case/when

```
case signal/variable is

 when wert1 =>
                    Sequentielle
                    Anweisungen
 when wert2 =>
                    Sequentielle
                    Anweisungen
 when others =>
                    Sequentielle
                    Anweisungen

end case;
```

Wie switch/case in C

Defaultfall others zwingend nötig

Kann geschachtelt werden, pro Fall Block von Anweisungen möglich

wert1, .., wertN müssen disjunkt sein

Speicher - FlipFlops

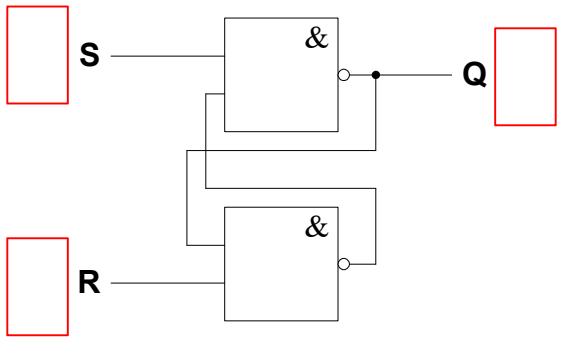
- neuer Zustand abhängig von Eingangssignalen **und** dem derzeitigen Zustand

- **zustandsgesteuerte** Flipflops ändern ihren Zustand aufgrund eines Signalpegels (statisch)
  - RS-Flipflop
  - Latch

- **flankengesteuerte** Flipflops ändern ihren Zustand aufgrund eines Pegelwechsels (dynamisch)
  - D-Flipflop

RS-FlipFlop

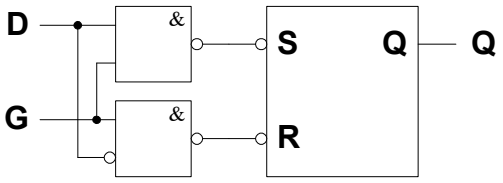
- **Bisher: Schaltnetze (kombinatorische Schaltungen)**
  - Eingangsbelegung legt Ausgangsbelegung eindeutig fest
  - Schaltung hat kein Gedächtnis
- **Gesucht: Schaltung mit Gedächtnis (Speicher)**



S	R	Q_n	Q_{n+1}

Latch

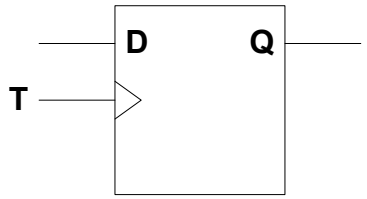
- schaltet während eines Taktsignalpegels den Eingang durch (das Latch ist transparent)
- hält während des anderen Taktsignalpegels den Ausgang auf dem zuletzt angenommen Pegel (das Latch ist gesperrt)



G	D	Q_{n+1}
0	-	Q_n
1	0	0
1	1	1

D-Flipflop

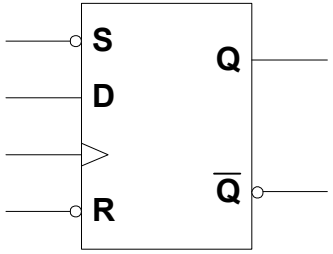
- Übernahme des Eingangs an den Ausgang nicht solange ein bestimmter Pegel anliegt sondern bei einem Pegelwechsel (Flanke)
- Nur eine Flanke ist die aktive Flanke



T	Q_{n+1}
0	Q_n
1	Q_n
⎍	D
⎏	Q_n

D-RS-Flipflop

- **Kombination aus**
  - taktflankengesteuertem D-Flipflop
  - zustandsgesteuertem RS-Flipflop



S und R haben Vorrang

S=0 und R=0 nicht definiert

S	R	T	Q_{n+1}	$\overline{Q}_{n+1}$
0	1	-	1	0
1	0	-	0	1
1	1	0	Q_n	$\overline{Q}_n$
1	1	1	Q_n	$\overline{Q}_n$
1	1	⌐	D	$\overline{D}$

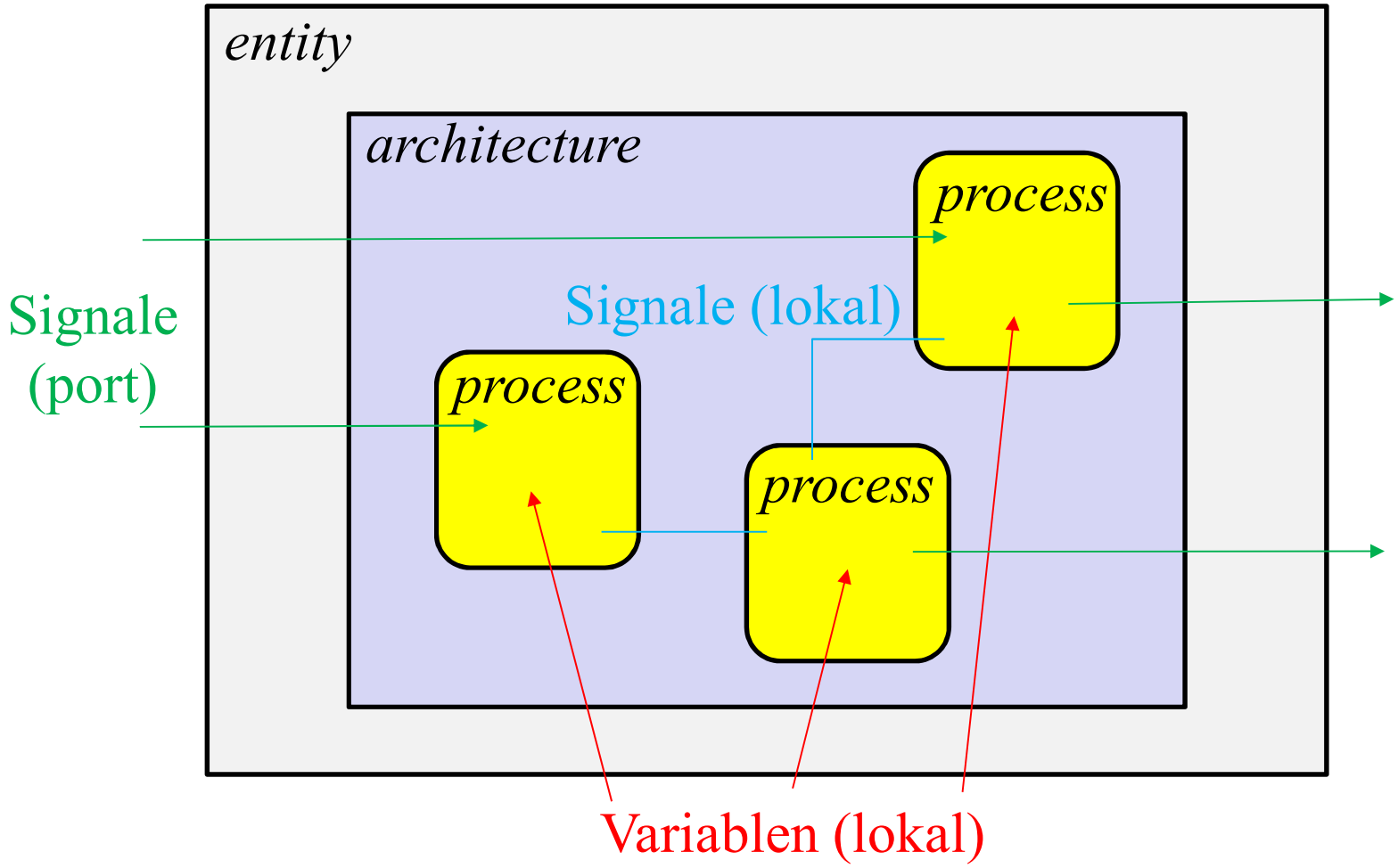
Prozess – Einsatzgebiete

- **Getakteter Prozess**
  - Beschreibung eines Speichers
  - Anwendung: Automat
- **Kombinatorischer Prozess**
  - Nutzung „bequemerer“ Anweisungen
- **Nicht synthetisierbarer Prozess**
  - Erzeugung von Signalverläufen im Simulator
  - Anwendung: Testbench

Prozess – Einordnung

- **Beinhaltet eine sequentielle Umgebung**
  - Andere Anweisungen (if/then, case/when)
  - Reihenfolge der Anweisungen hat Effekt
- **Ist nicht schachtelbar (kein Prozess im Prozess)**
- **Kann lokale Variablen deklarieren**
  - Formal wie Signale, aber andere Eigenschaften
- **Wird gleichzeitig mit anderen Prozessen ausgeführt**
  - Beliebig viele Prozesse in einer architecture möglich
- **Kommuniziert über Signale mit der Außenwelt**

Prozess – Einordnung



Prozess - Syntax

- Aufbau ähnlich einer architecture
- Kann lokale Deklarationen enthalten (keine neuen Signale)
- Kann eine Empfindlichkeitsliste enthalten,
  d. h. Liste aller Signale, auf die der Prozeß reagieren kann
- Kann mit einem Label versehen werden

```
label: process (Empfindlichkeitsliste)
  -- lokale Deklarationen

begin
  -- sequentielle Anweisungen

end process label;
```

Signale vs. Variablen

Eigenschaft	Signal	Variable
Deklaration	entity, architecture	process, function
Syntax Zuweisung	S <= …	V := …
Zeitverlauf (im Simulator)	ja	nein
Zuweisung wird im process wirksam	am Ende	sofort

Aufbau für synthetisierbare Schaltung

```
process (set, reset, clock)

begin

    if (bedingung aus set und reset)
    then
        -- asynchrone Aktionen
    elsif rising_edge(clock)
    then
        -- synchrone Aktionen
    end if;

end process;
```

Hinweise

- **Taktabfrage**
  - nur einmal im process (siehe Tempate)
  - nur eine Flanke rising_edge() oder falling_edge()
  - keine „Verheiratung“ des Taktsignals mit anderen Signalen, also nicht if rising_edge(clock) and hold=‘1‘
  - Taktsignal sollte vom Typ std_logic sein

- **Sonstiges**
  - Signale/Variablen vorbelegen, sie können in der sequentiellen Umgebung ja später überschrieben werden
  - Keine Anweisung außerhalb des if/then/else-Blocks (grauer Bereich im Template)
  - Keine wait-Anweisung (da Empfindlichkeitsliste vorhanden ist)