



Høgskulen  
på Vestlandet

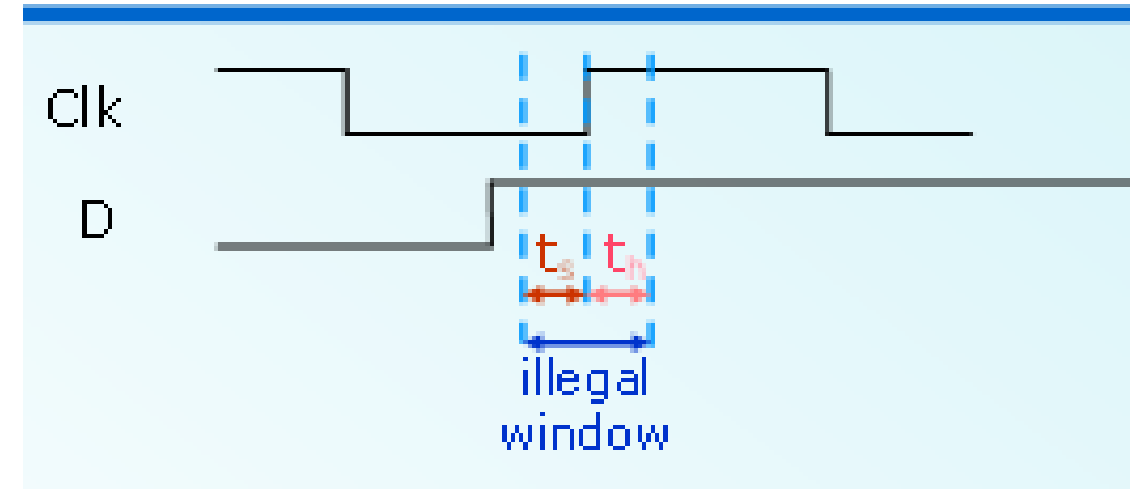
## ELE111 Digitale design

### F18\_000 Timing og synkronisering

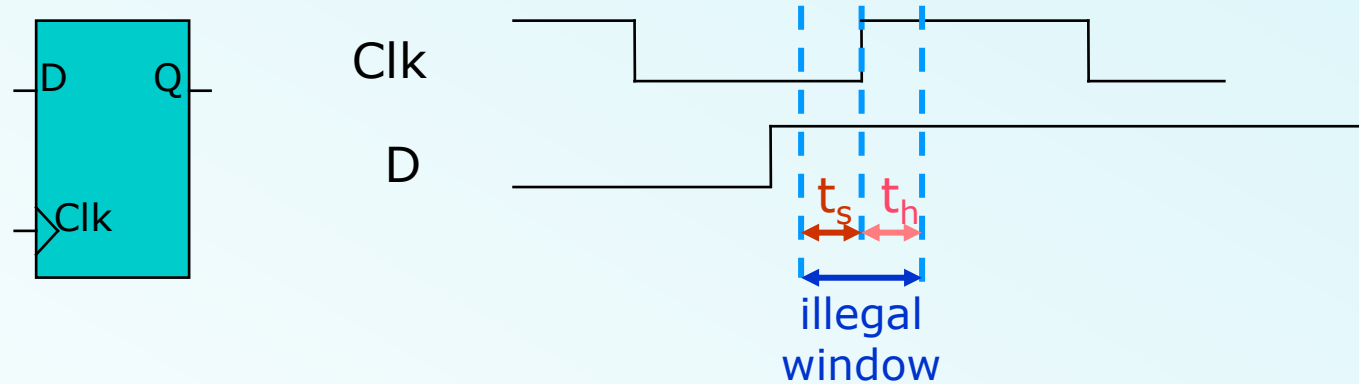
Basert på foredrag på FPGA-forum  
av Espen Tallaksen

---

Eivind Skjæveland  
esk@hvl.no



# Timing requirements

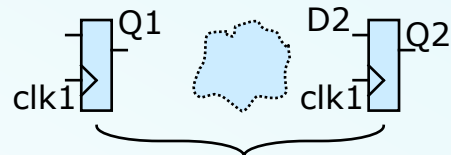


Setup-time ( $t_s$ ): The time D must be valid/stable before active C

Hold-time ( $t_h$ ): The time D must be held stable after active C

Illegal window: The time window from  $t_s$  to  $t_h$  where D must be stable

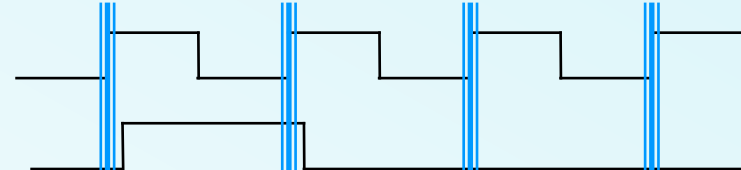
# Simple timing path / requirement



Pure synchronous  
timing path &  
timing requirement

clk1

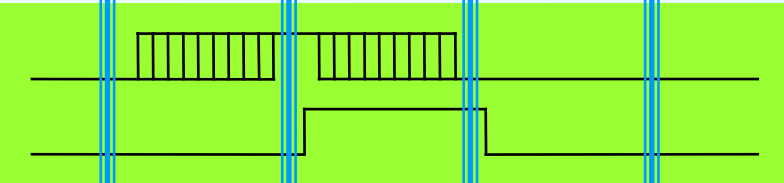
Q1



Timing is OK (D2 worst case)

D2-a

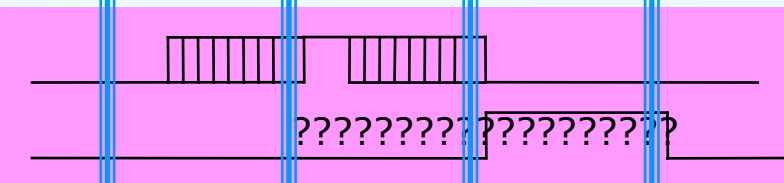
Q2-a



D2 (worst case) arrives too late,  
but after illegal window;  
- or ...?

D2-b

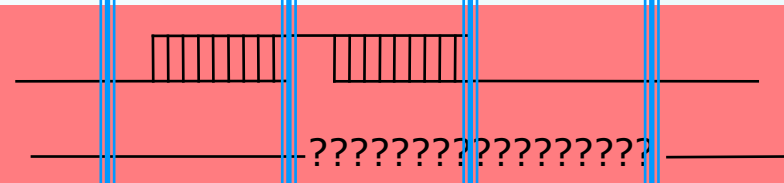
Q2-b



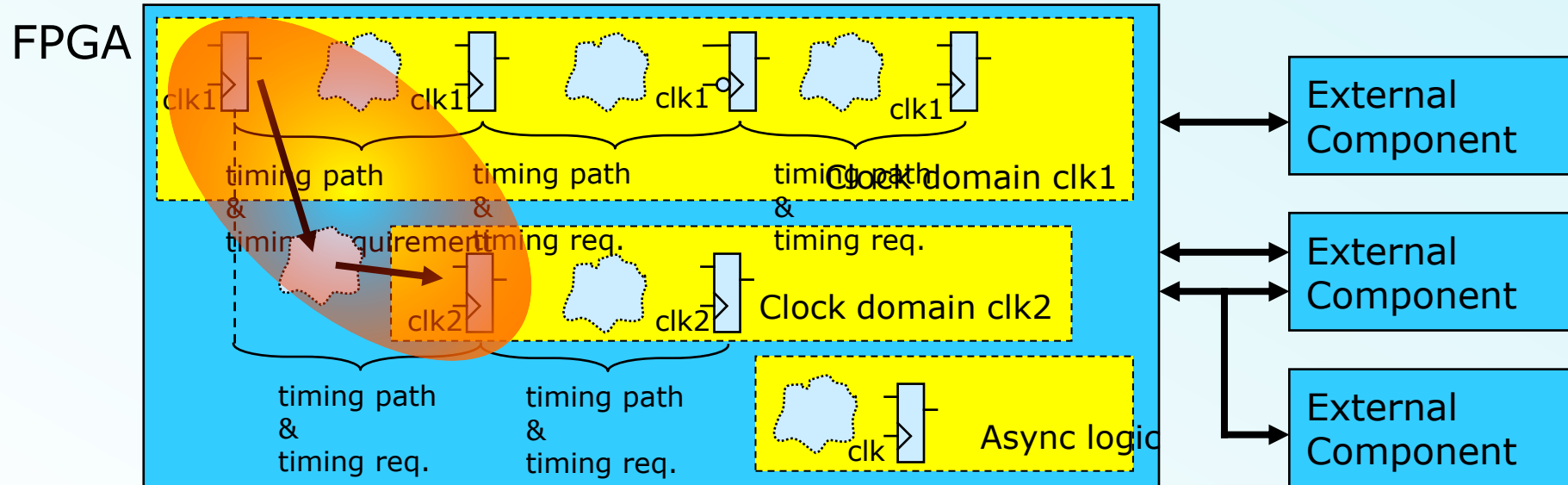
D2 arrives inside illegal window

D2-c

Q2-c



# FPGA Timing - Basics




Storage elements: Flops, memories, latch, black-box

Timing path: From active clock edge on source element to input on destination element plus setup-time

## Clock relations, synchronization, handshake, etc...

→ CDC: Clock Domain Crossing

I/O timing could be pure timing path or additional CDC – with different timing for each external component

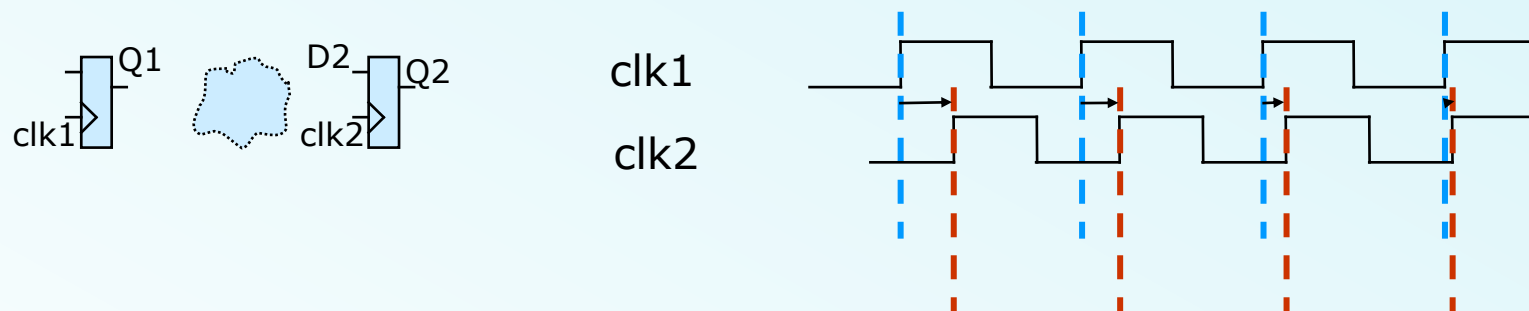


Storage element



Combinat.  
logic

# CDC: Clock Domain Crossing

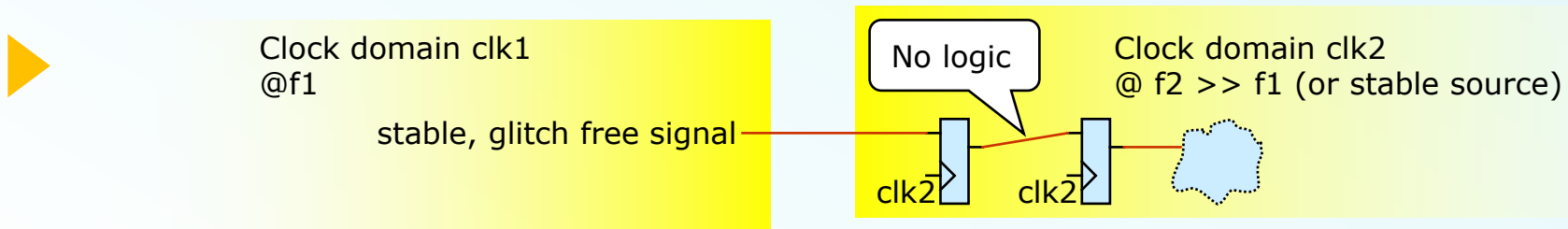


- Always assume sliding clock relations
  - ✓ Variable phase difference
    - » No control on or knowledge about actual phase diff.
  - ✓ D2 will arrive before, after and on clk2
  - ✓ Assume worst case for ALL signals going from one clock domain to another.
- Fixed clock relations may help
  - ✓ Handled later

# Single signal synchronization

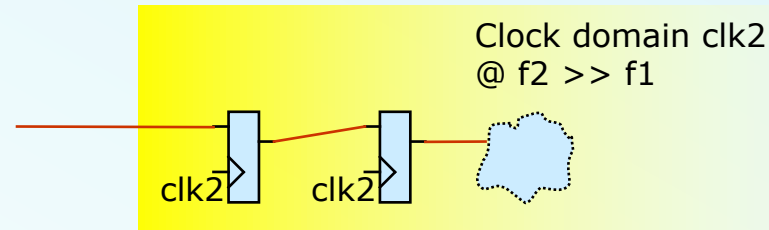
## - with faster destination

- Always:
  - ✓ Ensure stable and glitch free signal out of source domain
- For input to a faster domain:
  - ✓ Two synchronization flip-flops normally recommended
  - ✓ High frequencies or tight timing may require more
  - ✓ May utilise both edges to reduce latency



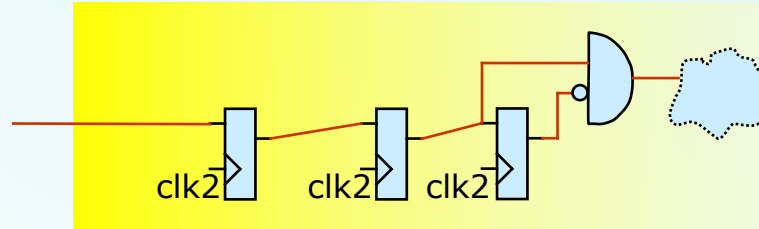
- For input to a slower domain (or unknown frequency):
  - ✓ Need handshake

# Why two flip-flops?



- Two flops – the rule of thumb for several decades...
- But, meta-stability characteristics has significantly improved
  - So why isn't a single flop sufficient?
    - ✓ In fact in many cases it would be..., but
      - » Would require tightened timing requirements out of flop 1
      - » Needs tighter follow-up throughout design-phase
      - » Does still have a higher risk of failure – for critical applications
      - » Why save a flop?
- ➔ Still a good rule: **Use two flip-flops for synchronization**
  - ➔ Really high frequencies might require more

# Pulse detection in destination domain



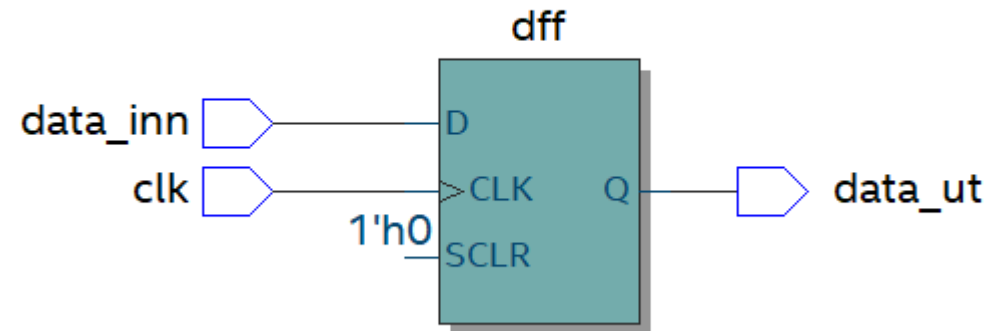
- Required when synchronized signal used as enable/trigger
  - ✓ To assure single enable/trigger
- Position after 2nd synch-flop



# Synkronisering av inn-signal – 1 register

- › Ved å setja på eit register på inngangen, blir data synkroniserte
- › Eit register løyser mange problem, men ikkje alltid nok
- › Timing-feil kan oppstå.

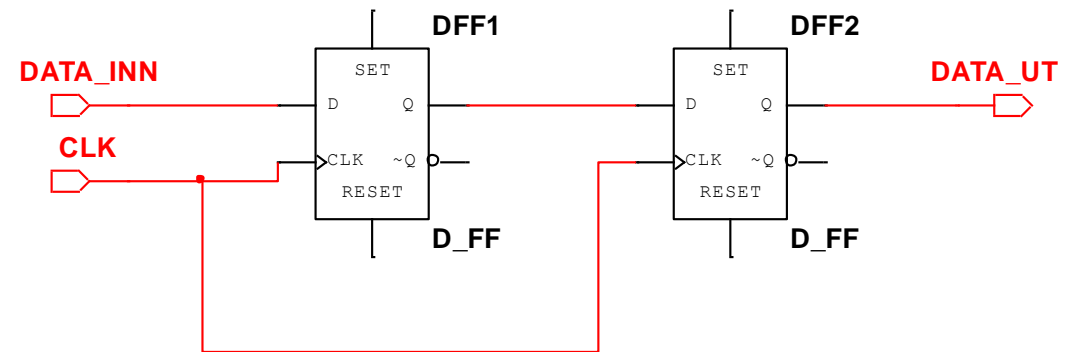
```
p_sync_et_register : process (clk) is
    begin
        if rising_edge(clk) then
            dff <= data_inn;
        end if;
    end process p_sync_et_register;
    data_ut <= dff;
```



# Synkronisering av innsignal – 2 register

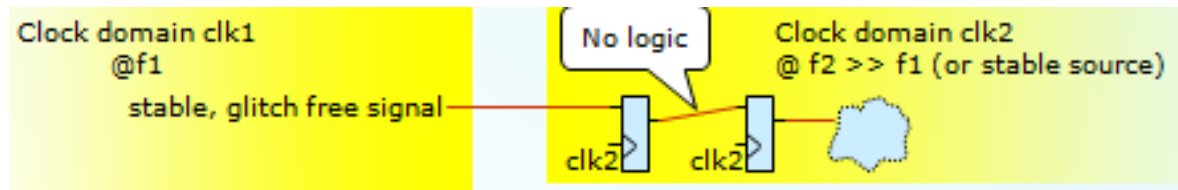
- › Mykje sikrare synkronisering med to register
- › "BEST PRAKSIS" for FPGA seier:
  - › Alle innsignal skal synkroniserast
  - › Bruk to register til synkronisering
  - › Register er billige !!

```
p_sync_to_register : process (clk) is
begin
    if rising_edge(clk) then
        dff1 <= data_inn;
        dff2 <= dff1;
    end if;
end process p_sync_to_register;
data_ut <= dff2;
```



# Synkronisering av to klokke domener

- Full sikkerhet ved synkronisering av to system:



```
p_sync_to_register : process (clk) is
```

```
begin
```

```
  if rising_edge(clk) then
```

```
    dff1 <= data_inn;
```

```
    dff2 <= dff1;
```

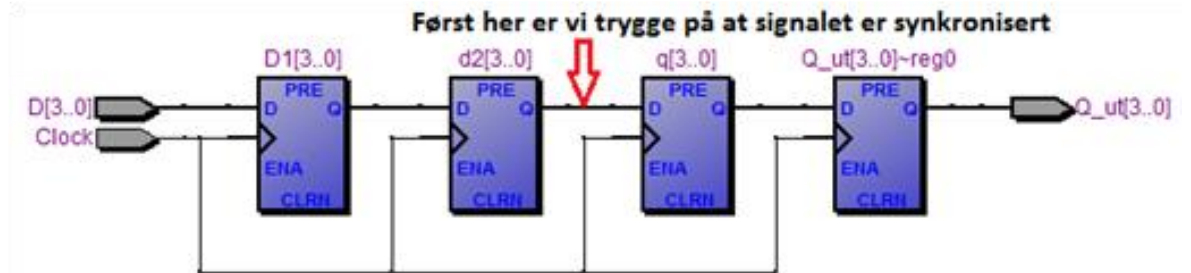
```
    -- her kan vi begynne å bruke inndataene
```

```
    Q <= dff2;
```

```
    Q_UT <= Q;
```

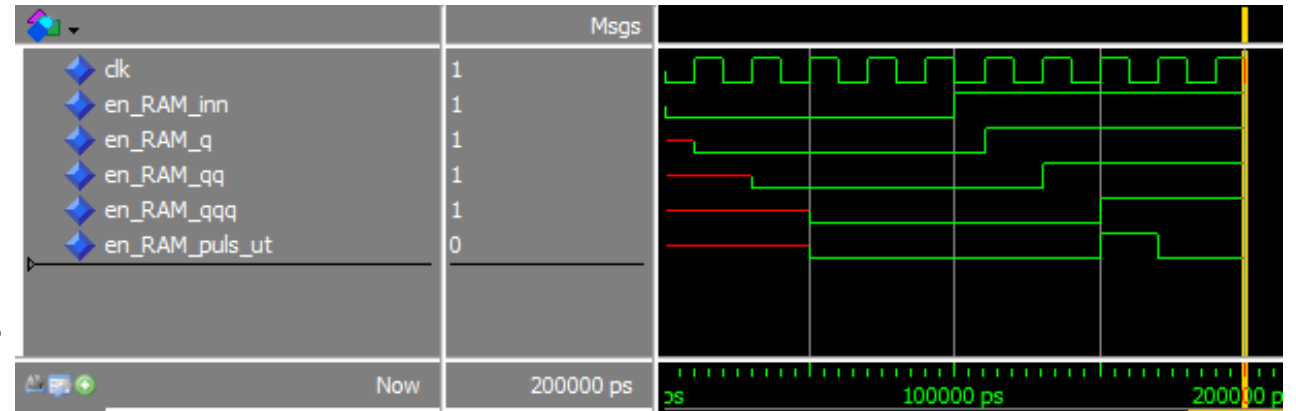
```
  end if;
```

```
end process p_sync_to_register;
```



# Flankedeteksjon

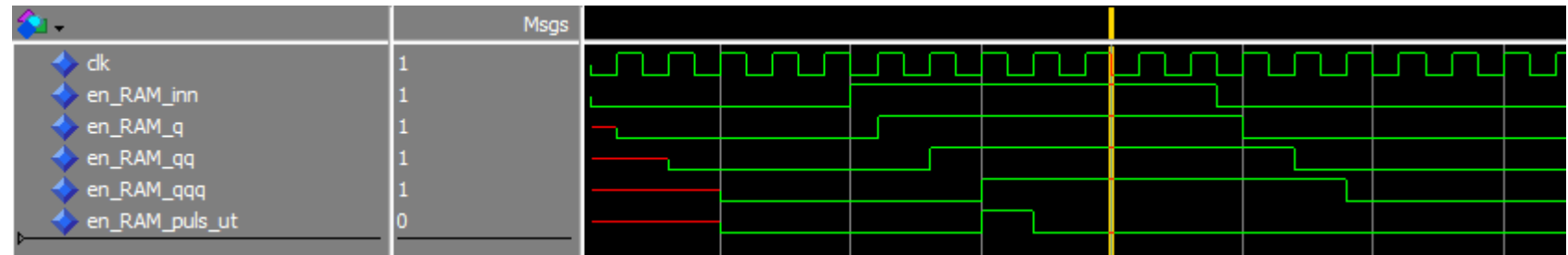
- › Vi trenger å detektera når eit innsignal går frå låg til høg
  - › Stigande flanke.
  - › Dette vil vi signalisera med ein puls som varer ein klokkeperiode .



```
p_sync_og_flankedeteksjon : process(clk) is
begin
  if rising_edge(clk) then
    en_RAM_q <= en_RAM_inn;
    en_RAM_qq <= en_RAM_q;

    -- her kommer flankedeteksjon

    en_RAM_qqq <= en_RAM_qq;
    -- vi har stigende flanke når en_RAM_qq er 1 og en_RAM_qqq er 0
    en_RAM_puls_ut <= (NOT en_RAM_qqq) AND en_RAM_qq;
  end if;
end process p_sync_og_flankedeteksjon;
```





Timing og synkronisering.