

VHDL

VHDL is a hardware description language.

- You can describe hardware at the level of individual gates and connections: **netlist**
- You can describe hardware as the flow of information between registers: **dataflow**
- You can describe hardware as the behavior of the circuit in response to inputs: **behavioral**

VHDL is a programming language

- There are variables that can hold values
- There are functions that take input and output parameters
- There are control structures that manage the sequence of operations

But,

All elements of the language have a connection with a piece of hardware.

Why do you want a hardware description language?

- To simulate hardware before implementing it
- To make design entry simpler than moving boxes and connecting wires
- To simplify verification and testing
- To simplify communication between different tools in a CAD environment
- To permit a standard for specification of inputs, outputs, and behavior of digital circuits

VHDL Structure

There are two types of digital circuits you can make: combinational and sequential

- Combinational: the output is a function of only the input variables (no memory).
- Combinational: the circuit does not have a clock or any clocked signals.
- Sequential: the output is a function of the current state (memory) and the inputs.
- Sequential: the circuit has something that functions as a clock and clocked signals.

The structure of a VHDL program

- The entity structure defines the input and output characteristics of the circuit
- The architecture structure defines the function of the circuit.

Example: Dataflow design for a 2-4 decoder that takes as input a binary number in [0, 3] and raises the specified output line high.

```
library ieee;                -- use the IEEE library
use ieee.std_logic_1164.all; -- specify which package in the library to include

entity V2to4dec is
  port ( EN: in std_logic;
        I: in std_logic_vector(0 to 1);
        Y: out std_logic_vector(0 to 3) );
end V2to4dec;

architecture dataflow of V2to4dec is
begin
  Y[0] <= EN when I = "00" else '0';
  Y[1] <= EN when I = "01" else '0';
  Y[2] <= EN when I = "10" else '0';
  Y[3] <= EN when I = "11" else '0';
end dataflow;
```

Example: Up-down binary counter circuit that has a reset, clock, and output signals

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity TheRings is
  port( reset, clock, direction: in std_logic;
        value: out std_logic_vector(7 downto 0) );
end TheRings;

architecture Lord of TheRings is
  signal Q: unsigned(7 downto 0);
begin
  process (reset, clock) begin
    if reset = '1' then
      Q <= "00000000";
    elsif clock = '1' and clock'event then
      if direction = '0' then
        Q <= Q + 1;
      else
        Q <= Q - 1;
      end if;
    end if;
  end process;

  value <= std_logic_vector(Q);
end Lord;
```

Things you need to know

- VHDL is strongly typed. Certain types can do certain things.
- Use the built-in type casting functions to convert from, for example, unsigned to std_logic_vector.
- To create a vector of bits, use quotes
- To specify a single value like '0' or '1', using single quotes
- To initialize different types, sometimes you use integers, sometimes you use strings
- Use the concatenation operator, &, to mess around with individual bits in a string
- The **clock = '1' and clock'event** is just a fancy way of saying “rising edge”.

State machine design

You state machines should all follow the same pattern, always.

```
entity three_consecutive is
  port(clk, r, x: in std_logic; Z: out std_logic);
end three_consecutive;

architecture moore of three_consecutive is
  type state is (nozeros, onezero, twozeros, threezeros);
  signal fsm_state: state := nozeros;
begin
  -- implement the state logic as a case statement
  process (clk, r, x) begin
    if r = 1 then
      fsm_state <= nozeros;
    elsif clk = 1 and clkevent then
      case fsm_state is
        when nozeros =>
          if x = 0 then fsm_state <= onezero;
          else fsm_state <= nozeros;
          end if;
        when onezero =>
          if x = 0 then fsm_state <= twozeros;
          else fsm_state <= nozeros;
          end if;
        when twozeros =>
          if x = 0 then fsm_state <= threezeros;
          else fsm_state <= nozeros;
        when others =>
          if x = 0 then fsm_state <= onezero;
          else fsm_state <= nozeros;
        end case;
      end if;
    end process;
    -- implement the output logic as a simple conditional signal assignment
    Z <= 1 when fsm_state = threezeros else 0;
  end moore;
```

VHDL is not a functional programming language

The following code snippets may seem like they do the same thing:

```
architecture A of C is
  signal x, y: std_logic;
begin
  x <= input_value;
  y <= x;
  output_value <= y;
end A;
```

```
architecture B of C is
  signal x, y: std_logic
begin
  process(trigger) begin
    x <= input_value;
    y <= x;
    output_value <= y;
  end process;
end B;
```

All statements outside of a process are called **concurrent statements**

- They are all running at the same time
- Any time the right side of a concurrent assignment statement changes, the statement executes
- Think of the assignment statement and the two variables on either side as all a single wire

All statements inside of a process are called **sequential statements**

- The assignments do **not** take place before the next line is executed
- The assignments are placed into a queue, with their current values in the order in which they are read
- The actual assignments are made at the end of the process statement
- The process statement only executes when the trigger changes

The cool thing about a hardware description language:

```
architecture B of C is
  signal x, y: std_logic
begin
  process(trigger) begin
    if trigger = '1' and trigger'event then
      x <= y; -- this works fine because the current value of y is stored in the queue
      y <= x; -- the current value of x is stored in the queue
    end if;
  end process; -- now both assignments take place simultaneously
end B;
```

Maxwell's Cardinal Rules of VHDL

1. Never use variables; always use signals
2. Never use for loops
3. Sequential circuits should always follow the state machine structure
4. Give yourself output signals that tell you what is going on
5. Use the **VHDL template** function in Quartus