

# Schieberegister

- Arbeitsweise
- Serien-Parallel Umsetzer
- Parallel-Serien Umsetzer
- Zähler mit Schieberegister
- Linear Rückgekoppelte Schieberegister



Prof. Dr. J. Reichardt  
Lehrbuch Digitaltechnik  
Eine Einführung mit VHDL, 3. A  
Oldenbourg Wissenschaftsverlag  
München 2013  
ISBN 978-3-486-72765-4

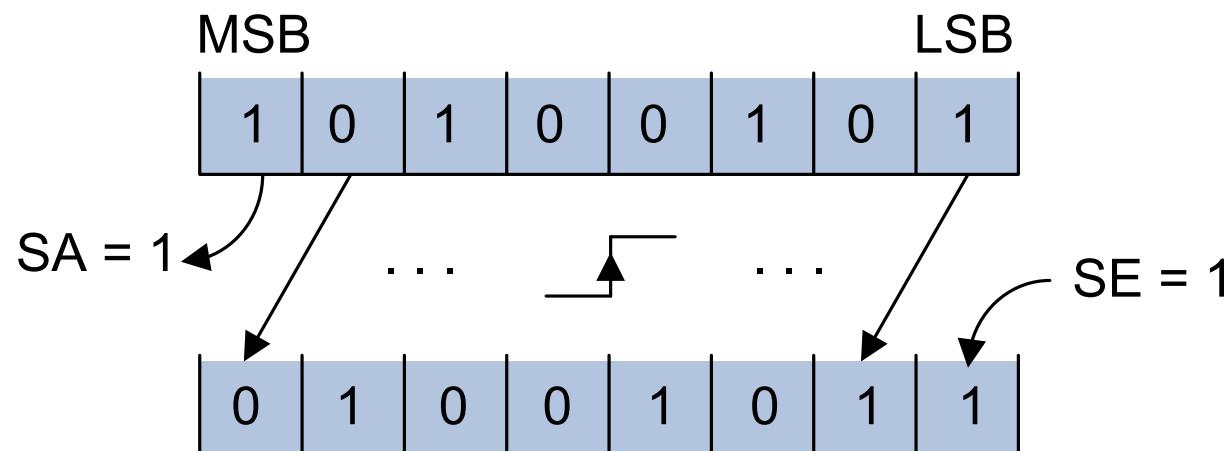
# Schieberegister

Schieberegister sind Schaltungen aus Flipflops, die kettenförmig aufgebaut sind und durch einen gemeinsamen Takt versorgt werden. Dadurch werden in jedem Takt Datenbits entweder nach links (in Richtung des MSB) oder nach rechts (in Richtung des LSB) verschoben.

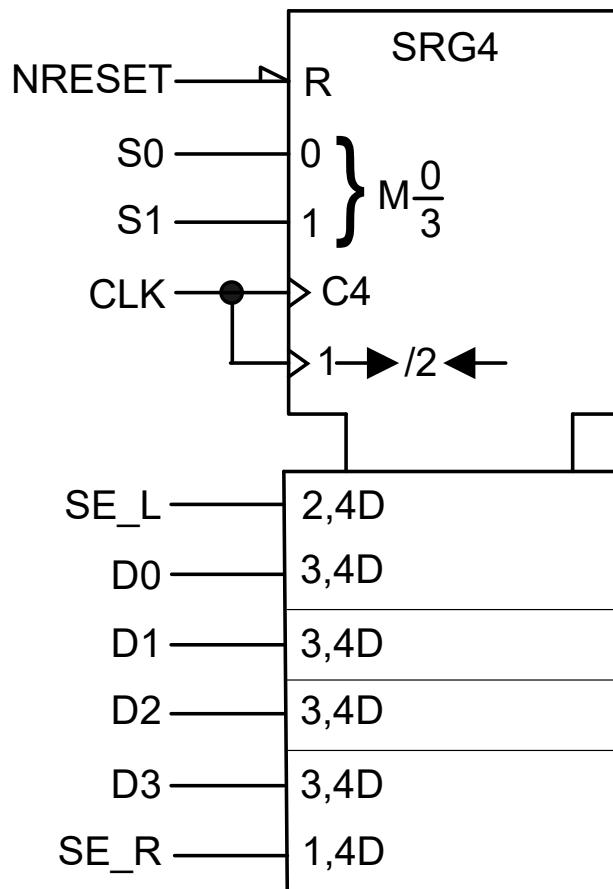
## Arbeitsweise eines Links-Schieberegisters:

Bei einer steigenden Taktflanke werden (gleichzeitig):

- das MSB auf den Schiebeausgang SA verschoben,
- alle anderen Datenbits auf die nächst höherwertige Bitposition verschoben
- das LSB mit dem Schiebeeingangsbit SE geladen.



# Konfigurierbares Schieberegister 74194



S1 S0	Betriebsart
0 0	Keine Aktion
0 1	Rechts schieben; Einlesen des SE_R-Bits nach Q(3)
1 0	Links schieben; Einlesen des SE_L-Bits nach Q(0)
1 1	Laden der Dateneingänge D(3)...D(0)

VHDL-Deklaration:

```
signal Q : bit_vector(3 downto 0);
```

Links-Schieben:  $Q \leftarrow Q(2 \text{ downto } 0) \& SE\_L;$

Rechts-Schieben:  $Q \leftarrow SE\_R \& Q(3 \text{ downto } 1);$

# Serielle Übertragungsprotokolle

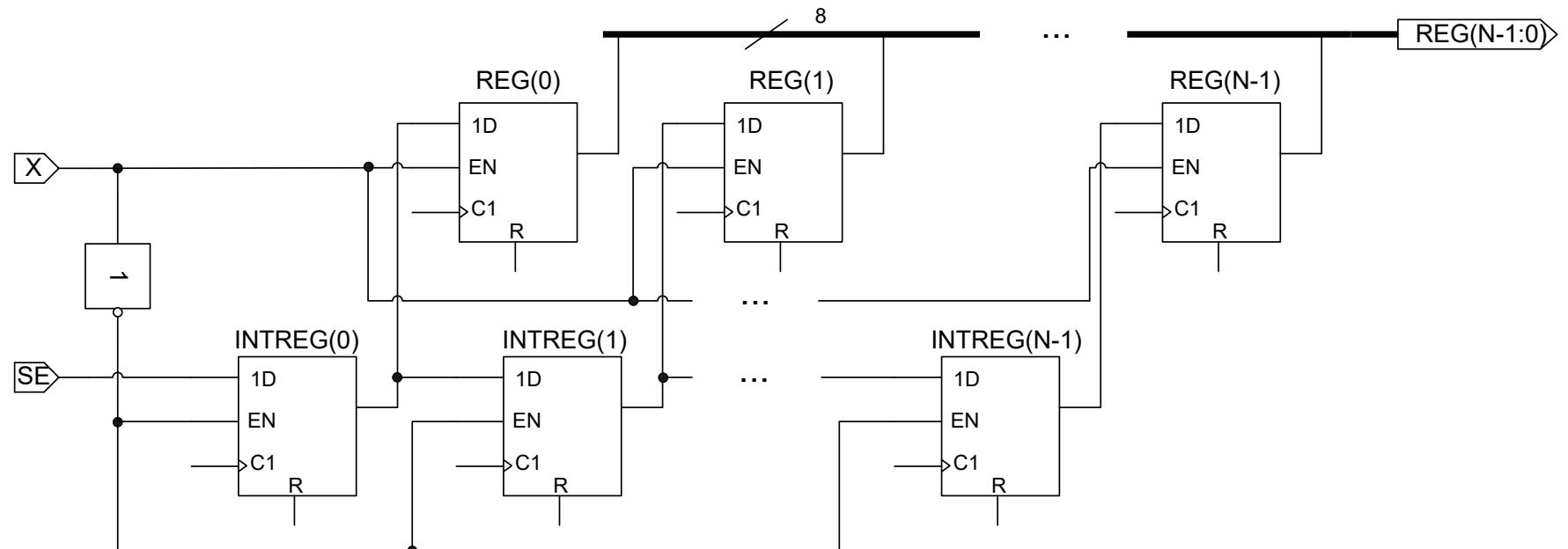
## Beispiele für serielle Protokolle:

- Der RS232C- bzw. COM-Standard des PCs dient der Kommunikation mit Peripheriegeräten.
- I<sup>2</sup>C-Bus-Schnittstellen werden insbesondere zur Konfiguration von ASICs in Konsumergeräten verwendet.
- Der CAN-Bus wurde zur Kommunikation von Halbleiterchips in mobilen Fahrzeugen entwickelt.
- Das SATA-Protokoll arbeitet mit seriellen Datenraten im GHz-Bereich und wird zum Anschluss schneller Festplatten eingesetzt.
- Das SPI-Bus-Interface erlaubt eine serielle Kommunikation zwischen verschiedenen Digitalkomponenten.
- ...

# Serien-Parallel-Umsetzer

Ein Serien-Parallel-Umsetzer hat die Aufgabe, die auf einer einzelnen Datenleitung seriell eintreffenden Datenbits in N-Bit-Datenworte umzusetzen.

- Zwei Gruppen von Flipflops:
  - N-Flipflops ( $\text{INTREG}(N-1) \dots, \text{INTREG}(0)$ ), die als Schieberegister beschaltet sind.
  - N-Flipflops ( $\text{REG}(N-1) \dots \text{REG}(0)$ ), die als paralleles Auffangregister (engl. buffer register) geschaltet sind.
- Mit dem Steuersignal X wird eingestellt, ob geschoben oder im Auffangregister gespeichert wird.



# VHDL-Modell des Serien-Parallel-Umsetzers

```
entity SER_PAR is
  generic(N : natural :=3);           -- Anzahl der Bits, von
  port( CLK, SE, RESET, X: in bit;
        REG: out bit_vector(N-1 downto 0));    -- N Register flipflops
end SER_PAR;
architecture VERHALTEN of SER_PAR is
signal INTREG: bit_vector(N-1 downto 0);    -- N Schieberegister Flipflops
begin
P1: process(CLK, RESET)
begin
  if RESET='1' then
    INTREG <= (others => '0') after 5 ns;      -- asynchroner Reset
    REG <= (others => '0') after 5 ns;
  elsif (CLK='1' and CLK'event) then          -- bei ansteigender Flanke
    if X='0' then
      INTREG <= INTREG(N-2 downto 0) & SE after 5 ns;    -- Schieben
    else
      REG <= INTREG after 5 ns;                  -- ins Pufferregister
    end if;
  end if;
end process P1;
end VERHALTEN;
```

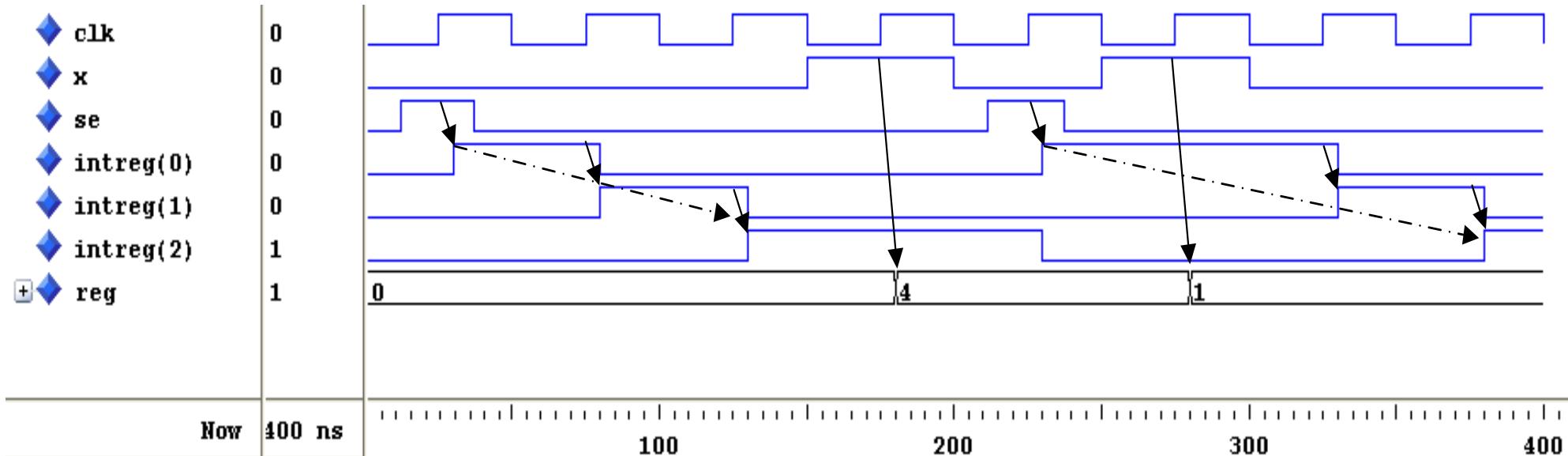
Mit dem generic wird die Breite des Schieberegisters dimensioniert.

Asynchroner Reset setzt beide Register zurück.

Taktsynchron Links Schieben

Taktsynchron im Pufferregister speichern

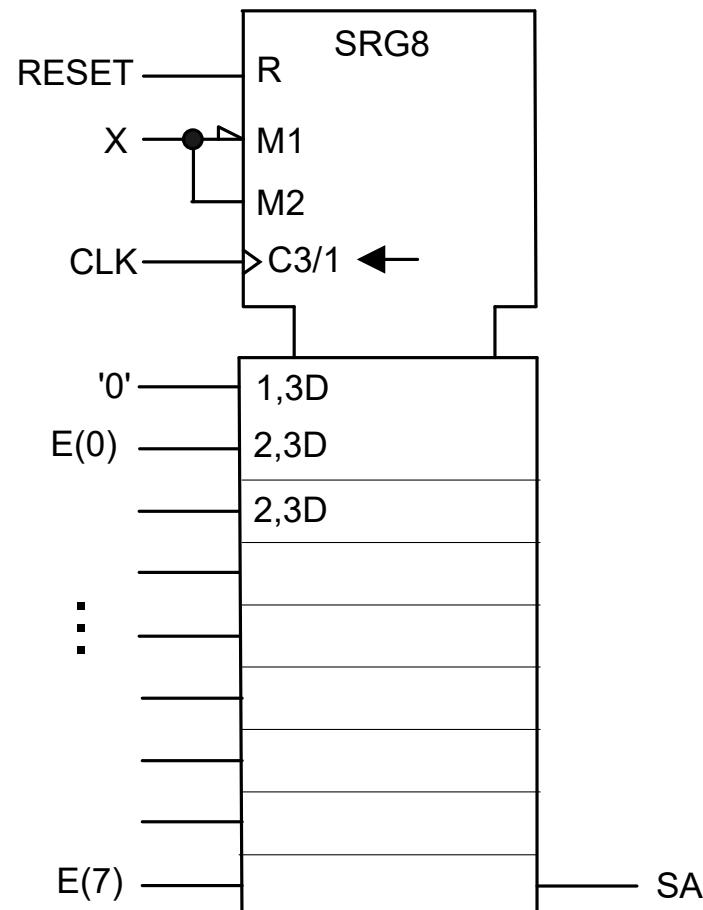
# Simulation eines 3-Bit-Serien-Parallel-Umsetzers



- Am Schiebeeingang wird zunächst die Folge 1 0 0 angelegt. Das 1-Bit wird durch das INTREG-Register geschoben → Taktsynchron erscheint mit X=1 das parallele Datenwort 0x4 im REG Auffangregister
- In der zweiten Folge wird das Schieben schon nach einem Takt beendet → Taktsynchron erscheint mit X=1 das parallele Datenwort 0x1 im Auffangregister

# Parallel-Serien-Umsetzer

Ein Parallel-Serien-Umsetzer hat die Aufgabe, ein paralleles N-Bit-Datenwort in einen seriellen Datenstrom von N Bit umzusetzen.



## Zwei Betriebsarten:

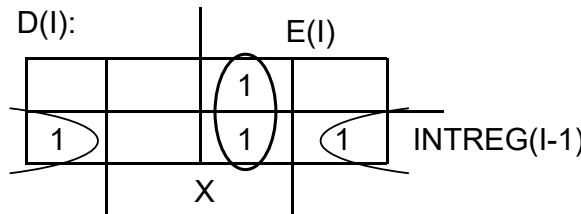
- M1 ( $X=0$ ): Schieben: Es wird in das niedrigwertigste Schieberegisterbit eine 0 geladen und die weiteren Bits werden taktsynchron zum jeweils nächst höherwertigen verschoben.
- M2 ( $X=1$ ): Laden: Es werden die am Dateneingang E anliegenden Bits taktsynchron in das Schieberegistersignal geladen.
- Das jeweilige MSB erscheint immer am SA-Ausgang.

# Übergangslogik im Parallel-Serien-Umsetzer

a)

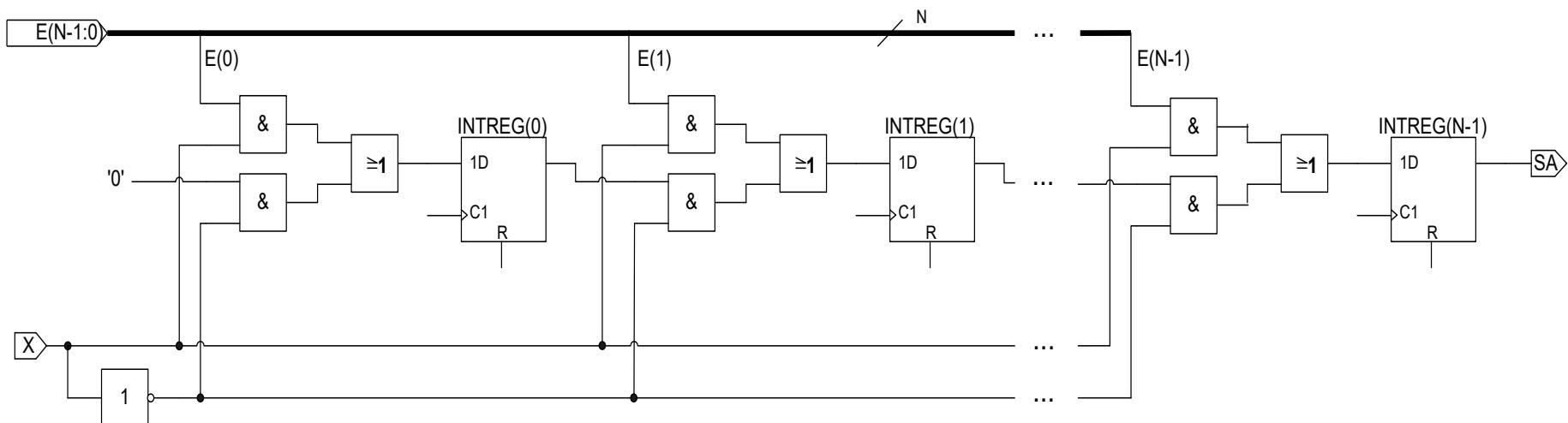


b)



**Die Wahlmöglichkeit M1 / M2 wird durch einen Multiplexer realisiert, der als Vorbereitungsschaltnetz dient.**

X	INTREG(I-1)	E(I)	D(I)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



# VHDL-Modell des Parallel-Serien-Umsetzers

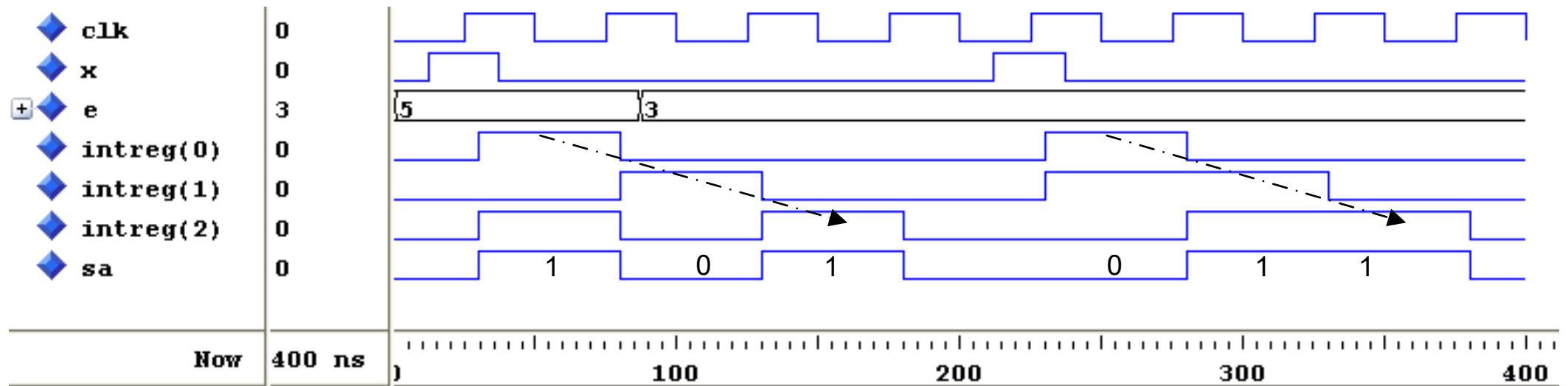
```
entity PAR_SER is
    generic(N : natural :=3); -- Anzahl der Bits, von
    port( CLK, RESET, X: in bit;
          E: in bit_vector(N-1 downto 0);
          SA: out bit);
end PAR_SER;
architecture VERHALTEN of PAR_SER is
signal INTREG: bit_vector(N-1 downto 0);
begin
P1: process(CLK, RESET)
begin
    if RESET='1' then
        INTREG <= (others => '0') after 5 ns;
    elsif (CLK='1' and CLK'event) then      -- takt synchron
        if X='1' then
            -- Daten laden:
            INTREG <= E after 5 ns;
        else
            -- LINKS schieben:
            INTREG <= INTREG(N-2 downto 0) & '0' after 5 ns;
        end if;
    end if;
end process P1;
SA <= INTREG(N-1);
end VERHALTEN;
```

Mit dem generic wird die Breite des Schieberegisters dimensioniert.

Laden oder Schieben.

Das MSB wird kombinatorisch am Schiebeausgang SA ausgegeben.

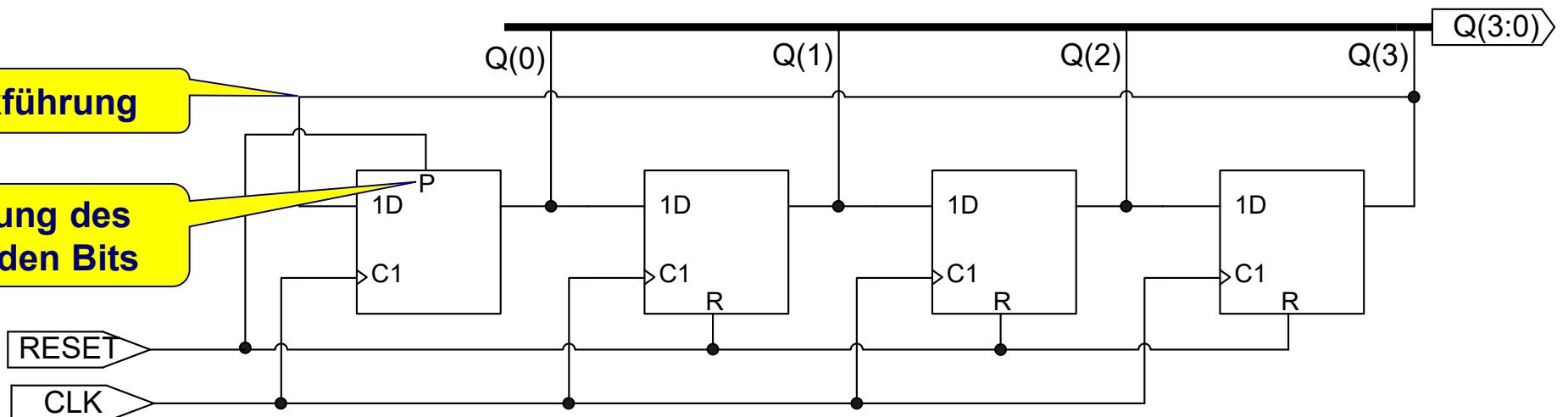
# Simulation eines 3-Bit-Parallel-Serien-Umsetzers



- Bei  $t = 25$  ns wird das Datenwort 0x5 geladen und in drei Takten seriell als 101 ausgegeben (MSB zuerst da Links-SRG).
  - Bei  $t = 225$  ns wird das Datenwort 0x3 geladen und seriell als 011 ausgegeben.

# Zähler mit Schieberegistern

- Mit einem Schieberegister muss ein zyklisches Zustandsdiagramm abgebildet werden.
- Vorteil:
  - Das Übergangsschaltnetz in Schieberegistern ist sehr einfach, daher kann eine sehr hohe Taktfrequenz erreicht werden.
- Nachteile:
  - Es lassen sich nicht alle Zustände im zyklischen Zustandsdiagramm erreichen → Die vielen Pseudozustände erfordern ein Korrekturschaltnetz
  - Die Schieberegisterbitkombinationen sind nicht binär aufsteigend.
- Beispiel: Ringzähler mit Links-SRG: Es soll ein einzelnes Bit zirkulieren.



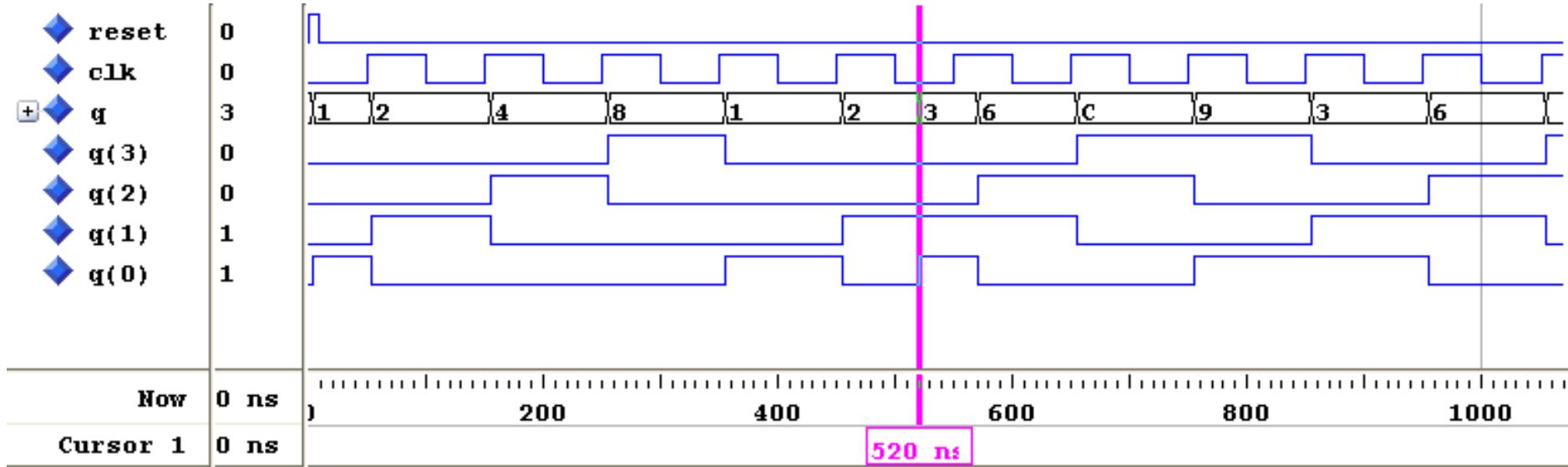
# VHDL-Code des Ringzählers

```
entity RING_CTR is
  port( CLK, RESET : in bit;
        Q : out bit_vector(3 downto 0)
      );
end RING_CTR;
architecture VERHALTEN of RING_CTR is
signal QINT: bit_vector(3 downto 0);
signal SERIAL_FEEDBACK : bit;
signal D: bit_vector(3 downto 0) := "0001";
begin
P1: process(CLK, RESET)
begin
  if RESET = '1' then
    QINT <= D after 5 ns;
  elsif CLK='1' and CLK'event then
    QINT <= QINT(2 downto 0) & SERIAL_FEEDBACK after 5 ns;
  end if;
end process P1;
SERIAL_FEEDBACK <= QINT(3), -- Zyklische Rückkopplung des MSB
Q <= QINT;                  -- Kopie als Ausgangssignal
end VERHALTEN;
```

The code is annotated with four yellow callout boxes:

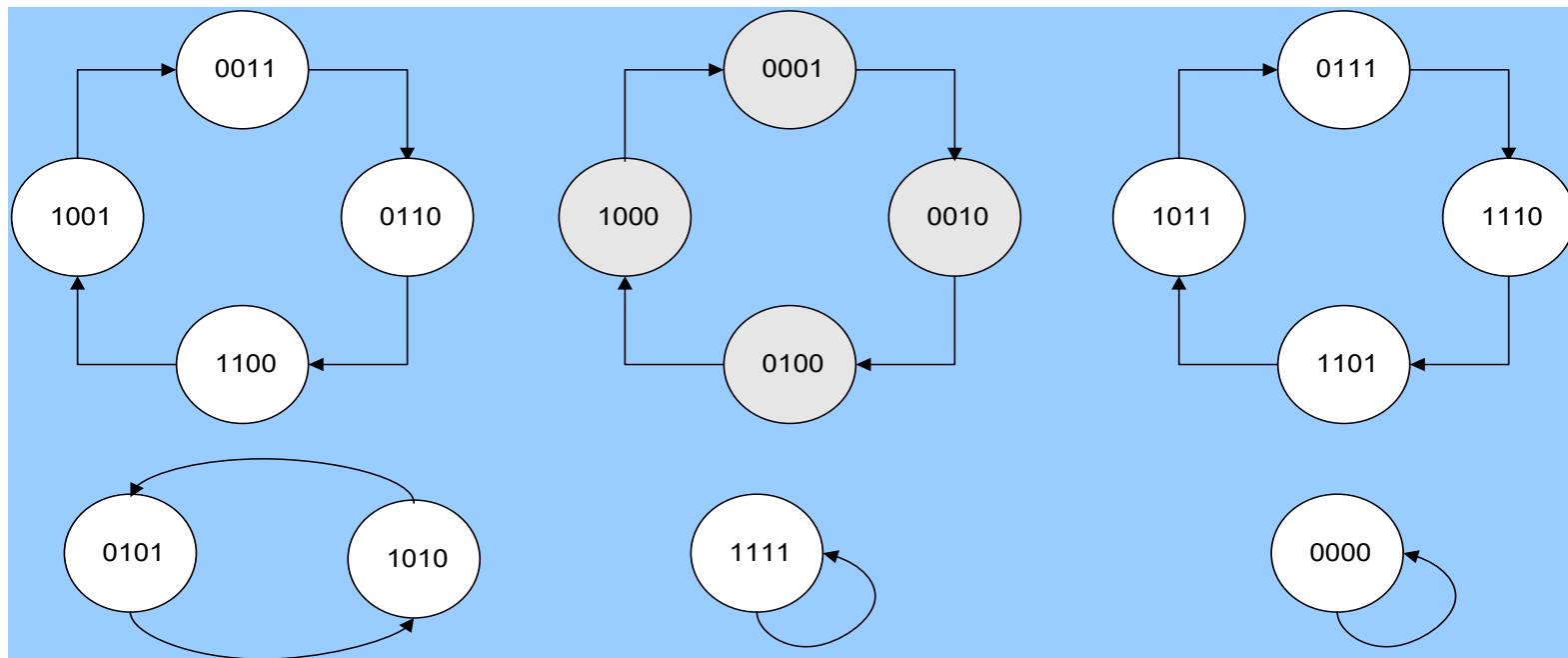
- A yellow box labeled "Serielles Eingangssignal" points to the signal `SERIAL_FEEDBACK`.
- A yellow box labeled "Initialisierungsmuster" points to the initial value assignment `signal D: bit_vector(3 downto 0) := "0001";`.
- A yellow box labeled "Links-SRG" points to the assignment `QINT <= D after 5 ns;`.
- A yellow box labeled "Zyklische Rückkopplung des MSB" points to the assignment `SERIAL_FEEDBACK <= QINT(3);`.

# VHDL-Simulation des Ringzählers



- Der RESET bei  $t = 0$  setzt nur das LSB.
- Bis  $t = 500$  ns zirkuliert genau ein Bit.
- Bei  $t = 520$  ns wird ein fehlerhaftes Bitmuster simuliert: Durch eine externe Störung sind plötzlich zwei Bit gesetzt ( $0x3$ ) → Ohne Korrekturschaltzeit kommt der Zähler niemals in seinen normalen Zyklus zurück !

# Vollständiges Zustandsdiagramm des Ringzählers

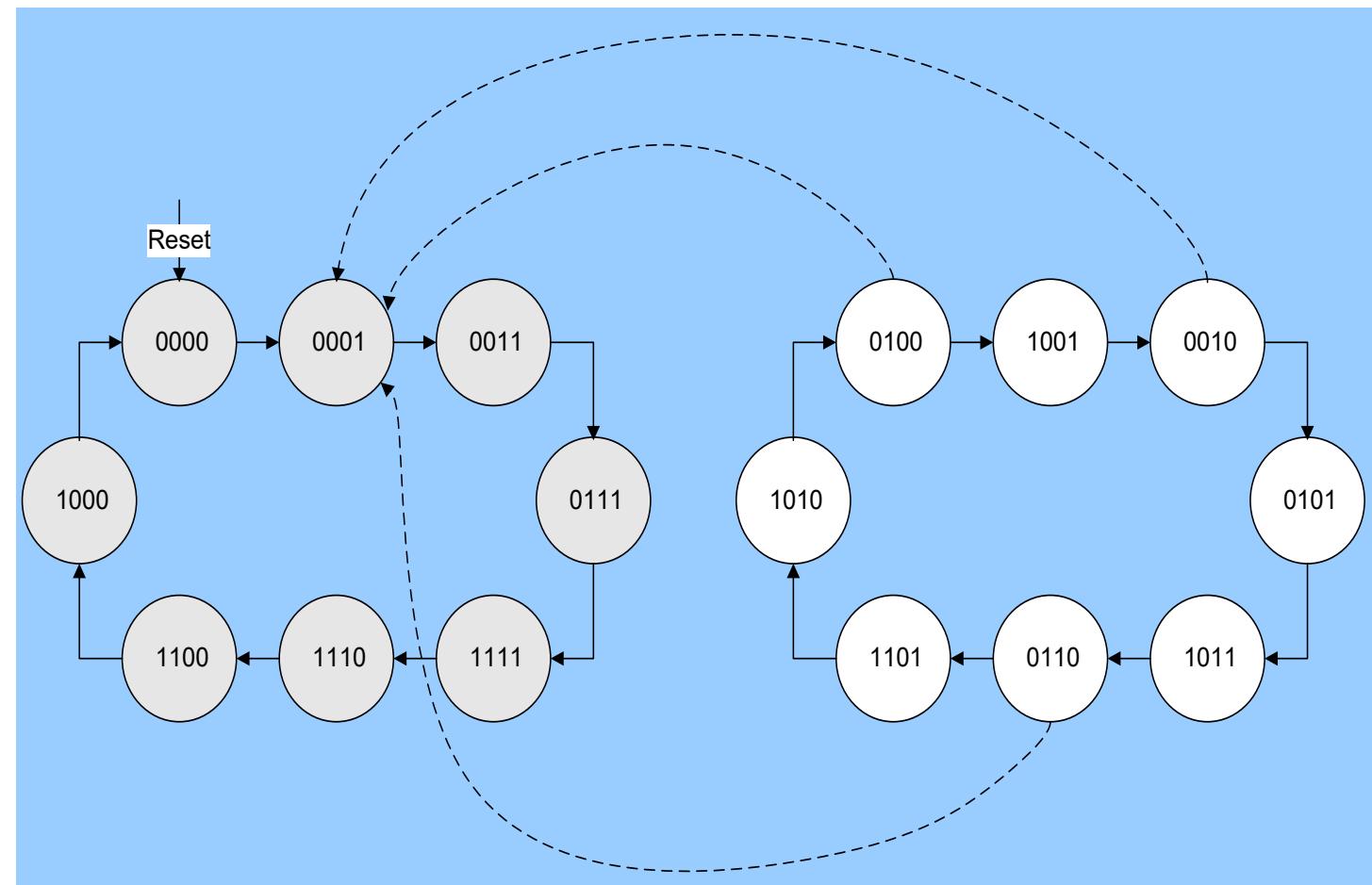


- Sechs völlig voneinander unabhängige geschlossene Zyklen!
  - Nur ein Zyklus enthält genau ein gesetztes Bit.
  - Für vier Zustände werden vier DFF benötigt → recht ineffiziente Nutzung der Flipflops!
- Da geht mehr....

# Johnson-Zähler

- Mit einem Inverter in der Rückführungsleitung kann im Vergleich zum einfachen Ringzähler die doppelte Anzahl von Zuständen gebildet werden:

- Im Normalfall existiert genau ein Zustand **0XX0**. Im gestörten Ablauf gibt es davon mehrere.
- Der Zustand **0XX0** kann als Ladebedingung im Korrekturschaltnetz verwendet werden.
- Die Rückkehr in den normalen Zyklus (Zustand **0001**) erfolgt nach spätestens drei Taktzyklen (gestrichelte Linien).



# VHDL-Modell des Johnson-Zählers mit Korrekturschaltnetz

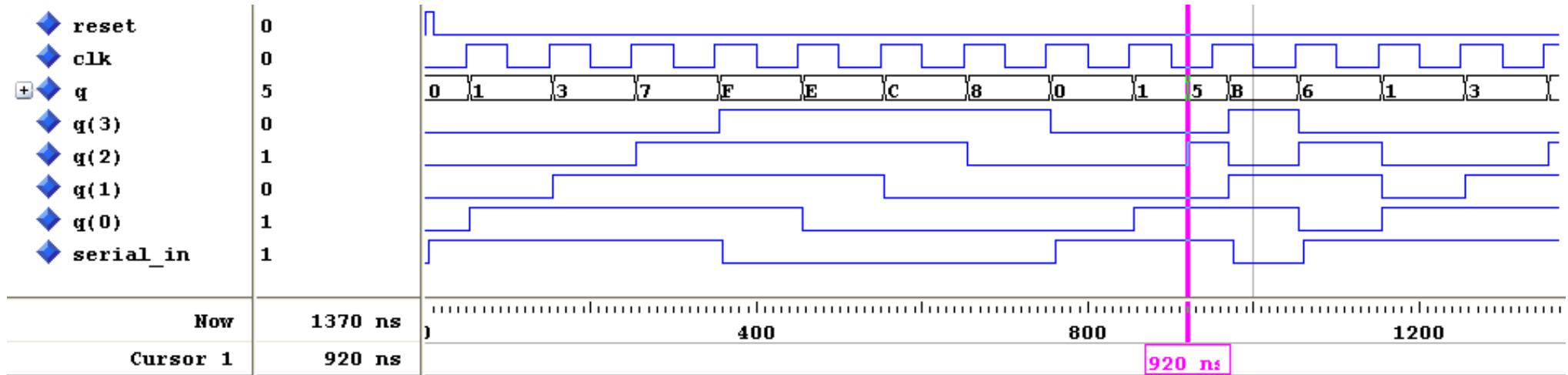
```
entity JOHNSON_CTR is
    generic(N : natural :=4);                      -- Anzahl der Bits, voreingestellt: 4
    port( CLK, RESET : in bit; Q : out bit_vector(N-1 downto 0) );
end JOHNSON_CTR;
architecture VERHALTEN of JOHNSON_CTR is
signal QINT: bit_vector(N-1 downto 0);
signal D: bit_vector(N-1 downto 0);
signal SERIAL_FEEDBACK: bit;
begin
D(N-1 downto 1) <= (others=>'0'); D(0) <= '1';
P1: process(CLK, RESET)
begin
    if RESET = '1' then
        QINT <= (others=>'0') after 5 ns;
    elsif CLK='1' and CLK'event then
        if (QINT(N-1)='0' and QINT(0)='0') then
            QINT <= D after 5 ns;
        else
            QINT <= (QINT(N-2 downto 0) & SERIAL_FEEDBACK) after 5 ns;
        end if;
    end if;
end process P1;
SERIAL_FEEDBACK <= not QINT(N-1) after 5 ns;
Q <= QINT;
end VERHALTEN;
```

Selbstkorrekturschaltnetz  
mit Korrekturbedingung  
0XX0

Initialisierungsmuster  
0001 wird synchron  
geladen

invertierende Rückkopplung

# Simulation eines 4-Bit-Johnson-Zählers



- Der asynchrone RESET bei  $t = 0$  löscht alle Bits.
- Bis  $t = 900$  ns erfolgt ein normaler zyklischer Ablauf.
- Bei  $t = 920$  ns wird durch Simulator-Stimulus das fehlerhafte Bitmuster 0101 injiziert.
- Bei der dritten Taktflanke nach der Störung befindet sich der Zähler wieder im normalen Zyklus.

# Linear Rückgekoppelte Schieberegister

- In der Rückkopplung des SRG befindet sich (meist einfache) Boole'sche XOR-Logik (Linear Feedback Shift Register LFSR)
- Anwendungen von LFSRs:
  - Kanalcodierung: Sog. CRC-Codes (engl. cyclic redundancy check) werden eingesetzt, um Übertragungsfehler zu erkennen bzw. zu korrigieren. CRC-Decoder werden als LFSR aufgebaut.
  - Erzeugung von „zufälligen“ Testmustern bzw. künstliche Erzeugung eines Rauschsignals beim Test bzw. der Selbstdiagnose integrierter Schaltungen.
  - Kryptologie (Verschlüsselungstechnik): Qualität der mit LFSRs erzeugten Pseudozufallszahlen ist jedoch außerordentlich schlecht, sodass die Rückkopplung nur in Verbindung mit nichtlinearen Rückkopplungsfunktionen eingesetzt wird.

Von besonderem  
Interesse sind  
Pseudozufallsgenera-  
toren mit maximaler  
Zykluslänge  $2^N-1$ . Diese  
ergeben sich (u.a.) durch  
die nebenstehenden  
Rückkopplungsschalt-  
netze:

N	SERIAL_IN	Zykluslänge
2	$Q(1) \leftrightarrow Q(0)$	3
3	$Q(2) \leftrightarrow Q(1)$	7
4	$Q(3) \leftrightarrow Q(2)$	15
5	$Q(4) \leftrightarrow Q(2)$	31
8	$Q(7) \leftrightarrow Q(3) \leftrightarrow Q(2) \leftrightarrow Q(1)$	255
16	$Q(15) \leftrightarrow Q(13) \leftrightarrow Q(12) \leftrightarrow Q(10)$	65535
32	$Q(31) \leftrightarrow Q(30) \leftrightarrow Q(29) \leftrightarrow Q(9)$	$2^{32} - 1$

# VHDL-Modell eines 4-Bit-Pseudozufallszahlengenerators

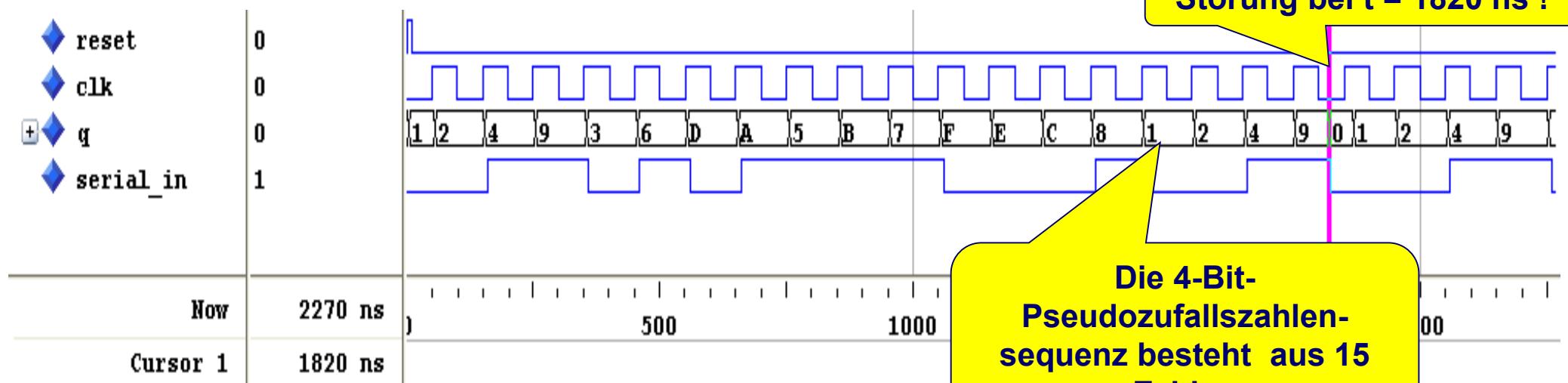
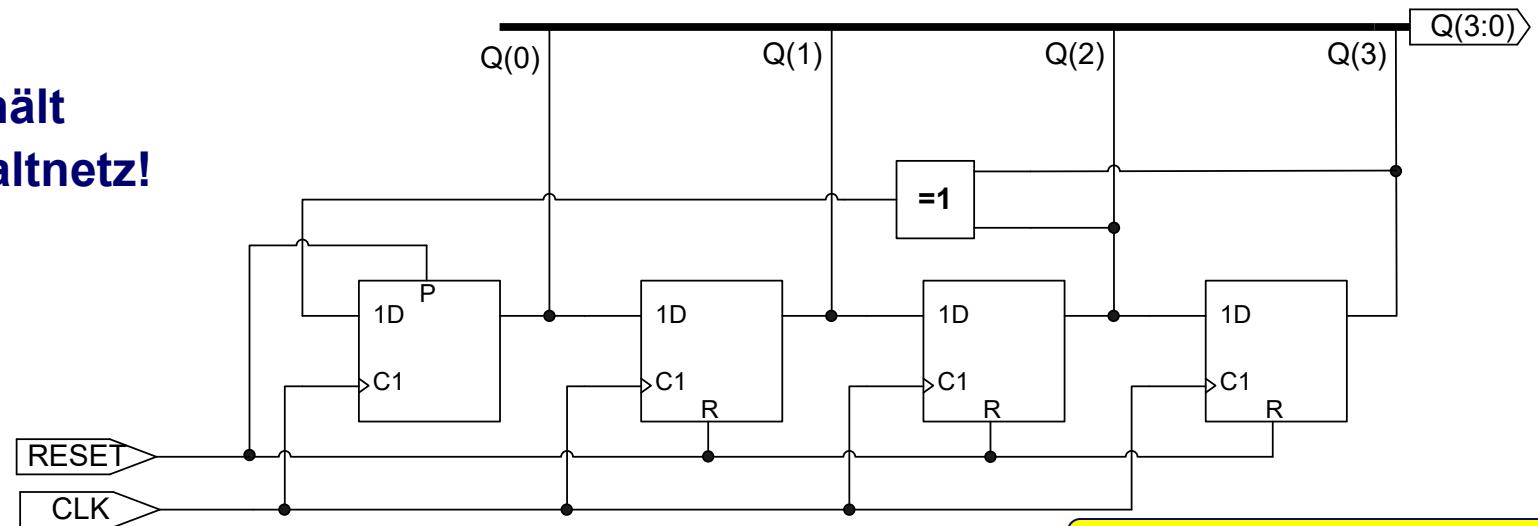
```
-- 4-Bit LFSR als Pseudozufallszahlengenerator
entity RAND_LFSR is
    port( CLK, RESET : in bit; Q : out bit_vector(3 downto 0));
end RAND_LFSR;
architecture VERHALTEN of RAND_LFSR is
signal QINT: bit_vector(3 downto 0);
signal SERIAL_FEEDBACK: bit;
begin
P1: process(CLK, RESET)
begin
    if RESET = '1' then
        QINT <= "0001" after 5 ns;
    elsif CLK='1' and CLK'event then
        if QINT = "0000" then
            QINT <= "0001" after 5 ns;
        else
            QINT <= (QINT(2 downto 0) & SERIAL_FEEDBACK) after 5 ns;
        end if;
    end if;
end process P1;
-- 4-Bit Rückkopplungsschaltnetz
SERIAL_FEEDBACK <= QINT(3) xor QINT(2) after 5 ns;
Q <= QINT;    -- Kopie als Ausgangssignal
end VERHALTEN;
```

Selbstkorrekturschaltnetz  
falls keine 1 geschoben  
werden kann.

Feedback-Schaltnetz für  
vier Bit

# Struktur und Simulation eines 4-Bit-LFSRs

Die Schaltung enthält  
kein Korrekturschaltnetz!



# Synchronisation Digitaler Systeme

- Kopplung von Signalen in zueinander synchronen Taktdomänen
  - Impulsverkürzung
  - Impulsverlängerung
- Synchronisierung Asynchroner Eingangssignale
  - Synchronisierung langer Eingangsimpulse
  - Synchronisierung kurzer Eingangsimpulse
  - Asynchrone Resets
- Datenaustausch zwischen Teilsystemen
  - Synchrone Datenübertragung
  - Asynchrone Datenübertragung



Prof. Dr. J. Reichardt  
Lehrbuch Digitaltechnik  
Eine Einführung mit VHDL, 3. A  
Oldenbourg Wissenschaftsverlag  
München 2013  
ISBN 978-3-486-72765-4

# Synchronisation Digitaler Systeme

In einem digitalen System werden evtl. unterschiedliche Schaltungsblöcke mit unterschiedlichem Takt betrieben. Unterscheide:

- Synchronisation zweier Taktdomänen mit gemeinsamem Grundtakt.
- Synchronisation völlig asynchroner Taktdomänen.

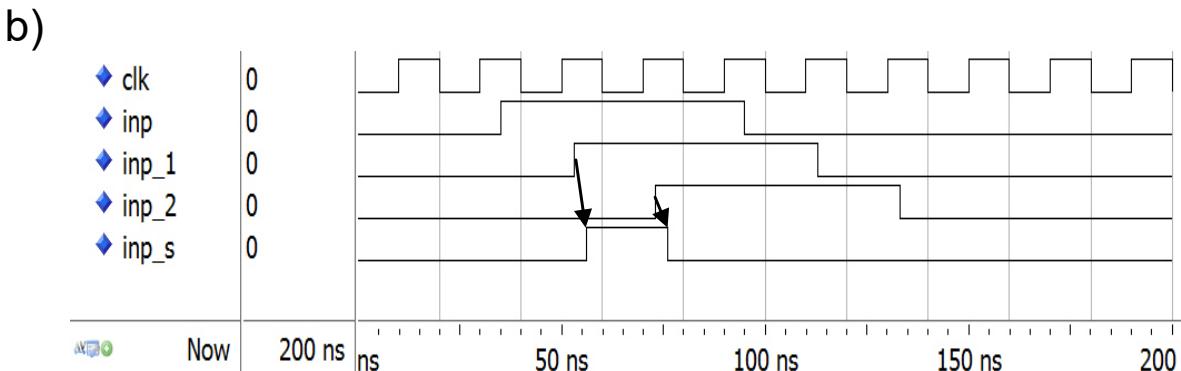
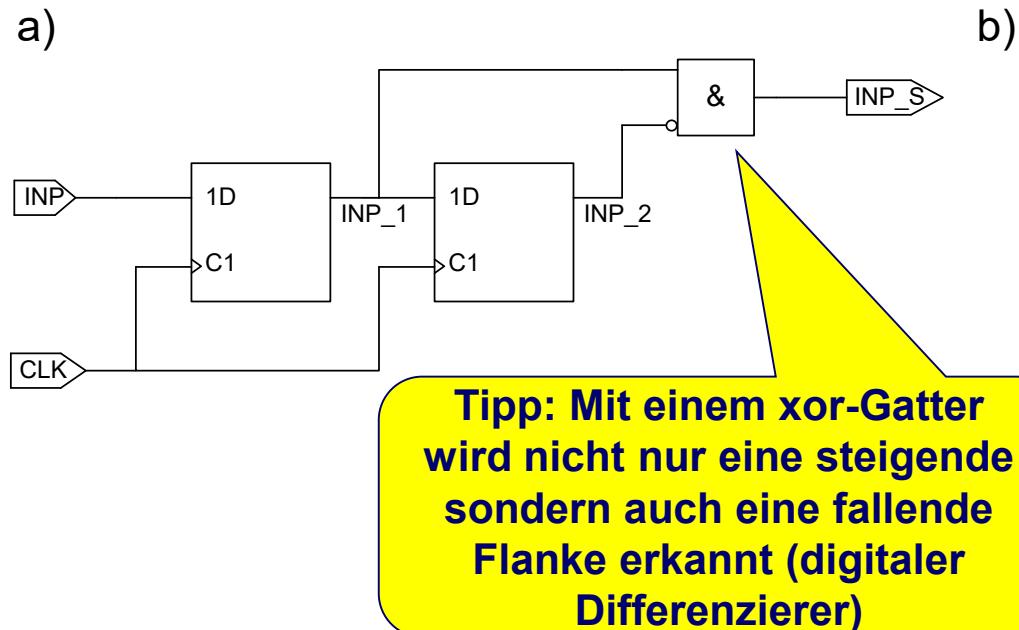
## Inhalt:

- Schaltungen zur Verkürzung von Eingangsimpulsen.
- Schaltungen zur Verlängerung von Eingangsimpulsen.
- Metastabilität.
- Konfiguration von Resets in CPLDs und FPGAs
- Datenaustausch zwischen zueinander synchronen oder asynchronen Teilsystemen mit Hilfe einer geeigneten Datenflussteuerung.

# Impulsverkürzungsschaltung (1)

## Schaltungsprinzip:

- Das Eingangssignal INP der langsameren Taktdomäne wird mit dem schnelleren Systemtakt „abgetastet“ und in eine 2-Bit Schieberegister-Pipeline geleitet. Der ältere Abtastwert befindet sich rechts. Ein UND-Gatter dekodiert eine alte 0 und einen neuen 1 am Eingang (Signalwechsel: steigende Flanke)
- Alternativer Name der Schaltung: **Flankendetektor**



# Impulsverkürzungsschaltung (2)

```
-- Pulse Shorter: Übertrage Impuls von langsamer in schnelle Taktdomäne
entity SHORTER is
port(  CLK, INP : in bit;
       INP_S: out bit);
end SHORTER;

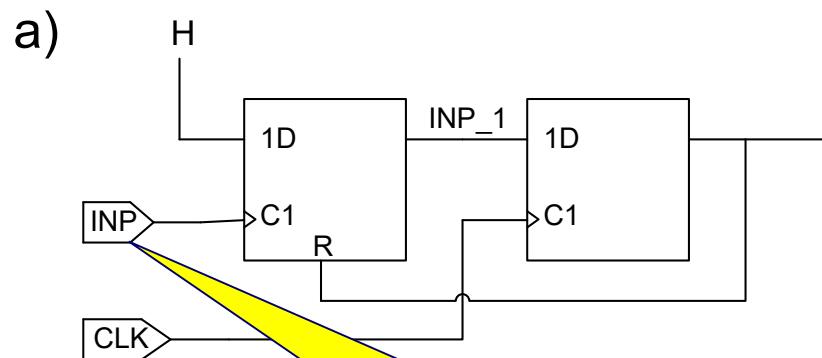
architecture VERHALTEN of SHORTER is
signal INP_1, INP_2: bit;
begin
PULSE_SHORTER: process(CLK)      -- 2 D-FFs
begin
  if CLK='1' and CLK'event then
    INP_1 <= INP after 3 ns;
    INP_2 <= INP_1 after 3 ns;
  end if;
end process PULSE_SHORTER;
INP_S <= INP_1 and (not INP_2) after 3 ns; -- verkürzter Impuls
end VERHALTEN;
```

Zweistufiges  
Pipelineregister

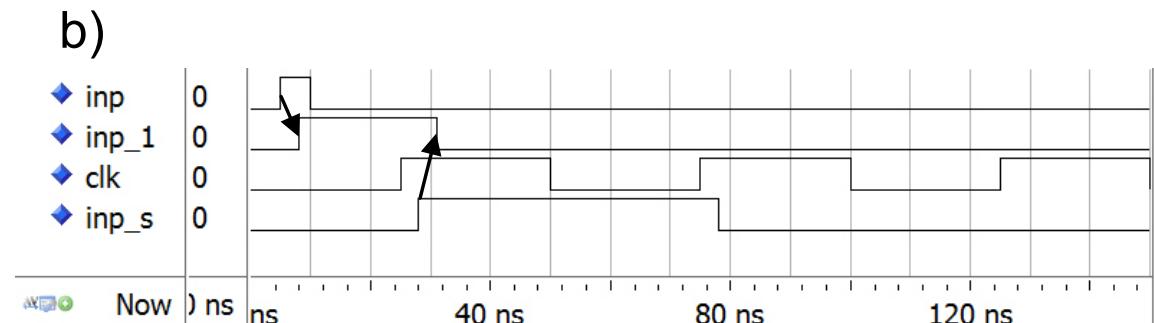
# Impulsverlängerungsschaltung (1)

## Schaltungsprinzip:

- Die steigende Flanke des schnellen Eingangssignals INP taktet eine 1 in das erste Flipflop INP\_1.
  - Das zweite Flipflop dient der Synchronisation auf die langsamere Taktdomäne.
  - Das synchronisierte Signal INP\_S setzt das Eingangsflipflop zurück.
  - Erst danach kann am Eingang eine weitere Flanke detektiert werden.



**beachte, dass hier  
am Takteingang das  
INP-Signal und nicht  
der Takt liegt !**



# Impulsverlängerungsschaltung (2)

```
entity STRETCHER is
port( CLK, INP : in bit);
end STRETCHER;

architecture VERHALTEN of STRETCHER is
signal INP_1, INP_S: bit;
begin
STRETCHER: process(INP, INP_S)
begin
  if INP_S = '1' then -- asynchroner Reset
    INP_1 <= '0' after 3 ns;
  elsif INP='1' and INP'event then
    INP_1 <= '1' after 3 ns;
  end if;
end process STRETCHER;

SYNCHRONIZER: process(CLK)
begin
  if CLK='1' and CLK'event then
    INP_S <= INP_1 after 3 ns;
  end if;
end process SYNCHRONIZER;
end VERHALTEN;
```

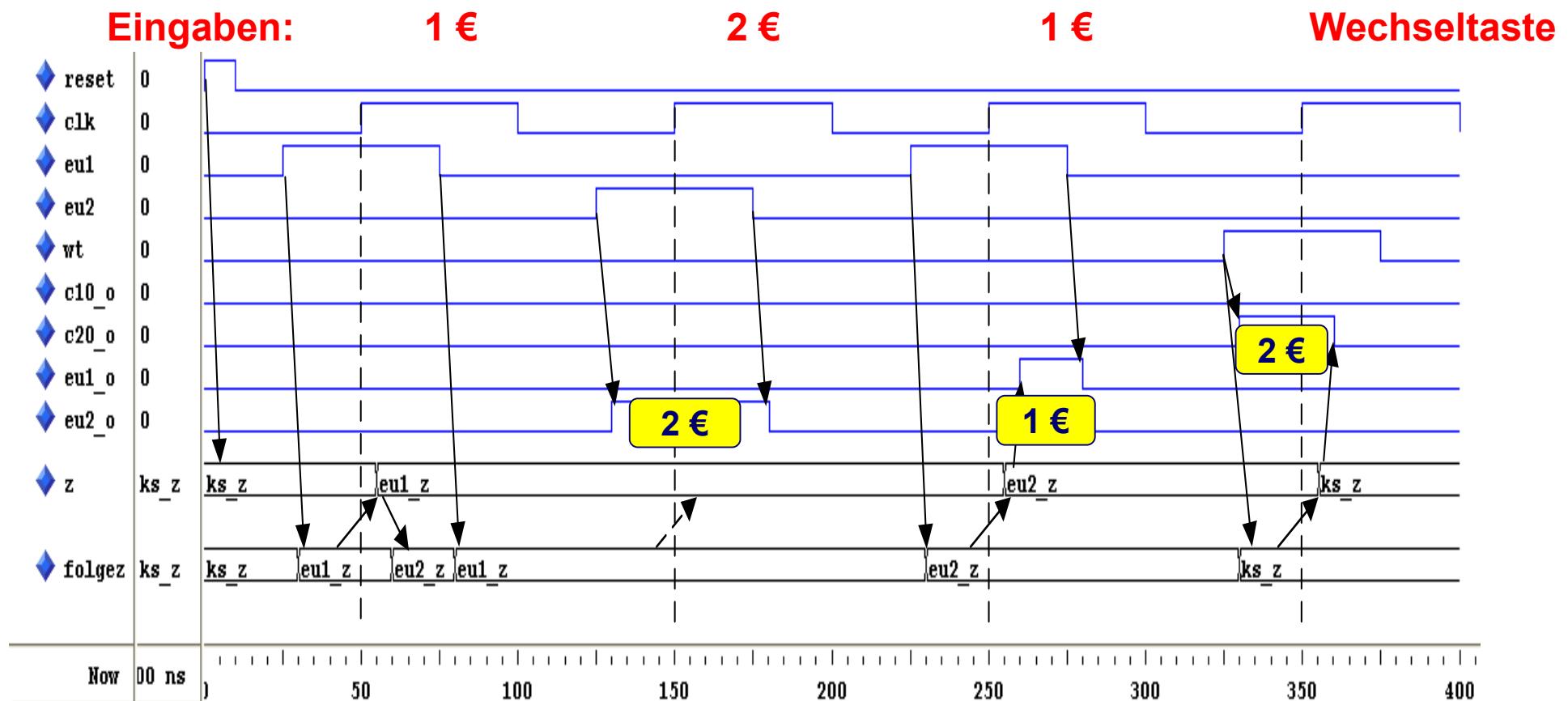
Asynchrones Rücksetzen des  
Stretcher-DFFs durch das  
synchronisierte Signal

Am Takteingangs des  
Stretcher-DFFs liegt  
das Eingangssignal !

# Unsynchronisierte Mealy Geldwechselautomat (GWA)

Das Problem des Mealy-Modells des Geldwechselautomaten aus Kap. 12 sind die nicht synchronisierten Ein- und Ausgangssignale:

- Nichtsynchronisierte Eingangssignale
- Zu kurze Ausgangssignale (typisch für ein Mealy-Modell)



# Synchronisierter Mealy Geldwechselautomat (1)

In der Praxis werden derartige Schaltungen mit hoher Taktfrequenz des Systemtakts CLK betrieben, die Eingangssignale der Sensoren sind dagegen erheblich langsamer und überdecken mehrere Taktperioden des Systemtakts. Die Ausgangssignale müssen für langsam arbeitende Aktoren, die mit dem Takt CLK\_SYNC arbeiten, verlängert werden.

## Lösungskonzept:

- Process PULSE\_SHORTER synchronisiert Signale EU1\_A, EU2\_A, WT\_A ein  
→ EU1, EU2, WT
- 4 Stretcher DFFs entdecken die Flanken von C10\_O, C20\_O, EU1\_O und EU2\_O am Ausgang  
→ C10\_O\_1, C20\_O\_2, EU1\_O\_2, EU2\_O\_2  
1 Register synchronisiert alle 4 Signale zum Ausgangstakt  
→ C10\_S, C20\_S, EU1\_S, EU2\_S

# Synchronisierter Mealy Geldwechselautomat (2)

Es werden hier nur die zusätzlichen Prozesse dargestellt, die die GWA-Schaltung ergänzen.

```
...
PULSE_SHORTER: process(CLK) -- Impulsverk. für drei Eingangssignale EU1_A, EU2_A, WT_A
begin
    if RESET = '1' then
        EU1_1 <= '0' after 1 ns; EU1_2 <= '0' after 1 ns;
        EU2_1 <= '0' after 1 ns; EU2_2 <= '0' after 1 ns;
        WT_1 <= '0' after 1 ns; WT_2 <= '0' after 1 ns;
    elsif CLK ='1' and CLK'event then
        EU1_1 <= EU1_A after 1 ns;
        EU1_2 <= EU1_1 after 1 ns;
        EU2_1 <= EU2_A after 1 ns;
        EU2_2 <= EU2_1 after 1 ns;
        WT_1 <= WT_A after 1 ns;
        WT_2 <= WT_1 after 1 ns;
    end if;
end process PULSE_SHORTER;
EU1 <= EU1_1 and not EU1_2 after 1 ns;
EU2 <= EU2_1 and not EU2_2 after 1 ns;
WT <= WT_1 and not WT_2 after 1 ns;
...
```

Ein gemeinsamer Prozess für alle mit dem Systemtakt betriebenen Signale

Auswertung der jeweils steigenden Flanken an den Eingangssignalen

# Synchronisierter Mealy Geldwechselautomat (3)

```
...
C10_STRETCHER: process(C10_O, C10_O_2)
begin
    if C10_O_2 = '1' then
        C10_O_1 <= '0' after 2 ns;
    elsif C10_O='1' and C10_O'event then
        C10_O_1 <= '1' after 2 ns;
    end if;
end process C10_STRETCHER;
...
SYNC_OUT: process(CLK_SYNC, RESET) -- Ausgangssync. auf langsame Taktdomäne
begin
    if RESET='1' then
        C10_O_2 <= '0' after 2 ns;
        C20_O_2 <= '0' after 2 ns;
        EU1_O_2 <= '0' after 2 ns;
        EU2_O_2 <= '0' after 2 ns;
    elsif CLK_SYNC ='1' and CLK_SYNC'event then
        C10_O_2 <= C10_O_1 after 2 ns;
        C20_O_2 <= C20_O_1 after 2 ns;
        EU1_O_2 <= EU1_O_1 after 2 ns;
        EU2_O_2 <= EU2_O_1 after 2 ns;
    end if;
end process SYNC_OUT;
C10_O_S <= C10_O_2;
C20_O_S <= C20_O_2;
EU1_O_S <= EU1_O_2;
EU2_O_S <= EU2_O_2;
```

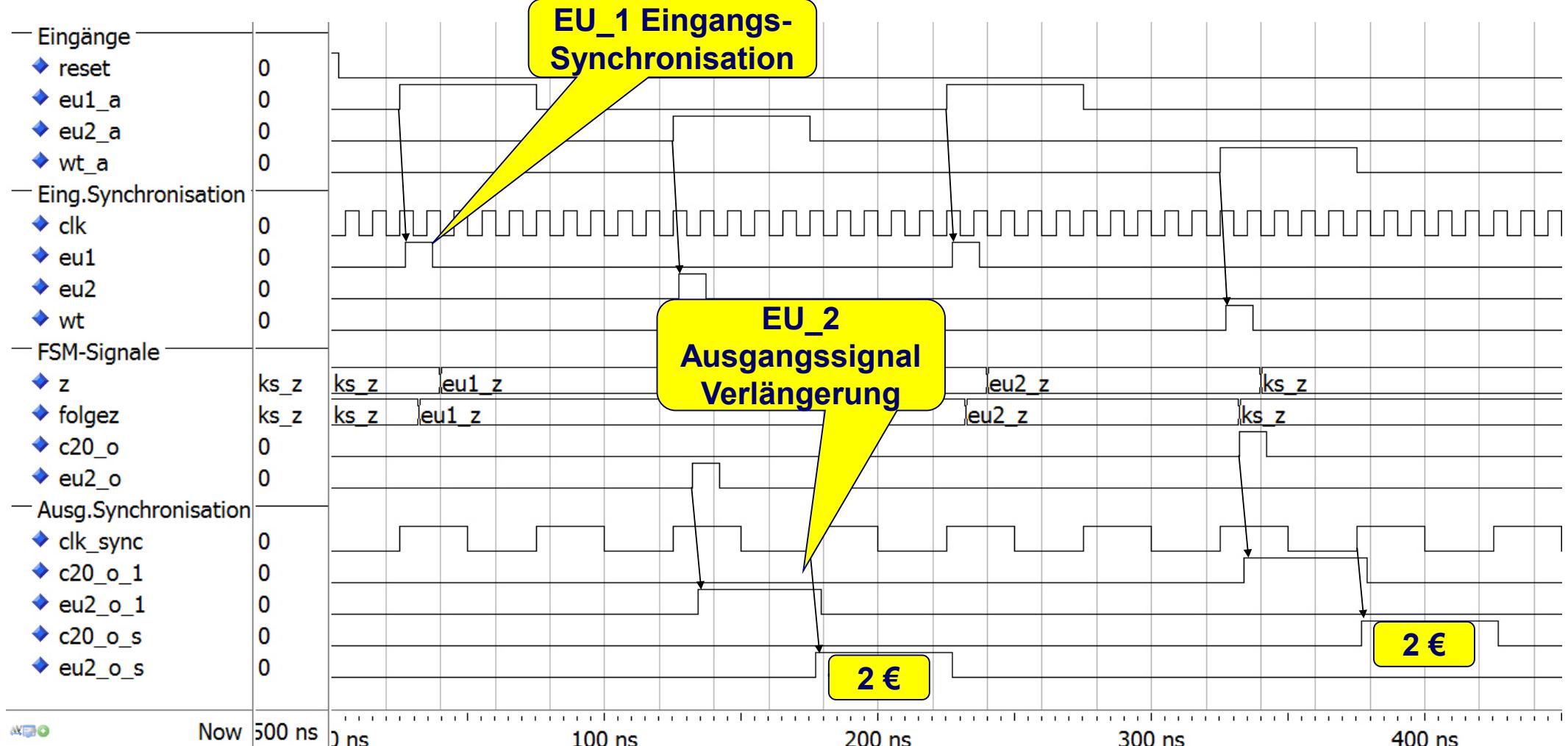
4 verschiedene  
Stretcher-Prozesse für  
die 4 Ausgangssignale.  
Rücksetzen durch das  
jeweilige sync. Signal.

Gemeinsame  
Synchronisation auf  
die langsame  
CLK\_SYNC  
Taktdomäne

Kopie an die Ausgänge  
zur Vermeidung von  
buffer Ports.

# Simulation des Synchronisierten GWA

Eingaben:      1 €      2 €      1 €      Wechseltaste

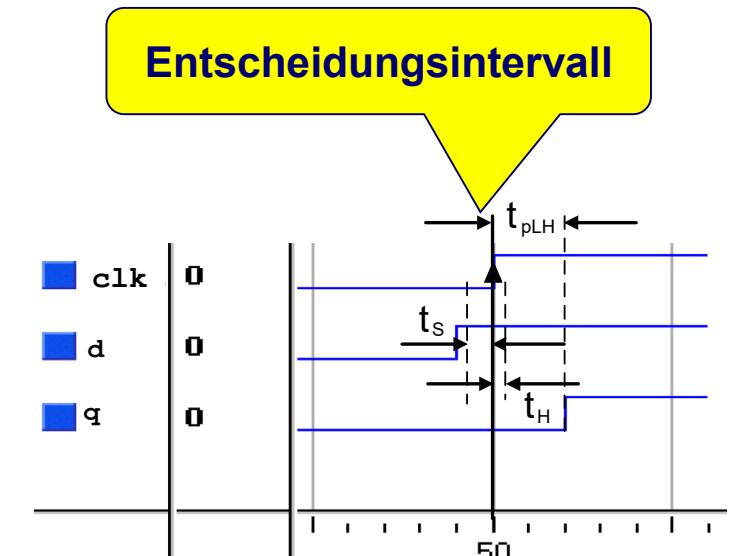


# Metastabilität durch Asynchrone Eingangssignale

Wenn Eingangssignale asynchron zum Takt erzeugt werden, kann sich z.B. der Folgezustand eines Automaten während des Entscheidungsintervalls des Zustandsflipflops ändern. Dies führt zu einer unbedingt zu vermeidenden Metastabilität des Zustandsflipflops!

## Exemplarische Beispiele für das Auftreten von Metastabilität:

- Wenn ein Daten-, Status- bzw. Steuersignal in einer Taktdomäne A gebildet wird und in einer anderen, unabhängigen Taktdomäne B ausgewertet werden muss.
- Wenn ein Eingangssignal völlig unabhängig vom Systemtakt eingelesen werden muss. Dies ist z.B. der Fall bei einem Interrupt in einem Mikroprozessorsystem.
- Wenn ein externer asynchroner Reset eine sequenzielle Schaltung mit vielen Flipflops zurücksetzen soll.



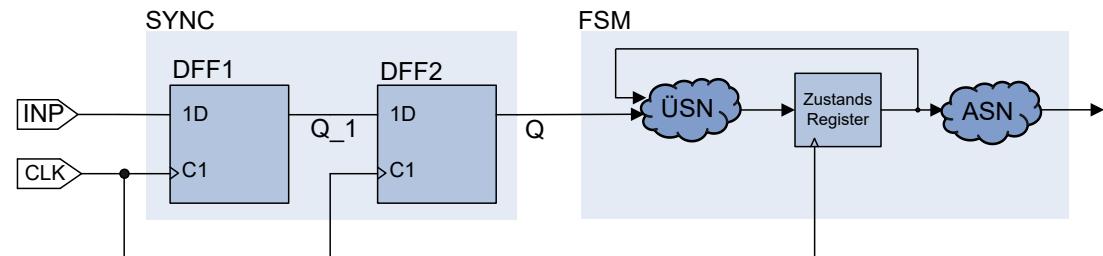
Bei der Timing-Simulation mit ModelSim können metastabile Zustände evtl. erkannt werden!

# Synchronisation Langer Async. Eingangsimpulse

Langer Eingangsimpuls: Länger als eine Taktperiodendauer

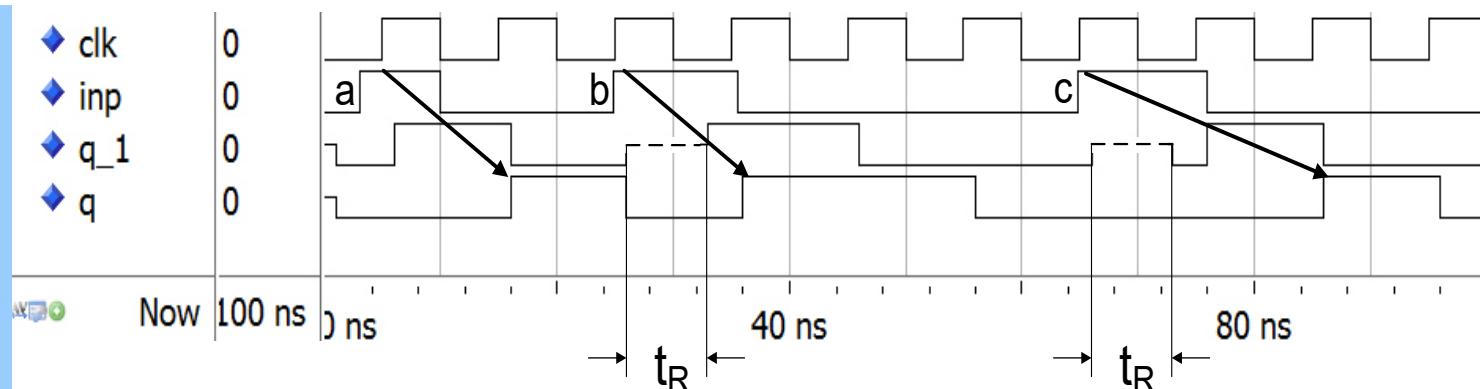
## Prinzip der Schutzschaltung SYNC:

- DFF1 wird unvermeidbar metastabil !
- Wenn die Taktperiodendauer länger als die **Erholungszeit  $t_R$**  des DFF1 ist, so wird in DFF2 ein definierter Wert (**0 oder ! 1**) eingelesen!



## VHDL-Modell der Schutzschaltung

```
SYNC: process(CLK, RESET)
begin
  if RESET = '1' then
    Q_1 <= '0' after 1 ns;
    Q <= '0' after 1 ns;
  elsif CLK='1' and CLK'event then
    Q_1 <= INP after 1 ns;
    Q <= Q_1 after 1 ns;
  end if;
end process REG;
```



- INP ändert sich außerhalb des Entscheidungsintervalls → keine Metastabilität
- INP ändert sich im Entscheidungsintervall → Der metastabile Zustand von DFF1 geht nach **1** → DFF2 übernimmt bei nachfolgender Flanke eine 1
- INP ändert sich im Entscheidungsintervall → Der metastabile Zustand von DFF1 geht nach **0** → DFF2 übernimmt erst bei der nächsten Flanke die am Eingang längere als eine Taktperiode anliegende 1



# Wahrscheinlichkeit für das Auftreten von Metastabilität

Die aus zwei in Reihe geschalteten Flipflops bestehende Synchronisationsschaltung für asynchrone Signale erfordert, dass das Eingangssignal länger als einen Takt andauert. Es ist nicht vorherzusagen, ob die Schaltung das Eingangssignal nach einem oder nach zwei Takten quittiert. Die Schaltung garantiert zwar keine Fehlerfreiheit, kann aber die Auftretenswahrscheinlichkeit drastisch reduzieren.

Wahrscheinlichkeit für das Auftreten eines Fehlers: MTBF (mean time between failures)

$$MTBF(t_R) = \frac{e^{t_R/\tau}}{T_0 \cdot f_{clk} \cdot f_{Data}}$$

darin sind:

- $\tau$  und  $T_0$  technologiespezifische Konstanten der DFFs
- $f_{CLK}$  Taktfrequenz des DFFs
- $f_{Data}$  die (mittlere) Frequenz der asynchronen Eingangssignale

$$t_R = 1/f - T_{SU} \dots \text{Recovery time}$$

- MTBF steigt exponentiell mit der zur Verfügung stehenden Erholungszeit  $t_R$ .
- MTBF sinkt mit steigender Takt- und Datenfrequenz.

Die Wahrscheinlichkeit des Auftretens der Metastabilität.  
Zeitfenster um die Taktflanke, in der Metastabilität eintreten kann.

Indiziert die Wahrscheinlichkeit, dass die auftretende Metastabilität in der Zeitperiode von  $t_R$  abklingt.

# Wahrscheinlichkeit für das Auftreten von Metastabilität

Familie	$\tau$ [ns]	$T_0$ [s]	$T_{SU}$ [ns]
74LS74	1.5	$4.0 \cdot 10^{-1}$	20
74HC74	1.82	$1.5 \cdot 10^{-6}$	25
XC95-CPLD	0.17	$9.6 \cdot 10^{-18}$	10

Beispiel: Mikroprozessorsystem, das mit  $f_{Clk}=10$  MHz Taktfrequenz arbeitet.

Die Interruptrate beträgt  $f_{Data} = 10^5$ /s. Die Interrupts sollen mit einem 74LS74 DFF synchronisiert werden:

- Berechne die Erholungszeit  $t_r = 1/f_{Clk} - T_{SU} = 80$  ns
- Berechne die MTBF =  $3.6 \cdot 10^{11}$  s  $\approx 11\ 513.5$  Jahre !
- Nun berechne die MTBF für  $f_{Clk}=16$  MHz!  $\rightarrow 3.15$  Sekunden
- Wähle nun für die Synchronisation eine XC95-CPLD Hardware MTBF erneut!

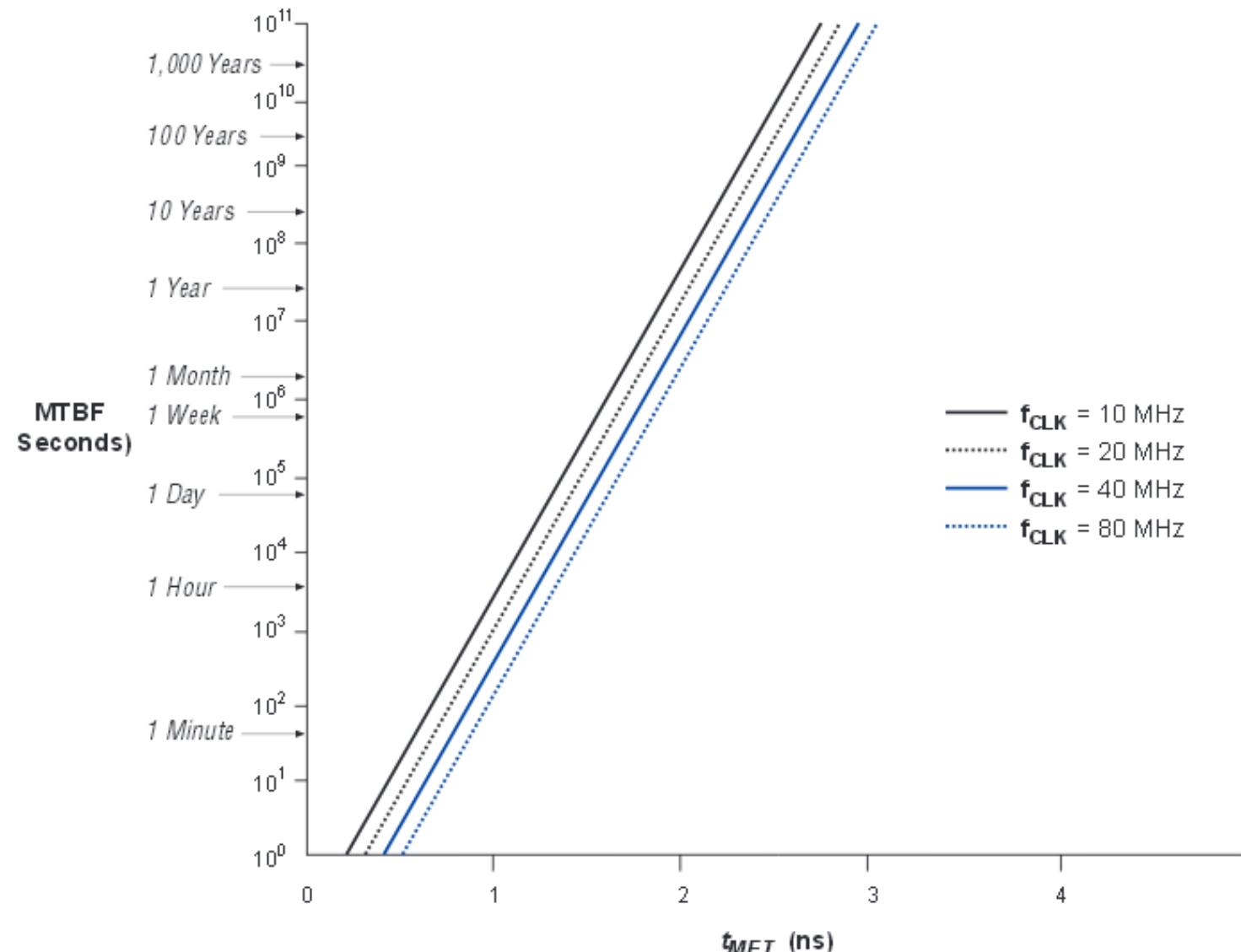
Welche Schlussfolgerungen müssen aus diesem Zahlenbeispiel gezogen werden?



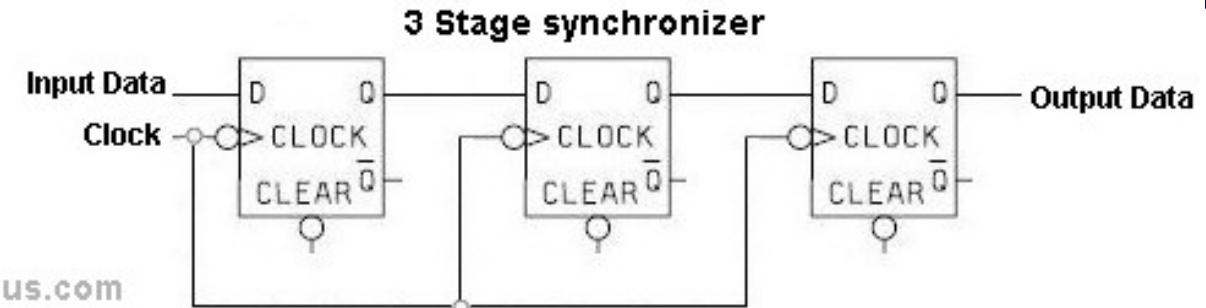
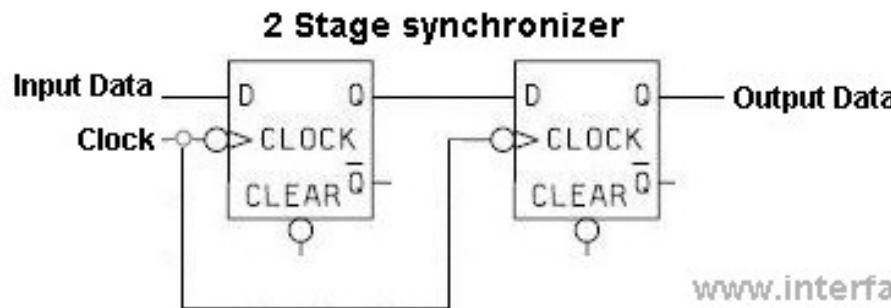
# Wahrscheinlichkeit für das Auftreten von Metastabilität

AN 42: Metastability in Altera Devices

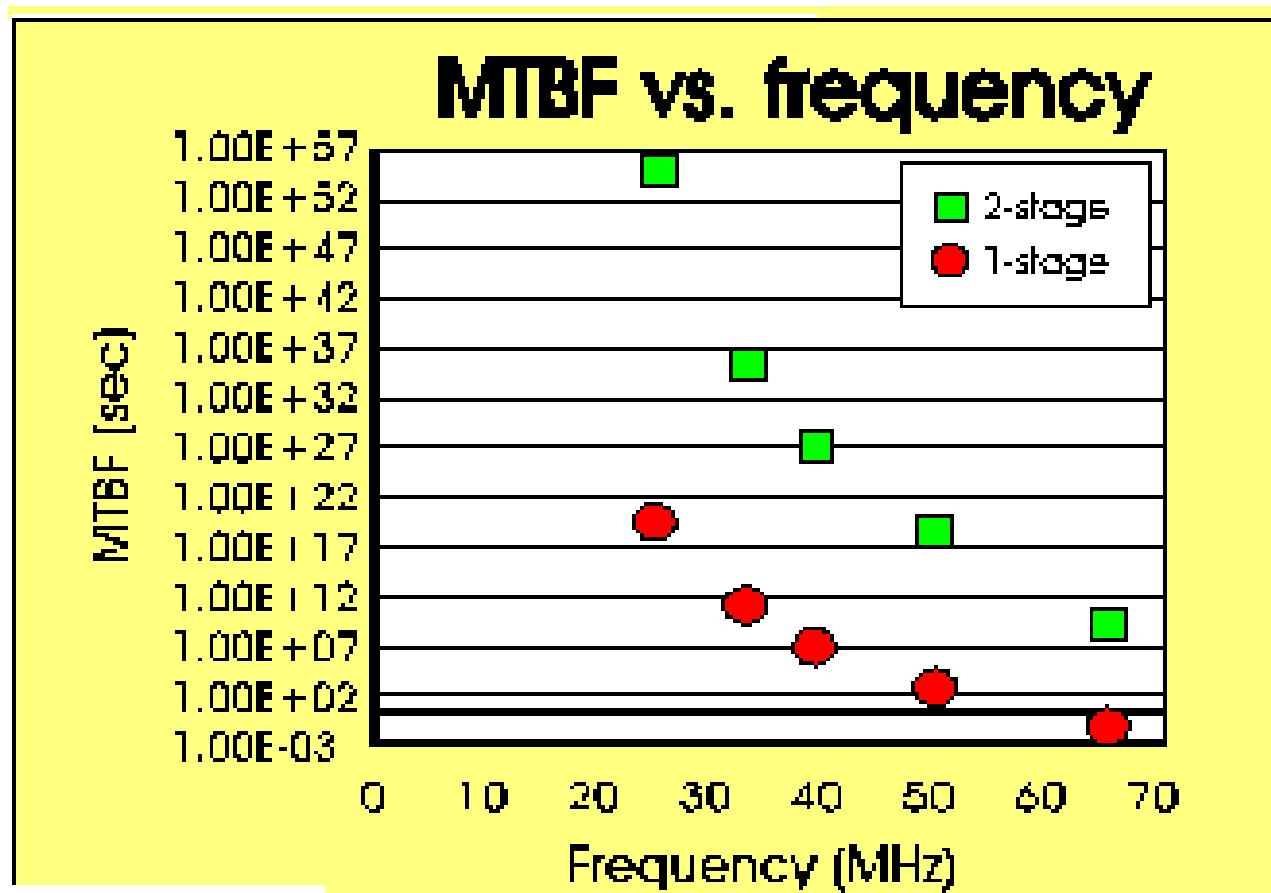
Figure 6. FLEX 10K, FLEX 8000 & FLEX 6000 MTBF Values



# Synchronizer mit 2 oder 3 Flip-Flops

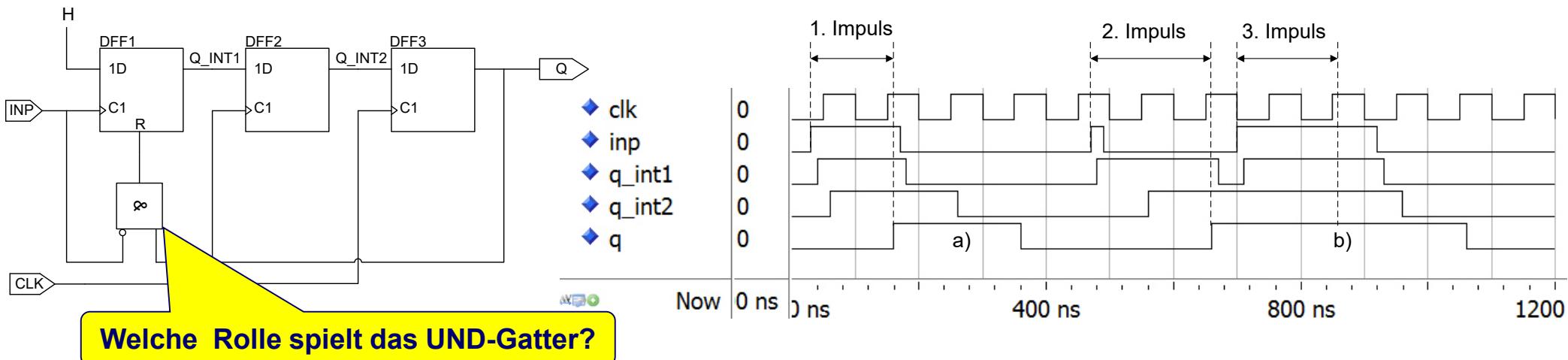


www.interfacebus.com



# Synchronisation Kurzer Eingangsimpulse

- Der vorhandenen Schutzschaltung muss ein „Stretcher-FF“ (DFF1) voran geschaltet werden.
- Das Rücksetzen des Stretcher-FFs erfolgt durch den synchronisierten Impuls



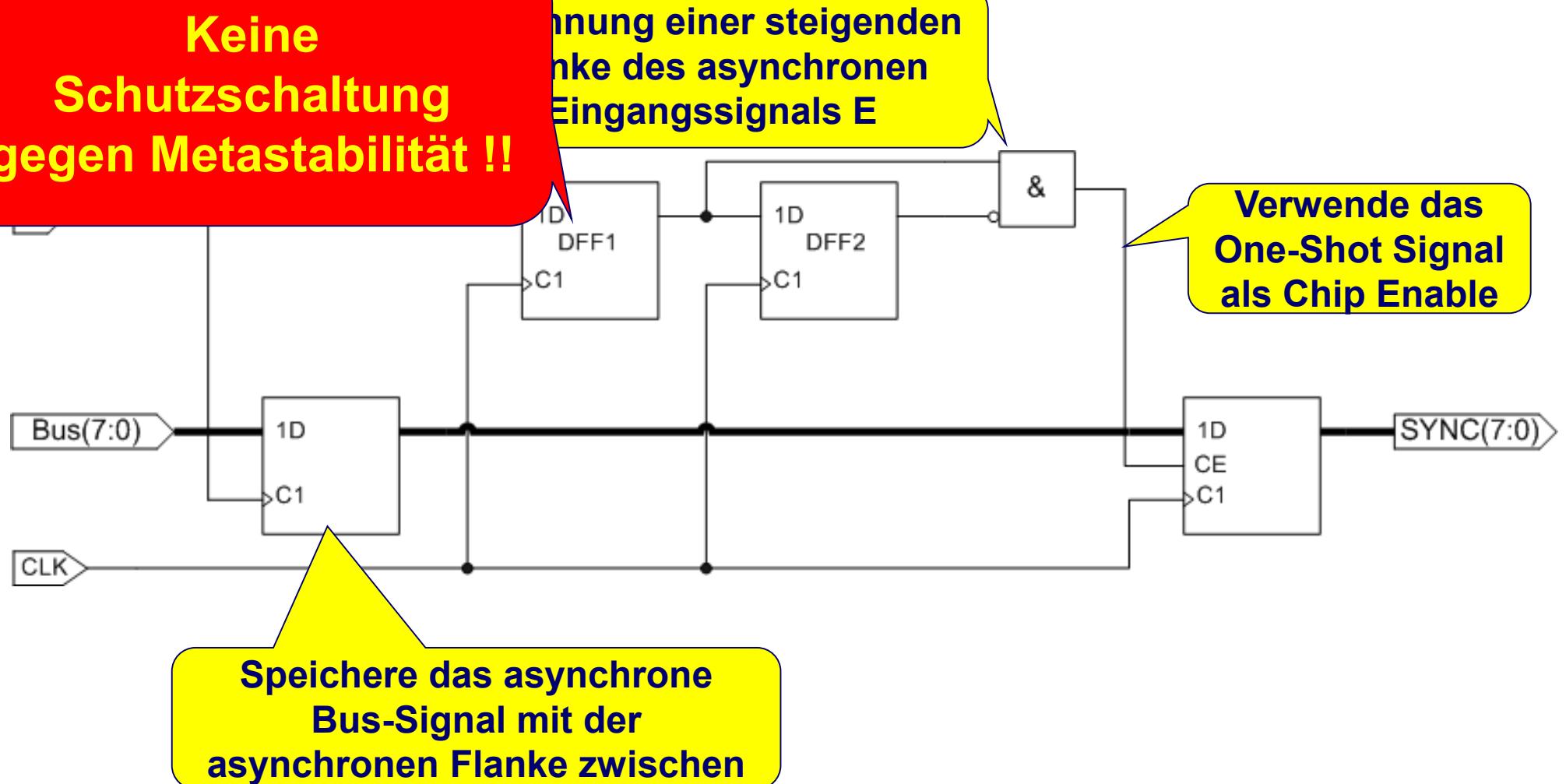
## Eigenschaften:

- Es können kurze (2. Impuls) aber auch lange Impulse (1. bzw. 3. Impuls) synchronisiert werden.
- Der zu einem Eingangsimpuls synchronisierte Ausgangsimpuls a) wird für die Dauer von zwei Taktzyklen gesetzt.
- Die Latenz des synchronisierten Ausgangssignals Q beträgt bis zu zwei Taktperioden.
- Wenn der Abstand der zu synchronisierenden Eingangssignale nicht mindestens drei Taktzyklen beträgt, so überlappen die Ausgangsimpulse. Im Bild erscheinen z.B. die Ausgangssignale der Eingangsimpulse 2 und 3 als ein Ausgangsimpuls b), der vier Taktzyklen lang ist.

# Erfassen Asynchroner Bus Signale (1)

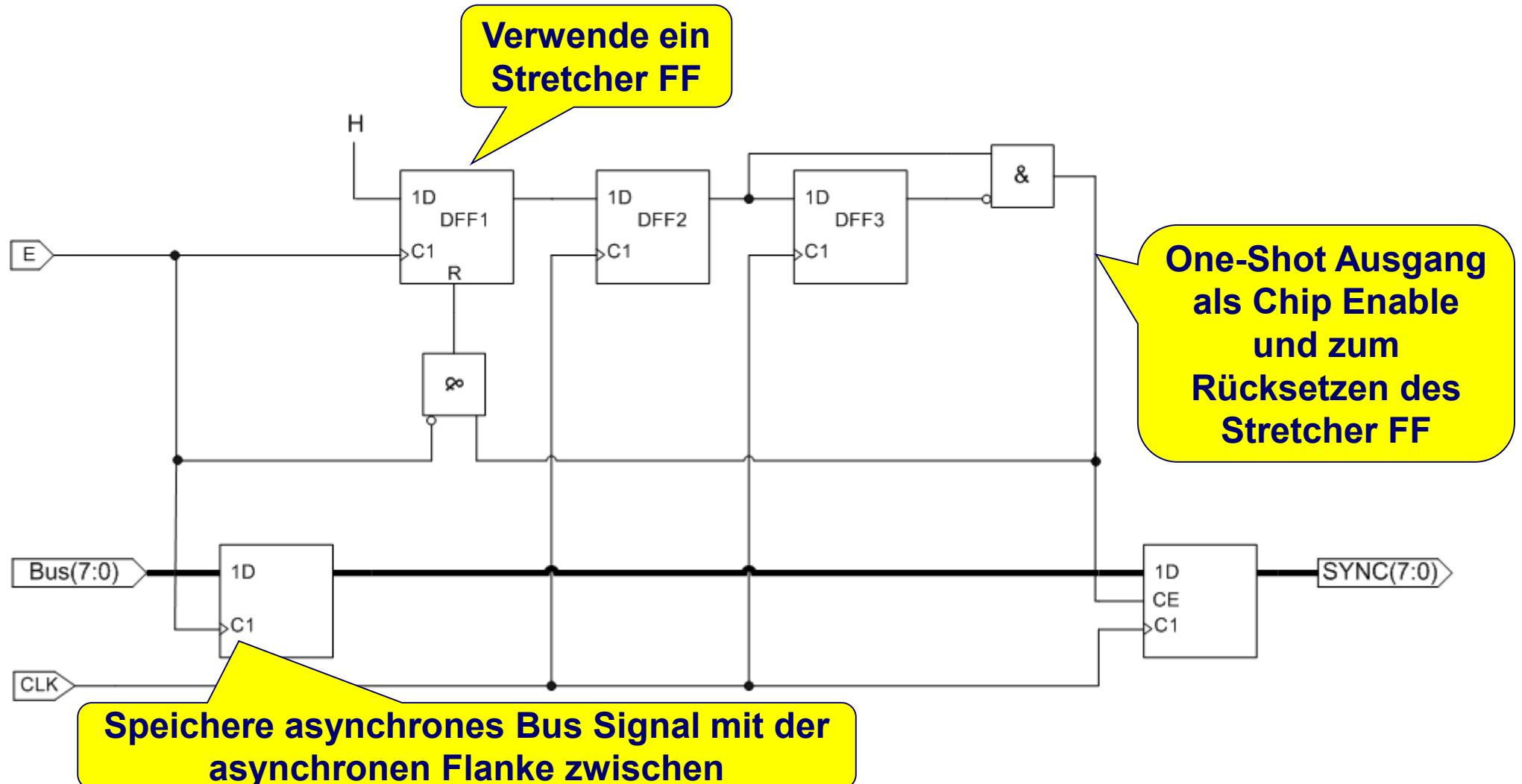
Zu verwenden, falls die Eingangsimpulse mindestens eine Taktperiode lang sind.

**Keine  
Schutzschaltung  
gegen Metastabilität !!**



# Erfassen Asynchroner Bus Signale (2)

Zu verwenden, falls die Eingangsimpulse kürzer als eine Taktperiode lang sind.



# Asynchrone Resets in CPLDs und FPGAs

## RESET-Konfigurationsmöglichkeiten in CPLDs und FPGAs:

- Power-ON Reset
- Asynchroner Reset durch eine Pegeländerung eines Eingangssignals (Pin-Reset)
- Synchroner Reset

### VHDL-Realisierung eines Pin-Resets

```
ASYNC: process(CLK, RESET)
begin
  if RESET = '1' then
    Q <= (others => '0');
  elsif CLK='1' and CLK'event then
    ...
  end if;
end process REG;
```

Bei gleichzeitigen Auftreten von Taktsignal und asynchronem Reset dominiert der Reset.  
Kritisch ist das Löschen des Rücksetzsignals :  
Dadurch kann das FF metastabil werden!

### VHDL-Realisierung eines Power-ON Resets im Deklarationsteil:

```
...
signal Q: bit_vector(7 downto 0) := (others => '0 ');
...
```

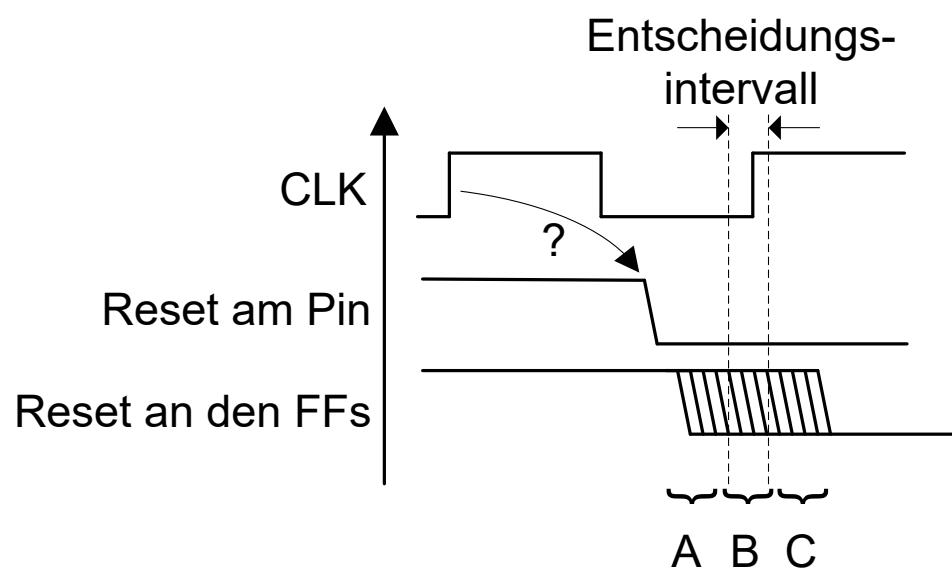
Der Power-ON Reset benötigt keine Verdrahtungsressourcen, er verwendet das ohnehin vorhandene GSR (Global Set/Reset - Netzwerk) im CPLD bzw. FPGA.

# Auftreten von Metastabilität beim Reset

In größeren Digitalschaltungen, deren Flipflops durch einen Pin zurück gesetzt werden, muss das Reset-Signal durch den FPGA geroutet werden. Das Reset-Signal erreicht nicht alle Flipflops gleichzeitig (Reset Verzug bzw. reset skew).

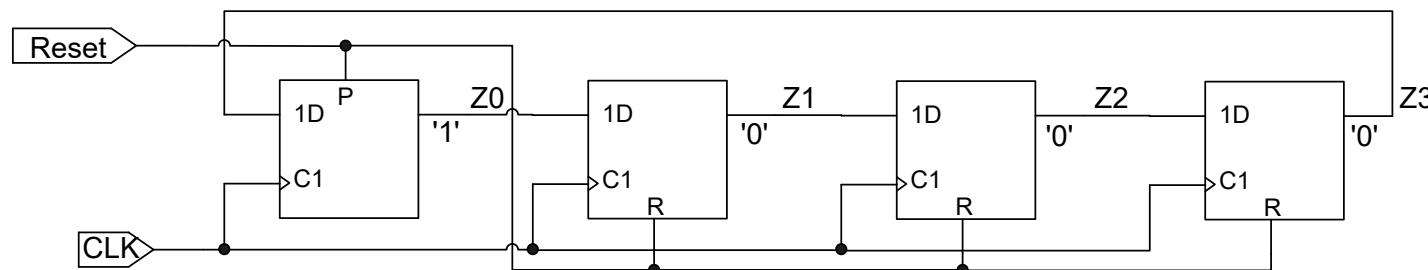
Zwischen dem Taktsignal und der Rücknahme des Resets gibt es keinen zeitlichen Zusammenhang. Der Reset am Pin wird „irgendwann“ gelöscht. Durch die Laufzeiten auf der Reset-Verdrahtung erreicht das gelöschte Signal die verschiedenen Flipflops zu unterschiedlichen Zeitpunkten.

- A: Die Flipflops können bereits bei der zweiten steigenden Taktflanke mit Daten geladen werden.
- B: Die Flipflops werden metastabil, es ist unklar, ob die Daten bei der 2. Taktflanke geladen werden können!
- C: Die Flipflops können erst bei der 3. Taktflanke mit Daten geladen werden!



# Beispiel: mod-4 Zähler

Der mod-4 Zähler wird als rückgekoppeltes 4-Bit Schieberegister aufgebaut, es soll immer exakt eine 1 im Schieberegister zyklisch kreisen (One-Hot Codierung).



Der Reset-Verzug kann zu einem Fehlverhalten des mod-4 Zählers führen!

## Diskussion des Schaltungsverhaltens:

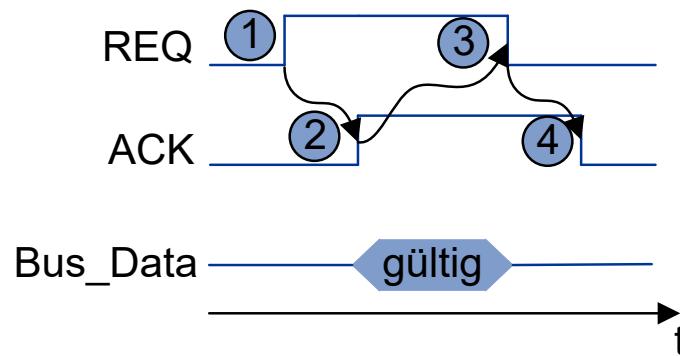
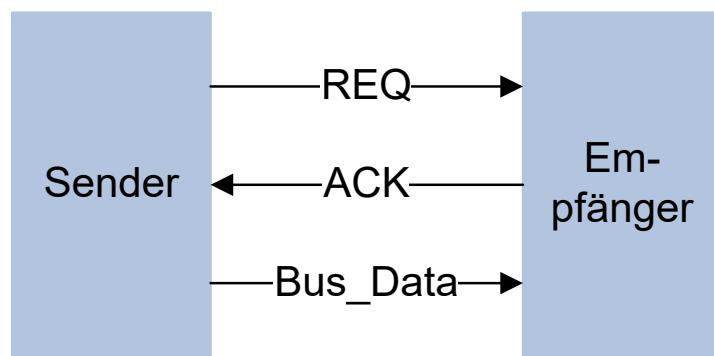
- Z0 wird durch den Reset gesetzt, alle anderen DFFs werden gelöscht.
- Annahme: Das gelöschte Reset-Signal erreicht DFF0 (Z0) eher, als das rechts daneben liegende DFF1 (Z1).
- Wenn kurz danach eine steigende Flanke an den DFFs anliegt, so wird die 1 aus Z0 nicht nach Z1 übernommen, wohl jedoch eine 0 von DFF3 nach DFF0.

# Der Vier-Phasen Handshake

Der Austausch von Daten zwischen unabhängig voneinander operierenden digitalen Teilsystemen erfordert eine Datenflussteuerung, die sicher stellt, dass keine Daten verloren gehen. Hierfür geeignet ist der Vier-Phasen Handshake mit den Steuersignalen REQ(uest) und ACK(nowledge).

Beschreibung der 4 Phasen:

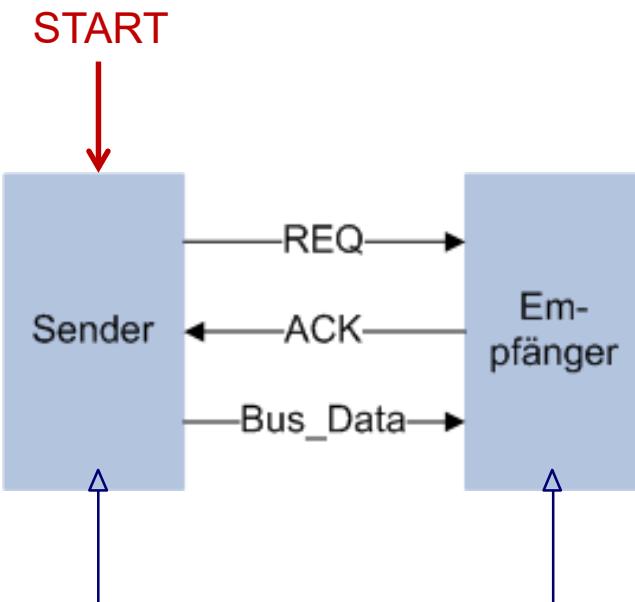
- ① Wenn der Sender Daten übertragen möchte, so setzt er das REQ-Signal.
- ② Der Empfänger liest das REQ-Signal und setzt, sofern er zum Datenempfang bereit ist, das ACK-Signal. Nach Empfang des ACK-Signals kann der Sender nun während  $\text{REQ} \wedge \text{ACK} = 1$  seine Daten übermitteln.
- ③ Nach Übertragung aller Daten nimmt der Sender das REQ-Signal zurück.
- ④ Der Empfänger quittiert diese Rücknahme, indem er seinerseits das ACK-Signal löscht. Der Sender ist nun für eine neue Datenübertragung bereit.



# Sync. Datenübertragung mit 4-Phasen Handshake

In dem VHDL-Modell soll auf einer Busleitung eine Folge von Pseudozufallsdaten vom Sender zum Empfänger übertragen werden.

- Realisierung beider entities
- Die Datenübertragung einer einzelnen Pseudozufallszahl wird durch ein externes Signal **START** der Testbench initiiert



```
-- Verhaltensmodell von Sender und Empfänger beim 4-Phasen Handshake
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-----
entity SENDER is
  port (CLK, START: in bit;
        ACK: in bit;
        REQ: out bit;
        BUS_DATA: out unsigned(3 downto 0)
      );
end SENDER;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity EMPFAENGER is
  port (CLK, REQ : in bit;
        BUS_DATA: in unsigned(3 downto 0);
        ACK: out bit
      );
end EMPFAENGER;
```

# Architektur des Senders

```
architecture VERHALTEN of SENDER is
signal DATA: unsigned(3 downto 0) := x"D"; -- Start einer Pseudozufallsfolge
signal REQ_int: bit;
begin
REQ_int <= START;
REQ <= REQ_int;

DATA_P: process(CLK) -- Erzeugung der Daten mit LFSR
begin
    if CLK='1' and CLK'event then
        if (REQ_int = '1' and ACK = '1') then
            DATA <= DATA(2 downto 0) & (DATA(3) xor DATA(2)) after 2 ns;
        end if;
    end if;
end process DATA_P;

DATASEND_P: process(REQ_int, ACK, DATA) -- Handshake
begin
    BUS_DATA <= (others=>'Z');
    if (REQ_int='1' and ACK='1') then
        BUS_DATA <= DATA after 2 ns;
    end if;
end process DATASEND_P;
end VERHALTEN;
```

Das externe START Signal einer TB initiiert die Übertragung

Getakteter Prozess zur Erzeugung der 4-Bit Pseudozufallszahlen bei jedem Request

Kombinatorischer Handshake Prozess

andernfalls wird das Bussignal hochohmig

Übertragungsbedingung: Die aktuelle Pseudozufallszahl wird auf den Bus gelegt.

# Architektur des Empfängers

```
architecture VERHALTEN of EMPFAENGER is
signal DATA: unsigned(3 downto 0);
signal ACK_int: bit;
begin

ACK <= ACK_int;
ACK_P: process(CLK) -- Handshake
begin
    if CLK='1' and CLK'event then
        if REQ = '1' then ACK_int <= '1' after 2 ns;
        else ACK_int <= '0' after 2 ns;
        end if;
    end if;
end process ACK_P;

DATAREC_P: process(CLK) -- Einlesen der Daten
begin
    if CLK='1' and CLK'event then
        if (REQ='1' and ACK_int='1') then
            DATA <= BUS_DATA after 2 ns;
        end if;
    end if;
end process DATAREC_P;
end VERHALTEN;
```

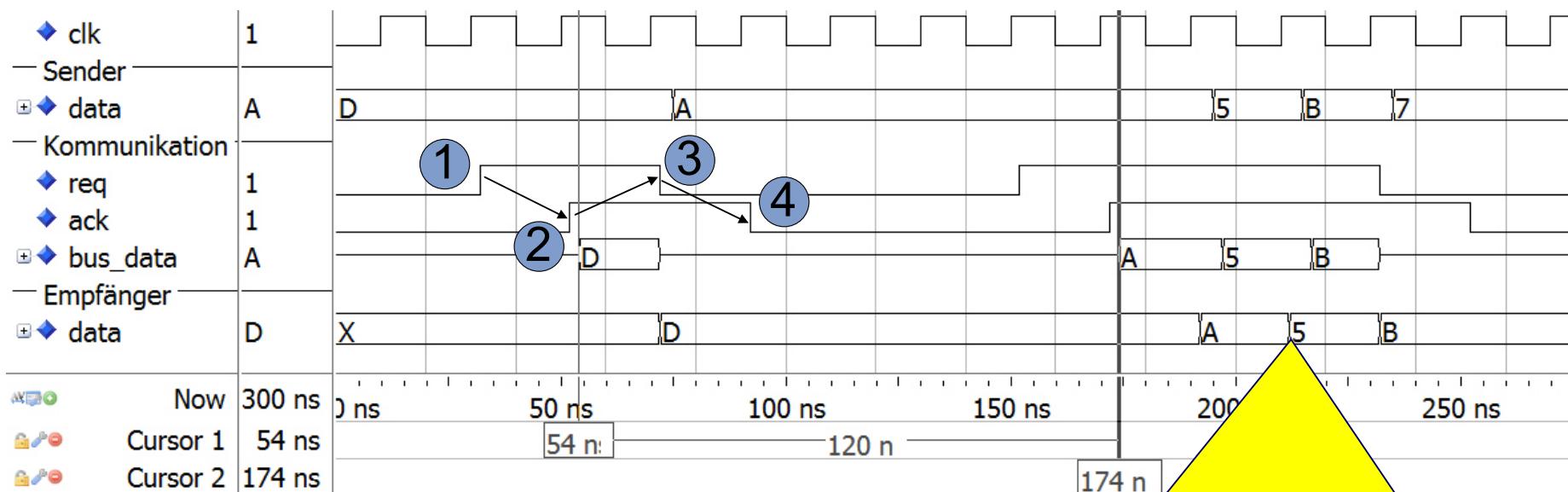
Taktsynchrones Auswerten des REQ-Signals im Empfänger , Setzen bzw. Löschen des ACK-Signals

Taktsynchrones Einlesen der Daten, falls Übertragungsbedingung  $REQ \wedge ACK$  erfüllt

# Simulation des Vier-Phasen Handshakes

Initiierung der Datenübertragung durch START Signal der Testbench:

- bei  $t = 30 \text{ ns}$  soll die erste Zufallszahl  $0xD$  übertragen werden  
Das REQ-Signal muss für 2 Takte gesetzt sein, die Dauer des Handshakes beträgt 3 Takte
- bei  $t = 150 \text{ ns}$  soll ein Burst von drei Zufallszahlen übertragen werden  
Das REQ-Signal muss für 4 Takte gesetzt sein, die Dauer des Handshakes beträgt 5 Takte



Für den Auf- und Abbau einer Datenverbindung wird beim 4-Phasen Handshake jeweils ein zusätzlicher Takt benötigt. Daher ist die effektive Datenrate bei Burst-Übertragungen höher als bei Einzelübertragungen!

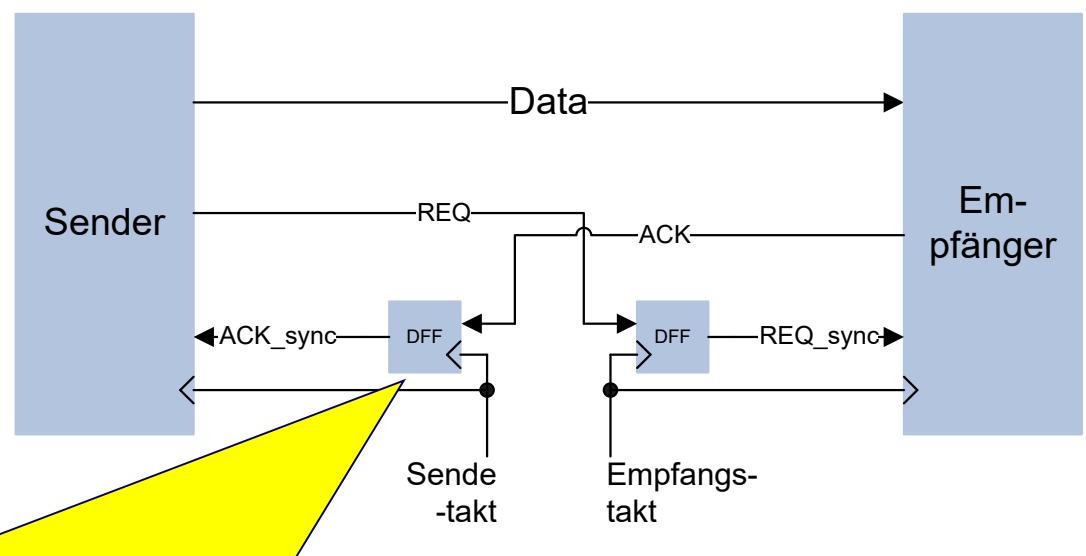
# Async. Datenübertragung mit 4-Phasen Handshake

Wenn Sender und Empfänger in unterschiedlichen Taktdomänen arbeiten, so stellt dies eine asynchrone Datenübertragung dar, es müssen Schutzmaßnahmen gegen metastabile Zustände eingeführt werden!

- Realisierung durch zwei kommunizierende Zustandsautomaten SENDER und EMPFÄNGER
- Zusätzliche Synchronisationsflipflops für die REQ- und ACK-Signale sind erforderlich

## Problem:

- Beim synchronen 4-Phasen Handshake werden bereits 2 zusätzliche Takte für den Auf- und Abbau der Verbindung benötigt.
- Eine Synchronisation mit jeweils 2 DFFs für die Handshakesignale würde die Datenbandbreite signifikant reduzieren.
- Wählte daher hier als Kompromiss eine einfachere Synchronisation mit nur einem DFF !



Das in diesem Konzept benötigte zweite DFF befindet sich hier hinter dem Übergangsschaltnetz innerhalb des Sender- bzw. Empfängerautomaten. Die zusätzliche Laufzeit  $t_{ÜSN}$  lässt sich durch einen statischen Timing-Analyse bestimmen bzw. durch ein Timing-Constraint in der UCF-Datei auch vorgeben !

Zur Verfügung stehende Erholungszeit:  
 $t_R = f_{CLK}^{-1} - t_{Setup} - t_{ÜSN}$

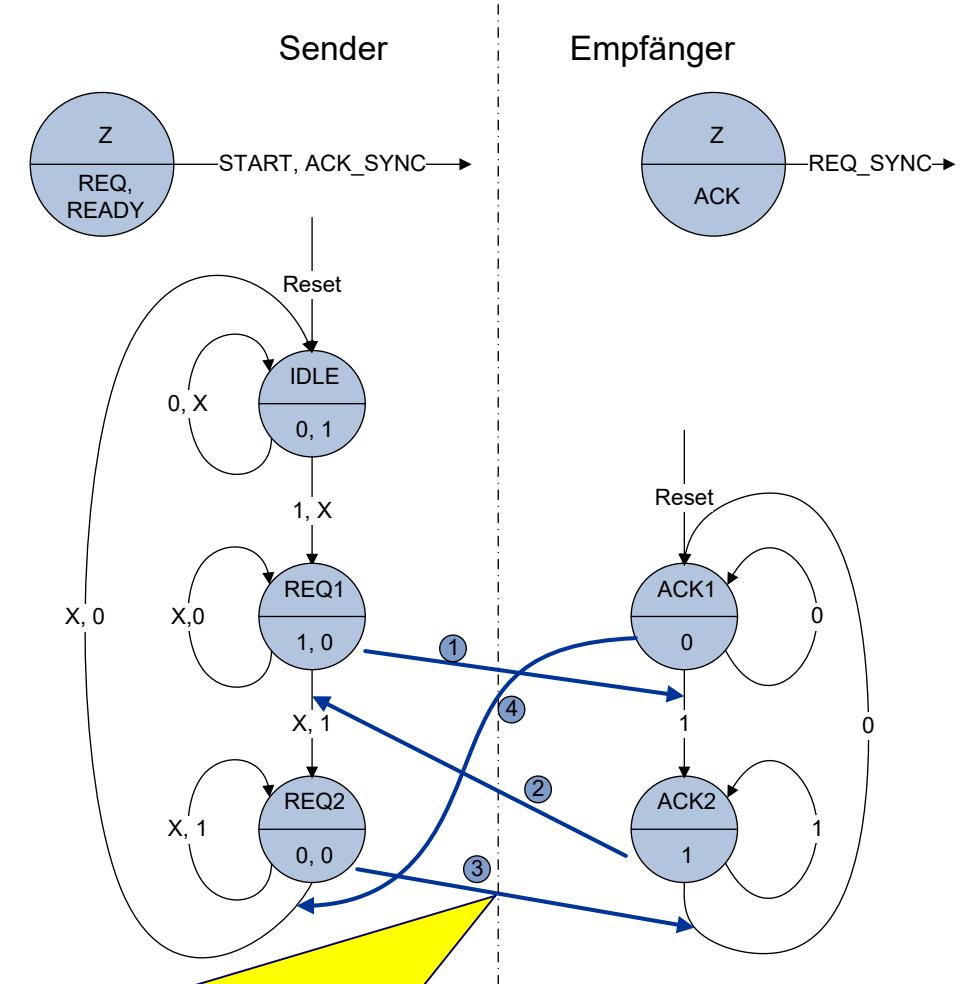
# Kommunizierende Zustandsautomaten

## Sender:

- Wartet im Zustand IDLE auf START=1, das REQ-Signal ist 0 das READY Signal ist 1. Warte auf START=1.
- Bei START=1 geht der Automat nach REQ1, REQ wird gesetzt. Warte auf ACK\_SYNC=1.
- Wenn das synchronisierte ACK\_SYNC=1 wird, geht der Automat nach REQ2, das REQ-Signal wird gelöscht. Warte auf ACK\_SYNC=0.
- Wenn in REQ2 das synchronisierte ACK\_SYNC Signal 0 ist, geht der Automat zurück in den IDLE-Zustand.

## Empfänger:

- Wartet im Zustand ACK1 auf das synchronisierte REQ\_SYNC-Signal.
- Bei REQ\_SYNC=1 geht der Automat nach ACK2, das ACK-Signal wird gesetzt.
- Wenn das synchronisierte REQ\_SYNC-Signal gelöscht wird geht der Automat zurück nach ACK1 und wartet dort auf eine neue Datenübertragung.



Beide Handshakesignale müssen jeweils in die aufnehmende Taktdomäne synchronisiert werden !

# VHDL Modell für den Senderautomaten (1)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-----
entity A_SENDER is
    port (CLK, RESET, START: in bit;
          ACK_SYNC: in bit;
          REQ, READY: out bit;
          BUS_DATA: out unsigned(3 downto 0)
        );
end A_SENDER;
architecture VERHALTEN of A_SENDER is
    signal DATA: unsigned(3 downto 0);
    signal REQ_int, READY_int: bit;
    type STATE_TYPE is (IDLE, REQ1, REQ2);
    signal STATE, NEXT_STATE: STATE_TYPE;
begin
DATA_P: process(CLK, RESET) -- Erzeugung der Daten
begin
    if RESET = '1' then
        DATA <= x"6" after 2 ns; -- Startwert
    elsif CLK='1' and CLK'event then
        if (START = '1' and READY_int = '1') then
            DATA <= DATA(2 downto 0) & (DATA(3) xor
                                         DATA(2)) after 2 ns;
        end if;
    end if;
end process DATA_P;
BUS_DATA <= DATA; ...
```

```
...
REG: process(CLK, RESET)
begin
    if RESET = '1' then
        STATE <= IDLE after 2 ns;
    elsif CLK = '1' and CLK'event then
        STATE <= NEXT_STATE after 2 ns;
    end if;
end process REG;
```

Zustandsregister des Senderautomaten

Erzeugung von Pseudozufallsdaten  
mit 4-Bit Links SRG.

# VHDL Modell für den Senderautomaten (2)

```
COMB: process(STATE , START, ACK_SYNC)
begin
    NEXT_STATE <= STATE after 2 ns; -- Default: verbleibe im Zustand
    REQ_int <= '0' after 2 ns;
    READY_int <= '0' after 2 ns;
    case STATE is
        when IDLE =>                      -- warte auf START
            READY_int <= '1' after 2 ns;
            if START = '1' then
                NEXT_STATE <= REQ1 after 2 ns;
            end if;
        when REQ1 =>                      -- warte auf ACK_SYN = 1
            REQ_int <= '1' after 2 ns;
            if ACK_SYNC = '1' then
                NEXT_STATE <= REQ2 after 2 ns;
            end if;
        when REQ2 =>                      -- warte auf ACK_SYN = 0
            if ACK_SYNC = '0' then
                NEXT_STATE <= IDLE after 2 ns;
            end if;
    end case;
end process COMB;
REQ <= REQ_int;
READY <= READY_int;
```

Übergangs- und Ausgangsschaltnetz des Senderautomaten.

Setze das REQ-Signal

# VHDL Modell für den Empfängerautomaten (1)

```
-- Asynchroner Empfänger für 4-Phasen Handshake
library ieee;
use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity A_EMPFAENGER is
  port (CLK, RESET: in bit;
        BUS_DATA: in unsigned(3 downto 0);
        REQ_SYNC: in bit;
        ACK: out bit;
        DATA: out unsigned(3 downto 0)
      );
end A_EMPFAENGER;

architecture VERHALTEN of A_EMPFAENGER is
  type STATE_TYPE is(ACK1, ACK2);
  signal ACK_int: bit;
  signal STATE, NEXT_STATE: STATE_TYPE;
begin
REG: process(CLK, RESET)
begin
  if RESET = '1' then
    STATE <= ACK1 after 2 ns;
  elsif CLK = '1' and CLK'event then
    STATE <= NEXT_STATE after 2 ns;
  end if;
end process REG;
```

Zustandsregister des Empfängerautomaten

# VHDL Modell für den Empfängerautomaten (2)

```
COMB: process(STATE , REQ_SYNC)
begin
    NEXT_STATE <= STATE after 2 ns;
    ACK_int <= '0' after 2 ns;
    case STATE is
        when ACK1 =>                      -- warte auf REQ_SYNC = 1
            if REQ_SYNC = '1' then
                NEXT_STATE <= ACK2 after 2 ns;
            end if;
        when ACK2 =>                      -- warte auf REQ_SYNC = 0
            ACK_int <= '1' after 2 ns;
            if REQ_SYNC = '0' then
                NEXT_STATE <= ACK1 after 2 ns;
            end if;
    end case;
end process COMB;
ACK <= ACK_int;

DATA_P: process(CLK, RESET) -- Empfang der Daten
begin
    if RESET ='1' then
        DATA <=x"0";
    elsif CLK='1' and CLK'event then
        if (REQ_SYNC = '1' and ACK_int = '1') then -- REQ1 und ACK2
            DATA <= BUS_DATA after 2 ns;
        end if;
    end if;
end process DATA_P;
end VERHALTEN;
```

Übergangs- und Ausgangsschaltnetz des Empfängerautomaten.

Setze das ACK-Signal

Falls Übertragungsbedingung erfüllt, so erfolgt das taktsynchrone Einlesen der Busdaten in den Empfänger

# VHDL Top-Level für Sender und Empfänger

```
architecture VERHALTEN of SENDER_EMPFAENGER_TOP is
signal BUS_DATA: unsigned(3 downto 0);
signal REQ, REQ_SYNC: bit;
signal ACK, ACK_SYNC: bit;
-- hier müssen die A_SENDER und A_EMPFAENGER Komponenten deklariert werden.
begin
ACK_SYNC_P: process(CLK1) -- einfaches Sync.FF!
begin
  if CLK1='1' and CLK1'event then
    ACK_SYNC <= ACK after 2 ns;
  end if;
end process ACK_SYNC_P;
SENDER: A_SENDER           -- Instanz der SENDER Komponente
port map(CLK1, RESET, START, ACK_SYNC,
         REQ, READY, BUS_DATA);

REQ_SYNC_P: process(CLK2) -- einfaches Sync.FF!
begin
  if CLK2='1' and CLK2'event then
    REQ_SYNC <= REQ after 2 ns;
  end if;
end process REQ_SYNC_P;

EMPFAENGER: A_EMPFAENGER -- Instanz der EMPFAENGER Komponente
port map(CLK2, RESET, BUS_DATA, REQ_SYNC, ACK, DATA);

end VERHALTEN;
```

The diagram illustrates the annotated VHDL code with four callout boxes:

- ACK Synchronisationsflipflop**: Points to the first process block for the ACK synchronization flip-flop.
- Sender Komponenteninstanziierung**: Points to the instantiation of the `A_SENDER` component.
- REQ Synchronisationsflipflop**: Points to the second process block for the REQ synchronization flip-flop.
- Empfänger Komponenteninstanziierung**: Points to the instantiation of the `A_EMPFAENGER` component.

# Testbench für 4-Phasen Kommunikation (1)

```
architecture TESTBENCH of SENDER_EMPFAENGER_TOP_TB is
signal DATA: unsigned(3 downto 0);
signal CLK1, CLK2, RESET, START, READY: bit;

component SENDER_EMPFAENGER_TOP is
port (CLK1, CLK2, RESET, START: in bit;
      READY: out bit;
      DATA: out unsigned(3 downto 0)
);
end component;
begin
RESET <='1', '0' after 7 ns;

CLOCK1_GEN: process
begin
  CLK1 <= '0'; wait for 4 ns; -- 125 MHz Schritt
  CLK1 <= '1'; wait for 4 ns;
end process CLOCK1_GEN;

CLOCK2_GEN: process
begin
  CLK2 <= '0'; wait for 10 ns; -- 50 MHz Schritt
  CLK2 <= '1'; wait for 10 ns;
end process CLOCK2_GEN;
...

```

**Toplevel Komponentendeklaration**

**Sender arbeitet mit 125 MHz Takt**

**Empfänger arbeitet mit 50 MHz Takt**

# Testbench für 4-Phasen Kommunikation (2)

Das Verhalten der Testbench ist zeitgesteuert:

- es wird eine Zeitvariable actual\_time definiert
- der Prozess zur Erzeugung ist taktgesteuert

```
START_P: process(CLK1)          -- START-Signal Erzeugung
variable actual_time: time; -- deklariere Zeitvariable
begin
    actual_time := now;           -- aktuelle Simulationszeit
    if actual_time < 10 ns then
        START <= '0';
    elsif actual_time < 250 ns then -- START-READY Handshake
        if CLK1='1' and CLK1'event then
            if READY ='1' then
                START <= '1' after 2 ns;
            else
                START <= '0' after 2 ns;
            end if;
        end if;
    else
        START <= '1';             -- ab jetzt START dauerhaft auf 1
    end if;
end process START_P;
```

Erzeugung des Startsignals durch  
START / READY Handshake

nach 250 ns wird das  
Startsignal dauerhaft 1

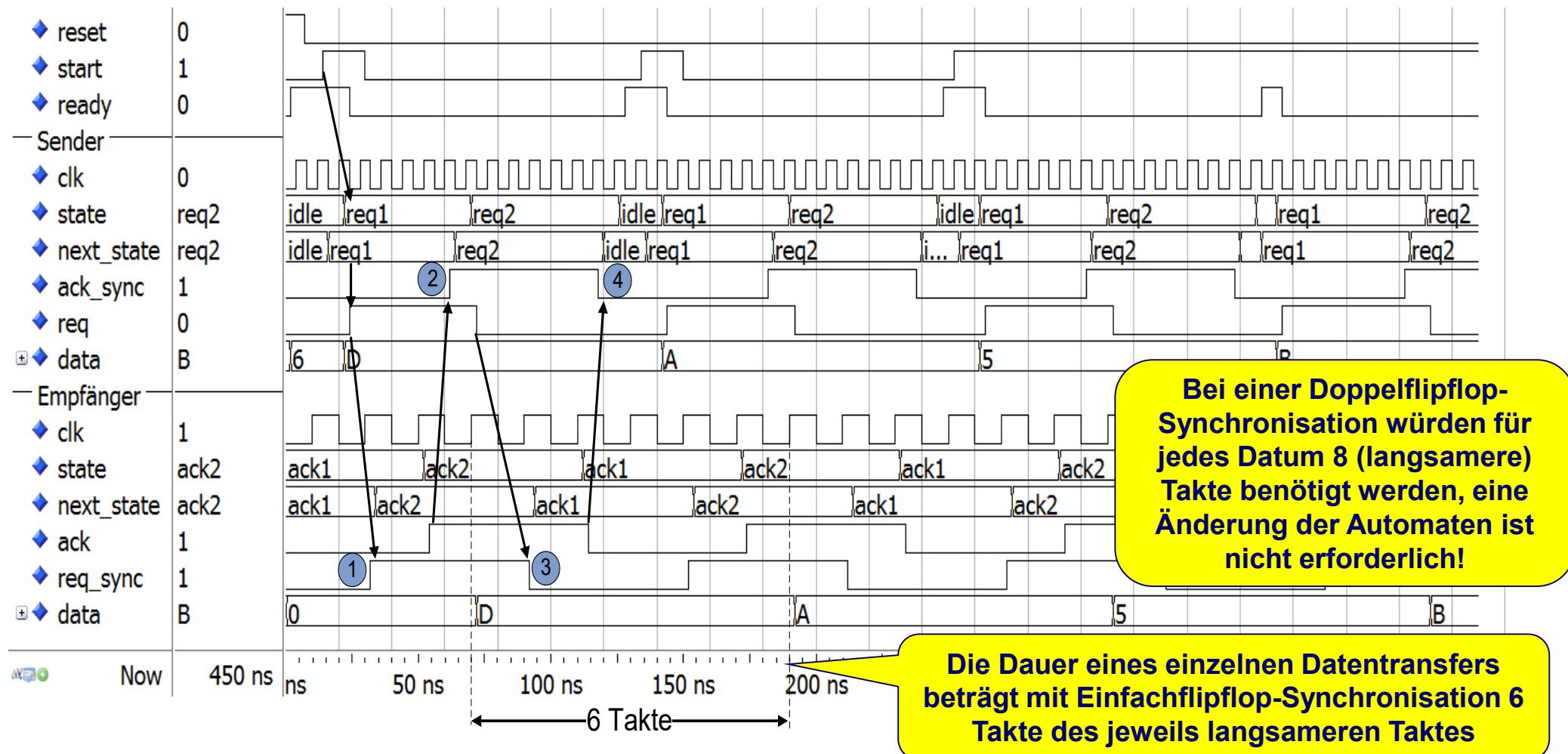
```
DUT: SENDER_EMPFAENGER_TOP
port map(CLK1, CLK2, RESET, START,
         READY, DATA);
```

Toplevel Komponenten-  
instanziierung

```
end TESTBENCH;
```

# Simulationsergebnis der 4-Phasen Kommunikation

- Sender und Empfänger arbeiten mit unterschiedlichem Takt.
  - Die REQ- und ACK-Handshakesignale werden in die jeweils andere Taktdomäne synchronisiert.
  - Die Datenübertragung erfolgt hier von der schnelleren in die langsamere Taktdomäne



# Synchronisation Digitaler Systeme

---

- **Kopplung von Signalen in zueinander synchronen Taktdomänen**
  - Impulsverkürzung
  - Impulsverlängerung
- **Synchronisierung Asynchroner Eingangssignale**
  - Synchronisierung langer Eingangsimpulse
  - Synchronisierung kurzer Eingangsimpulse
  - Asynchrone Resets
- **Datenaustausch zwischen Teilsystemen**
  - Synchrone Datenübertragung
  - Asynchrone Datenübertragung