# Contents

- The synthesis process
- VHDL as a standard for synthesis
- Synthesis aspects and guidelines
- Combinational synthesis
- Sequential synthesis

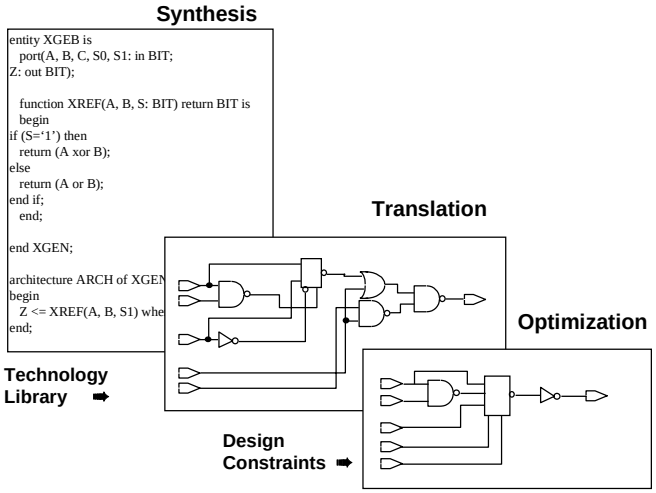




# Synthesis Process - 1

- Process of turning an abstract, technology independent, text description of a design into gates.
- Two crucial steps:
  - Translation
    acts as the automated bridge between two levels of abstraction
  - Optimization
    technology-specific design transformation to meet user's goals for the given design.

# Synthesis Process - 2








# Synthesis Process - 3

- Translation
  - In general, from an algorithmic-like description of the behavior to the register transfer level (RTL) to gate level.
- Optimization
  - Introduction of user's constraints.
  - Significant aspects: performance, area and test.

# Synthesis Process - 4

- The logic synthesis process begins with a validated behavioral design description.
- The design is translated into a design composed of:
  - control part, and
  - data path
- At the RTL level, the design description has been partitioned down to the implementation primitives of ALUs, ROMs, RAMs, data-path and control functions.





# Synthesis Process - 5

- A second component to synthesis design policy in the VHDL style that has been adopted for the specification of the device.
- Different VHDL descriptions of the same functionality can yield radically different results.
- The more the VHDL description yields an implementation (structural description style) the easier the translation phase is; the higher the abstraction level, the more fundamental the translation phase is.

# Synthesis Process - 6

- VHDL was aimed at simulation purposes.
  - The semantic is uniquely defined.
    - The same code processed with two different commercial simulators will provide the same results.
    - The simulation task requires no "interpretation" of the code.
  - The semantics of VHDL is expressed in terms of a canonical simulator and not in terms of equivalent hardware constructions.





# Synthesis semantics

- VHDL lacks of synthesis semantics.
  - The synthesis task requires an "interpretation" of the code to determine which hardware device can realize the desired function.
- Being synthesis a difficult task, commercial tools usually restrict the set of accepted statements to a tool-dependent subset.
  - No portability
  - No reusability

# Synthesis semantics

- To obtain satisfable results from the synthesis process, the user has to 'deeply' know the commercial tool.
- A basic idea of what VHDL constructs lead to is also required.
- A "Guide to VHDL synthesized constructs" is useful for selecting the appropriate constructs and having a general feeling of the possible results.

Level-0 VHDL Synthesis Subset





# Synthesis semantics

- VHDL Synthesis Subset
  - definition of a stable VHDL synthesis usage guide
  - allow portability
  - interchange format between tools, designers, companies …
  - allow VHDL to be used as an RT/logic specification language
  - facilitate reusability of descriptions

# Synthesis semantics

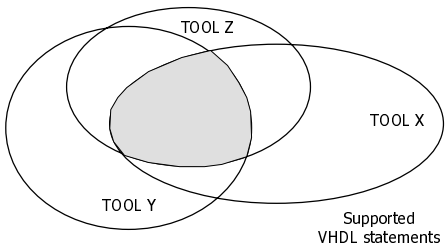
- Stable VHDL synthesis usage guide
  - It is important to understand how the tool interprets the VHDL source code
  - It is useful to agree on what the expected results of the synthesis of the VHDL source code should be:
    - less confusion
    - no misunderstanding





# Synthesis semantics

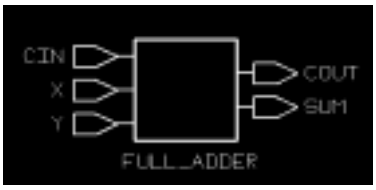
- VHDL portability




- Usually there are several possible styles to describe a behavior:
  - It is advisable to select the statement which is more widely adopted.

# VHDL styles

- Different “styles” describing the same behavior may lead to different realizations.
- An example: Full Adder
  - behavioral
  - data flow
  - structural








# VHDL styles: behavioral - 1

```
architecture FA_BEH of FULL_ADDER is
begin
   process (X, Y, CIN)
      variable BV: BIT_VECTOR(1 to 3);
      variable NUM: INTEGER range 0 to 3;
      variable STemp, CTemp: BIT;
    begin
          NUM := 0;
          BV := X & Y & CIN;
          for I in 1 to 3 loop
                if (BV(I) = '1') then
                      NUM := NUM + 1;
                end if;
          end loop;
```

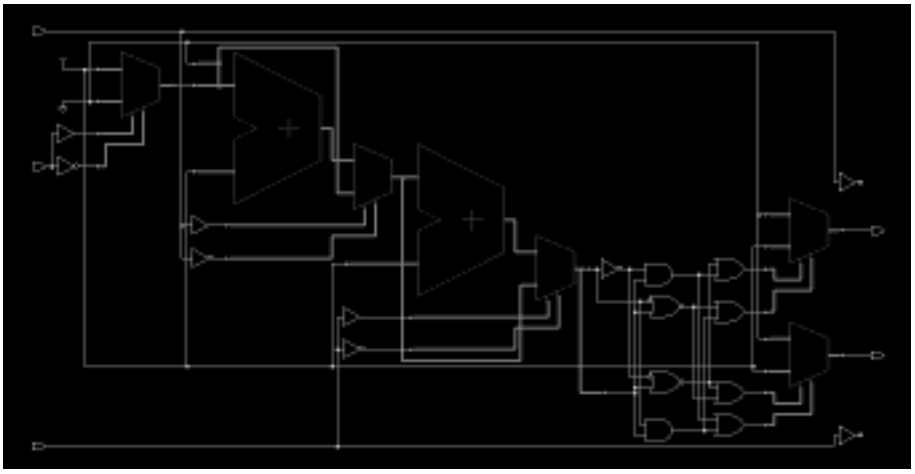
# VHDL styles: behavioral - 2

```
    case NUM is
              when 0 => CTemp:='0'; STemp:='0';
              when 1 => CTemp:='0'; STemp:='1';
              when 2 => CTemp:='1'; STemp:='0';
              when 3 => CTemp:='1'; STemp:='1';
           end case;
           SUM <= STemp ;
           COUT <= CTemp ;
       end process;
end FA_BEH;
```

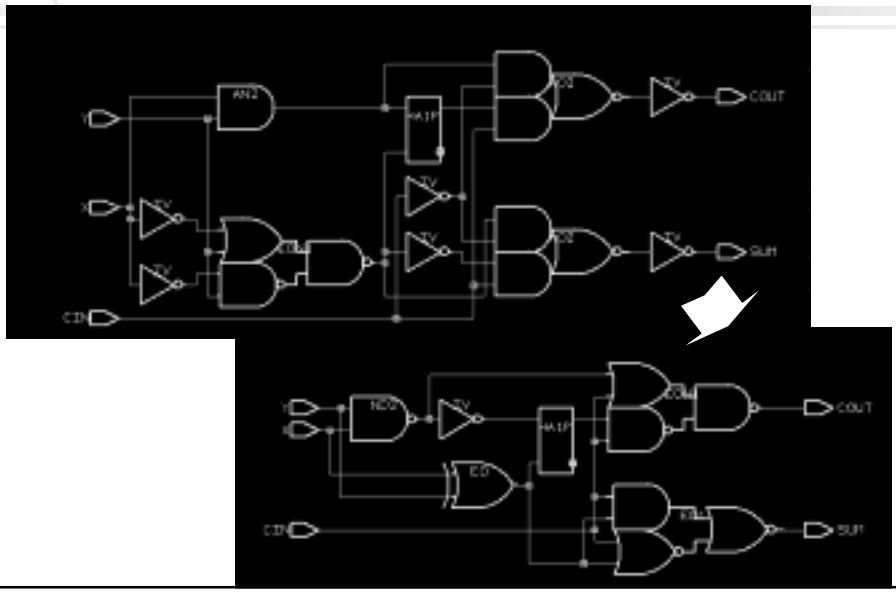




# VHDL styles: behavioral - 3



# VHDL behavioral - 4

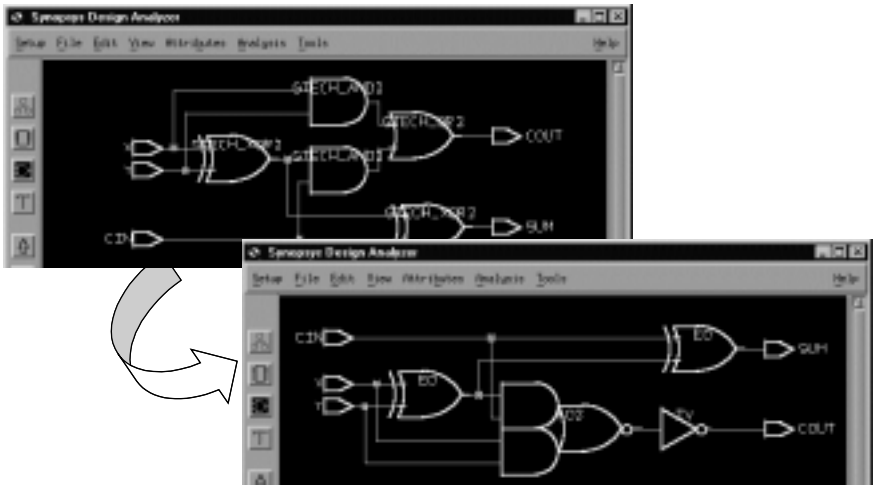





# VHDL styles: data flow - 1

```
architecture FA_BOOL of FULL_ADDER is
  signal S1, S2, S3: BIT ;
begin
      S1 <= X xor Y ;
      SUM <= S1 xor CIN after 1 ns;
      S2 <= X and Y ;
      S3 <= S1 and CIN after 1 ns;
      COUT <= S2 or S3 after 1 ns;
end FA_BOOL;
```

# VHDL styles: data flow - 2



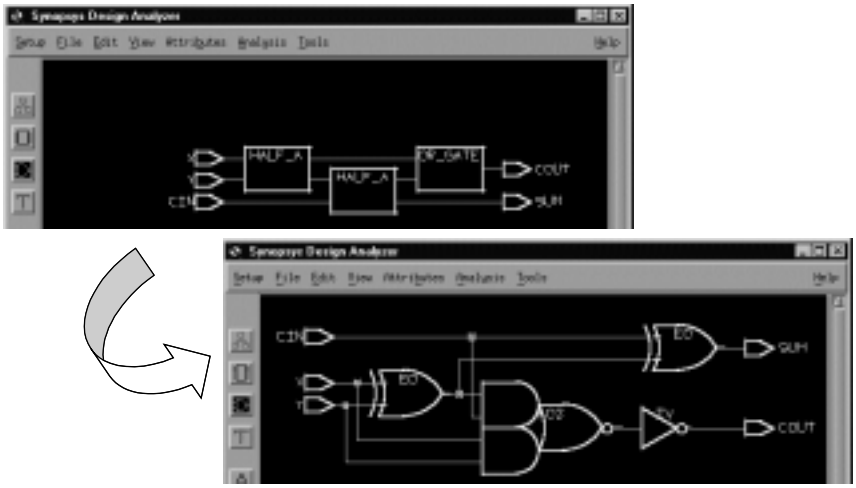





# VHDL styles: structural - 1

```
architecture FA_ST of FULL_ADDER is
     component HALF_A
        port(A, B: in BIT;
             S, C: out BIT);
     end component;
     component OR_GATE
        port(A, B: in BIT;
             O: out BIT);
     end component;
     signal C1, S1, C2: BIT;
begin
  HA1: HALF_A port map (A=>X, B=>Y, S=>S1, C=>C1);
  HA2: HALF_A port map (S1, CIN, SUM, C2);
  OR1: OR_GATE port map (C1, C2, COUT);
end FA_ST;
```

# VHDL styles: structural - 2








# Some considerations

- General aspects of some elements in relation with VHDL synthesis:
  - Vectors
  - Arithmetic, Logical & Relation operands
  - Subtypes, slices and functions

# VHDL: vectors

```
entity ex1 is
  port(A, B, C: in bit_vector(1 to 5);
       Z: out bit_vector(1 to 5));
end ex1;
architecture arch of ex1 is
begin
  process(A,B,C)
      variable TEMP: bit_vector(1 to 5);
  begin
      TEMP := A and B;
      Z <= TEMP xor C;
  end process;
end;
```

Bit-to-bit operations

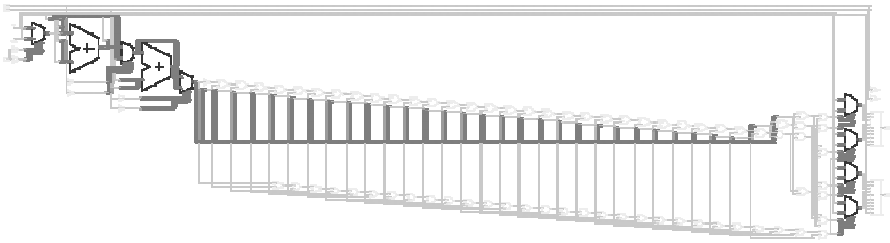




# VHDL: Turning Integer into Vectors

- Declare range for INTEGER objects for implementation of the minimum possible bits.
- If a signal (or port) or variable is declared to be of type INTEGER without specifying a range, a tool may implement the object using a full 32 bits.
  - The automatic tool warns the user of an unconstrained signal/variable declaration

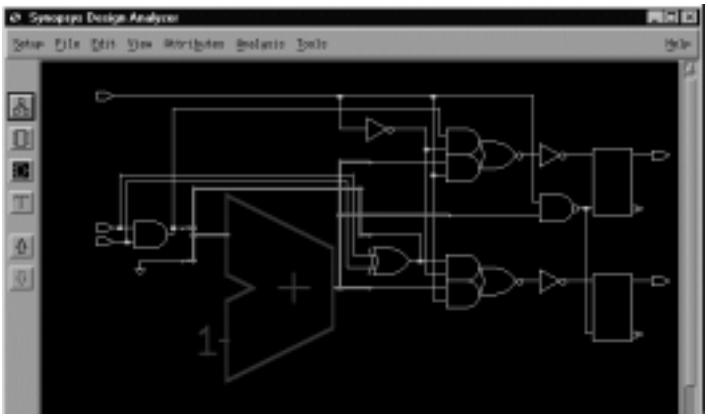
# VHDL: Turning Integer into Vectors - 1







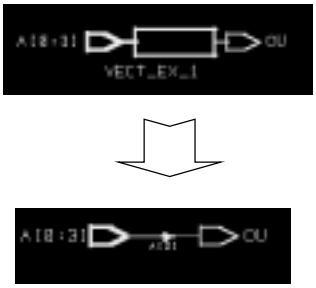
# VHDL: Turning Integer into Vectors - 2



# VHDL: vector indexes - 1

```
entity EX is
   port(A: in BIT_VECTOR(0 to 3);
        OU : out BIT) ;
end EX ;

architecture EX_1 of EX is
begin
   OU <= A(2) ;
end EX_1 ;
```








# VHDL: vector indexes - 2

```
entity EX_VAR is
  port(A : in BIT_VECTOR(0 to 7) ;
       INDEX : in INTEGER range 0 to 7;
       OUTPUT : out BIT) ;
end EX_VAR ;
architecture EX_1 of EX_VAR is
begin
     OUTPUT <= A(INDEX) ;
end EX_1 ;
```