VHDL Subprograms

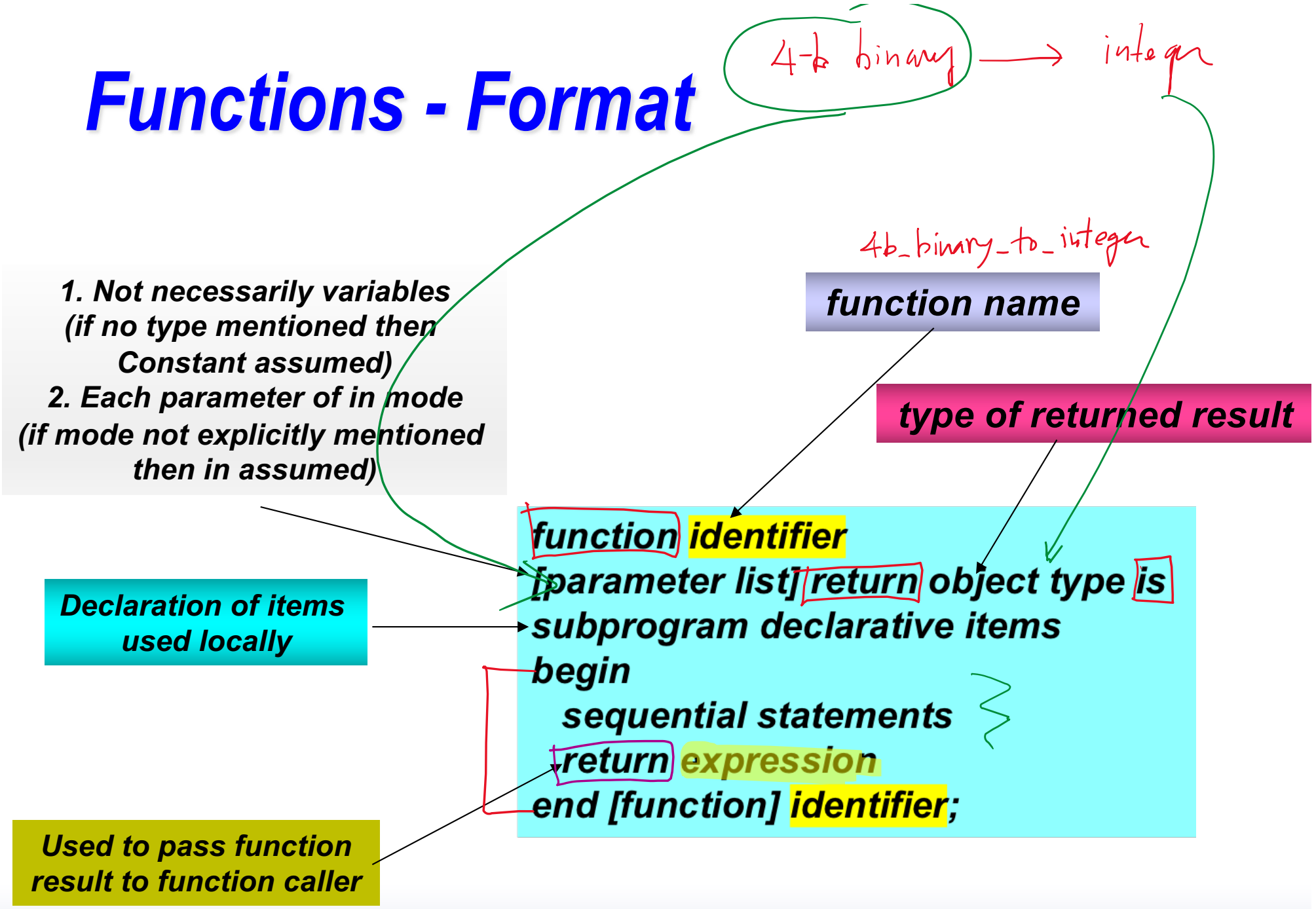
Subprograms

- Two types of subprograms
  - **Function**
    - Called within expression
    - Compute and return "single" output value
  - **Procedure**
    - Can be sequential or concurrent
    - Useful in partitioning large behavioral VHDL codes
    - Compute and return "zero or more" output values

Return Statement

- Operations in subprograms executed by sequential statements
- Subprograms finished by return statement
  - Termination of subprogram
  - Return control over program execution to calling program
  - Functions – require return statement
    - Value of expression returned to calling program

Functions - Format



Argument Class of Parameters for Subprograms

- Every parameter of subprogram has associated argument class
- Argument class
  - Determines what information is passed to subprograms
  - Determines whether passed parameter is constant, signal or variable
    - Three kinds of argument classes:
      - Constants, variables and signals

Example


```
library IEEE;
use IEEE.std_logic_1164.all;
entity dff is
  port(d, clock: in std_logic; q, qbar: out std_logic);
end entity dff;
architecture beh of dff is
  function rising_edge(signal clock:std_logic) return Boolean is
    variable edge: Boolean:=FALSE;
  begin
    edge:= (clock = '1' and clock'event);
    return (edge);
  end function rising_edge;
begin
  output: process is
  begin
    wait until (rising_edge(clock));
    q <= d after 5 ns;
    qbar <= not d after 5 ns;
  end process output;
end architecture beh;
```

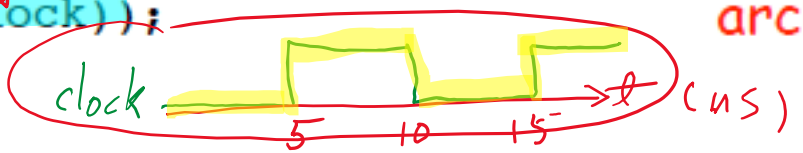
function name

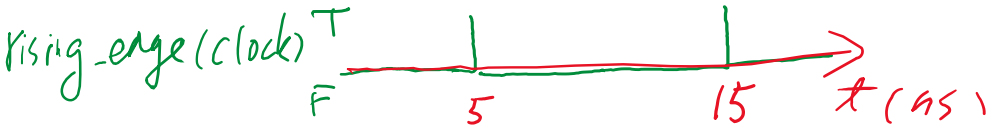
parameter list

returned data type

declarative region of architecture

no sensitivity list





Example

- A function to convert *bit_vector* to *integer*

```
FUNCTION to_integer (bin : BIT_VECTOR) RETURN INTEGER IS
  VARIABLE result: INTEGER;
BEGIN
  result := 0;
  FOR i IN bin'RANGE LOOP
    IF bin(i) = '1' THEN
      result := result + 2**i;
    END IF;
  END LOOP;
  RETURN result;
END to_integer;
```

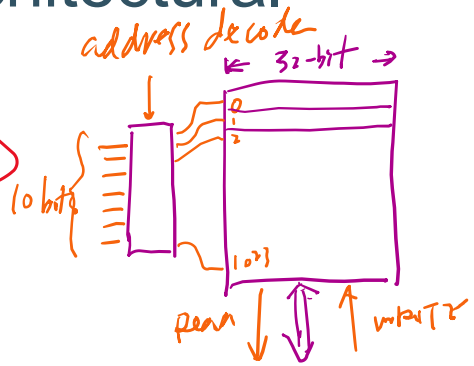
Functions have declarative and body parts

- *to_integer(x)* is an integer
  - Can be used anywhere an integer is expected

```
variable y: integer;
variable x: bit_vector(0 to 7);
y := to_integer(x) + 5;
```

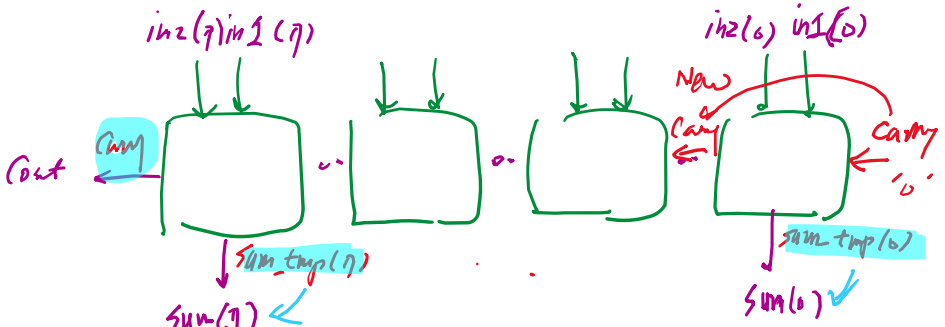
Procedures

- Allowing on modularization of large architectural bodies
- Returning zero or more output values
  - **In:** only readable by procedure
  - **Out:** only writable
  - **Inout:** both readable and writable
  - Returning parameters achieved through using parameters of **out** and **inout**
  - Procedures may change value of one or more parameters passed to it



Example: Procedure for Addition of Two Unsigned Numbers

```
procedure add_unsigned(in1,in2: in bit_vector(7 downto 0);
                       sum: out bit_vector(7 downto 0); Cout: out boolean) is
  variable sum_tmp: bit_vector(7 downto 0);
  variable carry: bit := '0';
begin
  for i in sum_tmp'reverse_range loop
    sum_tmp(i) := in1(i) xor in2(i) xor carry;
    carry := (in1(i) and in2(i)) or (carry and (in1(i)) or (carry and (in2(i))));
  end loop;
  sum := sum_tmp;
  Cout := carry = '1';
end procedure add_unsigned;
```

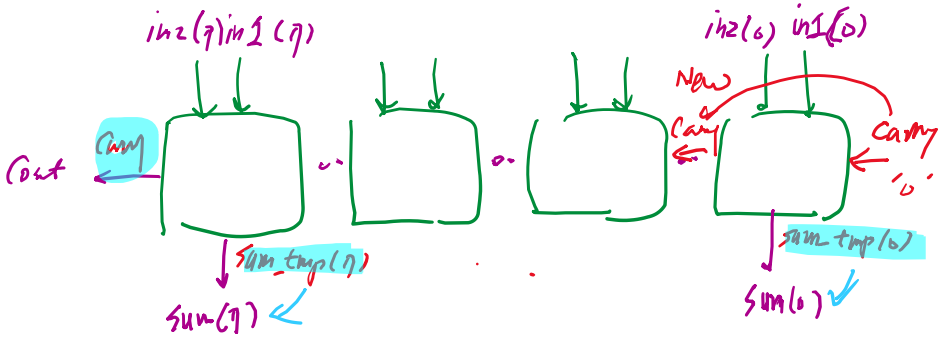


Example: Procedure for Addition of Two Unsigned Numbers

no initial carry_in

```
procedure add_unsigned(in1,in2: in bit_vector(7 downto 0);
                       sum: out bit_vector(7 downto 0); Cout: out bit) is
  variable sum_tmp: bit_vector(7 downto 0);
  variable carry: bit := '0';
begin
  for i in sum_tmp'reverse_range loop
    sum_tmp(i) := in1(i) xor in2(i) xor carry;
    carry := (in1(i) and in2(i)) or (carry and (in1(i)) or (carry and (in2(i)));
  end loop;
  sum := sum_tmp;
  Cout := carry;
end procedure add_unsigned;
```

sum_tmp REVERSE_RANGE = 0 to 7



Wait Statement in Procedures

- Wait statement allowed in procedure
  - "Wait" is forbidden in functions
- Functions used to compute values to be available instantaneously
  - Cannot wait
    - Not allowed to call procedure with wait statement from within function body