Opgaven

en uitwerkingen bij het boek Digitale Techniek

Jesse op den Brouw

Deel 2

Versie: 0.99pl4
Datum: 15 januari 2019

DE HAAGSE HOGESCHOOL

Suggesties en/of opmerkingen over dit boek kunnen worden gestuurd naar:
J.E.J.opdenBrouw@hhs.nl.

Inhoudsopgave

7 Opgaven hoofdstuk 7 1

8 Opgaven hoofdstuk 8 4

9 Opgaven hoofdstuk 9 7

10 Opgaven hoofdstuk 10 11

A Uitwerkingen 13

7

7.1. Geef de logische functie van onderstaande CSA:

```
1 s <= '1' when a = '0' else
2      '1' when b = '1' else
3      '1' when c = '0' else
4      '0';
```

Listing P7.1: *VHDL-beschrijving Conditional Signal Assignment.*

7.2. Geef de logische functie van onderstaande CSA:

```
1 s <= a when data = '0' else
2      b when data = '1' and en = '0' else
3      c when en = '0' else
4      d;
```

Listing P7.2: *VHDL-beschrijving Conditional Signal Assignment.*

7.3. Geef een VHDL-beschrijving van een EXOR d.m.v. een Conditional Signal Assignment en een Selected Signal Assignment.

7.4. Geef de VHDL-statement(s) om een 8-bits vector één plek naar rechts te schuiven.

7.5. Geef de VHDL-statement(s) om een 8-bits vector één plek naar links te roteren, waarbij het meest significante bit in het minst significante bit wordt geplaatst (dit wordt roteren genoemd).

7.6. Geef de volledige VHDL-beschrijving van een SR-latch met overheersende set.

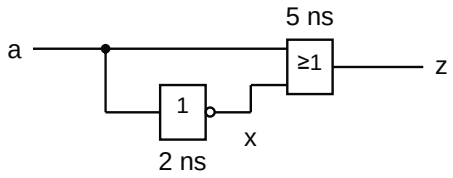
7.7. Geef een VHDL-beschrijving voor een EXOR-poort door middel van sequentiële VHDL-statements. Geef zoveel mogelijke oplossingen.

7.8. Geef een VHDL-beschrijving voor een negative edge triggered D-flipflop.

7.9. Geef een VHDL-beschrijving voor een 8-bits schuifregister dat schuift op de opgaande flank.

7.10. Eerder is code van een 8-input NOR gegeven. Geef nog twee andere mogelijkheden voor het beschrijven van de NOR. Gebruik de eerder gegeven code als leidraad.

7.11. Gegeven de schakeling in figuur P7.1. Alle signalen zijn van het type `bit` en op '0' geïnitialiseerd. Op tijdstip t = 3 ns wordt `a` logisch 1. Reken het schema door en stel de event-lijsten op voor elk event-tijdstip.



Figuur P7.1: *Schema voor doorrekenen tijdgedrag.*

7.12. Als opgave 7.11, maar nu wordt signaal `a` na 10 ns weer logisch 0.

7.13. Als opgave 7.11, maar nu als geen vertragingen zijn opgegeven.

7.14. Geef het schema van onderstaande VHDL-codes. Maak gebruik van multiplexers en poorten.

```
1    if sel = '1' then
2       y <= a and b;
3    else
4       y <= a or b;
5    end if;
```

Listing P7.3: *VHDL-code voor synthese.*

```
1    if x = y then
2        z <= '1';
3    else
4        z <= '0';
5    end if;
```

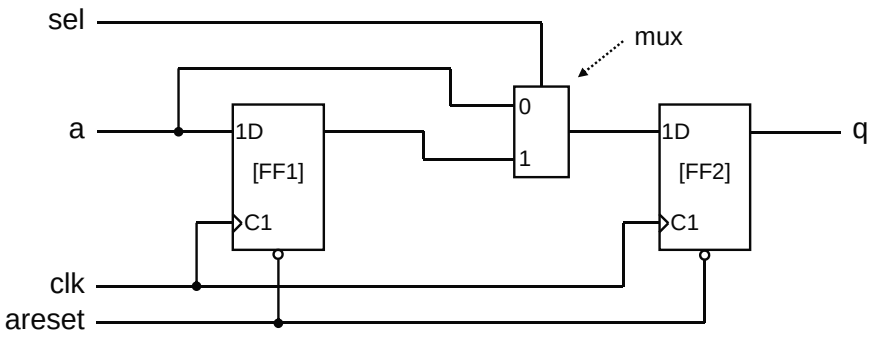
Listing P7.4: *VHDL-code voor synthese.*

7.15. Geef het schema van onderstaande VHDL-code:

```
1  if rising_edge(clk) then
2      ff1 <= x;
3      if ff1 = '1' and x = '0' then
4          ff2 <= '1';
5      else
6          ff2 <= '0';
7      end if;
8  end if;
```

Listing P7.5: *VHDL-code voor synthese.*

7.16. Geef de complete VHDL-beschrijving van onderstaand schema.



Figuur P7.2: *Schema voor ontwerp VHDL-code.*

7.17. Geef het schema van de onderstaande code. Probeer zo dicht mogelijk de originele code te volgen.

```
signal x : bit_vector (7 downto 0);
signal q : bit;
    ...
    process (x) is                  -- sensitive for x
    variable p : bit;               -- only p, no need to declare i
    begin
        p := '0';                   -- initialize to 0
        for i in 7 downto 0 loop    -- i = 7,6,5,4,3,2,1,0
            p := p or x(i);         -- OR p with each element
        end loop;
        q <= not p;                 -- signal assignment
    end process;
```

Listing P7.6: *VHDL-beschrijving van een 8-input NOR voor synthese.*

8

8.1. Ontwerp een VHDL-beschrijving van een 4-bits teller die ook een begintelstand kan laden.

8.2. Ontwerp een 4-bits omhoogteller met JK-flipflops en poorten.

8.3. Ontwerp een 4-bits omlaagteller met T-flipflops en poorten. Ga uit van een 4-bits omhoogteller.

8.4. Ontwerp een 6-teller, een teller die telt van 0 t/m 5, met losse poorten en T-flipflops.

8.5. Een teller is te ontwerpen met een register en een full-adder. Op één ingang van de full-adder wordt de telstand aangeboden, op de andere ingang wordt de constante 00..01 aangeboden. Implementeer een enable in deze teller zodat de telstand behouden blijft ook al passeert er een klokflank.

8.6. Gegeven een deel van de VHDL-beschrijving van de BCD-teller in listing P8.1. Wat doet de teller als deze per ongeluk in stand 12 terecht is gekomen?

```
1   process (clk, areset) is
2   begin
3       if areset = '1' then
4           count <= "0000";
5       elsif rising_edge(clk) then
6           if count = "1001" then
7               count <= "0000";
8           else
9               count <= count + 1;
10          end if;
11      end if;
12  end process;
13
14  tc <= '1' when count = "1001" and en = '1' else '0';
15  q <= count;
```

Listing P8.1: *De VHDL-beschrijving van een 1-decade BCD-teller.*

8.7. Pas de VHDL-code in listing P8.1 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.

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

8.9. 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.

8.10. Bepaal de resolutie van de Duty Cycle voor een n-bit digitaal PWM-systeem.

8.11. Het PWM-signaal is hoog als de referentiewaarde groter is dan de tellerwaarde (op enig moment). Daardoor is een DC van 100% niet haalbaar. Een student past de schakeling aan zodat het PWM-signaal is hoog als de referentiewaarde groter is dan of gelijk is aan de tellerwaarde (op enig moment). Is dit een slimme keuze? Zijn er andere problemen te verwachten? Motiveer het antwoord.

8.12. Bedenk een manier om tot een DC van 100% te komen zonder de vergelijkschakeling aan te passen.

8.13. Een ontwerper heeft de onderstaande code geschreven, zie listing P8.2. Het betreft hier een 1-decade BCD-teller. Merk op dat de beschrijving van de terminal count verweven is met de teller. Leg kort uit waarom de terminal count niet correct functioneert.

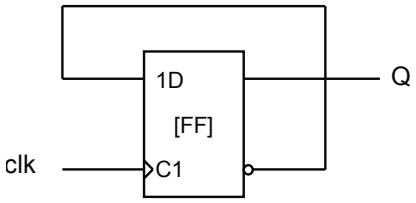
```
1    process (clk, areset) is
2    begin
3        if areset = '1' then
4            count <= "0000";
5        elsif rising_edge(clk) then
6            if en = '1' then
7                if count = "1001" then
8                    count <= "0000";
9                    tc <= '1';
10               else
11                   count <= count + 1;
12                   tc <= '0';
13               end if;
14           end if;
15       end if;
16   end process;
17
18   q <= count;
```

Listing P8.2: *Een deel van de beschrijving van de 1-decade BCD-teller.*

9

9.1. Van de tweedeler in figuur P9.1 zijn de volgende timingparameters gegeven.

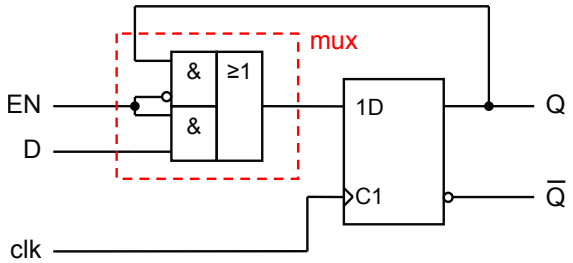
$$t_{su}(\text{FF}) = 15 \text{ ns}$$
$$t_{h}(\text{FF}) = 5 \text{ ns}$$
$$t_{P(min)}(\text{FF}) = 10 \text{ ns}$$
$$t_{P(max)}(\text{FF}) = 35 \text{ ns}$$



Figuur P9.1: *Timingparameters en schema van een tweedeler.*

Bepaal de maximale frequentie waarop dit systeem kan werken. Is betrouwbare dataoverdracht mogelijk?

9.2. Gegeven is een flipflop met enable in figuur P9.2. Bepaal de setup- en holdtijd voor signaal D t.o.v. van de actieve klokflank. Bepaal de maximale frequentie waarop dit systeem betrouwbare kan werken. Teken tijddiagrammen waarmee de berekeningen kunnen worden gemaakt.

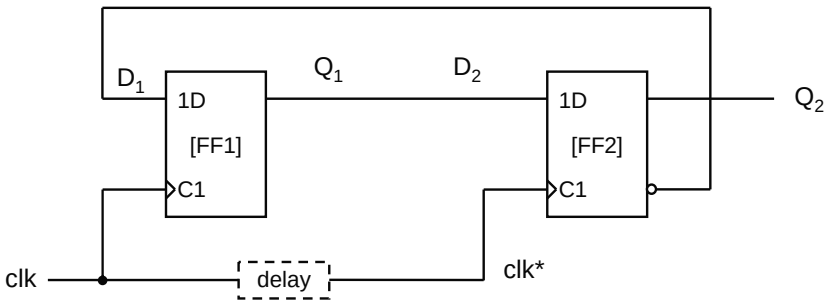


$$t_{su}(\text{FF}) = 14 \text{ ns}$$
$$t_{h}(\text{FF}) = 4 \text{ ns}$$
$$t_{P(min)}(\text{FF}) = 7 \text{ ns}$$
$$t_{P(max)}(\text{FF}) = 17 \text{ ns}$$
$$t_{P(min)}(\text{logic}) = 8 \text{ ns}$$
$$t_{P(max)}(\text{logic}) = 19 \text{ ns}$$

Figuur P9.2: *Schema en timing van een D-flipflop met enable.*

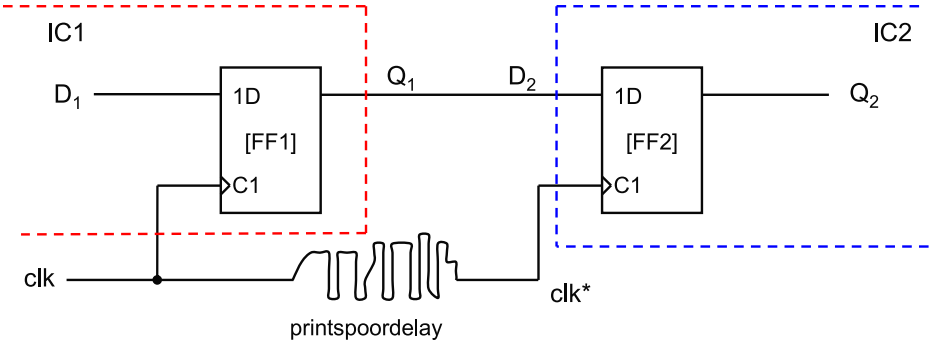
9.3. Gegeven is het schema in figuur P9.3. Het kloksignaal van FF2 is iets vertraagd t.o.v. de klok van FF1. Druk de maximale frequentie uit in de timingparameters

van de flipflops en de clock skew (delay). Analyseer de logische werking van dit systeem.



Figuur P9.3: *Twee flipflops met verbindingen en klokskew.*

9.4. Gegeven is het schema in figuur P9.4. Het laat dataoverdracht tussen twee flipflops van verschillende ic's zien.



Figuur P9.4: *Verbinding tussen verschillende ic's.*

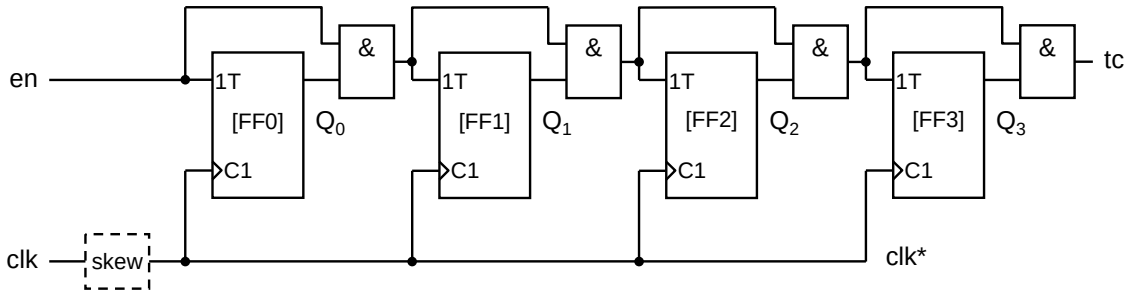
De timing van de flipflops is verschillend:

FF1: $t_{su}/t_h/t_{P(min)}/t_{P(max)} = 10/3/7/12$ ns

FF2: $t_{su}/t_h/t_{P(min)}/t_{P(max)} = 20/6/10/25$ ns

De skewtijd bedraagt 2 ns. Is betrouwbare dataoverdracht mogelijk?

9.5. Gegeven is de onderstaande 4-bits teller.



Figuur P9.5: *4-bits teller opgebouwd uit 1-bit tellersecties.*

Van de componenten zijn de volgende timingparameters gegeven:

Flipflops: $t_{su}/t_h/t_{P(min)}/t_{P(max)} = 2/1/4/7$ ns

AND: $t_{P(min)}/t_{P(max)} = 3/5$ ns

Skew: $t_P = 4$ ns

In dit systeem zijn meerdere paden te ontdekken waarlangs data wordt getransporteerd.

a) Geef alle paden waarlangs data van flipflop naar flipflop wordt getransporteerd.

Van dit systeem moet de maximale frequentie worden berekend.

b) Teken een tijddiagram met alleen relevante timingparameters waarmee de maximale frequentie berekend moet worden.

c) Bereken de maximale frequentie waarop dit systeem nog betrouwbaar werkt.

Het signaal en (enable) wordt vertraagd aangeboden aan de ingang van de flipflops. Dat heeft invloed op de setup- en holdtijden van signaal en t.o.v. het kloksignaal clk.

d) Teken een tijddiagram met alleen relevante timingparameters waarmee de setuptijd en holdtijd van en (enable) t.o.v. de actieve flank van clk berekend kan worden.

e) Bereken de setuptijd en holdtijd van en (enable) t.o.v. de actieve flank van clk.

De uitgang tc geeft aan of de teller op de maximale stand staat. Na het passeren van de actieve klokflank duurt het even voordat de nieuwe waarde op de uitgang beschikbaar is.

f) Teken een tijddiagram met alleen relevante timingparameters waarmee de minimale en maximale vertragingstijden van uitgang tc t.o.v. de actieve flank van clk berekend kan worden.

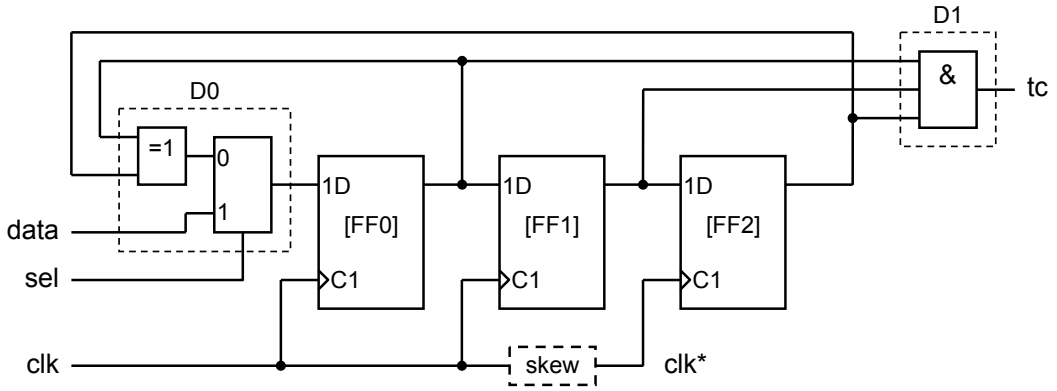
g) Bepaal de minimale en maximale vertragingstijden van de uitgang tc t.o.v. de actieve klokflank.

De opbouw van de teller is zeer elegant en makkelijk uitbreidbaar maar hoe zit het nou met de timing.

h) Beschrijf kort wat de invloed is van het aantal telsecties op de timing, bv. als het aantal secties wordt uitgebreid.

9.6. Gegeven de schakeling in figuur P9.6. Hierin zijn drie D-flipflops opgenomen (FF0, FF1 en FF2), twee stukken combinatoriek (D0, D1) en één klokvertraging (skew). D0 is combinatoriek voor de ingang van FF0 en D1 is combinatoriek na de uitgangen van FF0, FF1 en FF2. FF2 krijgt een vertraagd kloksignaal aangeboden (skew).

Van de bouwstenen zijn de volgende timingparameters gegeven:



Figuur P9.6: *Synchrone flipflop-schakeling*

D-flipflops: $t_{su}/t_h/t_{P(min)}/t_{P(max)} = 2/1/6/8$ ns

AND: $t_{P(min)}/t_{P(max)} = 2/3$ ns

EXOR: $t_{P(min)}/t_{P(max)} = 3/5$ ns

Mux: $t_{P(min)}/t_{P(max)} = 4/6$ ns

Skew: $t_P = 3$ ns

Van de skew is maar één vertragingstijd gegeven, er wordt geen onderscheid gemaakt tussen de minimale en maximale vertragingstijd. Het betreft hier namelijk clock skew.

Merk op dat er in dit systeem **vier** paden zijn tussen de flipflops waarlangs data wordt getransporteerd.

a) Geef aan welke paden er tussen de flipflops zijn waarlangs data wordt getransporteerd. Geef hierbij duidelijk aan welke stukken combinatorische logica wordt gepasseerd.

Van het systeem moet de maximale frequentie worden berekend. Na onderzoek blijkt het pad FF2 → D0 → FF0 de grootste minimale periodetijd op te leveren.

b) Teken een tijddiagram met alleen relevante timingparameters waarmee de maximale frequentie berekend moet worden.

c) Bereken de maximale frequentie waarop dit systeem nog betrouwbaar werkt.

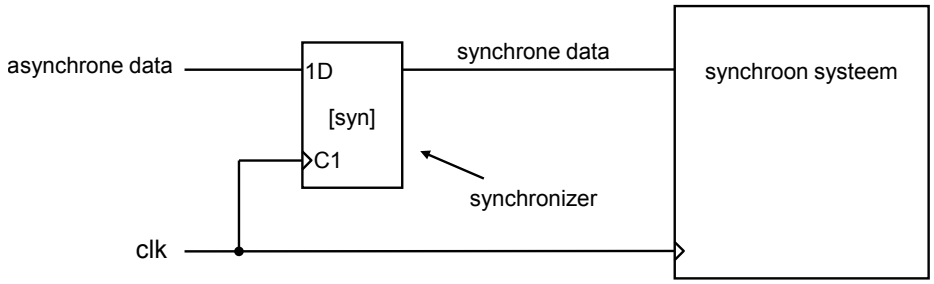
FF2 heeft last van klokskew, het kloksignaal komt vertraagd aan op de klokingang van FF2. Dat heeft invloed op de holdtijd van de dataingang van FF2.

d) Teken een tijddiagram met alleen relevante timingparameters waarmee de *hold slack* van de dataingang van FF2 berekend kan worden.

e) Bereken de *hold slack* van de dataingang van FF2.

10

10.1. Gegeven een synchroon systeem in figuur P10.1 met een frequentie van 20 MHz. Data komt asynchroon binnen met 100 kHz.



Figuur P10.1: *Een synchronisatiesysteem*

Van de synchronizer 74F50109 is gegeven: $\tau = 0{,}135$ ns, $T_0 = 9{,}8 \cdot 10^6$ s

De setuptijd van het achterliggende systeem is 20 ns. Bereken de MTBF van dit systeem.

10.2. Gegeven het systeem van in figuur P10.2.

Van flipflops FF1 en FF2 is gegeven $t_{su}/t_h/t_{P(min)}/t_{P(max)} = 5/2/4/10$ ns

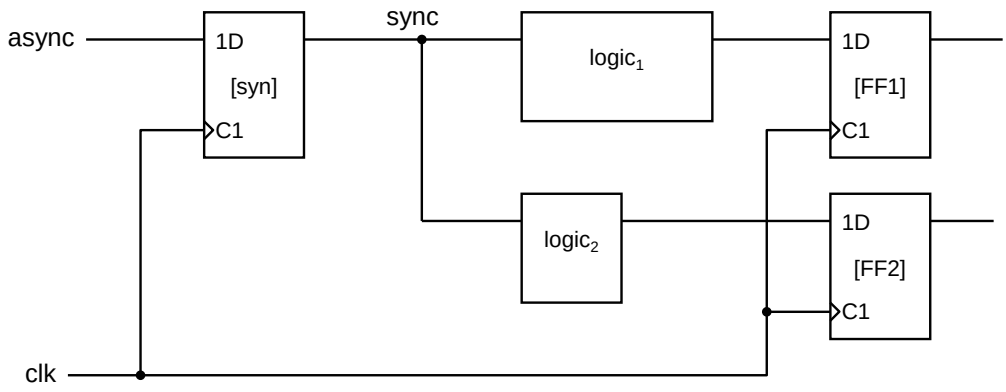
Van de synchronizer is gegeven $t_{su}/t_h/t_{P(min)}/t_{P(max)} = 2/0/3/5$ ns, $\tau = 0{,}135$ ns, $T_0 = 9{,}8 \cdot 10^6$ s.

Van logica_1 is gegeven $t_{P(min)}/t_{P(max)} = 10/20$ ns.

Van logica_2 is gegeven $t_{P(min)}/t_{P(max)} = 5/12$ ns.

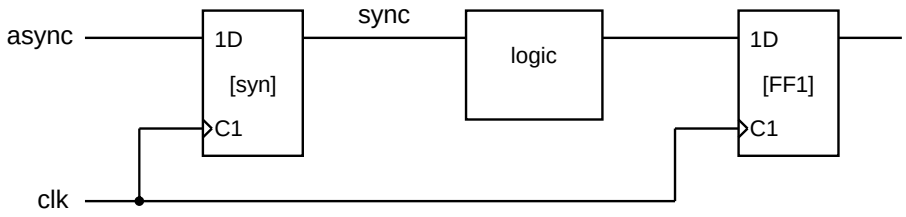
De systeemfrequentie is 25 MHz. De frequentie van de asynchroon binnenkomende data bedraagt 2 kHz.

Bepaal de MTBF van dit systeem.



Figuur P10.2: *Een synchronisatiesysteem*

10.3. Gegeven de schakeling in figuur P10.3.



Figuur P10.3: *Synchrone schakeling met synchronizer.*

Van flipflop FF1 is gegeven: $t_{su}/t_h/t_{P(min)}/t_{P(max)} = 5/2/4/10$ ns
Van de synchronizer is gegeven: $\tau = 0{,}15$ ns, $T_0 = 9{,}8 \cdot 10^6$ s
Van logica is gegeven: $t_{P(min)}/t_{P(max)} = 7/10$ ns

De systeemfrequentie is 40 MHz. Een ontwerper wil dat bij dit systeem de gemiddelde tijd tussen twee fouten 1 jaar (1 jaar = 365 dagen) bedraagt. Bereken de maximaal toegestane frequentie van het asynchroon binnenkomende signaal om aan de eis van de ontwerper te kunnen voldoen.

A

Uitwerkingen

Uitwerking opgave 7.1.
Elke regel levert een bijdrage aan de functie s. Eerst wordt de eerste regel uitgewerkt. De uitgang s wordt logisch '1' als $a = 0$. Dit stukje levert de functie $1 \cdot \overline{a}$ op (let op de 1). De tweede regel wordt uitgevoerd als de eerste regel niet waar is, dus als $a = 1$ én $b = 1$. De bijdrage van deze regel is $\overline{\overline{a}} \cdot 1 \cdot b$. De derde regel wordt uitgevoerd als de eerste twee voorwaarden niet waar zijn, dus als $a = 1$ en $b = 0$. De bijdrage is $\overline{\overline{a}} \cdot \overline{b} \cdot 1 \cdot \overline{c}$.

De hele functie is nu op te schrijven een te vereenvoudigen

$$\begin{aligned} s &= 1 \cdot \overline{a} + \overline{\overline{a}} \cdot 1 \cdot b + \overline{\overline{a}} \cdot \overline{b} \cdot 1 \cdot \overline{c} + \overline{\overline{a}} \cdot \overline{b} \cdot \overline{c} \cdot 0 \\ &= \overline{a} + a \cdot b + a \cdot \overline{b} \cdot \overline{c} \\ &= \overline{a} + a \cdot (b + \overline{b} \cdot \overline{c}) \\ &= \overline{a} + a \cdot (b + \overline{c}) \\ &= \overline{a} + (b + \overline{c}) \end{aligned} \tag{A.1}$$

Het is natuurlijk ook mogelijk om een waarheidstabel in te vullen:

Uitwerking opgave 7.2.
Het uitwerken van deze functie m.b.v. schakelalgebra is een tijdrovende en lastige zaak. Slimmer is om direct een functietabel op te zetten. Hierin zijn de uitgangswaarden geen constanten maar variabelen. Het opzetten gaat op dezelfde manier als een waarheidstabel. De uitgang s wordt a als $data = 0$. De bijdrage aan de functie is $a \cdot \overline{data}$. De tweede regel levert b op als $data = 1$ en $en = 0$. Nu is het zo dat de tweede regel alleen maar kan worden uitgevoerd (denk aan simulatie) als de eerste regel niet wordt uitgevoerd en dat is als $data = 1$. De voorwaarde $data = 1$ in de tweede regel is dus overbodig! De bijdrage van de tweede regel is $b \cdot data \cdot \overline{en}$. Nog leuker is de derde regel. Deze regel

Tabel A.1: *Waarheidstabel voor functie s, met uitleg.*

a	b	c	s	
0	0	0	1	eerste regel $a = 0$
0	0	1	1	eerste regel $a = 0$
0	1	0	1	eerste regel $a = 0$
0	1	1	1	eerste regel $a = 0$
1	0	0	1	derde regel $abc = 100$
1	0	1	0	de laatste regel
1	1	0	1	tweede regel $ab = 10$
1	1	1	1	tweede regel $ab = 10$

wordt alleen maar uitgevoerd als $en = 0$ maar deze voorwaarde staat ook al in de tweede regel! Dus als de tweede regel wordt uitgevoerd moet $en = 0$ waar zijn (natuurlijk moet $data = 1$ zijn). De derde regel wordt *nooit* uitgevoerd! Al met al de onderstaande tabel:

Tabel A.2: *Waarheidstabel voor functie s, met uitleg.*

$data$	en	s	
0	0	a	eerste regel $data = 0$
0	1	a	eerste regel $data = 0$
1	0	b	tweede regel $data = 1$ en $en = 0$
1	1	d	laatste regel $data = 1$ en $en = 1$

De functie kan heel makkelijk gevonden worden.

$$s = a \cdot \overline{data} + b \cdot data \cdot \overline{en} + d \cdot data \cdot en \tag{A.2}$$

De functie is op een andere manier te schrijven waardoor het gebruik van multiplexers duidelijk te zien is.

$$s = \underbrace{\overline{data} \cdot a + data \cdot \underbrace{(\overline{en} \cdot b + en \cdot d)}_{\text{mux}}}_{\text{mux}} \tag{A.3}$$

Uitwerking opgave 7.3.

Hieronder zijn de twee mogelijkheden weergegeven. Links is de versie met CSA en rechts de versie met SSA. Aangezien de EXOR een '1' geeft als de ingangen ongelijk zijn, moeten eerst twee regels worden beschreven waarbij de uitgang '1' wordt. Dit geeft gelijk al aan dat de EXOR niet vereenvoudigd kan worden. In de SSA-variant wordt eerst een vector samengesteld uit de twee afzonderlijke ingangen. Daarna kan EXOR worden beschreven

als een waarheidstabel.

```
s <= '1' when a = '0' and b = '1' else
     '1' when a = '1' and b = '0' else
     '0';
```

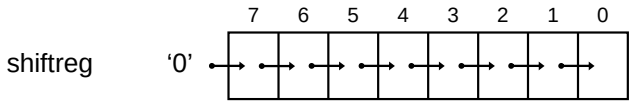
Listing A.1: *CSA-code voor functie s.*

```
ab_vec <= a & b;
with ab_vec select
    s <= '0' when "00",
         '1' when "01",
         '1' when "10",
         '0' when "11",
         'X' when others;
```

Listing A.2: *SSA-code voor functie s.*

Uitwerking opgave 7.4.

Zie de figuur hieronder. De toekenning verloopt als volgt: de '0' wordt toegekend aan element 7. Element 7 wordt toegekend aan 6. Zo gaat dat verder.



Figuur A.1: *Uitbeelding van naar rechts schuiven.*

In code wordt dat:

```
shiftreg(7) <= '0';
shiftreg(6) <= shiftreg(7);
shiftreg(5) <= shiftreg(6);
shiftreg(4) <= shiftreg(5);
shiftreg(3) <= shiftreg(4);
shiftreg(2) <= shiftreg(3);
shiftreg(1) <= shiftreg(2);
shiftreg(0) <= shiftreg(1);
```

Listing A.3: *Code voor het naar links schuiven van een 8-bits waarde.*

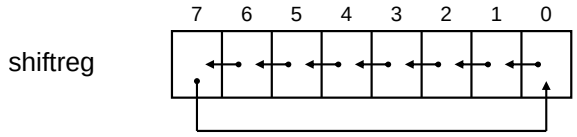
Of sneller:

```
shiftreg <= '0' & shiftreg(7 downto 0);
```

Listing A.4: *Code voor het naar links schuiven van een 8-bits waarde.*

Uitwerking opgave 7.5.

In de onderstaande figuur is weergegeven wat er moet gebeuren. Alle elementen moeten één plaats naar links worden geschoven en het hoogste element moet toegekend worden aan het laagste element. Dit wordt roteren genoemd. Elke microprocessor heeft een rotate-instructie aan boord.



Figuur A.2: *Uitbeelding van het linksom roteren van een 8-bits waarde.*

De VHDL-code moet zo geschreven worden dat element 6 toegekend wordt aan element 7, element 5 aan element 4 etc. Element 0 moet toegekend worden aan element 1. Element 7 wordt toegekend aan element 0.

```
1 shiftreg(7) <= shiftreg(6);
2 shiftreg(6) <= shiftreg(5);
3 shiftreg(5) <= shiftreg(4);
4 shiftreg(4) <= shiftreg(3);
5 shiftreg(3) <= shiftreg(2);
6 shiftreg(2) <= shiftreg(1);
7 shiftreg(1) <= shiftreg(0);
8 shiftreg(0) <= shiftreg(7);
```

Listing A.5: *Code voor het linksom roteren van een 8-bits waarde.*

Deze acht toekenningen kunnen veel sneller opgeschreven worden door gebruik te maken van vector slices.

```
1 shiftreg(7 downto 1) <= shiftreg(6 downto 0);
2 shiftreg(0)          <= shiftreg(7);
```

Listing A.6: *Code voor het linksom roteren van een 8-bits waarde.*

Of nog sneller (als in: minder code):

```
1 shiftreg(7 downto 0) <= shiftreg(6 downto 0) & shiftreg(7);
```

Listing A.7: *Code voor het linksom roteren van een 8-bits waarde.*

Of nóg sneller (als in: nóg minder code):

```
1 shiftreg <= shiftreg(6 downto 0) & shiftreg(7);
```

Listing A.8: *Code voor het linksom roteren van een 8-bits waarde.*

Uitwerking opgave 7.6.

De code is hieronder weergegeven.

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

entity sr_latch_w_domset is
    port ( s : in    std_logic;
           r : in    std_logic;
           q : out   std_logic);
end sr_latch_w_domset;

architecture logic of sr_latch_w_domset is
signal q_int : std_logic;
signal notq  : std_logic;
begin

q     <= q_int;
q_int <= s nand notq;
notq  <= r nand q_int;

end logic;
```

Listing A.9: *Code voor een SR-latch met overheersende set.*

Uitwerking opgave 7.7.

Hieronder is een aantal oplossingen gegeven. De eerste twee oplossingen gaan uit van de definitie van de EXOR-poort en zijn min of meer recht-toe-recht-aan.

```vhdl
if a = '1' and b = '0' then
    f <= '1';
elsif a = '0' and b = '1' then
    f <= '1';
else
    f <= '0';
end if;
```

Listing A.10: *Code voor een EXOR-poort.*

```vhdl
f <= '0'   -- default
if a = '1' and b = '0' then
    f <= '1';
elsif a = '0' and b = '1' then
    f <= '1';
end if;
```

Listing A.11: *Code voor een EXOR-poort.*

Bij onderstaande linker oplossing is gebruik gemaakt van het feit dat in VHDL ook relationele tests kunnen worden beschreven. Hier wordt gekeken of de waarde van a *ongelijk* is aan de waarde van b. In de rechter oplossing is de functie in feite als een waarheidstabel

beschreven.

```
if a /= b then     -- not equal
    f <= '1';
else
    f <= '0';
end if;
```

Listing A.12: *Code voor een EXOR-poort.*

```
ab_vec <= a & b;
case ab_vec is
    when "00" => f <= '0';
    when "01" => f <= '1';
    when "10" => f <= '1';
    when "11" => f <= '0';
    when others => f <= 'X';
end case;
```

Listing A.13: *Code voor een EXOR-poort.*

Uitwerking opgave 7.8.

Hieronder de complete VHDL-beschrijving van de negative edge triggered D-flipflop.

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

entity dffneg is
port (clk    : in std_logic;
      d      : in std_logic;
      q      : out std_logic
     );
end entity dffneg;

architecture behavioral of dffneg is
begin

    process (clk) is
    begin
        if falling_edge(clk) then
            q <= d;
        end if;
    end process;

end architecture behavioral;
```

Listing A.14: *Code voor een negative edge triggered D-flipflop.*

Uitwerking opgave 7.9.

Een schuifregister bestaat uit een groep D-flipflops waarvan de uitgang van een flipflop gekoppeld is aan de ingang van een volgende flipflop. Afhankelijk van de exacte aansluitingen (en het naar buiten brengen van de flipflopinhouden) kan je rechtsom schuiven of linksom schuiven. Er moet gebruik gemaakt worden van flipflops, dat zijn flankgevoelige geheugenelementen. Er is dus een flankbeschrijving nodig in de VHDL-code. Zie onderstaande listing.

```
process (clk) is
begin
    if rising_edge(clk) then
        shiftreg <= shiftreg(6 downto 0) & '0';
    end if;
end process;
```

Listing A.15: *Code voor het naar links schuiven van een 8-bits register.*

Uitwerking opgave 7.10.

Hieronder staan twee listings voor het probleem. Bij de listing links wordt niet de hele array (vector) afgelopen. Variabele `p` wordt eerst gelijk gemaakt aan de waarde van het meest significante element. Daarna moeten de overige elementen verwerkt worden. Dit scheelt één lusdoorgang.

De listing rechts maakt gebruik van de definitie van de OR-poort: de uitgang wordt '1' als één of meer ingangen '1' zijn. Als een array-element '1' blijkt te zijn, is de uitgang ook '1'. Zijn geen van de array-elementen '1' dan wordt variabele `p` nooit op '1' gezet. Na het doorlopen van de lus is `p` dan nog steeds '0'. Merk op dat er geen **`else`** gebruikt mag worden.

```
p := x(7);
for i in 6 downto 0 loop
    p := p or x(i);
end loop;
q <= not p;
```

Listing A.16: *Code 8-input NOR-poort.*

```
p := '0';
for i in 7 downto 0 loop
    if x(i) = '1' then
        p := '1';
    end if;
end loop;
q <= not p;
```

Listing A.17: *Code 8-input NOR-poort.*

Uitwerking opgave 7.11.

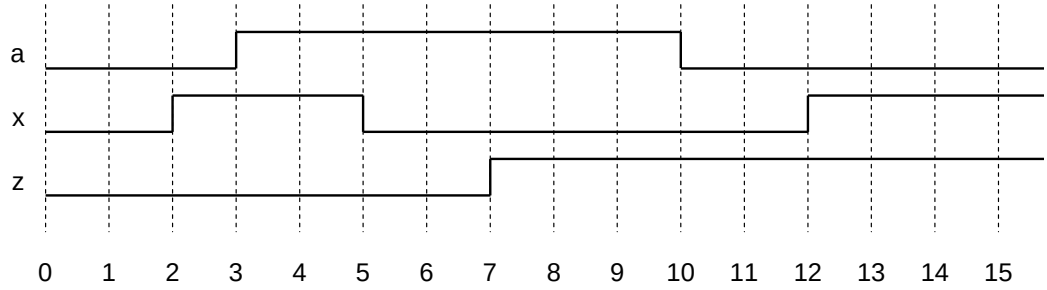
Zie uitwerking van opgave 7.12

Uitwerking opgave 7.12.

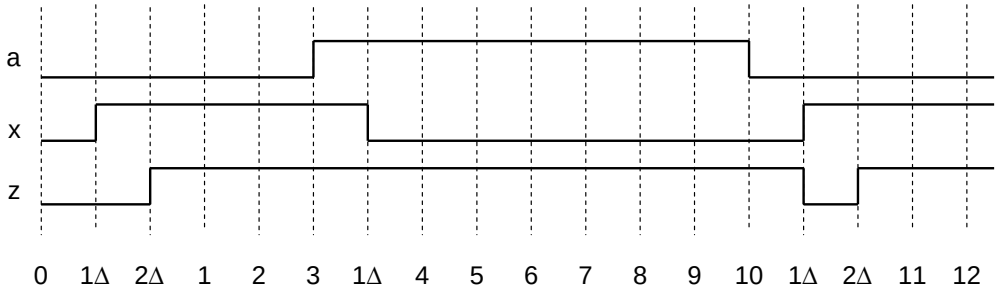
Hieronder is het tijddiagram te zien. Let erop dat de inverter signaalpulsen korter dan 2 ns niet op de uitgang laat zien. Voor de OR-poort is dat 5 ns. Dat laatste kan goed gezien worden in onderstaand tijddiagram. Op tijdstip 10 ns wordt ingang a logisch '0' en 2 ns later (tijdstip 12 ns) wordt signaal x logisch '1'. Dat levert een puls van 2 ns (de puls is laag) die de OR-poort niet doorgeeft.

Uitwerking opgave 7.13.

Het tijddiagram heeft zowel 'echte' tijden als delta-delays. Er zijn geen vertragingen opgegeven, dus worden alle toekenningen één delta later actief. Alle pulsen worden doorgegeven, want de minimale pulsbreedte is één delta.



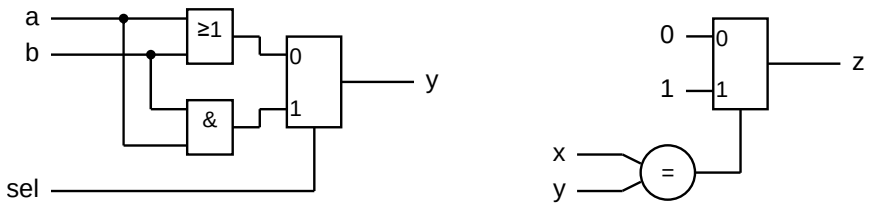
Figuur A.3: *Timingdiagram.*



Figuur A.4: *Timingdiagram.*

Uitwerking opgave 7.14.

In beide stukken code wordt een **`if ... then ... else`** gebruikt zodat er multiplexers worden gesynthetiseerd. In de code links is signaal `sig` één bit breed, dus het signaal kan direct dienen om de multiplexer aan te sturen. Afhankelijk van de waarde van `sig` moet een OR-functie of een AND-functie gerealiseerd worden. In de code rechts wordt de relationele operator = gebruikt. Dat houdt in dat er een vergelijkschakeling moet worden gesynthetiseerd. Verder is te zien dat als `x` gelijk is aan `y` de uitgang logisch '1' wordt en anders logisch '0'. De constanten '0' en '1' worden op de data- ingangen van de multiplexer aangesloten. In feite hoeft de multiplexer niet gesynthetiseerd te worden, de vergelijkschakeling geeft al een logische '1' als `x` gelijk is aan `y`. De signalen `x` en `y` kunnen vectoren zijn (beide met hetzelfde aantal elementen) of kunnen beide uit één bit bestaan. In dat laatste geval kan de gehele rechter schakeling vereenvoudigd worden tot een EXNOR-poort.

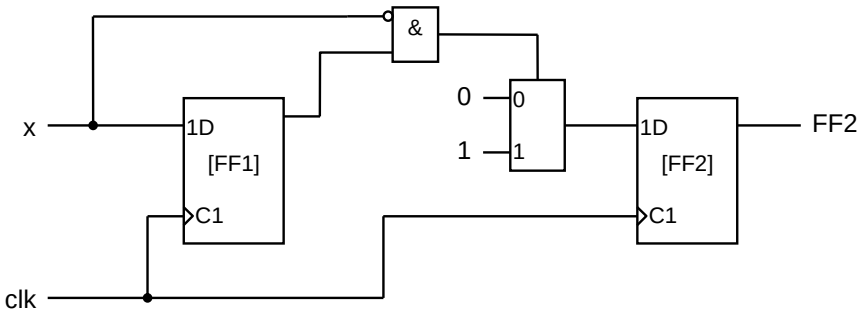


Figuur A.5: *Gerealiseerde schema's voor VHDL-code.*

Uitwerking opgave 7.15.

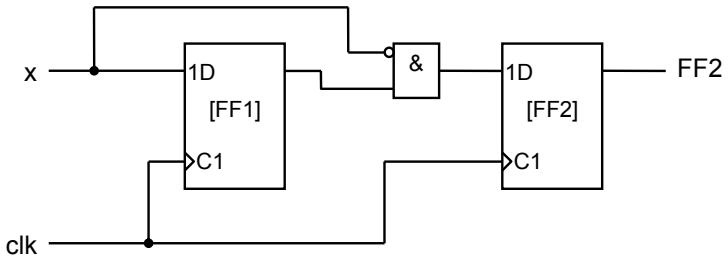
De schakeling bestaat uit twee D-flipflops. Er worden namelijk toekenningen gedaan aan twee signals onder besturing van een klokflank. Er wordt een **`if...then...else`**

gebruikt, dat levert een multiplexer op. In de test van de **`if`** wordt een AND-constructie gebruikt (samen met een inverter), dat worden dan logische poorten. Zie onderstaande schakeling.



Figuur A.6: *Gerealiseerd schema voor VHDL-code.*

De constanten '0' en '1' worden op de data-ingangen van de multiplexer aangesloten. In feite hoeft de multiplexer niet gesynthetiseerd te worden, de schakeling voor de test geeft al een logische '1' als deze waar is. Zie onderstaande schakeling.



Figuur A.7: *Geoptimaliseerd schema voor VHDL-code.*

Uitwerking opgave 7.16.

In de figuur zijn twee flipflops te zien. Dat houdt in dat de toekenningen onder flankbesturing moet worden geplaatst. De flipflops hebben een asynchrone reset, actief laag, dat kan met een **`if`**-statement worden beschreven. De multiplexer kan worden beschreven met een **`if ... then .. else ... end if`**. Zie onderstaande code. Een en ander kan wat compacter beschreven worden. Zo kan intern signaal `ff2` vermeden worden en vervangen worden door signaal `q` zelf.

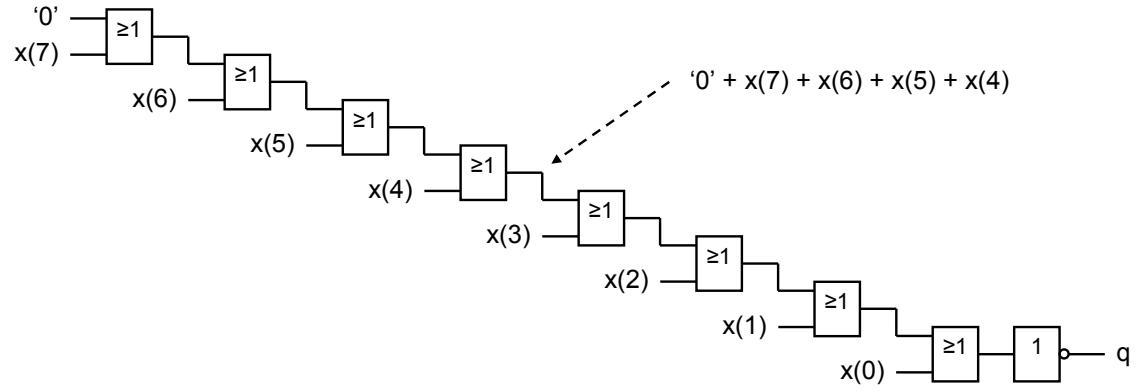
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity opgave33 is
5      port (clk : in std_logic;
6            areset : in std_logic;
7            sel : in std_logic;
8            a : in std_logic;
9            q : out std_logic);
10 end entity opgave33;
```

```
11
12 architecture behav of opgave33 is
13 signal ff1, ff2 : std_logic;
14 begin
15
16     process (clk, areset) is
17     begin
18         if areset = '0' then
19             ff1 <= '0';
20             ff2 <= '0';
21         elsif rising_edge(clk) then
22             ff1 <= a;
23             if sel = '1' then
24                 ff2 <= ff1;
25             else
26                 ff2 <= a;
27             end if;
28         end if;
29     end process;
30
31     q <= ff2;
32
33 end architecture behav;
```

Listing A.18: *VHDL-code voor de schakeling.*

Uitwerking opgave 7.17.

In de code wordt de variabele `p` eerst op '0' gezet. De eerste keer dat de **`for`**-lus wordt doorlopen, wordt de (huidige) waarde van `p` (en die is '0') ge-OR-d met `x(7)` met als resultaat dat `p` nu gelijk aan `x(7)`. In de volgende iteratie wordt `p` ge-OR-d met `x(6)` . Dat gaat zo door tot de lus is afgelopen, dus als laatste wordt `p` ge-OR-d met `x(0)`. We noemen dit ook wel *loop unrolling*. Daarna wordt de inverse van `p` toegekend aan `q`. In onderstaande figuur is het schema te zien. Merk op dat een slimme compiler na uitwerking van de **`for`**-lus en de NOT direct een 8-input NOR-poort genereert.



Figuur A.8: *Schema voor de 8-input NOR-poort op basis van loop unrolling.*

Uitwerking opgave 8.1.

Een mogelijke oplossing staat hieronder. Hierin is `data_in` een 4-bit ingang waarop de

te laden stand staat. Door signaal `load` logisch '1' te maken worden de data-ingangen geladen in de interne stand.

```
process (clk) is
variable count : unsigned(3 downto 0);
begin
    if rising_edge(clk) then
        if load = '1' then
            count := data_in;
        elsif enable = '1' then
            count := count + 1;
        end if;
    end if;
    q <= count;
end process;
```

Listing A.19: *VHDL-code voor een laadbare teller.*

Uitwerking opgave 8.2.

Een JK-flipflop met beide ingangen (J en K) aan elkaar aangesloten, gedraagt zich als een T-flipflop. Zie ook http://nl.wikipedia.org/wiki/Flipflop en bekijk de tabellen van de JK-flipflop en de T-flipflop.

Uitwerking opgave 8.3.

Van een zuivere binaire omhoog-teller waarbij alle telstanden worden doorlopen (0 t/m $2^n - 1$ met n telbits) kan een omlaag-teller worden gemaakt door alle uitgangen van de telbits te inverteren. Dat kan niet als niet de volledige cyclus wordt doorlopen. Zie onderstaande 3-bit teller. De linker kolom laat de opeenvolgende telstanden zien van een omhoog-teller, de rechter kolom van een omlaag-teller.

Tabel A.3: *Telvolgorde van een omhoog- en omlaagteller.*

up	down
000	111
001	110
010	101
011	100
100	011
101	010
110	001
111	000

Uitwerking opgave 8.4.

Eerst even de T-flipflop bespreken. Een T-flipflop is een flankgevoelig geheugenelement dat van stand verandert (een 0 wordt een 1 en een 1 wordt een 0) als de stuuringang (T) logisch '1' is. De stand blijft ongewijzigd als de stuuringang logisch '0' is.

De 6-teller begint bij 000_2 en dan naar 001_2, 010_2, etc. Na stand 101_2 volgt weer stand 000_2 want de teller is aan het einde van zijn cyclus. In de onderstaande tabel zijn de doorlopen telstanden onder elkaar gezet.

Tabel A.4: *Telcyclus van de 6-teller.*

telstand	opmerking
000	begin nieuwe cyclus
001	
010	Q_1 toggelt
011	
100	Q_2, Q_1 toggelt
101	
000	Q_2 toggelt, begin nieuwe cyclus

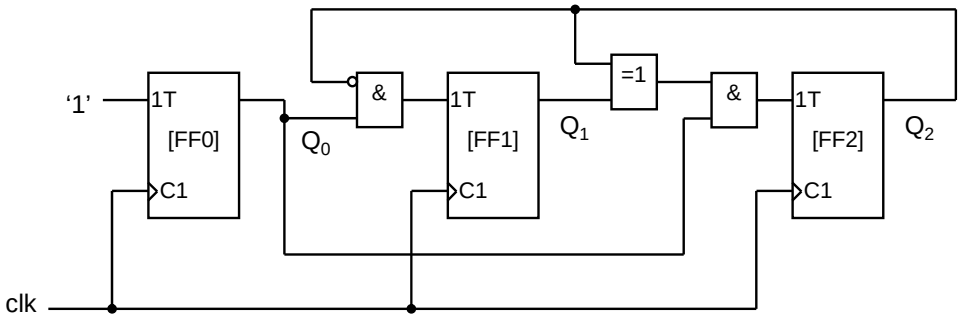
Wat direct opvalt is dat het minst significante telbit (Q_0) bij elke volgende stap wisselt. Dat houdt in dat de bijbehorende flipflop continue moet toggelen. Dat kan eenvoudig gedaan worden door de T-ingang van de flipflop op logisch '1' te zetten. Verder is te zien dat dat het middelste telbit (Q_1) moet toggelen bij stand 001_2 en 011_2 is. De meest significante telbit (Q_2) moet toggelen bij stand 011_2 en 101_2.

De standen waarbij de flipflops moeten toggelen moeten uitgecodeerd worden. Er worden schakelfuncties gerealiseerd voor de T-ingangen van de flipflops. Een logische '1' doet een T-flipflop toggelen. Dus wanneer een flipflop moet toggelen moet de bijbehorende schakelfunctie een logische '1' geven.

Na enig rekenwerk worden de volgende functies gevonden:

$$\begin{aligned} T_0 &= 1 \\ T_1 &= \overline{Q_2}\cdot\overline{Q_1}\cdot Q_0 + \overline{Q_2}\cdot Q_1\cdot Q_0 = \overline{Q_2}\cdot Q_0\cdot(\overline{Q_1}+Q_1) = \overline{Q_2}\cdot Q_0 \\ T_2 &= \overline{Q_2}\cdot Q_1\cdot Q_0 + Q_2\cdot\overline{Q_1}\cdot Q_0 = Q_0\cdot(Q_2\oplus Q_1) \end{aligned} \tag{A.4}$$

De volledige teller is in onderstaande figuur weergegeven.

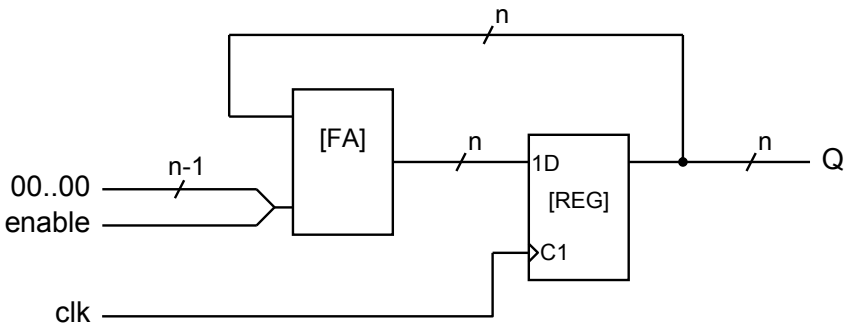


Figuur A.9: *Schema van de 6-teller op basis van T-flipflops.*

Uitwerking opgave 8.5.

Een teller is ook op te bouwen met een full adder en een register. Het register onthoudt

dan de laatst ingebrachte telstand. Het vermeerderen van de telstand wordt gedaan door de full adder. Op één set ingangen wordt de huidige telstand aangeboden, op de andere set ingangen wordt het binaire getal $00\ldots01_2$ aangeboden. Na een actieve klokflank wordt de nieuwe telstand ingeklokt. Uiteraard werkt de teller modulo 2^n waarbij n het aantal telbits is. Door het getal $00\ldots01_2$ aan te passen kunnen andere telvolgorden realiseren, bijvoorbeeld $00\ldots11_2$ om de teller steeds met drie te vermeerderen.



Figuur A.10: *Realisatiw van een enable bij een teller op basis van een opteller.*

Uitwerking opgave 8.6.

Hieronder de (ingekorte) listing van de eerder beschreven BCD-teller.

```
1  begin
2      process (clk, areset) is
3      begin
4          if rising_edge(clk) then
5              if count = "1001" then -- teller in hoogste stand?
6                  count <= "0000"; -- ... dan teller weer 0
7              else
8                  count <= count + 1; -- anders gewoon tellen
9              end if;
10         end if;
11     end process;
12     tc <= '1' when count = "1001" and en = '1' else '0';
13     q <= count;
14 end architecture rtl;
```

Listing A.20: *Verkorte code van de BCD-teller.*

In de beschrijving wordt de gelijkheid van `count` met 9_{10} getest. Dan wordt `count` gelijk aan 0_{10}. Bij ongelijkheid, dus ook als de telstand 12_{10} is, wordt `count` verhoogd. De teller zal doorlopen tot 15_{10} (het is immers een 4-bit teller) en dan naar 0_{10} "omklappen". De uitgang `tc` wordt dan *niet* logisch '1', dat gebeurt alleen als de telstand 9_{10} is (en `en` is '1').

Uitwerking opgave 8.7.

In feite staat de teller in een niet gebruikte telstand en moet daar zo snel mogelijk uit komen. De teller moet dan geladen worden met 0_{10}. Dat kan door de = te vervangen door >= bij beide tests:

```
1 if count >= "1001" then ...
```

Listing A.21: *Test op groter dan of gelijk aan 9.*

```
1 tc <= '1' when count >= "1001" and en = '1' else '0';
```

Listing A.22: *Aangepaste code voor de terminal count.*

Uitwerking opgave 8.8.

De test op het hoogste getal moet aangepast worden, net als de de functie voor de `tc`. Hier wordt ook gebruik gemaakt van de `>=` operator.

```
1 if count >= "1100" then ...
```

Listing A.23: *Test op groter dan of gelijk aan 12.*

```
1 tc <= '1' when count >= "1100" and en = '1' else '0';
```

Listing A.24: *Aangepaste code voor de terminal count.*

Uitwerking opgave 8.9.

Natuurlijk kunnen de twee benodigde bits getest worden via onderstaande code:

```
1 if count(3) = '1' and count(0) = '1' then ...
```

Listing A.25: *Test voor het patroon 1– –1.*

Maar de bibliotheek `numeric_std` heeft ook een functie `stdmatch` aan boord die twee vectoren vergelijkt met don't cares:

```
1 if stdmatch(count, "1--1") then ...
```

Listing A.26: *Test voor het patroon 1– –1 met stdmatch.*

Uitwerking opgave 8.10.

Met een n-bit teller kunnen 2^n stappen worden gemaakt. De resolutie is de reciproke daarvan. Duty cycles worden altijd in procenten gegeven (maar er wordt niet met procenten gerekend), dus de resolutie bij n bits is

$$DC = \frac{1}{2^n} \cdot 100\% \tag{A.5}$$

Tabel A.5: *Bitbreedte vs. resolutie.*

bits	resolutie (%)
4	6,25
8	0,39
10	0,098
16	0,0015

In onderstaande tabel zijn de resoluties van enkele bitbreedtes gegeven.

Uitwerking opgave 8.11.

Een groter-of-gelijk-schakeling lost het probleem bij de maximale waarde op. Als de waarde van de teller gelijk is aan de referentiewaarde, zal de vergelijker een logische '1' geven. Het probleem is nu dat het PWM-signaal niet meer volledig "uit" kan. Als de referentiewaarde 0 is, zal de schakeling namelijk een logische '1' geven als de teller ook op 0 staat. De Duty Cycle is dan minimaal $1/2^n \times 100\%$.

Uitwerking opgave 8.12.

Dit kan heel eenvoudig door de teller die gebruikt wordt voor generatie van het PWM-signaal niet helemaal een volledige 2^n-telcyclus te laten doorlopen, maar bijvoorbeeld een $2^n - 1$-telcyclus. De cyclus loopt van 0 t/m $2^n - 2$ en de gebruiker kan wél $2^n - 1$ als referentiewaarde opgeven. Neem bijvoorbeeld een 8-bit PWM-teller. De teller loopt van 0 t/m 254 en begint dan weer opnieuw. De gebruiker kan wel 255 instellen als referentiewaarde.

Uitwerking opgave 8.13.

De toekenning aan `tc` gebeurt onder klokflanksturing. Dat betekent dat de toekenning pas één klokflank later wordt uitgevoerd, er wordt immers een flipflop voor `tc` gesynthetiseerd. In de praktijk is `tc` dus logisch 1 op telstand 0000_{BCD}. Voor correcte werking moet voor `tc` een combinatorische functie worden gerealiseerd.

Uitwerking opgave 9.1.

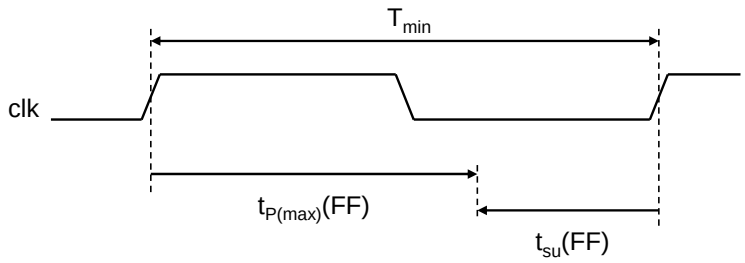
Voor het bepalen van de maximale frequentie moet gekeken worden welk pad de data volgt van flipflop naar flipflop. Er is er maar één, dus dat is makkelijk. Vanuit de uitgang van de flipflop komt de data aan bij de ingang van de flipflop. Het pad is dus FF1 →FF1. Er is geen sprake van klokskew. Dat was ook niet interessant geweest, want de data komt bij dezelfde flipflop aan. De minimale periodetijd van dit systeem wordt opgebouwd uit de vertraging van de data van de flipflop en de setuptijd van de flipflop. Zie figuur A.11.

De minimale periodetijd wordt berekend.

$$T_{min} = t_{P(max)}(\text{FF}) + t_{su}(\text{FF}) = 35 + 15 = 50 \text{ ns} \tag{A.6}$$

De maximale frequentie is 20 MHz.

De holdtijd is kleiner dan de minimale propagatietijd van de flipflop, dat houdt in dat er



Figuur A.11: *Timing maximale frequentie van de tweedeler.*

geen *hold time violation* is:

$$t_{slack,h} = t_{P(min)}(\text{FF}) - t_h(\text{FF}) = 10 - 5 = 5 \text{ ns} \tag{A.7}$$

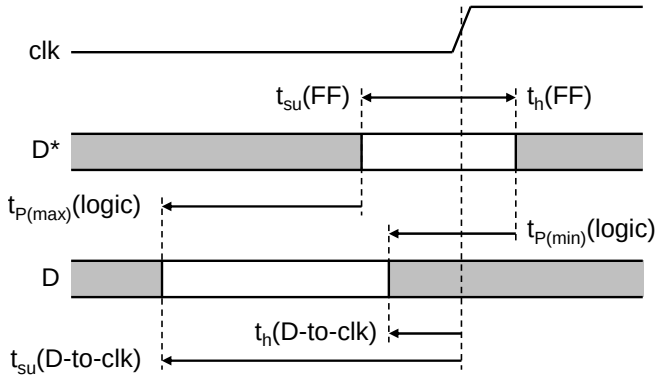
Er is betrouwbare dataoverdracht mogelijk.

Uitwerking opgave 9.2.

De flipflop heeft een setup- en holdtijd ten opzichte van de actieve flank van de klok (de opgaande flank in dit geval). In dat gebied moet de D-ingang van de flipflop (`D*`) stabiel zijn. Voor de flipflop zit nog wat logica en die heeft een bepaalde vertragingstijd. Het zal dan duidelijk zijn dat de setuptijd van de D-ingang (de D-ingang bij de logica, niet de flipflop) t.o.v. van de opgaande flank van de klok "naar voren" wordt verschoven. In dit geval moet de maximale propagatietijd genomen worden. Dat is ook te verklaren: als de data op de D-ingang stabiel wordt gehouden, dan duurt het op zijn langst nog de maximale vertragingstijd voordat de stabiele data bij `D*` aankomt. Precies op dat moment gaat de setuptijd van `D*` in.

Voor de holdtijd geldt eenzelfde verhaal, maar nu moet de minimale vertragingstijd genomen worden. Ook dat is te verklaren. Als de stabiele data weer gaat veranderen, dan zal op zijn vroegst de data bij `D*` na de minimale vertragingstijd veranderen. Precies op dat moment is de holdtijd bij `D*` verstreken.

Eén en ander is in figuur A.12 getekend.



Figuur A.12: *Setup- en holdtijd van de D-flipflop met enable.*

Voor de setup- en holdtijd t.o.v. de klok zijn twee formules op te stellen.

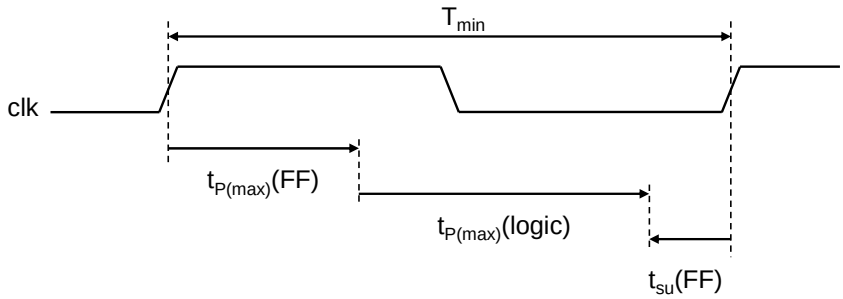
$$\begin{aligned} t_{su}(\text{D-to-clk}) &= t_{su}(\text{FF}) + t_{P(max)}(\text{logic}) \\ t_h(\text{D-to-clk}) &= t_h(\text{FF}) - t_{P(min)}(\text{logic}) \end{aligned} \tag{A.8}$$

Merk op dat in de formule van de holdtijd een minteken staat. Dat komt omdat de holdtijd naar rechts wordt getekend (positieve waarde). Invullen en uitrekenen levert op:

$$\begin{aligned} t_{su}(\text{D-to-clk}) &= 14 + 19 = 33 \text{ ns} \\ t_h(\text{D-to-clk}) &= 4 - 8 = -4 \text{ ns} \end{aligned} \tag{A.9}$$

Merk op dat de holdtijd *negatief* is. Dat houdt in dat de data op D nog voor de klokflank mag veranderen. Zie ook de bovenstaande figuur.

Voor het bepalen van de maximale frequentie moet het pad worden bepaald dat de data aflegt van uitgang naar ingang. Het pad is FF → logica → FF. Dat duurt minimaal één klokperiode. In figuur A.13 is dat goed te zien.



Figuur A.13: *Bepaling minimale periodeduur van de D-flipflop met enable.*

De bijbehorende formules:

$$\begin{aligned} T_{min} &= t_{P(max)}(\text{FF}) + t_{P(max)}(\text{logic}) + t_{su}(\text{FF}) \\ &= 17 + 19 + 14 = 50 \text{ ns} \end{aligned} \tag{A.10}$$

De maximale frequentie bedraagt dan 20 MHz.

Uitwerking opgave 9.3.

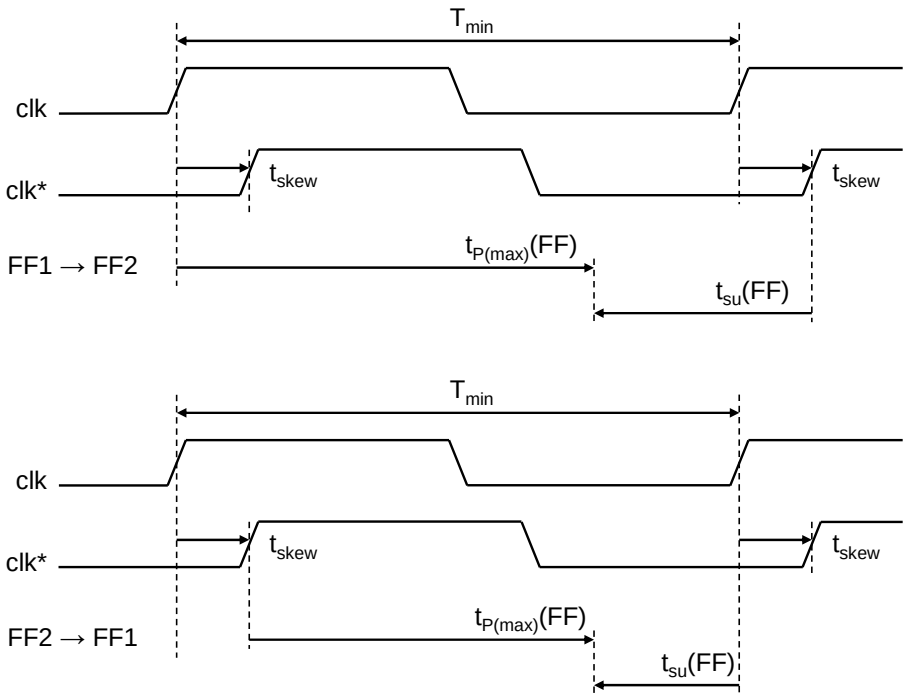
Er zijn hier twee paden te ontdekken waarlangs data wordt getransporteerd: FF1 → FF2 en FF2 → FF1. Van beide moet de timing bepaald worden. In figuur A.14 zijn die weergegeven.

Beide paden leveren een T_{min}. De grootste waarde van T_{min} levert de minimale periodetijd waarop het gehele systeem nog precies betrouwbaar werkt (als er geen hold violation is).

$$\begin{aligned} \text{FF1} \rightarrow \text{FF2}: &\quad T_{min} + t_{skew} = t_{P(max)}(\text{FF}) + t_{su}(\text{FF}) \\ \text{FF2} \rightarrow \text{FF1}: &\quad T_{min} - t_{skew} = t_{P(max)}(\text{FF}) + t_{su}(\text{FF}) \end{aligned} \tag{A.11}$$

Na herschikking van de term t_{skew} in beide formules wordt duidelijk dat de formule voor het pad FF2 → FF1 de grootste T_{min} levert (natuurlijk geldt dat $t_{skew} > 0$).

De logische werking (gedrag) is als volgt te bepalen. De beide flipflops hebben de beginstand $Q_2Q_1 = 00$. Dat betekent dat na de klokflank de inhoud van flipflop 1 logisch 1



Figuur A.14: *Timing van twee rondgekoppelde D-flipflops.*

wordt, immers wordt de inverse stand van Q_2 ingeklokt. De inhoud van flipflop 2 wordt/-blijft logisch '0'. De inhoud van de flipflops is $Q_2Q_1 = 01$. Na een volgende klokflank is de inhoud van de flipflops gewijzigd in $Q_2Q_1 = 11$. Daarna wordt de inhoud (ook wel stand genoemd) gewijzigd in $Q_2Q_1 = 10$. Hierna volgt dan nog $Q_2Q_1 = 00$. Dat was ook de beginstand van de flipflops. De flipflops doorlopen de cyclus $00 \rightarrow 01 \rightarrow 11 \rightarrow 10$. Dat is de bekende Gray-codering. Dit is dan ook een Gray-teller[1].

Uitwerking opgave 9.4.

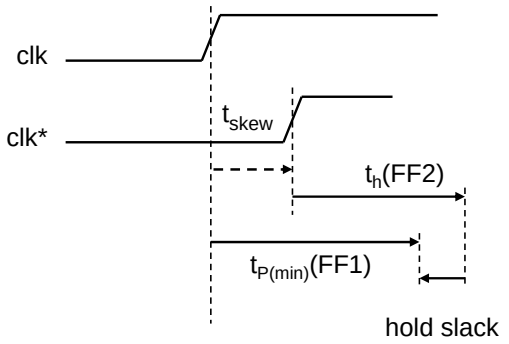
Er zijn twee voorwaarden waaraan betrouwbare dataoverdracht moet voldoen:

$$\begin{aligned} \text{setup slack groter dan/gelijk aan 0}: &\quad t_{slack,su} \geq 0 \\ \text{hold slack groter dan/gelijk aan 0}: &\quad t_{slack,h} \geq 0 \end{aligned} \tag{A.12}$$

De setup slack komt ter sprake bij de berekening van de minimale periodetijd. Daar is niets over gezegd (d.w.z. dat de minimale periodetijd prima kan worden berekend). Er is dan ook geen *setup time violation*.

Voor het berekenen van de hold slack moet gekeken worden op welk kloksignaal de ontvangende flipflop zijn data inklokt. In dit geval is dat `clk*`. Deze klok is vertraagd met 2 ns. De holdtijd van de tweede flipflop wordt dus "naar achteren" geduwd. De eerste flipflop, die geen last heeft van skew, haalt zijn data op zijn vroegst na de minimale propagatietijd weg op de actieve flank van `clk`. De tijden kunnen worden getekend in een figuur, zie figuur A.15.

[1] De oplettende lezer ziet in deze code ook de telcode van een Johnson counter.



Figuur A.15: *Hold violation bij data-overdracht tussen twee flipflops.*

Uit de figuur kan de volgende formule worden opgemaakt:

$$\begin{aligned} t_{slack,h} &= t_{P(min)}(\text{FF}) - t_{skew} - t_h(\text{FF2}) \\ &= 7 - 2 - 6 = -1 \text{ ns} \end{aligned} \tag{A.13}$$

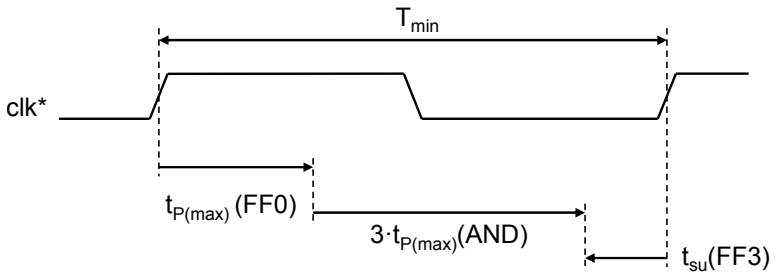
De hold slack is negatief en dat betekent dat er een *hold time violation* is. Er is geen betrouwbare dataoverdracht mogelijk.

Uitwerking opgave 9.5.

(Opgave a) In de schakeling zijn de volgende zes paden te ontdekken:

FF0 → 1x AND → FF1
FF0 → 2x AND → FF2
FF0 → 3x AND → FF3
FF1 → 1x AND → FF2
FF1 → 1x AND → FF3
FF2 → 1x AND → FF3

(Opgave b) De flipflops hebben onderling geen last van klokskew, er kan uitgegaan worden van kloksignaal `clk*`. Het langste pad in tijd is onmiskenbaar FF0 → 3x AND → FF3. Hiervan is een timingdiagram te maken, zie figuur A.16.

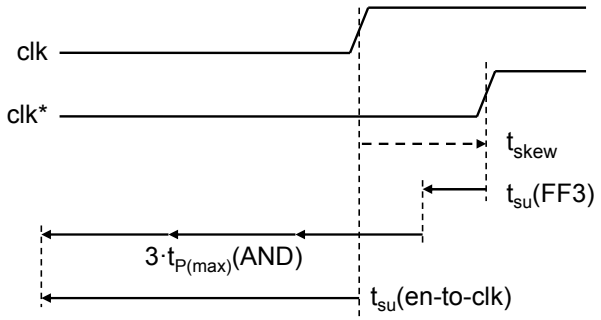


Figuur A.16: *Timingdiagram voor berekening van de minimale periodeduur.*

(Opgave c) De bijbehorende formule voor de berekening van de minimale periodetijd is

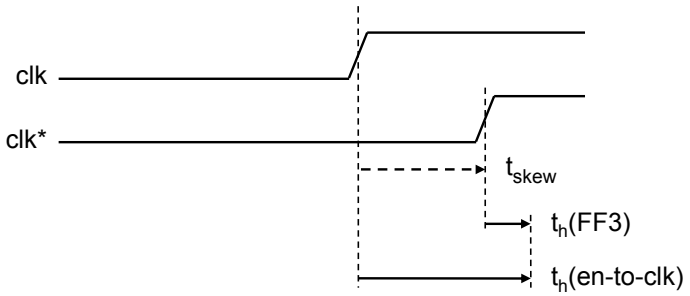
$$\begin{aligned} T_{min} &= t_{P(max)}(\text{FF0}) + 3 \cdot t_{P(max)}(\text{AND}) + t_{su}(\text{FF3}) \\ &= 7 + 3 \cdot 5 + 2 \\ &= 24 \text{ ns} \end{aligned} \tag{A.14}$$

(Opgave d) Om de setuptijd van signaal `enable` t.o.v. van de data-ingang van een willekeurige flipflop te berekenen, moet eerst gekeken worden wat het langste pad (in tijd) is van `enable` tot aan een flipflop. Dat is onmiskenbaar de data-ingang van FF3. Het is logisch dat hier het langste pad (in tijd) gekozen moet worden want een verandering op `enable` komt op z'n laatst bij FF3 aan. Het signaal op de data-ingang moet één setuptijd voor de klokflank stabiel zijn. Het betreft hier natuurlijk het kloksignaal `clk*`. De AND-poorten leveren na een maximale vertragingstijd zeker een definitieve waarde. T.o.v. van `clk*` moet signaal `en` dus ééns setuptijd en drie vertragingen van de AND-poorten eerder stabiel zijn. Nu wordt wordt echter als referentie het signaal `clk` gebruikt en niet `clk*`. Het enable-signaal moet dus één skewtijd later stabiel zijn. Zie figuur A.17.



Figuur A.17: *Timingdiagram voor berekening van de setuptijd.*

Voor de berekening voor de holdtijd volgt iets vergelijkbaars. Nu moet echter het kortste pad (in tijd) gekozen worden. Dat is natuurlijk de data-ingang van FF0. Dat is te verklaren, want een verandering komt op z'n vroegst bij FF0 aan. De holdtijd van FF0 wordt gemeten vanaf signaal `clk*`. De holdtijd van signaal `en` is pas verstreken na één skewtijd en één holdtijd. Er is geen vertraging van signaal `en` tot aan de data-ingang van FF0, er zit immers geen logica tussen. Zie figuur A.18.



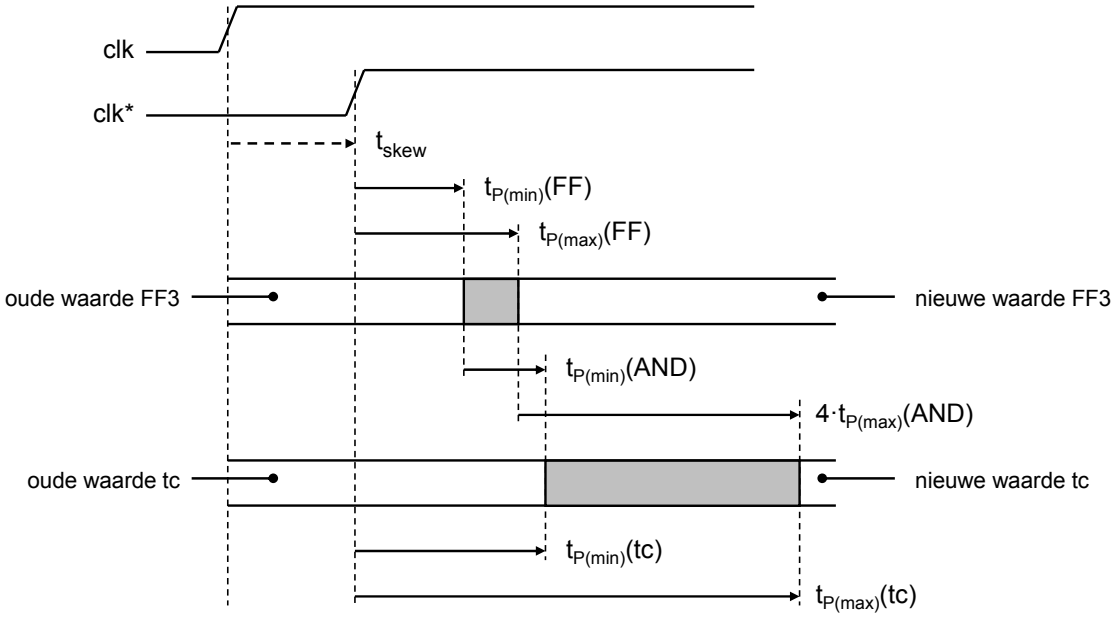
Figuur A.18: *Timingdiagram voor berekening van de holdtijd.*

(Opgave e) Uit het bovenstaande kunnen de setuptijd en de holdtijd berekend worden.

$$\begin{aligned} t_{su}(\text{en-to-clk}) &= t_{su}(\text{FF3}) + 3 \cdot t_{P(max)}(\text{AND}) - t_{skew} \\ &= 2 + 3 \cdot 5 - 4 \\ &= 13 \text{ ns} \end{aligned} \tag{A.15}$$

$$\begin{aligned} t_{h}(\text{en-to-clk}) &= t_{skew} + t_{h}(\text{FF3}) \\ &= 4 + 1 \\ &= 5 \text{ ns} \end{aligned} \tag{A.16}$$

(Opgave f) Na de actieve flank van `clk*` moet even gewacht worden tot de uitgangen van flipflops veranderden. Dat is op z'n vroegst na de minimale propagatietijd. Dan is er nog één poortvertraging nodig om vanaf FF3 bij uitgang `tc` te komen. Voor de maximale propagatietijd van `tc` ligt het anders. Het langste pad van een willekeurige flipflop naar uitgang `tc` begint bij de uitgang van FF0. Dan zijn er nog vier(!) poortvertragingen van de AND-poort nodig om bij uitgang `tc` aan te komen. Vergeet uiteraard de clock skew niet. Zie figuur A.19.



Figuur A.19: *Timingdiagram voor het berekenen van de minimale en maximale vertragingstijd.*

(Opgave g) Het bovenstaande kan in twee formules gezet worden. Daarmee zijn de tijden uit te rekenen.

$$t_{P(min)}(\text{clk-to-tc}) = t_{skew} + t_{P(min)}(\text{FF3}) + t_{P(min)}(\text{AND}) = 4+3+4 = 11 \text{ ns}$$
$$t_{P(max)}(\text{clk-to-tc}) = t_{skew} + t_{P(max)}(\text{FF3}) + 4 \cdot t_{P(max)}(\text{AND}) = 4+5+28 = 37 \text{ ns} \quad \text{(A.17)}$$

(Opgave h) Opmerkingen over de teller

De structuur van de teller is bijzonder elegant. Grotere tellers zijn eenvoudig te realiseren. Op het gebied van timing is dit ontwerp geen succes. De maximale frequentie neemt af met het aantal secties. De setuptijd van signaal `enable` en de minimale en maximale vertragingstijd van signaal `tc` neemt toe met het aantal secties.

Uitwerking opgave 9.6.

(Opgave a) De volgende paden van flipflop naar flipflop zijn in de schakeling te herkennen:

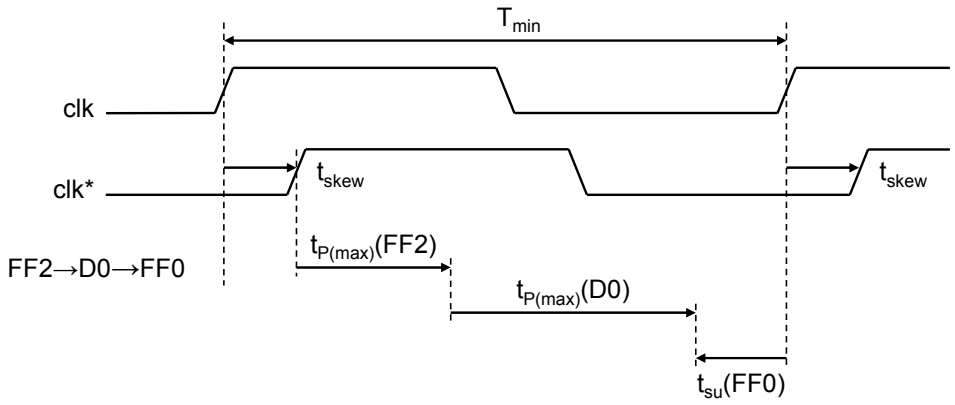
FF0 → D0 → FF0
FF0 → FF1

FF1 → FF2
FF2 → D0 → FF0

Merk op dat D1 in geen van de paden voorkomt want dat een logica voor een uitgang.

(Opgave b) In figuur A.20 is het timingdiagram getekend. Het langste pad in tijd is van FF2 via D0 naar FF0. De klokingang van FF2 heeft last van skew en die moet meegenomen worden in de berekening van de minimale periodetijd.

Noot: de andere drie paden leveren na uitwerking een kleinere minimale periodetijd dan het hier boven genoemde pad.



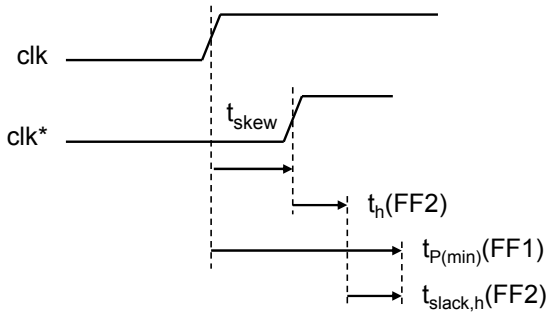
Figuur A.20: *Timingdiagram voor berekening minimale periodetijd.*

(Opgave c) Zie onderstaande berekening.

$$\begin{aligned} T_{min} - t_{skew} &= t_{P(max)}(\text{FF2}) + t_{P(max)}(\text{D0}) + t_{su}(\text{FF0}) \\ T_{min} &= t_{P(max)}(\text{FF2}) + t_{P(max)}(\text{D0}) + t_{su}(\text{FF0}) + t_{skew} \\ &= 8 + (6+5) + 2 + 3 \\ &= 24 \text{ ns} \end{aligned} \tag{A.18}$$

De klokskew heeft negatieve invloed op de minimale periodetijd.

(Opgave d) De *hold slack* is de tijd die over is na het verstrijken van de holdtijd van FF2 en verstrijken van de minimale propagatietijd van FF1. Let er op dat de holdtijd van FF2 naar "achter" wordt geduwd als gevolg van de skew. Zie figuur A.21.



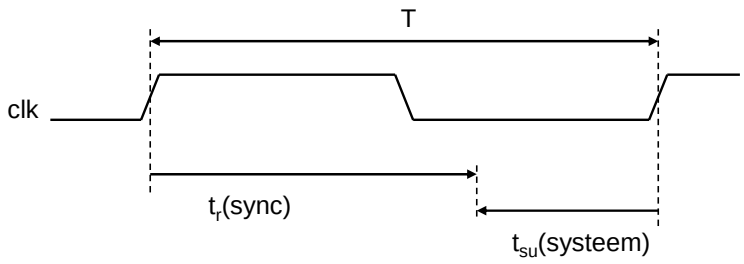
Figuur A.21: *Timingdiagram voor berekening hold slack.*

(Opgave e) De hold slack kan als volgt berekend worden:

$$\begin{aligned} t_{skew} + t_h(\text{FF2}) + t_{slack,h}(\text{FF2}) &= t_{P(min)}(\text{FF1}) \\ t_{slack,h}(\text{FF2}) &= t_{P(min)}(\text{FF1}) - t_{skew} - t_h(\text{FF2}) \\ &= 6 - 3 - 1 \\ &= 2 \text{ ns} \end{aligned} \tag{A.19}$$

De hold slack is positief, dat betekent dat de holdtijd eerder is verstreken dan de minimale propagatietijd en dat houdt in dat er betrouwbare data-overdracht mogelijk is (als de maximale frequentie niet overschreden wordt).

Uitwerking opgave 10.1.
In onderstaande figuur is het timingdiagram getekend.



Figuur A.22: *Timingdiagram van het synchronisatiesysteem.*

Uit deze figuur kan de volgende formule worden opgemaakt.

$$T = t_r(\text{sync}) + t_{su}(\text{systeem})$$

Nu kan de resolutietijd worden bepaald

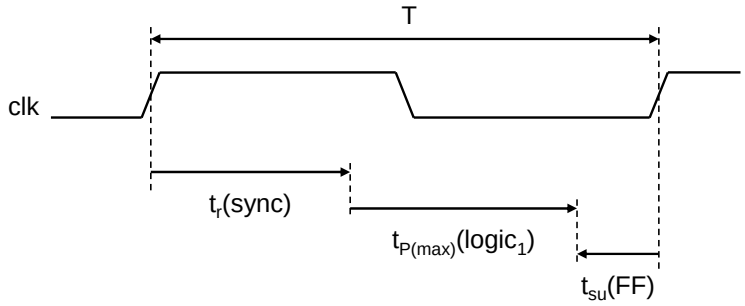
$$\begin{aligned} t_r(\text{sync}) &= T - t_{su}(\text{systeem}) \\ &= 50 - 20 \\ &= 30 \text{ ns} \end{aligned}$$

De synchronizer heeft dus 30 ns om uit z'n metastabiele toestand te raken. Invullen in formule voor berekenen van de MTBF geeft:

$$\begin{aligned} \text{MTBF}(t_r) &= \frac{e^{\frac{t_r}{\tau}}}{T_0 \cdot f_{sys} \cdot f_{data}} = \frac{e^{\frac{30}{0{,}135}}}{9{,}8 \cdot 10^6 \cdot 20 \cdot 10^6 \cdot 100 \cdot 10^3} = \frac{e^{222{,}222}}{19{,}6 \cdot 10^{18}} \\ &= 1{,}651 \cdot 10^{77} \end{aligned}$$

Merk op dat t_r en τ beide in ns gegeven zijn. De gemiddelde tijd tussen twee fouten is $1{,}651 \cdot 10^{77}$ seconden.

Uitwerking opgave 10.2.
In onderstaande figuur is het timingdiagram getekend.



Figuur A.23: *Timingdiagram van het synchronisatiesysteem.*

Uit deze figuur kan de volgende formule worden opgemaakt.

$$T = t_r(\text{sync}) + t_{P(max)}(\text{logic}_1) + t_{su}(\text{FF})$$

Nu kan de resolutietijd worden bepaald

$$\begin{aligned} t_r(\text{sync}) &= T - t_{P(max)}(\text{logic}_1) - t_{su}(\text{FF}) \\ &= 40 - 20 - 5 \\ &= 15 \text{ ns} \end{aligned}$$

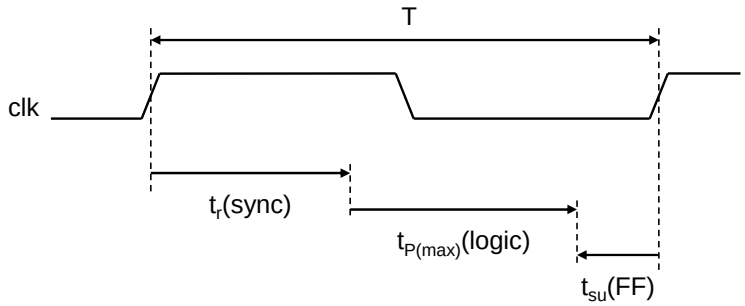
De synchronizer heeft dus 15 ns om uit z'n metastabiele toestand te raken. Invullen in formule voor berekenen van de MTBF geeft:

$$\begin{aligned} \text{MTBF}(t_r) &= \frac{e^{\frac{t_r}{\tau}}}{T_0 \cdot f_{sys} \cdot f_{data}} = \frac{e^{\frac{15}{0{,}135}}}{9{,}8 \cdot 10^6 \cdot 25 \cdot 10^6 \cdot 2 \cdot 10^3} = \frac{e^{111{,}111}}{490 \cdot 10^{15}} \\ &= 3{,}671 \cdot 10^{30} \end{aligned}$$

De gemiddelde tijd tussen twee fouten is $3{,}671 \cdot 10^{30}$ seconden. Dat is omgerekend $1{,}164 \cdot 10^{23}$ jaar.

Uitwerking opgave 10.3.

We moeten eerst de maximale resolution time van de synchronizer bepalen. We tekenen een timingdiagram waarin de relatie tussen de diverse tijden duidelijk wordt. Zie figuur A.24.



Figuur A.24: *Timingdiagram van het synchronisatiesysteem.*

Uit deze figuur kan de volgende formule worden opgemaakt.

$$T = t_r(sync) + t_{P(max)}(\text{logic}) + t_{su}(\text{FF})$$

Nu kan de resolutietijd worden bepaald

$$\begin{aligned} t_r(sync) &= T - t_{P(max)}(\text{logic}) - t_{su}(\text{FF}) \\ &= 25 - 10 - 5 \\ &= 10 \text{ ns} \end{aligned}$$

De synchronizer heeft dus 10 ns om uit z'n metastabiele toestand te raken. De MTBF is ook bekend, dat is 1 jaar. Dit moet worden omgerekend naar seconden, dus

$$\begin{aligned} \text{MTBF}(10 \text{ ns}) &= 1 \text{ jaar} = 1 \times 365 \times 24 \times 60 \times 60 = 31536000 \\ &= 3{,}1536 \cdot 10^7 \text{ s} \end{aligned}$$

De formule voor het berekenen van de MTBF moet aangepast worden zodat de frequentie van het asynchroon binnenkomend signaal berekend kan worden. Dat kan eenvoudig door MTBF en f_{data} uit te wisselen:

$$\begin{aligned} f_{data} &= \frac{e^{\frac{t_r}{\tau}}}{T_0 \cdot f_{sys} \cdot \text{MTBF}(t_r)} = \frac{e^{\frac{10}{0{,}15}}}{9{,}8 \cdot 10^6 \cdot 40 \cdot 10^6 \cdot 3{,}1536 \cdot 10^7} = \frac{e^{66{,}667}}{1236{,}2112 \cdot 10^{19}} \\ &= 7{,}258933 \cdot 10^6 \end{aligned}$$

De maximale toegestane frequentie van het asynchroon binnenkomend signaal is 7,259 MHz.