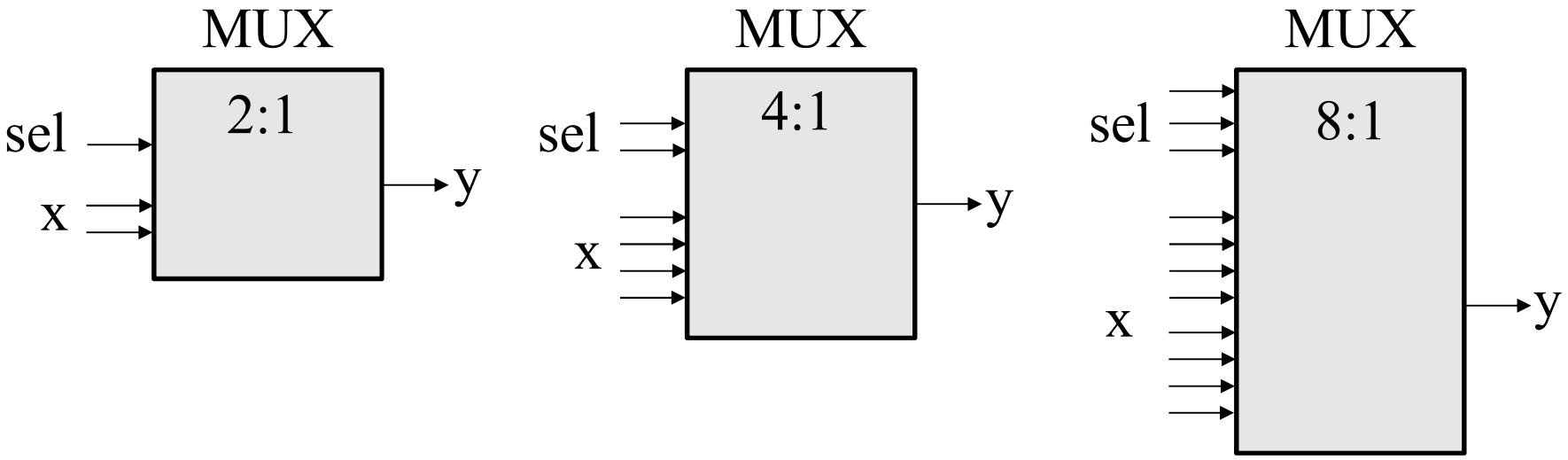
Parametrisierung - Idee

1. Entwurf eines Universalmoduls
2. Konfiguration bei der Instanziierung



Parametrisierung - Entwurf

```
entity modul is
  generic
  (
    parameterliste;
  );
  port
  (
    signalliste;
  );
end;
```

Entwurfszeit

Sind in der fertigen Schaltung nicht mehr enthalten, keine Änderung im Betrieb möglich

Betriebszeit

Sind in der fertigen Schaltung noch enthalten, können sich im Betrieb ändern

Parametrisierung - Instanziierung

```
I1: MUX generic map (zuweisungsliste_k1)
        port map (zuweisungsliste);

I2: MUX generic map (zuweisungsliste_k2)
        port map (zuweisungsliste);
```

Die Werte zur Konfiguration
(zuweisungsliste_k1, zuweisungsliste_k2)
müssen bei der Schaltungssynthese bekannt sein

Package

- **Entspricht einem Header in C**
- **Deklarationsteil (immer vorhanden)**
  - package name is / end name;
  - Deklaration von Datentypen, Konstanten, Funktionen, Komponenten
- **Definitionsteil (nur falls nötig)**
  - package body name is / end name;
  - Definitionen von Funktionen
- **Einbindung in andere Design Units**
  - library work; -- wo zu finden, hier in work (Projekt)
  - use work.package.all; -- was wird eingebunden (hier alles)

Unterscheidung Umgebungen

- Im Betrieb werden alle Signale gleichzeitig verarbeitet

- Beim Entwurf gibt es **zwei Umgebungen**

  – **Nebenläufig**
    Die Reihenfolge der Anweisungen spielt keine Rolle

  – **Sequentiell**
    Die Anweisungsreihenfolge kann einen Einfluss haben

Unterschiedliche Syntax

Nebenläufig	Sequentiell
Signale	Signale Variablen
Zuweisung `<=` (an Signal)	Zuweisung `<=` (an Signal) `:=` (an Variable)
when/else	if/then/else
with/select	case/when

if/then/else

Vergleichbar mit when/else, aber beliebig schachtelbar.

```
if bedingung1
then
   anweisungen;
else
  if bedingung2
  then
     anweisungen;
  end if;
end
```

```
if bedingung1
then
   anweisungen;
elsif bedingung2
then
   anweisungen;
end if;
```

Funktion

Funktionen werden zwar wie in einer Programmier-sprache benutzt, werden aber bei der Schaltungs-synthese bei jedem Aufruf instanziiert.

N Funktionsaufrufe -> N-facher Aufwand

Deklaration / Definition meist in einem package, aber auch lokal in einer architecture möglich

In der Funktion ist die sequentielle Umgebung aktiv

Funktion - Syntax

- Aufbau ähnlich einer Kombination aus entity und architecture
- Hat immer einen Rückgabewert
- Rückgabe ist immer erforderlich
- Kann lokale Deklarationen enthalten

```
function name (signalliste) return datentyp is
  -- lokale Deklarationen
begin
  --Funktionskörper
  return wert; -- Rückgabe ist zwingend
end;
```

Prozess

- Eröffnet eine neue sequentielle Umgebung
- Hauptanwendung ist die Beschreibung von Schaltungen mit einem Takt
- Beliebig viele Prozesse in einer nebenläufigen Umgebung möglich (laufen alle gleichzeitig)
- Kommunikation in/aus einem Prozess über Signale
- In einem Prozess nur Deklaration neuer lokaler Variablen möglich

Prozess - Syntax

```
Name:
process (Empfindlichkeitsliste)
  -- Lokale Deklarationen
begin
  -- Sequentielle Umgebung
end process Name;
```

- Der Name ist optional.
- Die Empfindlichkeitsliste enthält alle Signale,
  bei deren Änderung eine Reaktion im prozess eintreten kann.
  Das ist sehr häufig nur der Takt.