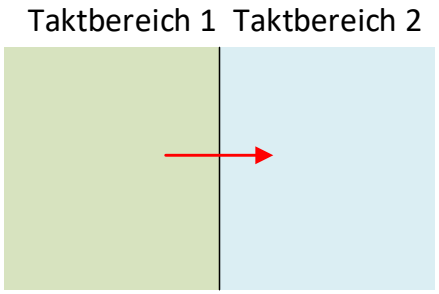
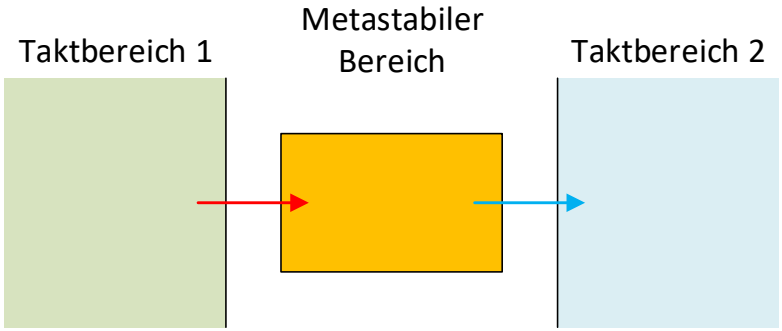
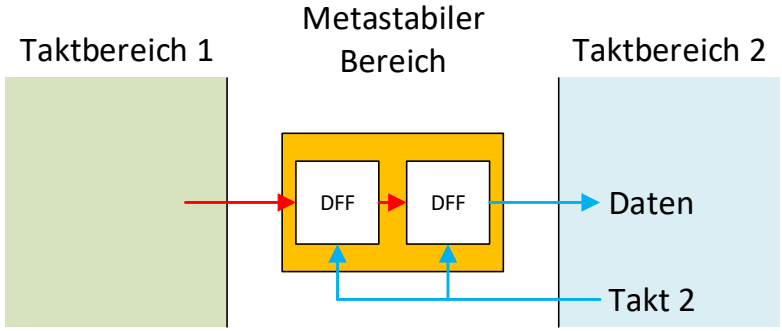
Übergang zwischen Taktbereichen



Taktbereich 1
kann auch extern sein



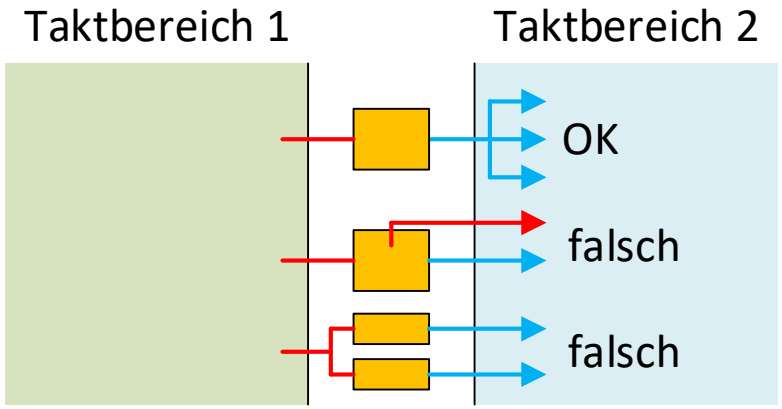
Einfügen eines
Synchronisierers



Typische DFF-Kette

Anzahl der Stufen abhängig von
- Takt 2
- Technologie der DFF

Anwendung für ein Signal



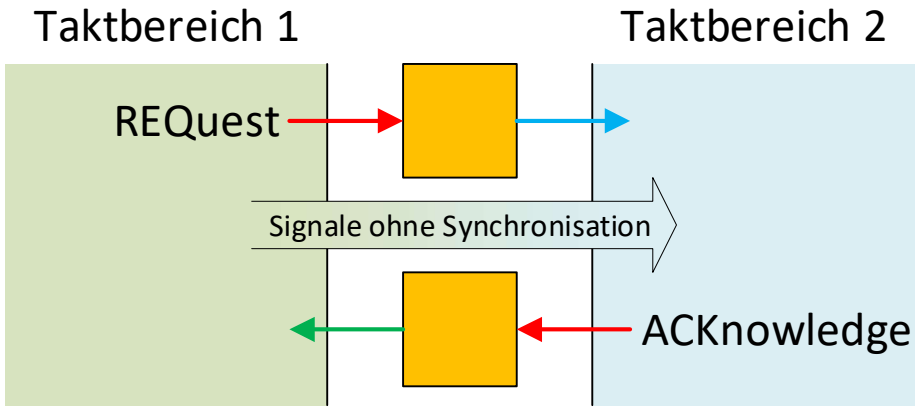
Nicht auf Signale im Synchronisierer zugreifen

Ein Signal nur **einmal** in einen TB einsynchronisieren

Anwendung für mehrere Signale

Handshake

- Übergabe mehrerer Signale (Bus)
- Übergang zwischen Taktbereichen unbekannter Frequenzverhältnisse



Request (REQ)

- Anforderung T1->T2

Acknowledge (ACK)

- Bestätigung T2 -> T1

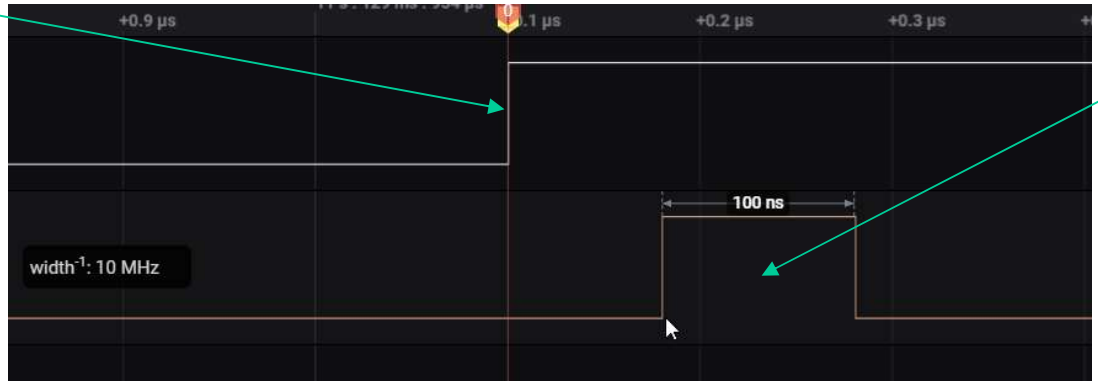
Signale ohne Synchronisation

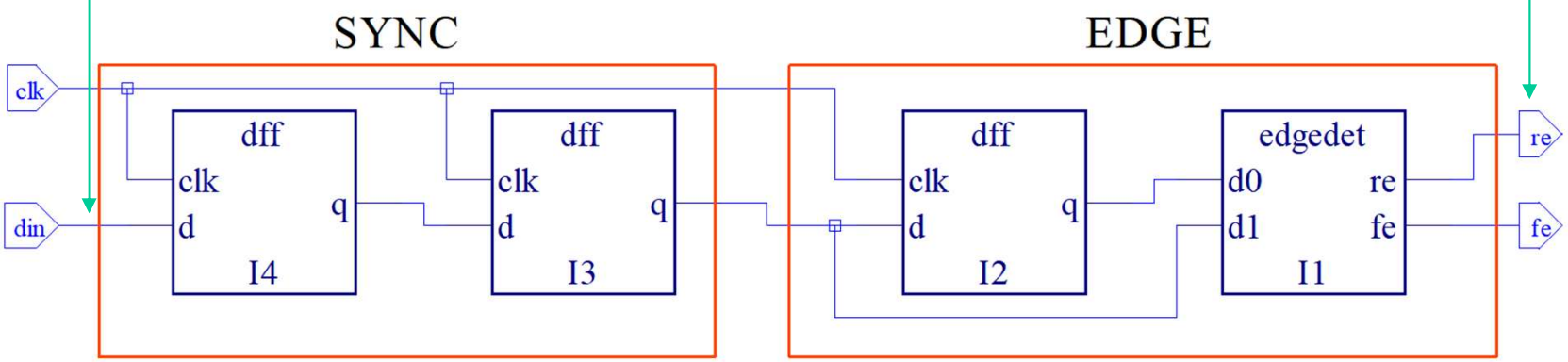
- müssen ab REQ bis ACK stabil bleiben (Aufgabe für Taktbereich 1)

Flankenerkennung

Asynchrones Taktsignal

Synchronisiertes Enable-Signal





Array

- **Deklaration durch neuen Typ**
  - Im Deklarationsteil (vor *begin*) bzw. in einem *package*
  - Anzahl der Elemente kann offen bleiben (unconstrained)

```
type name is array (range x to y) of elementyp;
type name is array (range x downto y) of elementyp;
type byte is array (7 downto 0) of std_logic;
signal ledreihe: byte; -- Byte hat immer 8 Bits, schon festgelegt

type name is array (integertyp range <>) of elementtyp;
type memory is array (natural range <>) of byte; -- Speichergröße variabel
signal ledmatrix: memory (0 to 15);              -- Speicher für 16 Bytes
-- Zugriff auf Elemente
ledmatrix(11) <= x"00101010";
ledmatrix(11)(2 downto 0) <= x"001";
```

- **Nutzung wie bei Vektoren (das sind arrays)**
  - Beliebige Bereiche (range) können in einer Anweisung bearbeitet werden
  - Zweidimensionale Felder bei der Synthese als Speicher

Record (Struktur)

- **Deklaration durch neuen Typ**

```
type record_typname is
record
  element1_name : typ1;
  element2_name : typ2;
end record;
```

- **Nutzung**

```
signal signame: record_typname;
…
signame.element1_name  <= …
```

- **Keine individuellen Richtungen der Elemente möglich**
  (erst ab VHDL 2019)

Schleifen

- Schleifen werden **bei der Kompilation** ausgeführt
- Jeder Schleifendurchlauf erzeugt (in der Regel) Hardware, d.h. **Schaltungsaufwand**
- Diese Hardware arbeitet dann **nebenläufig**
- Die Schleifenvariable muss nicht deklariert werden und ist **außerhalb der Schleife nicht sichtbar**
- **Syntaktische Unterscheidung** zwischen Anwendung in der sequentiellen und nebenläufigen Umgebung
- Abfrage der Variable mit **if** für Sonderfälle ist **in beiden Fällen** möglich

Schleifen

- In der sequentiellen Umgebung: for loop

```
for i in range
loop
  sequentielle Anweisungen;
  exit; -- when bedingung
  next; -- when bedingung;
end loop;
```

- Range (z.B. 0 to 7) muss konstant sein
- exit und next können optional eine Bedingung haben
- Schleifen können geschachtelt werden

- In der nebenläufigen Umgebung: for generate loop

```
label: for i in range
generate
  multireg : reg port map (clk, d(i), q(i));
end generate label;
```