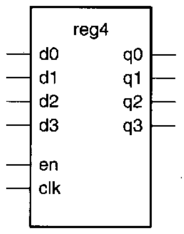
Reg4 Body

- an architecture body is a description of the internal implementation of an entity.
- Alternative entity implementations correspond to different architecture
  - ◇ more than one architecture may represent one entity
  - ◇ Behavioral architecture (process, concurrent statemets)
  - ◇ Strutural architecture (interconnections of gates)
  - ◇ Mixed architecture (both)

Example

- A four-bit register module reg4
  - ◇ Six inputs, dO, d1, d2, d3, en and clk.
  - ◇ Four outputs, qO, ql, q2 and q3.



- Design entity reg4 with 10 ports

```
entity reg4 is
    port ( d0,d1,d2,d3,en,clk:  in bit;
        q0,ql,q2,q3:out bit);
end entity reg4;
```

VHDL Concepts

VLSI CAD

Case Western Reserve University

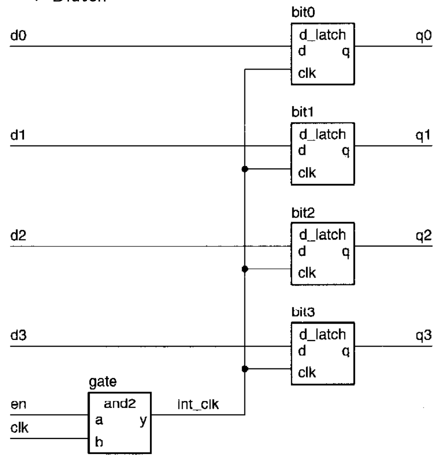
Dan Saab

Bahavior of D-latch

```
entity dlat is
    port (d, clk:  in bit; q out bit);
end dlat;
architecture basic of dlat is
begin
    latbehav:  process is
    begin
        if clk = '1' then
            q <= d after 2 ns;
        end if;
        wait on clk, d;
    end process latbehav;
end architecture basic;
```

Structure

- Two basic cells:
  - ◇ AND
  - ◇ Dlatch



Behavioral

```
architecture behav of reg4 is
begin
    storage:process is
    variable sd0, sd1, sd2, sd3:  bit;
    begin
        if en = '1' and clk = '1' then
            sd0 :=d0;
            sd1 :=d1;
            sd2 :=d2;
            sd3 :=d3;
        end if;
        q0 <= sd0 after 5 ns;
        ql <= sd1 after 5 ns;
        q2 <= sd2 after 5 ns;
        q3 <= sd3 after 5 ns;
        wait on d0, d1, d2, d3, en, clk;
    end process storage;
end architecture behav;
```

Testbench

- VHDL test bench consists of
  - ◇ An architecture body containing
  - ◇ An instance of the component to be tested
  - ◇ A Process that generate the sequence
- Use monitoring facility of the simulator to observe circuit output ports.

reg4 structural architecture body

```
architecture struct of reg4 is
signal cntclock:  bit;
begin
    bit0 : entity work.djatch(basic)
        port map (d0, cntclock, q0);
    bit1 :  entity work.djatch(basic)
        port map (d1,cntclock, q1);
    bit2 :  entity work.djatch(basic)
        port map (d2, cntclock, q2);
    bit3 :  entity work.djatch(basic)
        port map (d3, nt clk, q3);
    gate :  entity work.and2(basic)
        port map (en clk cntclock);
end architecture struct;
```

Bahavior of AND2

```
entity and2 is
    port(a,b:in bit; y:outbit);
end and2;
architecture basic of and2 is
begin
    and2beh:  process is
    begin
        y <= a and b after 2 ns;
        wait on a, b;
    end process and2beh;
end architecture basic;
```

Comments

- Code is written once but is read by
  - ◇ The authors
  - ◇ Other Engineer
- Help readers understand the VHDL model
  - ◇ Structure
  - ◇ Logic
- A comment extends from the two dashes to the end of the line
  - ◇ Example

    −− a descriptive comments

Lexical Elements

- Comments
- Identifier
- Special symbols
- Numbers
- Characters
- Strings and bit strings

Testbench

```
entity testbench is
end entity testbench;

architecture testreg4 of testbench is
signal d0, d1, d2, d3, en, clk, q0, q1, q2, q3 :  bit;
begin
    dut:  entity work.reg4(behav)
        port map (d0, d1, d2, d3, en, clk, q0, q1, q2, q3);
    stimulus :  process is
    begin
        d0<='1'; d1<='1'; d2<='1';d3<='1';
        en <= '0'; clk <= '0';
        wait for 20 ns;
        en <= '1'; wait for 20 ns;
        clk <= '1'; wait for 20 ns;
        d0<= '0' ; d1 <= '0' ; d2<= '0' ; d3<= '0'; wait for 20 ns;
        en <= '0'; wait for 20 ns;
        ...

        wait;
    end process stimulus;
end architecture testjeg4;
```

Reserved words or keywords

- Reserved for special use in VHDL
  - ◇ Examples: abs, entity next, nor, access, etc..

Examples

- Valid identifiers

```
Try try -- are the same name
Try_this Trythis -- are different
A
ZO
counter
next_index
\clk__int\ -- Extended identifier is enclosed between '\'
\Clk__int\ and \clk__int\ -- are different
```

- invalid identifiers

```
clk__int -- two successive underlines
a@value -- contains an illegal character for an identifier
5bit_reg -- starts with a non-alphabetic character
_A -- starts with an underline
A_ -- ends with an underline
```

Identifier

- Used to name items in a VHDL model
  - ◇ may only contain
    - alphabetic letters ('A' to 'Z' and 'a' to 'z')
    - decimal digits ('0' to '9')
    - underline character ('_')
- must start with an alphabetic letter
- may not end with an underline character
- may not include two successive '_' characters.

Characters and Strings

- A character literal is enclosed in single quotation marks.
  - ◇ Example:

```
'A' -- uppercase letter
'z' -- lowercase letter
',' -- the punctuation character comma
''' -- the punctuation character single quote
' ' -- the separator character space
```

- A string literal represents a sequence of characters. It is written by enclosing the characters in double quotation marks. It must be on written on one line.
  - ◇ Example:

```
"one string"
"The string may include any number of character(including 0)"
"" -- empty string
"C character:""char""." -- double quotation to include quotation mark
"Longer than one line string," &"can be concatinated"
```

Numbers

- Integer literals: An integer literal simply represents a whole number and consists of digits without a decimal point
  - ◇ Example:

```
10 23 0
40E6 1e+10
2#101# 16#fd# 8#023# -- base is between 2 and 16
2#1#e50 16#3#e3
```

- real literals: They always include a decimal point, which is preceded by at least one digit and followed by at least one digit.
  - ◇ Example:

```
23.1 0.0 3.14159
2.24E03 65.6E+11 34.0e-08
0.5 2#0.100# 8#0.4# 12#0.6# -- base is express in decimal
```

- underline character is included in long number to aid readability. It may not appear at the beginning or end of a number, nor may two appear in succession.

```
123_456 3.141_592_6 2#1111_1100_0000_0000#
```

Special Symbols

- Operator symbols
- Language construct and punctuation
  - ◇ One char symbols: & ( ) * + , - . / : < = > !
  - ◇ Two char symbols: <= ** := /= >= <= <>

Types

- A type is a set of values.
- The type of each object is static; it can not be changed during simulation.
  - ◇ Only certain operators are defined for a type.
- Commonly used types and subtypes are pre-defined:
  - ◇ bit, bit_vector
  - ◇ character, string
  - ◇ integer
  - ◇ real, time
- Additional types can be constructed using pre-defined types.

Objects

- Objects are containers of values (VHDL):
  - ◇ constant : have a single value may not change
  - ◇ variables : have a single which can be change
  - ◇ files: contains sequence of values that can be read or written
- Signals: have a past history of value, a present value, a a set of projected future values: only future values can be changed by assignment.
- All objects are of some types.

Bit Strings

- Includes values that represents bits. This is represented by a string of digits, enclosed by double quotation mark and preceded by character that specifies the base (B for binary, O for octal, X for hexadecimal)
  - ◇ Example:

```
B"010101" B"01" b"111_001_10101"
O"372" B"011_111_010"
o"01" B"000_001"
X"FF" B"1111_1111"
x"af" B"1010_1111"
```

Example

```
architecture EXMPL of TYPE_TIME is
signal clock :  bit := '0';

signal A, B, Z : integer range 0 to 3;
constant PERIOD : time := 50 ns;

begin
    A <= 2;
    B <= 3;
    process
    begin
        wait for 100 ns;
        Z <= A;
        wait for PERIOD*2 ;
        Z <= B;
        wait for PERIOD * 5.5;
    end process;

    clock <= not clock after PERIOD/2 ;
end EXMPL;
```

'time' Datatypes

- Time units
  - ◇ fs, ps, ns, us, ms, sec, min, hr
- Can be divided or multiplied by a real or integer
- Specify delay and manipulate simulation time

STANDARD Data Types

- type BOOLEAN is (FALSE,TRUE);
- type BIT is ('0','1');
- type CHARACTER is (– ascii set);
- type INTEGER is range – implementation
- type REAL is range – implementation