F3: Grunder i VHDL

- **Innehåll:**
  - **Modellering för simulering**
  - **Simuleringscykel**
  - **VHDL dataobjekt**
    - **Identifierare, datatyper, attribut**

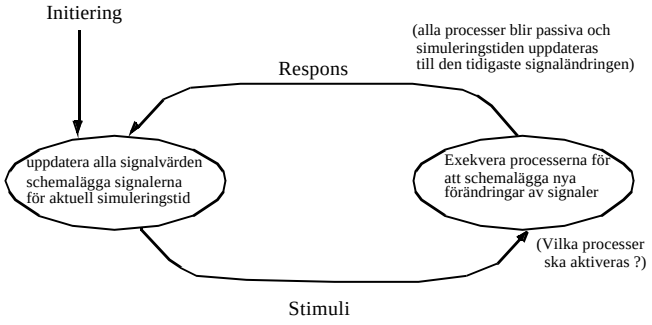
Modellering för simulering

- **I den här kursen betonas modellering för syntes**
- **Koden måste alltid simuleras för att funktionaliteten ska verifieras**
- **Koden måste också skrivas så att den kan simuleras**
- **De viktigaste begreppen för simulering som tas upp är:**
  - **Simuleringscykel**
  - **Timing i simulering**
  - ***sensitivity list***
  - **drivning av signaler**

Simuleringscykel

- **VHDL kod består av ett antal parallella satser eller processer**
- **Stimuli / Respons**
  - **stimuli (ingångar) förändras vid specifik simuleringstid**
  - **processer evaluerar de resulterande logiska förändringarna**
  - **värdena för signalerna ändras efter det en specificerad simuleringstid har förflutit**
- **Samtliga parallella processer exekveras upprepat parallellt ("samtidigt") enligt en standard simuleringscykel**

Simuleringscykel



Initieringsfas

- Aktuell simuleringstid sätts till noll
- Signaler initieras till:
  - värden som explicit har satts i VHDL koden
  - om signalerna inte har initieras i koden så sätts följande värden
    - '0' för bit
    - FALSE för boolean
    - 'U' (unknown) för datatypen std_logic och std_ulogic
    - -2,147,483,647 (-2^{11} -1) för en 32 bitars integer
    - *first* (det mest vänstra) värdet för *enumeration* typ
- Varje process körs tills den går i viloläge (suspended), vilket sker då den hamnar i ett läge då den väntar på en förändring på en insignal
  Processen schemalägger (schedules) nya händelser för framtida simuleringstid
- Simuleringstiden är fortfarande noll

Uppdateringsfas

- Alla signaler som har en övergång (transition) schemalagd till nuvarnade simuleringstid uppdateras (signalen har en händelse (event))
- Övriga signaler förändras inte (signalerna har ingen händelse)
- Simuleringstiden uppdateras ej

Processexekvering

- Varje process exekveras OM den är känslig (sensitive) för en signal som har en händelse (event) i nuvarande simuleringscykel
- Simuleringstiden för nästa cykel bestäms. Det är den tidigaste tidpunkten av följande två fall:
  - den tidigaste tidpunkt någon signal har en händelse schemalagd
  - den tidigaste tidpunkt någon process som är schemalagd ska aktivares
- Om den nya tidpunkten är ett delta delay, en ny simuleringscykel startas med samma simuleringstid som nuvarande simuleringscykel
- I annat fall så uppdateras simuleringstiden och en ny simuleringscykel startar

Processexekvering

- En process kan antingen vara exekverande eller i viloläge
- Exekvering börjar då någon av signalerna i processens sensitivitetslista ändras (*event*)
- Under exekveringen så exekveras satserna i processen sekventiellt i den ordning de står
- Exekvering pågår tills processen går i viloläge. Att processen går i viloläge kan orsakas av:
  - den sista satsen i processen har exekverats, eller
  - Ett *wait*-statement har exekverats

Processexekvering

- **Det finns en ekvivalens mellan *sensitivity-list* och *wait*-sats**
- **Nytt värde tilldelas signalen *x* vid simuleringstiden (nuvarande tidpunkt + *delta delay*)**

```
p1: PROCESS (a,b)
BEGIN
  x <= a AND b;
END PROCESS p1;
```

⟺

```
p2: PROCESS
BEGIN
  x <= a AND b;
  WAIT ON a,b;
END PROCESS p2;
```

- **Nytt värde tilldelas signalen *x* vid simuleringstiden (nuvarande tidpunkt + 5 ns)**
  - **detta gäller endast för simulering inte syntes**

```
p1: PROCESS (a,b)
BEGIN
  x <= a AND b AFTER 5 ns;
END PROCESS p1;
```


Schemaläggning av händelser och drivning av signaler

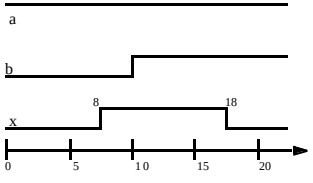
- **Det sista och bestående effekten från en process är schemaläggning av kommande händelser för dess utgångssignaler**
- **När en händelse har schemalagts för en signal, en eller flera tid/värde-par läggs till den signalens *utgångs-vågform***
- ***utgångs-vågformen* är helt enkelt en lista med tid/värde-par (och inte en grafisk vågform)**
- **utgångs-vågformen kontrollerar en *signal drivar* mekanism som ändrar signalens värde vid de tidpunkter som specificeras av tid/värde-paren under uppdateringsfasen**

Drivning av signaler: Exempel 1

- **ändlig fördröjning specificeras i processen p4**
- **antag att `a = '1', b = '0', x = '0'` vid simuleringstiden = 0**
- **Vid tidpunkten 10 ns uppdateras b till '1'**

```
p4: PROCESS (a,b)
BEGIN
  x <= a AND b AFTER 5 ns,
       NOT b AFTER 8 ns
END PROCESS p1;
```

tid	a, b, x	drivare	kommentar
0 ns	1, 0, 0	(0, 0ns) (0, 5ns) (1, 8ns)	p4 körs p.g.a initiering
5 ns	1, 0, 0	(0, 5ns) (1, 8ns)	
8 ns	1, 0, 1	(1, 8ns)	
10 ns	1, 1, 1	(1, 10ns) (1, 15ns) (0, 18ns)	p4 körs p.g.a att b ändras
15 ns	1, 1, 1	(1, 15ns)(0,18ns)	
18 ns	1, 1, 0		

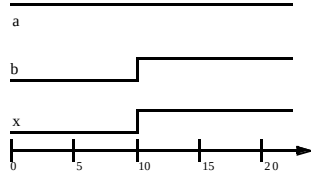


Drivning av signaler: Exempel 2

- **Ingen fördröjning specificeras i koden i p5 - *delta delay* används**
- **Antag att `a = '1', b = '0', x = '0'` vid simuleringstiden = 0**
- **Vid tidpunkten 10 ns uppdateras b till '1'**

```
p5: PROCESS (a,b)
BEGIN
  x <= a AND b;
END PROCESS p1;
```

tid	a, b, x	drivare	kommentar
0 ns	1, 0, 0	(0, 0ns) (0, 0ns+Δ)	p5 körs p.g.a initiering
0 ns Δ	1, 0, 0	(0, 0ns+Δ)	
10 ns	1, 1, 0	(0, 10ns)(1, 10ns+Δ)	p5 körs p.g.a att b ändras
10 ns Δ	1, 1, 1	(1, 10ns+ Δ)	

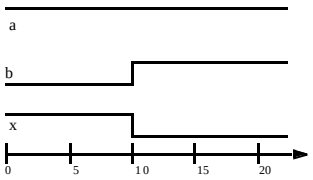


Drivning av signaler: Exempel 3

- Två sekventiella tilldelningar till signalen *x* - resulterande signalen blir det sist tilldelade värdet
- Ingen fördröjning är specificerad i p6 - *delta delay* används
- Antag att `a = '1', b = '0', x = '0'` vid simuleringstiden = 0
- Vid tidpunkten 10 ns uppdateras b till '1'

```
p6: PROCESS (a,b)
BEGIN
  x <= '1';
  IF b = '1' THEN
    x <= '0';
  END IF;
END PROCESS p6;
```

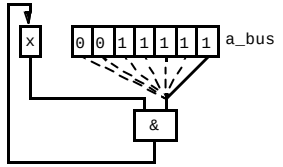
tid	a, b, x	drivare	kommentar
0 ns	1, 0, 0	(0, 0ns) (1, 0ns+Δ)	p6 körs p.g.a initiering
0 ns Δ	1, 0, 1	(1, 0ns+Δ)	
10 ns	1, 1, 1	(1, 10ns) ~~(1, 10ns+Δ)~~ (0, 10ns+Δ)	p6 körs p.g.a att b ändras
10 ns Δ	1, 1, 0	(0, 10ns+Δ)	



Vi kan få problem ...

- Skriv en model för en 8-ingångars AND grind: x = AND(a_bus(7..0))
- Problem p.g.a schemaläggning och timing - x blir ALLTID 0 !
- Varför ? och ...
- ... hur kan man fixa det ?

```
ARCHITECTURE wont_work OF my_and IS
BEGIN
  anding: PROCESS (a_bus)
  BEGIN
    x <= '1';
    FOR i IN 7 DOWNTO 0 LOOP
      x <= a_bus(i) AND x;
    END LOOP;
  END PROCESS anding;
END wont_work;
```



Identifierare

- Identifierare = namn på saker DU skapar
  - signaler, variabler, konstanter
  - *architecture* namn, *entity* namn, *process* namn, *component* namn
- Regler
  - Får inte vara ett reserverat ord i VHDL
  - Stora och små bokstäver är ekvivalenta (*case-insensitive*)
  - Första och sista tecknet får INTE vara underscore (_)
  - Två på varandra följande underscores är inte tillåtet (__)
- Vilka identifierare är legala
  - foobar_5, 7bits, carry/, _load, Execute14_more, doitnow_, Endif, add_subtract, process, StrongFuzzyLogicDriver, don't_do_it, exor#and

Objekt

- Objekt är saker som kan hålla ett värde
  - de har *class* och *type*
  - Kan tilldelas ett initialvärde (användbart för syntes ?)
  - Deklareras i *package*, *entity*, *architecture* eller *process*
  - Räckvidden begränsad till den region den är deklarerad i
- *Class* bestämmer vilka slags operationer man kan göra på objektet
- *Type* bestämmer vilkla giltiga värden objektet kan anta
- *Class*
  - signal - värdet ändras som funktion av tiden, har en drivare och kan ses som en fysisk ledning
  - variable - värdet ändras omedelbart, inga tidsbegrepp relateras
  - constant - värdet kan ej ändras
  - file - värden från extern disk fil kan accesseras

Deklaration av objekt

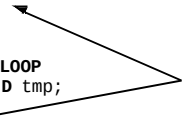
- **Syntax**
  - ***Class Identifier : Type :=*** *InitialValue****;***
- **Exempel**

```
CONSTANT MyNumber: real := 3.14159;
SIGNAL Load_Reg_n: std_logic := 'X';
VARIABLE counter: bit_vector;

ENTITY ename IS
  Ports
  Declarations   -- no variables
END ename

ARCHITECTURE x OF y IS
  Declarations   -- no variables
BEGIN  ...  END x;

PROCESS (a,b)
  Declarations   -- no signals
BEGIN   ...   END PROCESS;
```


Variables

- **Enbart giltiga i processer och subprogram**
- **Används vanligtvis för hög-nivå beskrivningar och beräkningar i algoritmer**
- **Lätt att skriva men svårt att syntetisera**
- **direkt tilldelning:** `x := a OR b;`
- **Korrekt 8-ingångars AND gate modell med variabler**

```
ARCHITECTURE will_work OF my_and IS
BEGIN
  anding: process(a_bus)
    VARIABLE tmp: bit;
  BEGIN
    tmp := 1;
    FOR i IN 7 DOWNTO 0 LOOP
      tmp := a_bus(i) AND tmp;
    END LOOP;
    x <= tmp;
  END PROCESS anding;
END will_work;
```



TYPES

- **datatypen för ett objekt bestämmer vilka värden det kan anta**
- **VHDL är ett språk som har hårda krav på hur olika typer får användas**
  - **Objekt av olika grundtyper kan inte tilldelas varandra direk (konverteringsfunktioner måste användas)**
  - **Vilka typer har följande variabler ?**
    `a <= 14;  b <= '1';  c <= 'X';  d <= TRUE;`
- **Det finns två huvudkategorier av typer**
  - **Skalärer : kan anta ett enda värde (*enumeration , integer, float, physical*)**
  - **Sammansatta : kan anta flera värden (*array, record*)**
- **Typerna kan vara fördefinierade eller definierade av användaren**
  - **fördefinierade typer definieras i VHDL standard 1076 och 1164**
  - **Användardefinierade typer skapas av dig**

Skalärer (Scalar)

- ***Enumeration***
  - **en lista med distinkta värden som objektet kan anta**
- ***Integer***
  - **ett set med heltal, positiva och negative, fördefinierad typ "integer"**
  - **32-bitars värden med tecken, $-(2^{11}-1)$ till $+(2^{11}-1)$, -2 147 483 647 till +2 147 483 647**
  - **ett reducerat område används ofta vid syntes, t.ex:**
    `VARIABLE num: integer RANGE -64 TO +64;`
- ***Float***
  - **Flyttal, fördefinierad typ "real"**
  - **32-bitars enkel precision**
  - **INTE för syntes, resulterar i för komplex hårdvara**
- ***Physical***
  - **måttenheter, fördefinierad typ "time" (fs, ps, ns, ... min, hr)**
  - **har ingen betydelse för syntes**

Enumeration

- **Enumeration typ är en lista med distinkta värden där ordningen på dessa har betydelse**
- **användar-definierade typer, exempel:**

```
TYPE instruct IS (nop, add, sub, jump);
SIGNAL instruction: instruct; ...

  IF instruction > nop THEN ...
  instruction <= jump;
```

- **Fördefinierade 1076 typer**

```
TYPE boolean IS (FALSE, TRUE);
TYPE bit IS ('0', '1');
```

- **Fördefinierade 1164 typer**
  - **std_logic, std_ulogic, + deras arrayer och under-typer**
  - **tillgång till dessa fås genom *LIBRARY* och *USE* satserna:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```


Enumeration typer - 1164

- **grundtypen är std_ulogic (*unresolved*)**

```
TYPE std_ulogic IS ('U', -- Uninitilized
                    'X', -- Forcing unknown
                    '0', -- Forcing zero
                    '1', -- Forcing one
                    'Z', -- High impedance (float)
                    'W', -- Weak unknown
                    'L', -- Weak zero
                    'H', -- Weak one
                    '-'  -- Don't care
                    );

SUBTYPE std_logic IS resolved std_ulogic;
```

- **Värdena 'U', 'X', 'W' stöds inte av syntes**
- **värden som anges med bokstäver är case-sensitive: 'z' är INTE samma som 'Z'**

Sammansatta typer (*Composite*)

- **Array - många element av samma typ**
  - **Mest använda fördefinierade array typen (1076 och 1164)**

```
TYPE bit_vector IS ARRAY (natural RANGE <>) OF bit

TYPE std_ulogic_vector IS ARRAY (natural RANGE <>) OF
std_ulogic

TYPE std_ulogic_vector IS ARRAY (natural RANGE <>) OF
std_ulogic
```

  - **Dessa är obegraänsade (*uncontrained*) array-typer - godtyckligt antal bitar kan användas**
  - **Då man verkligen ska använda arrayer måste antal element bestämmas**

```
SIGNAL mybus: std_logic_vector (7 DOWNTO 0);
```

eller

```
TYPE yourword IS ARRAY (7 DOWNTO 0) OF bit;
SIGNAL yourbus, hisbus, herbus: yourword;
```


Sammansatta typer

- **Två-dimensionell tabell**

```
TYPE table6x2 IS ARRAY (0 TO 5, 1 DOWNTO 0) OF bit;

CONSTANT mytable: table6x2 := ("00","01","10","11","01","01");
```

- **Tilldelning av värden i arrayer**

```
mybus <= "0011XXZZ";   -- Bit7='0' och Bit0='Z'
```

- **Bit-strängar för binära, oktala och hexadecimala tal**

```
X"A3"   -- = B"1010_0011" för en 8 bitars array
O"27"   -- = B"010_111" för en 6 bitars array
```

Sammansatta typer

- ***Record* typer**
- **Records används sällan. Det finns inga fördefinierade record typer**

```
TYPE myrec IS RECORD
  ctl: std_logic;
  inp: bit_vector (3 DOWNTO 0);
  outp: bit_vector (7 DOWNTO 0);
END RECORD;

SIGNAL device1, device2: myrec;
SIGNAL final: bit_vector (7 DOWNTO 0);

device1.ctl <= '1';
device2.ctl <= device1.ctl;
device2.inp <= "1011";
final <= device1.outp;
device1 <= device2;
```


Typer: Signed och unsigned

- **IEEE 1076.3 är syntesstandarden och definierar two *packages*; numeric_std och number_bit vilka**
  - **definierar nya typer för för signed och unsigned integers**
  - **overload operatorer för dessa typer - aritmetiska, relationer och logiska**
  - **definierar nya funktioner för dessa typer - funktioner för typkonvertering**

```
-- numeric_std
TYPE unsigned IS ARRAY (natural RANGE <>) of std_logic;
TYPE signed IS ARRAY (natural RANGE <>) of std_logic;

-- numeric_bit
TYPE unsigned IS ARRAY (natural range <>) of bit;
TYPE signed IS ARRAY (natural range <>) of bit;
```


Aliases

- **Alternativt namn för existerande objekt (eller del av objekt)**
- **Kan ge bättre läsbarhet och dokumentation**
- **Exempel:**

```
SIGNAL instruction: std_logic_vector (15 DOWNTO 0);

ALIAS opcode: std_logic_vector (5 DOWNTO 0)
              IS instruction (15 DOWNTO 10);

ALIAS operands: std_logic_vector (9 DOWNTO 0)
                IS instruction (9 DOWNTO 0);
```


Attribut

- **Tillhandahåller information eller karakteristik för en signal, variabel, typ, funktion, etc.**
- **Format: *ItemName'attribute***
- **Används som en konstant (*read-only* värde) på ställen som villkor och på högra sidan i en tilldelningssats**
- **Exempel på användbara attribut:**

Attribut	Betydelse
t'left	det värde som är längst till vänster i en typ
t'right	det värde som är längst till höger i en typ
t'low	det mista värdet i en typ
t'high	det största värdet i en typ
a'length	antal element i en array
a'range	området för en array (x TO y) eller (x DOWNTO y)
s'event	Sann (true) om en transition just har ägt rum på en signal

Attribut: Exempel

```
TYPE bitcount IS integer RANGE -3 TO +5;
TYPE opcode IS (Add, Sub, Jump, Call, Nop);
TYPE byte IS ARRAY (7 DOWNTO 0) OF std_logic;
SIGNAL clk: std_logic;
```

	bitcount	opcode	byte	clk
'left	-3	Add	7	
'right	+5	Nop	0	
'low	-3	Add	0	
'high	+5	Nop	7	
'length			8	
'range			7 DOWNTO 0	
'event				TRUE eller FALSE