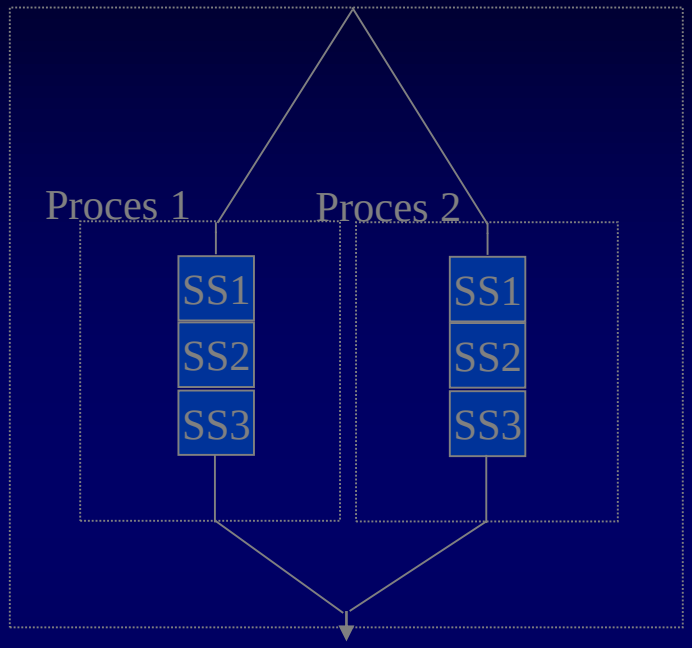
# VHDL

Beschrijving van digitale systemen

# VHDL

Het proces

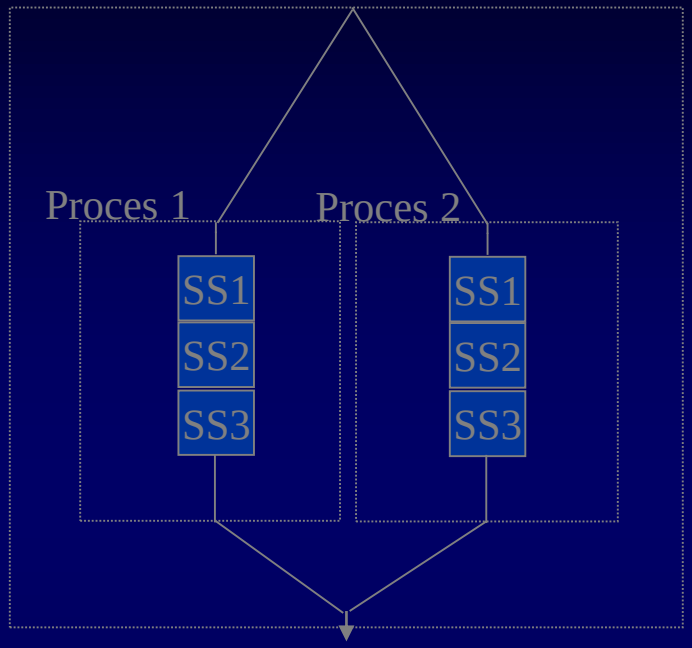
# Het proces




sequentiële uitdrukkingen worden hoofdzakelijk gebruikt in processen

```
PROCESS  (d0,  d1,  sel )
BEGIN
        IF sel = '0'
        THEN  out  <=  d0 ;
        ELSE   out  <=  d1 ;
END IF ;
END PROCESS ;
```

# Het proces

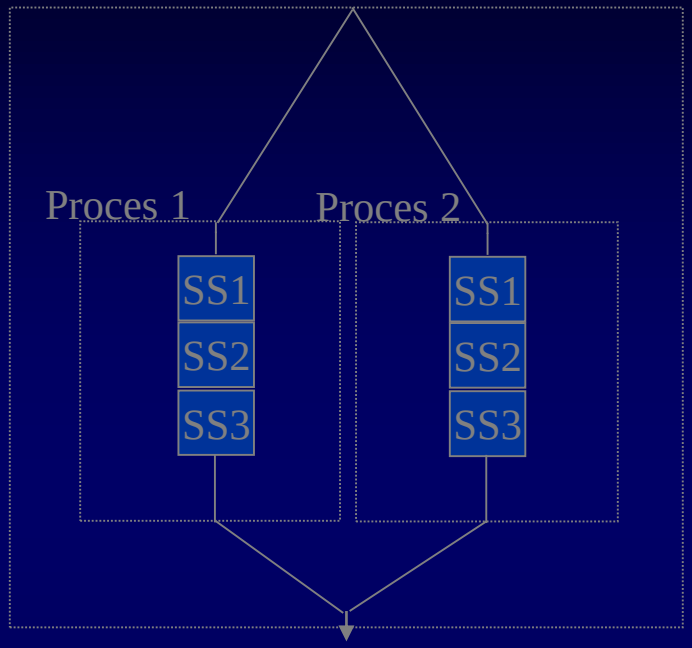



sequentiële uitdrukkingen worden hoofdzakelijk gebruikt in processen

een architectuur kan zoveel processen bevatten als noodzakelijk voor de beschrijving

```
PROCESS  (d0,  d1,  sel )
BEGIN
        IF sel = '0'
        THEN  out  <=  d0 ;
        ELSE   out  <=  d1 ;
END IF ;
END PROCESS ;
```

# Het proces



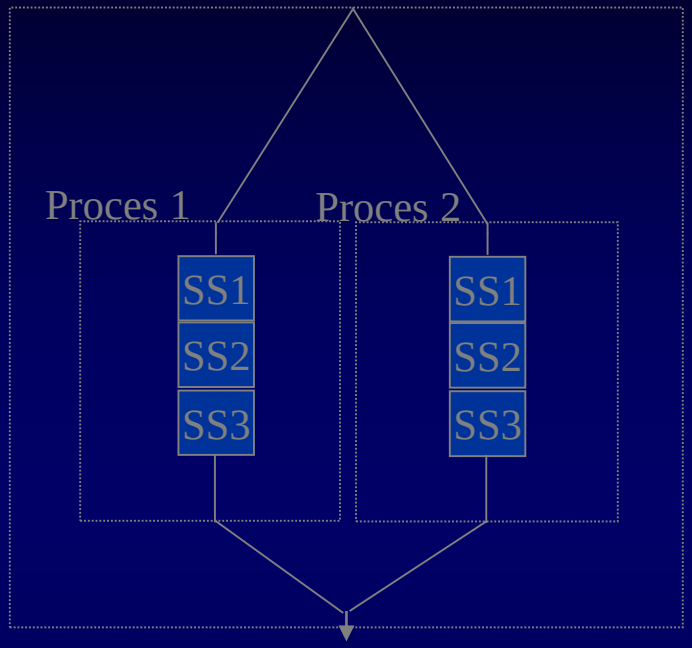

sequentiële uitdrukkingen worden hoofdzakelijk gebruikt in processen

een architectuur kan zoveel processen bevatten als noodzakelijk voor de beschrijving

een proces wordt uitgevoerd zoals een concurrente uitdrukking

```
PROCESS  (d0,  d1,  sel )
BEGIN
        IF sel = '0'
        THEN  out  <=  d0 ;
        ELSE   out  <=  d1 ;
END IF ;
END PROCESS ;
```

# Het proces




sequentiële uitdrukkingen worden hoofdzakelijk gebruikt in processen

een architectuur kan zoveel processen bevatten als noodzakelijk voor de beschrijving

een proces wordt uitgevoerd zoals een concurrente uitdrukking

alle processen worden concurrent uitgevoerd

```
PROCESS  (d0,  d1,  sel )
BEGIN
        IF sel = '0'
        THEN  out  <=  d0 ;
        ELSE   out  <=  d1 ;
END IF ;
END PROCESS ;
```

# Het proces

alg. structuur

de algemene structuur van een proces ziet eruit zoals hier getoond

```
procesnaam: process (sensitivity list)

        lokale definities en toewijzingen

BEGIN

        sequentiële uitdrukkingen

END PROCESS;
```

# Het proces

## alg. structuur

de algemene structuur van een proces ziet eruit zoals hier getoond

in een voorbeeld bekijken we de verschillende delen van de processtructuur

```
procesnaam: process (sensitivity list)

        lokale definities en toewijzingen

BEGIN

        sequentiële uitdrukkingen

END PROCESS;
```

# Het proces

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

voor de duidelijkheid, vooral in architecturen met verschillende processen, is het wenselijk een procesnaam te voorzien

# Het proces

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

de signaalveranderingen die het proces moeten starten vinden we terug in de sensitivity list (proces gevoeligheidslijst)

# Het proces

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

eventuele veranderlijken die enkel voor het proces zichtbaar moeten zijn worden hier gedefinieerd

# Het proces

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

tussen BEGIN en END PROCESS worden de sequentiële uitdrukkingen geplaatst die het gedrag van het te beschrijven systeem modelleren

# Het proces

## uitvoering

een proces wordt telkens slechts 1 maal uitgevoerd en daarna wordt de uitvoering ervan onderbroken

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

# Het proces

uitvoering

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

pas na de onderbreking van het proces zijn de signaaltoekenningen die in het proces gebeurd zijn beschikbaar voor andere processen of concurrente uitdrukkingen

# Het proces

uitvoering

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

pas na de onderbreking van het proces zijn de signaaltoekenningen die in het proces gebeurd zijn beschikbaar voor andere processen of concurrente uitdrukkingen

wanneer processen of concurrente uitdrukkingen die gebruik maken van signalen waarvan de waarde afkomstig is van een proces in uitvoering of van een niet gestart proces, dan wordt de daarvoor opgeslagen signaalwaarde gebruikt

# Het proces

## uitvoering

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        voorb: process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

een proces wordt slechts opnieuw gestart wanneer een signaal dat opgenomen is in de sensitivity list een waardeverandering vertoont

( bvb. door de uitvoering van een ander proces of concurrente uitdrukking)

# Het proces

## onderbrekingen

processen zonder gevoeligheidslijst zijn echter ook mogelijk omdat bij het starten van een simulatie alle processen 1 keer worden uitgevoerd

```
process
begin
        in1 <= ‘ 0’;
        in2 <= ‘ 0’;
        wait;
end process;
```

# Het proces

## onderbrekingen

om de signalen die door het proces worden gegenereerd te kunnen gebruiken in andere processen moeten we het onderbreken

```
process
begin
        in1 <= ‘ 0’;
        in2 <= ‘ 0’;
        wait;
end process;
```

# Het proces

## onderbrekingen

wait;

```
process
begin
        in1 <= ‘ 0’;
        in2 <= ‘ 0’;
        wait;
end process;
```

het wait commando zorgt voor de onderbreking

# Het proces

## onderbrekingen

wait;

nadat dit proces 1 keer is uitgevoerd en door het wait commando onderbroken werd, wordt het niet meer gestart

```
process
begin
        in1 <= ‘ 0’;
        in2 <= ‘ 0’;
        wait;
end process;
```

# Het proces

## onderbrekingen

wait;

nadat dit proces 1 keer is uitgevoerd en door het wait commando onderbroken werd, wordt het niet meer gestart

een typische toepassing van dit soort proces is de initialisatie van een systeem

```
process
begin
        in1 <= ‘ 0’;
        in2 <= ‘ 0’;
        wait;
end process;
```

# Het proces

## onderbrekingen

wait for *wachttijd*;

```
process
begin
        clk <= ‘ 0’;
        wait for 50 ns ;
        clk <= ‘ 1’;
        wait for 50 ns ;
end process;
```

met het wait for commando kunnen we een proces gedurende een bepaalde tijd onderbreken

# Het proces

## onderbrekingen

wait for *wachttijd*;

```
process
begin
        clk <= ‘ 0’;
        wait for 50 ns ;
        clk <= ‘ 1’;
        wait for 50 ns ;
end process;
```

wanneer de onderbrekingstijd voorbij is wordt het proces verder uitgevoer tot aan het volgende onderbrekingscommando

# Het proces

## onderbrekingen

wait for *wachttijd*;

```
process
begin
        clk <= ‘ 0’;
        wait for 50 ns ;
        clk <= ‘ 1’;
        wait for 50 ns ;
end process;
```

als end process wordt bereikt wordt het proces opnieuw opgestart

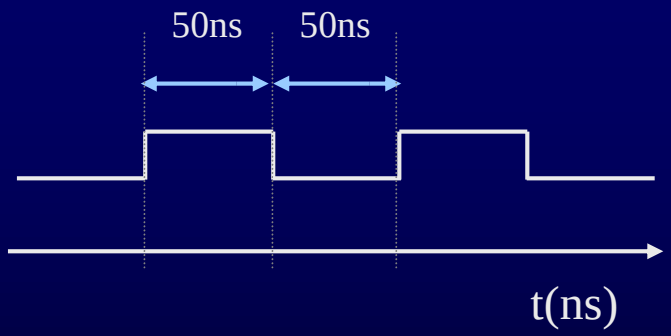
# Het proces

## onderbrekingen

wait for *wachttijd*;

in dit voorbeeld wordt hierdoor een kloksignaal (clk) met een periode van 100 ns opgewekt

```
process
begin
        clk <= ‘ 0’;
        wait for 50 ns ;
        clk <= ‘ 1’;
        wait for 50 ns ;
end process;
```



# Het proces

## onderbrekingen

wait until *uitdrukking*;

```
process
begin
        wait until enable='1'
        out <= data_in;
end process;
```

met de wait until variante van het wait commando blijft een proces onderbroken tot het resultaat van de uitdrukking waar is

# Het proces

## onderbrekingen

wait on *signaallijst*;

```
process
begin
        wait on  A, B
        out <= data_in;
end process;
```

met de wait on variante blijft het proces onderbroken tot één van de vermelde signalen van waarde verandert

(dit komt op hetzelfde neer als een proces met gevoeligheidslijst)

# Het proces

## signalen/variabelen

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

buiten de gebruikelijke signalen, kunnen we in een proces ook veranderlijken gebruiken

# Het proces

## signalen/variabelen

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

de declaratie van de veranderlijken gebeurt in het declaratiedeel (voor BEGIN) van het proces

# Het proces

## signalen/variabelen

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 := a and b;
                        tmp2 := Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```

de veranderlijken worden gebruikt voor tijdelijke opslag van informatie binnen een proces

dit verbetert de duidelijkheid of de structuur van een proces

# Het proces

## signalen/variabelen

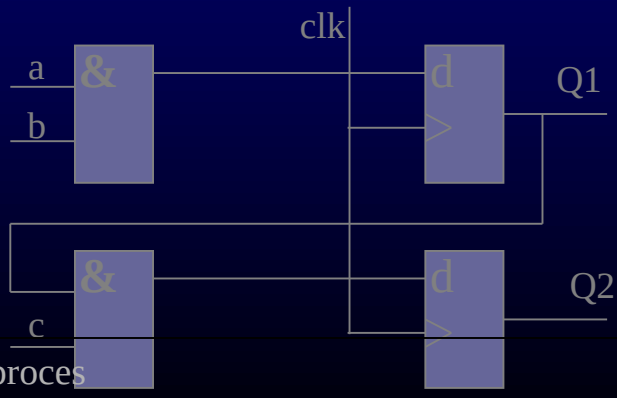
```
architecture versie1 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
        begin
                if rising_edge(clk)
                        Q1 <= a ar
                        Q2 <= Q1

                end if;
        end process;
end versie 1;
```

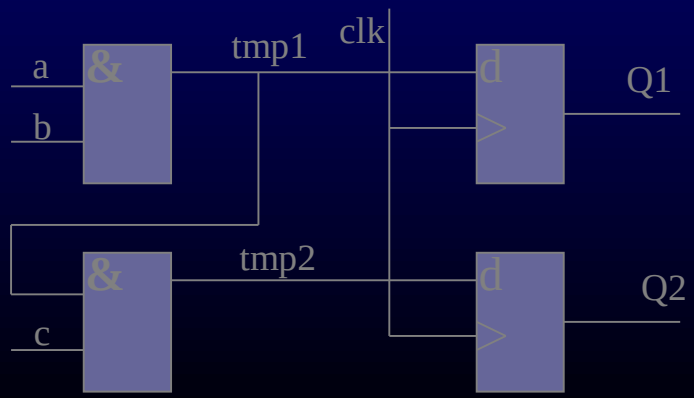
**Opgelet !!!**

De waardetoekenning aan een veranderlijke gebeurt onmiddelijk bij het uitvoeren van het commando

De waardetoekenning aan een signaal gebeurt slechts na onderbreking van het proces

```
        Q1 <= tmp1;
        Q2 <= tmp2;
    end process;
end versie 2;
```



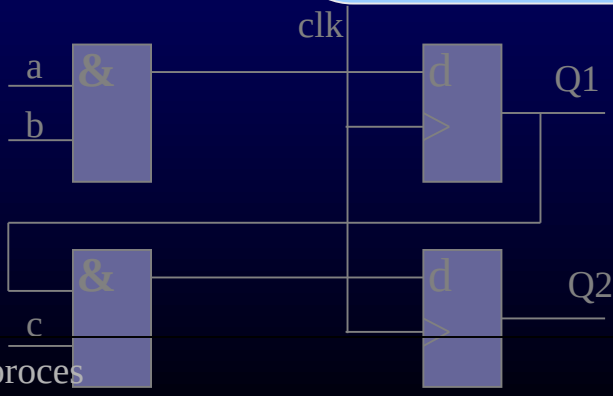
# Het proces

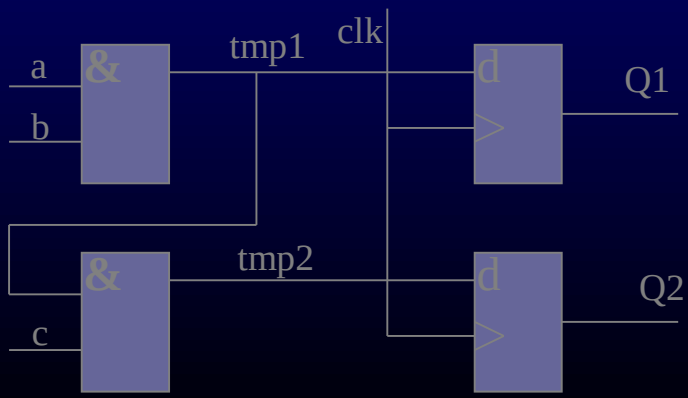
## signalen/variabelen

```
architecture versie1 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
        begin
                if rising_edge(clk)
                        Q1 <= a and b;

        end proces
end versie 1;
```

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 <= a and b;
                        tmp2 <= Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        ;
```

hierdoor kan bij synthese van een schakeling het al of niet gebruiken van veranderlijken een verschillend synthese-resultaat opleveren,



# Het proces

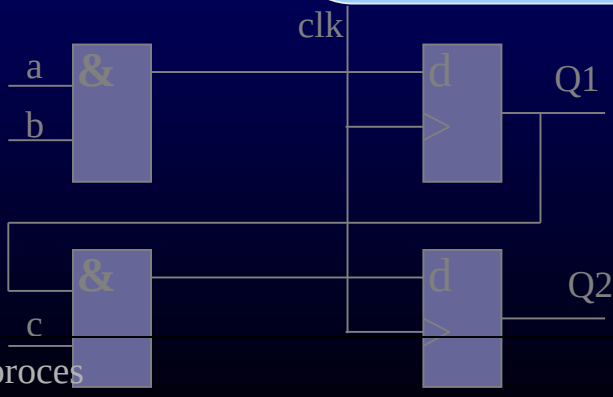
## signalen/variabelen

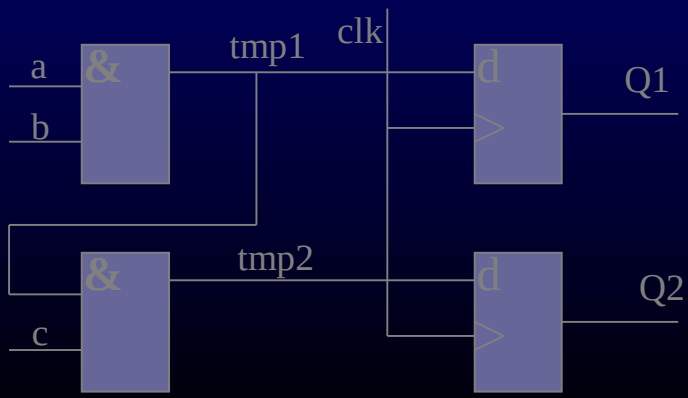
```
architecture versie1 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
        begin
                if rising_edge(clk)
                        Q1 <= a and b;

        end proces
end versie 1;
```

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 <= a and b;
                        tmp2 <= Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        ;
```

hierdoor kan bij synthese van een schakeling het al of niet gebruiken van veranderlijken een verschillend synthese-resultaat opleveren,

dit wordt in het volgende voorbeeld geillustreerd



# Het proces

## signalen/variabelen

```
architecture versie1 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
        begin
                if rising_edge(clk)
                        Q1 <= a and b;
                        Q2 <= Q1 and c;
                end if;
        end process;
end versie 1;
```

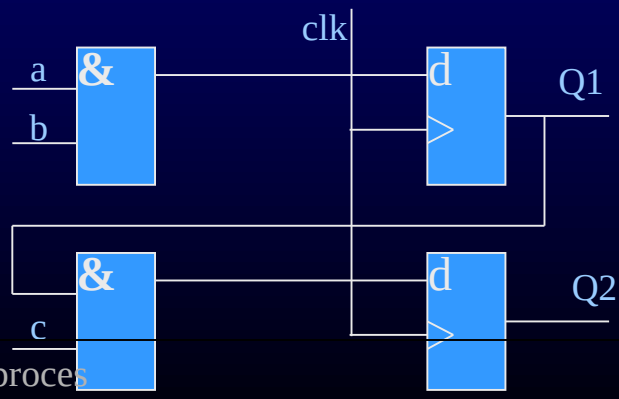
dit proces is enkel met signalen uitgevoerd

# Het proces

## signalen/variabelen

```
architecture versie1 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
        begin
                if rising_edge(clk)
                        Q1 <= a and b;
                        Q2 <= Q1 and c;
                end if;
        end process;
end versie 1;
```

dit proces is enkel met signalen uitgevoerd




het proces

synthese levert dit schema op

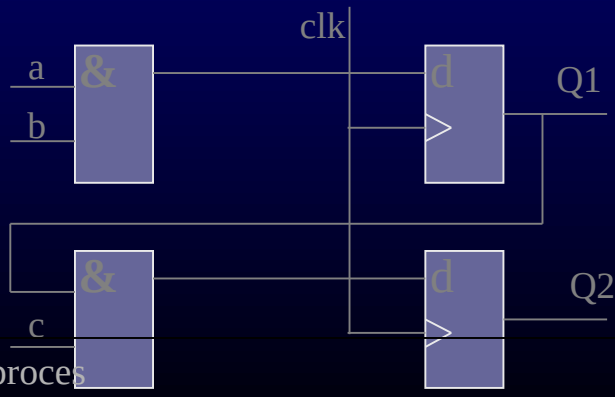
# Het proces

## signalen/variabelen

```
architecture versie1 of circuit is
        signal Q1,Q2: std_logic;
begin
                                    e(clk)
                                        and b;
                                    Q2 <= Q1 and c;
                end if;
        end process;
end versie 1;
```

hetzelfde proces uitgevoerd met veranderlijken

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 <= a and b;
                        tmp2 <= Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```



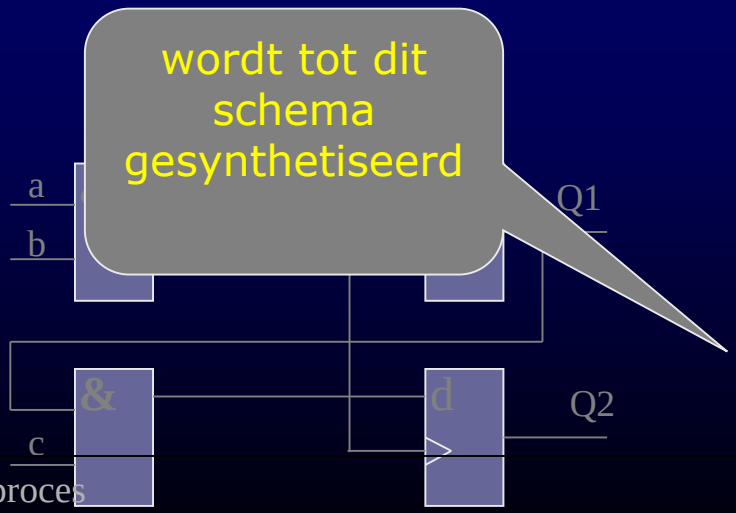

het proces

# Het proces

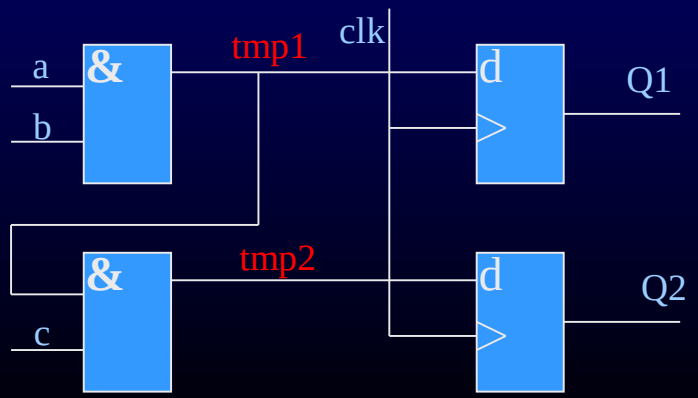
## signalen/variabelen

```
architecture versie1 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
        begin
                if rising_edge(clk)
                        Q1 <= a and b;
                        Q2 <= Q1 and c;
                end if;
        end process;
end versie 1;
```

```
architecture versie2 of circuit is
        signal Q1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 <= a and b;
                        tmp2 <= Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```




het proces



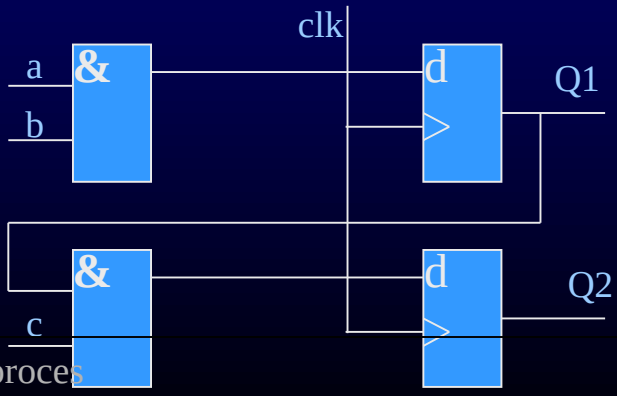
# Het proces

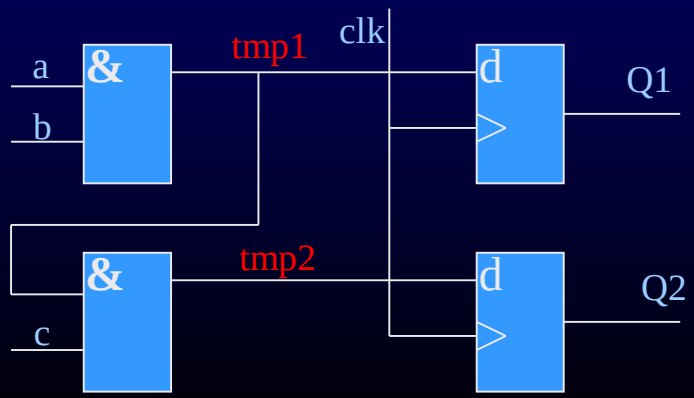
## signalen/variabelen

beide schema's zijn duidelijk verschillend

```
architecture versie1 of circuit is
        signal Q1,Q2: std_log
begin
        process (clk)
        begin
                if rising_edge(clk)
                        Q1 <= a and b;
                        Q2 <= Q1 and c;
                end if;
        end process;
end versie 1;
```

```
f circuit is
1,Q2: std_logic;
begin
        process (clk)
                variable tmp1,tmp2: std_logic
        begin
                if rising_edge(clk)
                        tmp1 <= a and b;
                        tmp2 <= Q1 and c;
                end if;
                Q1 <= tmp1;
                Q2 <= tmp2;
        end process;
end versie 2;
```



# VHDL

EINDE