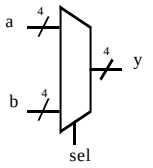
F2: Grunder i VHDL

- **Innehåll:**
  - **Kodmodell**
    - **Deklaration av entity**
    - **Architecture**
  - **Port deklaration**
  - **Deklaration av *Entity***
  - **Architecture**
    - **VHDL kodningsstilar**
    - **Kodningsstilens effekt på syntesresultatet**

VHDL exempel

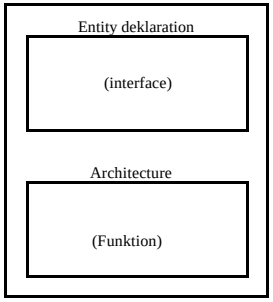
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENITIY mymux1 IS
  PORT (a, b: IN bit_vector(3 downto 0); --
inputs
        sel: IN bit;
       y: OUT bit_vector(3 downto 0)); -- outputs
END mymux1;


ARCHITECTURE behavior OF mymux1 IS
BEGIN
mux: process (a, b, sel)
  BEGIN
    IF sel = '0' THEN y <= a;
    ELSE y <= b;
    END IF;
  END PROCESS mux;
END behavior;
```

Symbol



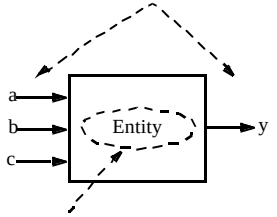
Kodmodell

Entity



Komponentmodell

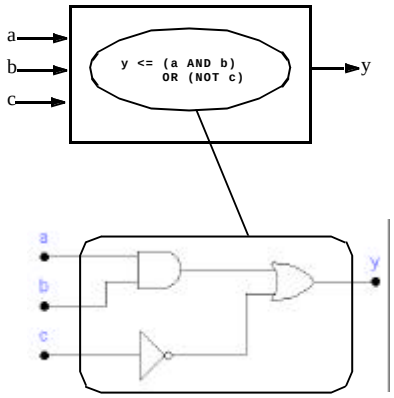
Ports - externa anslutningar som har en *class*, *type* och *mode*



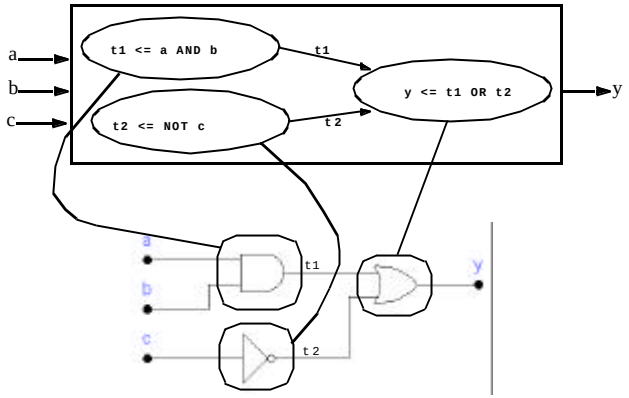
Entity - en samling a parallela processer som beskriver funktionen

Komponentmodell

Entity med en process



Entity med flera processer



Entity deklaration

- **Entity deklaration**
  - **definierar externt interface till *entity* eller komponenten**
  - **kommer före *architecture* sektionen**
- **Grundläggande *entity* syntax:**

```
ENTITY entity_name IS
  PORT (port1, port2: MODE TYPE;
        port3, port4: MODE TYPE;
        port5: MODE TYPE);
END entity_name;
```

riktningen på dataflödet

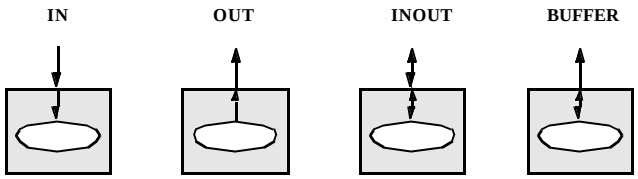
vilka typer av värden porten kan ha

Ports

- **Port deklarationen är det viktigaste i *entity* deklarationen**
- **Varje port representerar antingen**
  - **komponenternas externa pinnar eller**
  - **ledningar som kopplar samman två eller flera *entities* som finns i en och samma komponent**
- **Varje port har**
  - **port namn (identifierare som du skapar)**
  - **mode (riktning)**
  - **type (vilka värden porten kan anta)**

Ports

- Varje port kan vara i av fyra typer
  - IN, OUT, INOUT och BUFFER



Port moder

- IN : data går enbart <u>in till</u> entiteten som drivs av någon annan entitet
  - används på höger sida i en tilldelning (`z <= a NAND inport`)
- OUT : data går enbart ut från den här entiteten in till andra entiteter
  - det går ej att läsa signalen intern i den här entiteten
  - används på vänstra sidan i en tilldelning (`outport <= x OR y`)
- BUFFER : data går enbart ut från från den här entiteten in till andra entiteter
  - kan läsas av den här entiteten
  - kan enbart ha en drivande signal
  - används på båda sidorna i en tilldelning (`bufport <= z;`
    `IF bufport = '1' THEN ...`)
- INOUT : data kan gå i bägge riktningarna (bi-direktionell), antingen in eller ut eller bägge
  - kan läsas av den här entiteten
  - kan också drivas av extern drivare
  - används på båda sidorna i en tilldelning

Port typer

- PORTAR är <u>alltid</u>signaler
- TYPER som är användbara för syntes och simulering
  - bit, bit_vector
  - std_logic, std_logic_vector
  - std_ulogic, std_ulogic_vector
  - boolean
  - integer
- Typer som enbart är användbara vid simulering
  - real
  - time

Architecture

- Architecture
  - definierar entitetens <u>funktion</u>
  - måste följa entitet - deklarationen
- Grundläggande *architecture* syntax:

```
ARCHITECTURE arch_name OF entity_name IS
BEGIN
  CONCURRENT_STATEMENT1;
  CONCURRENT_STATEMENT2;
END arch_name;
```

Architecture

- Abstraktionsnivåer (VHDL kodningsstilar)
  - Beteende (behavioral)
  - Dataflöde (dataflow)
  - strukturell (structural)
- En architecture kan skrivas helt i en stil eller i en blandning av kodningsstilar
- Olika stilar tillåter konstruktören att beskriva funktionen för en komponent på det mest naturliga sättet

Beteende (behavioral)

- Högnivåbeskrivning, algoritmer
- Enkelt att skriva och förstå (som kod i ett högnivåspråk - C, Pascal, etc.)
- VHDL process med sekventiella satser - ordningsföljden är viktig!
- Exekveras i noll (zero) simuleringstid

```
ARCHITECTURE behavior OF mymux1 IS
BEGIN
  mux: PROCESS (a,b,sel)
  BEGIN
    IF sel = '0' THEN y <= a;
    ELSE y <= b;
    END IF;

  END PROCESS mux;
END behavior;
```


Beteende (behavioral)

- Ytterligare ett exempel på beteendebeskrivning av samma komponent

```
ARCHITECTURE behavior2 OF mymux1 IS
BEGIN
  PROCESS (a,b,sel)
  BEGIN
    y <= a;
    IF sel = '1' THEN y <= b;
    END IF;
  END PROCESS;
END behavior2;
```


Dataflöde (dataflow)

- "mellan-nivå" beskrivning
- kallas för RTL-beskrivning (Register Transfer Level)
- kan vara svårare att skriva och förstå
- Flera parallella signaltilldelningar
- Exekveras i icke-noll simuleringstid

```
ARCHITECTURE dataflow OF mymux1 IS
BEGIN
  y <= a WHEN (sel = '0') ELSE b;
END dataflow;
```

Dataflöde (dataflow)

- Ytterligare en dataflödesbeskrivning av samma komponent
- Samtidiga satser - ordningsföljden ej relevant

```
ARCHITECTURE boolean_dataflow OF mymux1 IS
BEGIN
  y(3) <= (a(3) AND NOT sel) OR (b(3) AND sel);
  y(2) <= (a(2) AND NOT sel) OR (b(2) AND sel);
  y(1) <= (a(1) AND NOT sel) OR (b(1) AND sel);
  y(0) <= (a(0) AND NOT sel) OR (b(0) AND sel);
END boolean_dataflow;
```


Strukturell beskrivning

- Låg-nivå, VHDL nätlista - komponentinstantiering och sammankoppling
- Egentligen en text version av ett schema
- Hierarkisk
- Kan vara svår att skriva och förstå (mycket detaljer)
- Ingen VHDL process eller parallella satser

Strukturell beskrivning

```
USE work.gates_pkg.all;
ARCHITECTURE structureal OF mymux1 IS
  SIGNAL ta, tb: bit_vector (3 downto 0);
  SIGNAL seln: bit;

BEGIN
  u0: and2 PORT MAP (a(3), seln, ta(3));
  u1: and2 PORT MAP (a(2), seln, ta(2));
  u2: and2 PORT MAP (a(1), seln, ta(1));
  u3: and2 PORT MAP (a(0), seln, ta(0));
  u4: and2 PORT MAP (b(3), seln, ta(3));
  u5: and2 PORT MAP (b(2), seln, ta(2));
  u6: and2 PORT MAP (b(1), seln, ta(1));
  u7: and2 PORT MAP (b(0), seln, ta(0));
  u8: or2 PORT MAP (ta(3), tb(3), y(3));
  u9: or2 PORT MAP (ta(2), tb(2), y(2));
  u10: or2 PORT MAP (ta(1), tb(1), y(1));
  u11: or2 PORT MAP (ta(0), tb(0), y(0));
  u12: not PORT MAP (sel, seln);
END structural;
```


Kodningsstilens effekt på syntesresultatet

- För komplexa kretsar kan kodningsstilen påverka implementeringen som görs av syntesverktyget
- VHDL koden för en funktion kanske inte är skriven på det optimala sättet
- Syntesverktyget kan generera logik som inte passar den komponent på bästa sätt
- Place & Route verktyget kan välja ett sätt som inte utnyttjar komponentens resurser på bästa sätt

Kodningsstilens effekt på syntesresultatet

- Alltså, olika beskrivningar kan resultera i olika implementeringar av kretsen
  - de kan funktionellt vara ekvivalenta men ha samma ELLER olika prestanda
  - de kan funktionellt vara ekvivalenta men kräver mer grindar
  - de kan till och med vara funktionellt olika på grund av buggar i verktygen eller konstruktionsfel
  - Ett bra syntesverktyg ska generera ekvivalenta kretsar med nära optimal lösning

- Strukturell kod är lämplig för hierarkisk nedbrytning
  - man kan använda färdiga biblioteksselement från komponentleverantör för att få bättre prestanda - bör dock undvikas
  - man tappar oberoendet för en viss tillverkare eller komponent
  - ökar konstruktionstiden om man måste lära sig ett specifikt bibliotek
  - kan resultera i icke-optimal konstruktion
  - det är svårt ett debugga en nätliste-baserad konstruktion
  - Den viktigaste anledningen till att använda VHDL är att kunna skriva beteende- eller dataflödes-kod

Kodningsstilens effekt på syntesresultatet

- Generella rekommendationer:
- Starta med en beteende- och dataflödes- beskrivning
  - lättare att skriva, förstå och debugga

- Använd strukturell kod där det är naturligt att bryta ner komponenten till separata funktionsenheter

- If konstruktionen inte uppnår uppställda mål (hastighet/area) lägg till direktiv till syntesverktyget

- Om målen fortfarande inte är uppfyllda så lägg till detaljerad RTL beskrivning och/eller använd leverantörspecifika bibliotekskomponenter