Exercise: how can we do?

❑ **Create special inter-process communication channels: the signals**

❑ **Interleave execution phases and signal update phase**

❑ **Signal assignments during execution phase are recorded and delayed; no immediate effect on the signal**

❑ **Signal value change only during update phase**

Exercise: examples of processes

- ❑ **The process models a 2 inputs or:** `Z = X or Y.`
  - ❑ `MAJ: process (X, Y, Z)`
    `begin`
    `  M <= X and (Y or Z) or Y and Z;`
    `end process MAJ;`

  - ❑ `FA: process (A, B, CI)`
    `begin`
    `  CO <= A and (B or CI) or B and CI;`
    `  S <= A xor B xor CI;`
    `end process FA;`

- ❑ **The clock is** `Z`**, the input is** `X` **and the output is** `Y`**. On every event on** `Z`**, if it's a rising edge (event on** `Z`**,** `Z='1'` **and** `Z` **is a** `BIT` **=> rising edge),** `Y` **is assigned** `X`**.**
  - ❑ `DFF: process(CLK, RSTN)`
    `begin`
    `  if RSTN = '0' then Q <= '0';`
    `  elsif CLK = '1' and CLK'event then`
    `    Q <= D;`
    `  end if;`
    `end process DFF;`

```
signal X, Y, Z: BIT;
P1: process(X, Y)
begin
  if X = '1' then
    Z <= '1';
  elsif Y = '1' then
    Z <= '1';
  else
    Z <= '0';
  end if;
end process P1;
```

```
signal X, Y, Z: BIT;
P2: process(Z)
begin
  if Z = '1' then
    Y <= X;
  end if;
end process P2;
```

Exercise: type attributes

- REVERSE_COLORS'BASE = COLORS

- REVERSE_COLORS'LEFT = ORANGE

- REVERSE_COLORS'RIGHT = RED

- REVERSE_COLORS'HIGH = ORANGE

- REVERSE_COLORS'LOW = RED

Exercise

❑ **Signal S assigned twice in same execution phase: second assignment overwrites first one**

❑ **Signal S thus read before being written: memory elements in a combinational process**

❑ **Computed value is A+B+S instead of A+B+2**

```
process(A, B)
  variable V: integer;
begin
  V := A + B;
  V := V + 2;
  S <= V;
end process;
```

Exercise

```
P1: process(CP, RST)
begin
  if (RST = '0') then
     Q <= '0';
  elsif (CP = '1' and CP 'event) then
     Q <= D;
  end if;
end process P1;


QN <= not Q;
```

Exercise

```
function VEC2NAT(VAL: BIT_VECTOR) return NATURAL;
```

```
function VEC2NAT(VAL: BIT_VECTOR) return NATURAL is
  variable TMP: BIT_VECTOR(VAL'LENGTH - 1 downto 0) := VAL;
  variable RES: NATURAL := 0;
begin
  assert (VAL'LENGTH < 32)
    report "VEC2NAT: overflow"
    severity WARNING;
  for I in VAL'LENGTH - 1 downto 0 loop
    RES := RES * 2;
    if (TMP(I) = '1') then RES := RES + 1; end if;
  end loop;
  return RES;
end function VEC2NAT;
```

Exercise

```vhdl
function PP(VAL1, VAL2: NATURAL) return NATURAL;
function PP(VAL1, VAL2: BIT_VECTOR) return BIT_VECTOR;
```

```vhdl
function PP(VAL1, VAL2: NATURAL) return NATURAL is
begin
  if (VAL1 < VAL2) then return VAL1 else return VAL2;
end PP;

function PP(VAL1, VAL2: BIT_VECTOR) return BIT_VECTOR is
begin
  assert ((VAL1'LENGTH = VAL2'LENGTH) and (VAL1'LENGTH < 32))
    report "PP: overflow or illegal use"
    severity WARNING;
  return NAT2VEC(PP(VEC2NAT(VAL1), VEC2NAT(VAL2)), VAL1'LENGTH);
end function PP;
```

Exercise

```vhdl
function PG(VAL1, VAL2: NATURAL) return NATURAL;
function PG(VAL1, VAL2: BIT_VECTOR) return BIT_VECTOR;
```

```vhdl
function PG(VAL1, VAL2: NATURAL) return NATURAL is
begin
  if (VAL1 > VAL2) then return VAL1 else return VAL2;
end PG;

function PG(VAL1, VAL2: BIT_VECTOR) return BIT_VECTOR is
begin
  assert ((VAL1'LENGTH = VAL2'LENGTH) and (VAL1'LENGTH < 32))
    report "PG: overflow or illegal use"
    severity WARNING;
  return NAT2VEC(PG(VEC2NAT(VAL1), VEC2NAT(VAL2)), VAL1'LENGTH);
end function PG;
```

Exercise

```
type ST is (S0, S1, S2, S3, S4);

signal STATE, NEXT_STATE: ST;
```

```
P1: process(CP, RSTN)

begin

  if RSTN = '0' then
    STATE <= S0;
  elsif CP = '1' and CP'event then
    STATE <= NEXT_STATE;
  end if;

end process P1;
```

```
P2: process(STATE)

begin

  if (STATE = S1) then
    INIT <= '1';
  else
    INIT <= '0';
  end if;
  if (STATE = S3) then
    ADD <= '1';
  else
    ADD <= '0';
  end if;
  if (STATE = S4) then
    SHIFT <= '1';
  else
    SHIFT <= '0';
  end if;

end process P2;
```

EDC - VHDL Language, from Specification to model - Renaud PACALET

Exercise

```
P3: process(STATE, STB, DONE, LSB)
begin
  case STATE is
    when S0 => if (STB = '0') then
                    NEXT_STATE <= STATE;
               else
                    NEXT_STATE <= S1;
               end if;
    when S1 => NEXT_STATE <= S2;
    when S2 => if (DONE = '1') then
                    NEXT_STATE <= S0;
               elsif (LSB = '0') then
                    NEXT_STATE <= S4;
               else
                    NEXT_STATE <= S3;
               end if;
    when S3 => NEXT_STATE <= S4;
    when S4 => NEXT_STATE <= S2;
  end case;
end process P3;
```

```
entity SM is
  port(LSB, STB, DONE,
       CP, RSTN: in BIT;
       INIT, SHIFT,
       ADD: out BIT);
end entity SM;
architecture ARC of SM is
  type ST is (S0, S1, S2,
              S3, S4);
  signal STATE,
         NEXT_STATE: ST;
begin
  P1: process(CP, RSTN)
  begin
  …
  end process P1;
  P2: process(STATE)
  …
  P3: …
end architecture ARC;
```