan bevatten. Welke twee bitpatronen worden door de machine herkend (meest linkse bit komt eerst) ?

- **a.** 10110 en 00111
- **b.** 10110 en 11001
- **c.** 01100 en 00111
- **d.** 01100 en 11001

Vraag 9



	y2	y1	y0
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0
F	1	0	1
G	1	1	0
H	1	1	1

Als de toestandscodering uit de tabel rechts geldt voor het toestandsdiagram van vraag 8, en een realisatie m.b.v. een PROM wordt uitgevoerd, zoals hierboven links is aangegeven, wat is dan de inhoud van de PROM op adres 0100 ?

- **a.** 1011
- **b.** 1100
- **c.** 1101
- **d.** 0111

Vraag 10

Met 512 geheugenchips van 4kx4 en een decoder wordt een groter geheugensysteem voor woorden van 32 bits samengesteld. Hoeveel adreslijnen heeft dit nieuwe systeem ?

a. 14 **b.** 15 **c.** 16 **d.** 18

Vraag 11

Gegeven de volgende Delta I assembly code:

```
set   c
ld    10100111b
and   11111010b
xor   01010100b
add   00001111b
xor   00000101b
add   01101001b
```

Wat gebeurt met A, Z en C na de executie van de laatste instructie?

- **a.** A = $6A_H$, C =0, Z = 1
- **b.** A = 69_H , C = 0, Z = 1
- **c.** A = 69_H , C = 0, Z = 0
- **d.** A = $6D_H$, C = 0, Z = 0

Hieronder is een entity gegeven met daarbij zes verschillende architectures (deze gegevens hebben betrekking op vraag 12 t/m 15).

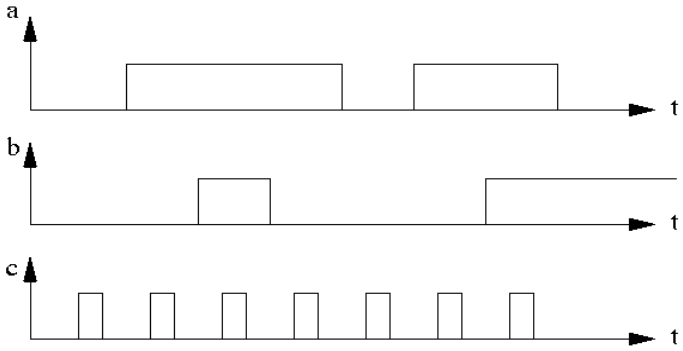
```
entity schakeling is
    port(a, b, c: in bit; d: out bit);
end schakeling;

architecture een of schakeling is
begin
    process(a, b, c)
      begin
         if ( b = '0') then d<= '0';
         elsif ( c'event and c = '0') then d <= a;
         end if;
      end process;
end een;

architecture twee of schakeling is
begin
    process( a, c)
      begin
        if ( c'event and c='1') then d<= a;
        else d <= '0';
        end if;
      end process;
end twee;

architecture drie of schakeling is
begin
   process ( a, b)
      begin
         if ( a = '1' ) then d<= b;
         else d <= '0';
         end if;
      end process;
end drie;
```

```
architecture vier of schakeling is
begin
    process(a, b)
       begin
          if ( a = '0') then d<= '1';
          else d <= not b;
          end if;
       end process;
end vier;

architecture vijf of schakeling is
begin
    process ( b )
       begin
         if ( b = '0') then d<= a;
         end if;
       end process;
end vijf;

architecture zes of schakeling is
begin
    process(a, b)
      begin
          if (b = '0') then d<= a;
          end if;
      end process;
end zes;
```

Vraag 12

- **a.** architecture drie beschrijft de combinatorische functie a nand b.
- **b.** architecture vier beschrijft de combinatorische functie a and b.
- **c.** architecture zes beschrijft een latch.
- **d.** geen van de drie genoemde mogelijkheden is goed.

In onderstaande figuur 1 zijn de binaire signalen a, b en c gegeven als functie van de tijd.



figuur 1

Vraag 13

Gevraagd wordt, als a, b en c verlopen zoals in figuur 1, hoe vaak in architecture vier het statement d<= not b wordt gescheduled en hoe vaak d<= '0' in architecture drie wordt gescheduled.

- **a.** d<= not b in architecture vier: 5 keer en d<= '0' in architecture drie: 2 keer.
- **b.** d<= not b in architecture vier: 5 keer en d<= '0' in architecture drie: 5 keer.
- **c.** d<= not b in architecture vier: 2 keer en d<= '0' in architecture drie: 5 keer.
- **d.** geen van de drie genoemde mogelijkheden is goed.

Vraag 14

- **a.** Architecture twee beschrijft een flipflop met signaal c als klok die werkt op de opgaande flank, en een synchrone reset.
- **b.** Architecture een beschrijft een flipflop met signaal c als klok die werkt op de neergaande flank, en een asynchrone reset.
- **c.** Bij de in figuur 1 gegeven signalen wordt d<= '0' in architecture twee, 10 keer gescheduled.
- **d.** Geen van de drie genoemde mogelijkheden is goed.

Vraag 15

Als de signalen a, b en c verlopen zoals in figuur 1, hoe vaak wordt dan d<= a gescheduled in architecture een?

- **a.** 6 keer
- **b.** 4 keer
- **c.** 2 keer
- **d.** geen van de drie genoemde mogelijkheden is goed.

Vraag 16

Gevraagd wordt hoe functies en procedures kunnen voorkomen in de taal VHDL.

- **a.** Een functie kan alleen concurrent statements bevatten en alleen voorkomen in packages.
- **b.** Een procedure call kan alleen als een concurrent statement voorkomen.
- **c.** Een functie body bevat alleen concurrent statements en kan voorkomen in architectures zowel als in packages.
- **d.** Geen van de drie genoemde mogelijkheden is goed.

Vraag 17

Gevraagd wordt hoe een statement kan voor komen in VHDL.

- **a.** Een sequentieel statement kan niet voorkomen in een process.
- **b.** Een sequentieel statement kan niet voorkomen in een procedure en in een funktie.
- **c.** Een concurrent statement kan niet voorkomen in een architecture body.
- **d.** Geen van de drie genoemde mogelijkheden is goed.

Vraag 18

In een architecture hebben we te maken met de vijf std_logic signalen a, b, c, d en e.
a heeft de waarde 'H', b heeft de waarde '1', c heeft de waarde 'L' en d heeft de waarde 'L'. In de architecture komen de volgende twee concurrent statements voor:

```
e <=  a and b;
e <= c or d;
```

Welke waarde krijgt e (zie ook uittreksel van std_logic op volgende bladzijde)?

- **a.** e krijgt de waarde 'X'.
- **b.** e krijgt de waarde '0'.
- **c.** e krijgt de waarde '1'.
- **d.** geen van de drie mogelijkheden is goed.

Uittreksel van “std_logic_1164”

```
    ------------------------------------------------------------------
    -- tables for logical operations
    ------------------------------------------------------------------
    -- truth table for "and" function
    CONSTANT and_table : stdlogic_table := (
    --      ---------------------------------------------------
    --      |  U    X    0    1    Z    W    L    H    -      |   |
    --      ---------------------------------------------------
            ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ),  -- | U |
            ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
            ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | 0 |
            ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 1 |
            ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | Z |
            ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | W |
            ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | L |
            ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | H |
            ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- | - |
    );
    -- truth table for "or" function
    CONSTANT or_table : stdlogic_table := (
    --      ---------------------------------------------------
    --      |  U    X    0    1    Z    W    L    H    -      |   |
    --      ---------------------------------------------------
            ( 'U', 'U', 'U', '1', 'U', 'U', 'U', '1', 'U' ),  -- | U |
            ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | X |
            ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 0 |
            ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ),  -- | 1 |
            ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | Z |
            ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | W |
            ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | L |
            ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ),  -- | H |
            ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' )   -- | - |
    );
    -- truth table for "not" function
    CONSTANT not_table: stdlogic_1d :=
    --  -------------------------------------------------
    --  |   U    X    0    1    Z    W    L    H    -   |
    --  -------------------------------------------------
         ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' );

    ------------------------------------------------------------------
    -- overloaded logical operators ( with optimizing hints )
    ------------------------------------------------------------------

    FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
    BEGIN
        RETURN (and_table(l, r));
    END "and";

    FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
    BEGIN
        RETURN  (not_table ( and_table(l, r)));
    END "nand";

    FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
    BEGIN
        RETURN (or_table(l, r));
    END "or";

    FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
    BEGIN
        RETURN  (not_table ( or_table( l, r )));
    END "nor";

    FUNCTION "not"  ( l : std_ulogic ) RETURN UX01 IS
    BEGIN
        RETURN (not_table(l));
    END "not";
```

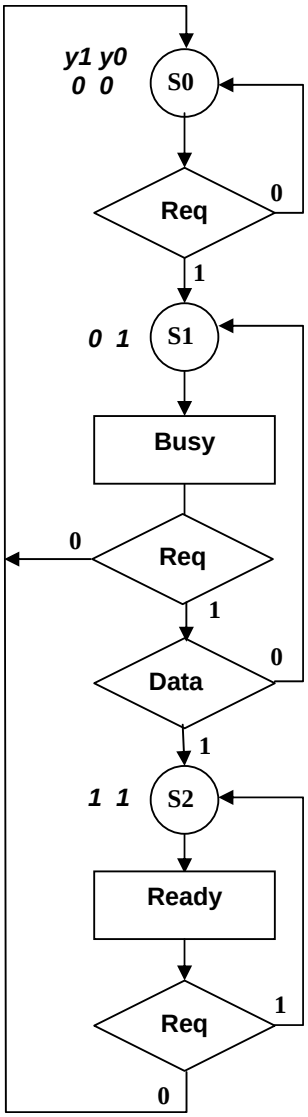
B. Open vragen (gewicht: 37%)

Vraag 19 (10 punten)

Gegeven de ASM zoals hiernaast weergegeven.
Stel dat we de toestand Si gecodeerd hebben met 2 variabelen y0, y1 volgens:

y1	y0	state
0	0	S0
0	1	S1
1	1	S2

Stel dat de ASM met D flip-flops moet worden gerealiseerd.
Teken de K-maps (voor in ieder geval **y1**$^+$ en **y0**$^+$) en leidt minimale expressies af voor **y1**$^+$, **y0**$^+$, **Busy** en **Ready**.



Vraag 20 (10 punten)

Een vruchtenleverancier wil een besturingssysteem ontwerpen voor een machine die appels kan sorteren in 3 verschillende soorten dozen. De machine bestaat uit een invoergedeelte dat de appels in de machine haalt, een sorteergedeelte, en een dozenverdeelgedeelte dat de appels in de dozen zet. De appelsorteermachine moet de appels op de volgende manier sorteren in de drie dozen:

Doos 1 - Alle kleine appels
Doos 2 - Alle middelgrote appels
Doos 3 - Alle grote appels

Deze sorteermachine heeft één sensor die drie soorten appels kan onderscheiden: Klein, Middelgroot, en Groot. De appelgrootte wordt als volgt door de sensor gecodeerd:

S1	**S0**	betekenis
0	**0**	Klein
0	**1**	Middelgroot
1	**0**	Groot
1	**1**	Sensor fout.

Het systeem dat de appels naar de dozen verdeelt wordt als volgt gecontroleerd door twee signalen:

D1	D0	betekenis
0	**1**	Stuur appel naar doos 1
1	**0**	Stuur appel naar doos 2
1	**1**	Stuur appel naar doos 3
0	**0**	Hou de appel vast op zijn plek

Het systeem kan de volgende appel pas verwerken als de huidige in de corresponderende doos is gearriveerd. De status van de dozenverdeelgedeelte is beschikbaar door middel van het invoersignaal **Done**:

- als **Done 1** is, dan is de appel in de corresponderende doos gearriveerd
- als **Done 0** is, dan is de appel nog niet in de corresponderende doos gearriveerd

De status van het invoersysteem is beschikbaar door middel van het invoersignaal **Next**:

- **Next 1**: er is een nieuwe appel beschikbaar voor verwerking door het sorteersysteem.
- **Next 0**: er is geen appel beschikbaar voor verwerking door het sorteersysteem.

Het besturingsysteem werkt als volgt: wanneer een nieuwe appel arriveert in het sorteersysteem, moet deze gestuurd worden naar de corresponderende doos gebaseerd op de sensorinformatie. Als er een sensor fout wordt gedetecteerd moet het signaal **Error** geactiveerd worden (**Error =1**). Hierna moet de machine blijven wachten totdat de fout handmatig opgelost is, waarna de huidige appel, die al in de machine ingevoerd is, opnieuw gescand wordt.

Specificeer het gewenste gedrag van het besturingssysteem voor deze appelsorteermachine m.b.v. een toestandsdiagram (FSD) voor een FSM volgens het Moore model (dit *moet* een **toestandsdiagram voor een *FSM* volgens het Moore model** zijn!) gebruikmakend van de eerder beschreven signalen **Next**, **Done**, **S1**, **S0**, **D1**, **D0**, en **Error.** Geef duidelijk de in en uitgangsignalen aan die horen bij toestanden/toestandsovergangen.

Vraag 21 (7 punten)
Schrijf de VHDL code op van een D-flipflop met gecombineerd een asynchroon reset-signaal en een synchroon set-signaal. De entity is gedefinieerd als:

```
entity Dff is
   port (ingang, reset, set, clk: in bit;
         uitgang: out bit);
end Dff;
```

Vraag 22 (10 punten)
Maak gebruik van de in Vraag 21 geïntroduceerde entity Dff en bouw met dit element een 3-bit rotator (bit3_rotator) rotator (i.e., een shifter waarbij de beide uiteinden met elkaar zijn verbonden – dus bij naar links shiften wordt de hoogste bit in de laagste positie ingeschoven en bij het naar rechts shiften de laagste bit naar de hoogste positie ingeschoven) die ook de mogelijkheid heeft om een nieuwe 3-bit waarde in te laden als een load-signaal hoog is. Hou tevens rekening met het volgende:

- vergeet niet de uitgang van de D flip-flops ook zichtbaar te maken als een port van de bit3_rotator entity.
- geef duidelijk aan welke ingangssignalen je nodig hebt en hoe je deze hebt benoemd in je VHDL code.
- vergeet de component declaratie van de Dff entity niet.
- het is niet toegestaan om processen te gebruiken.
- HINT: denk aan de multiplexer en hoe deze kan worden gebouwd met een combinatorisch netwerk.
- je moet gebruik maken van de volgende entity beschrijving:

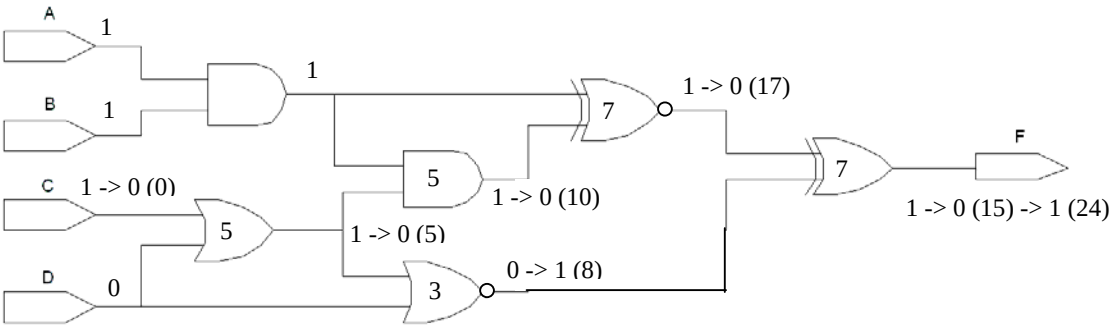
```
entity bit3_rotator is
   port (links, niets, rechts, load, reset, clk: in bit;
         -- van de signalen links, niets, rechts, load worden
         -- aangenomen dat nooit twee tegelijk '1' kunnen zijn.
         ingang : in bit_vector (2 downto 0);
         uitgang: out bit_vector (2 downto 0));
end bit3_rotator;
```

– o – o – o – o – o – o – o –

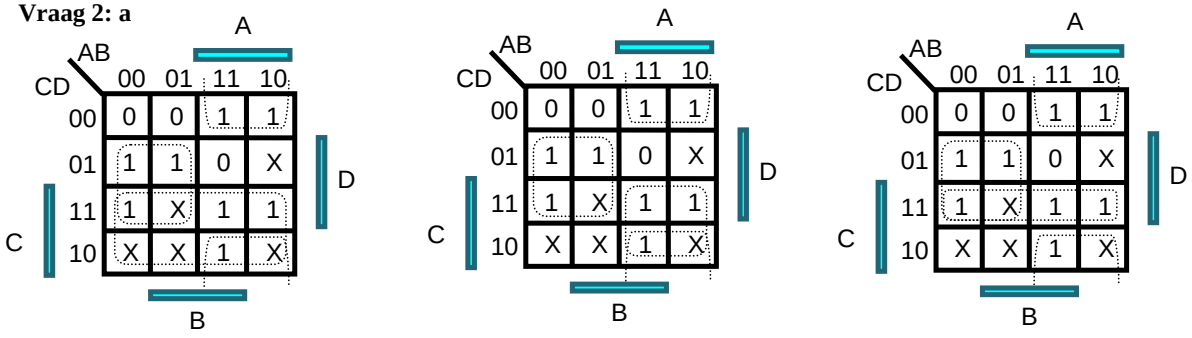
Uitwerkingen Eindtentamen Digitale Systemen (ET1405) 18 juni 2008

MC-vragen:

Vraag 1: b



Vraag 2: a

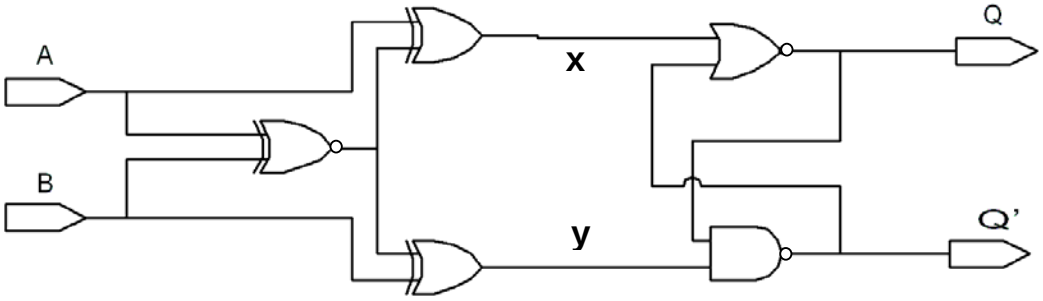


F = C + A'D + AD'

F = A'D + AC + AD'

F = CD + A'D + AD'

Vraag 3: a



A	B	X	Y	Q	Q'	
0	**0**	**1**	**1**	**0**	**1**	**reset**
0	**1**	**0**	**1**	**Q_{t-1}**	**Q'_{t-1}**	**hold**
1	**0**	**1**	**0**	**0**	**1**	**reset**
1	**1**	**0**	**0**	**0**	**1**	**reset**

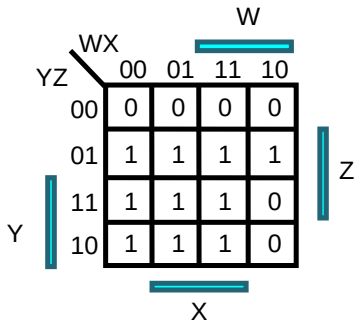
Vraag 4: d

A = 100010 = -(011101 + 1) = -30, B = 011010 = 26. Dan zou C -56 moeten zijn maar dat past niet als 2's complement waarde in een 6 bits getal.

Vraag 5: c
FF 1 en 4 lezen in op de opgaande klokflank (t1, t3, t5,..), FF 2 en 3 op de neergaande (t2, t4, t6, …)

+/- 3ns na:	Q2 XOR Q3	Q1	Q2	Q3	Q4
	0	**1**	**1**	**1**	**1**
t1	**0**	**0**	**1**	**1**	**1**
t2	**1**	**0**	**0**	**1**	**1**
t3	**1**	**1**	**0**	**1**	**1**
t4	**1**	**1**	**1**	**0**	**1**
t5	**1**	**1**	**1**	**0**	**0**

Vraag 6: c
Uit de PAL: F = x'y'z + xy + xz + y'z + y'z + x'yw'. Invullen in K-map geeft:



Bij c**: F =** y(x + w') + z(x + y') = xy + yw' + xz + y'z en dit is ook een beschrijving van de K-map.

Vraag 7: c
G = X.D2 = X.A0' = XY'
Met A1=Z en A0=Y volgt verder S1 = D1 = Y+Z en S0 = D0 = YZ
Dus F= O0 + O3= G.(S1'S0' + S1.S0) = XY'((Y + Z)'(YZ)' + (Y + Z)YZ) = XY'((Y'Z'(Y' + Z') + YZ) = XY'((Y'Z'+ YZ) = XY'Z'

Vraag 8: b
Begin bij de overgangen wanneer de uitgang een 1 geeft en volg het spoor terug: 10110 en 11001. Aan de laatste 2 bits van elke reeks is al te zien dat de andere antwoorden niet goed kunnen zijn.

Vraag 9: a
Op adres 0100 betekent ''huidige toestand'' = 010 = C en x = 0.
Uit diagram volgt: ''volgende toestand'' = F = 101 en z = 1 => 1011, dus antwoord **a**

Vraag 10: d
Om een geheugensysteem van 32 bits te maken moeten 8 4kx4 chips naast elkaar worden gezet om de 32-bits woorden samen te stellen (8 x 4 = 32). Er kunnen dan 512/8 = 64 rijen gemaakt worden van 8 4kx4 chips. Om alle woorden te adresseren hebben we dus 12(voor 4k) + 6 (voor 64 rijen) bits nodig.

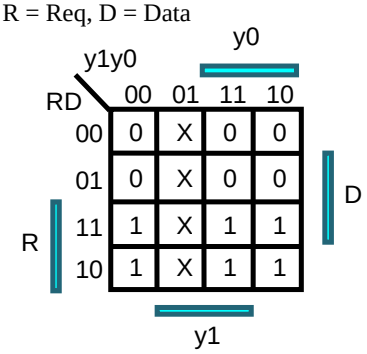
Vraag 11: d

		A	C	Z
set	C	?	1	?
ld	10100111b	10100111	1	?
and	11111010b	10100010	1	0
xor	01010100b	11110110	1	0
add	00001111b	00000110	1	0
xor	00000101b	00000011	1	0
add	01101001b	01101101	0	0

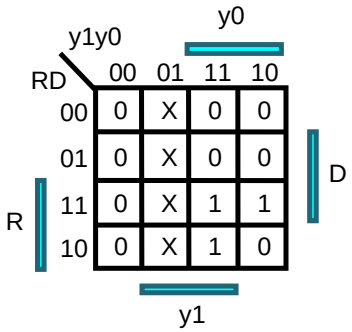
Vraag 12: c
Vraag 13: a

Vraag 14: b
Vraag 15: c
Vraag 16: d
Vraag 17: d
Vraag 18: a

Vraag 19:
R = Req, D = Data



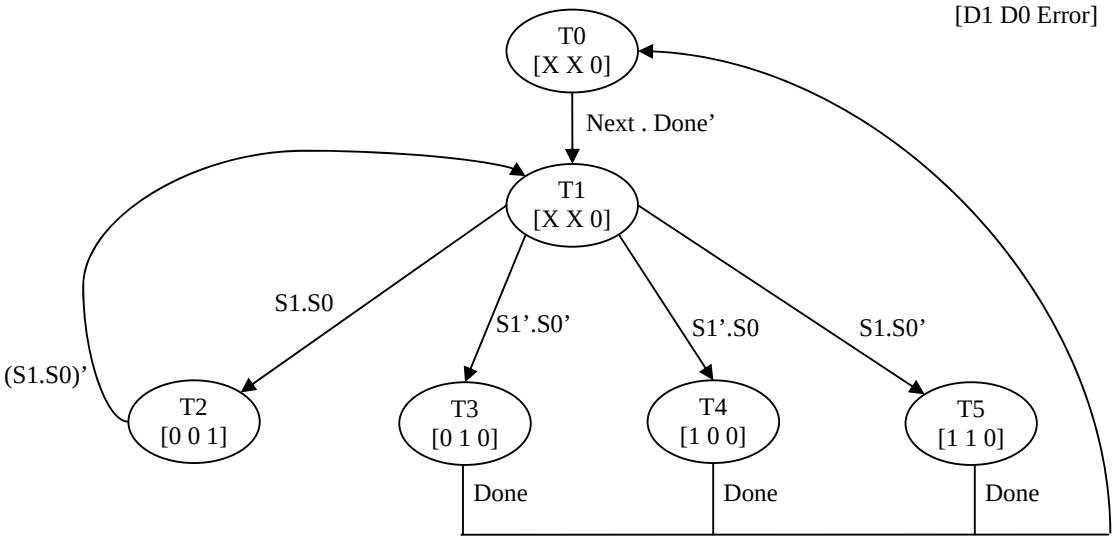
$\mathbf{y0}^{+} = R$



$\mathbf{y1}^{+} = y1.R + y0.R.D$

De uitgangen hangen alleen van de huidige toestand af en zijn makkelijk te bepalen:
Busy = y1'.y0
Ready = y1

Vraag 20:



Vraag 21:

```
entity Dff is
   port (ingang, reset, set, clk: in bit;
         uitgang: out bit);
end Dff;

architecture behaviour of Dff is
begin
   lbl1:
   process(reset, clk)
   begin
      if (reset = '1') then
         uitgang <= '0';
      elsif (clk'event and clk = '1') then  -- deze is positive-edge triggered
         if (set = '1') then
            uitgang <= '1';
         else
            uitgang <= ingang;
         end if;
      end if;
   end process clk_process;
end behaviour;
```

Vraag 22:

```
entity bit3_rotator is
   port (links, niets, rechts, load, reset, clk: in bit;
         -- van de signalen links, niets, rechts, load wordt
         -- aangenomen dat nooit twee tegelijk '1' kunnen zijn.
         ingang : in bit_vector (2 downto 0);
         uitgang: out bit_vector (2 downto 0));
end bit3_rotator;

architecture struct of bit3_rotator is
component Dff
   port (ingang, reset, set, clk: in bit;
         uitgang: out bit);
end component;
signal din0, din1, din2 : bit;
signal duit0, duit1, duit2 : bit;
begin
   inst0: Dff port map (din0, reset, '0', clk, duit0);
   inst1: Dff port map (din1, reset, '0', clk, duit1);
   inst2: Dff port map (din2, reset, '0', clk, duit2);
   din0 <= (links and duit2) or (rechts and duit1)
            or (niets and duit0) or (load and ingang(0));
   din1 <= (links and duit0) or (rechts and duit2)
            or (niets and duit1) or (load and ingang(1));
   din2 <= (links and duit1) or (rechts and duit0)
            or (niets and duit2) or (load and ingang(2));
   uitgang(0) <= duit0;
   uitgang(1) <= duit1;
   uitgang(2) <= duit2;
end struct;
```