Design Rules

A VHDL process:
- describes a hardware structure
- describes what happens in one clock cycle
- this happens in every clock cycle

Design Rules:
- synchronous design (flip flops)
- no latches
- no logic in the clock path

Partly Synthesisable Constructs

For, while:
- the number of iterations must be a compile time constant. All loops will be unrolled and calculated in one clock cycle.
- for all other repetitions, use a state machine (any multi cycle computation)

Variable:
- a value used must have been created in the same clock cycle (temporary result in a calculation)

```
tmp := foo();
a <= tmp;
b <= bar(tmp);
tmp := a*b;
result <= tmp(32 to 63);
```
Separate Comb. And Sync. Behaviour

comb: process(data_reg, acc_reg)
begin
  acc_next <= data_reg + acc_reg;
end process;

seq: process(clk)
begin
  if rising_edge(clk) then
    acc_reg <= acc_next;
  end if;
end process;

Reset and Chip Enable

seq: process(clk, reset_async)
begin
  if reset_async = '0' then
    acc_reg <= (others => '0');
  elsif rising_edge(clk) then
    if ce = '1' then
      acc_reg <= acc_next;
    end if;
  end if;
end process;

Sequential Processes

a record may not contain
• clock
• reset
Combinatorial Processes

**may not have any memory**
- all signals used must be in the sensitivity list
- any use of a variable must have a reaching assignment from this clock cycle
- assign a signal in all possible control paths

A use of a variable is when it appears on the right-hand side of an assignment or in a condition.
If you make a misstate there will be a warning about latches

Assign In All Possible Ctrl Paths

```vhdl
process(a_reg, b_reg)
begin
  if(a_reg < b_reg) then
    result_next <= foo(a_reg, b_reg);
  end if;
end process;
```

**ERROR !!!**

```vhdl
process(a_reg, b_reg)
begin
  if(a_reg < b_reg) then
    result_next <= foo(a_reg, b_reg);
  end if;
end process;
```

Assign In All Possible Ctrl Paths
Assign In All Possible Ctrl Paths

Divide the process into three parts:
- default values
- computation
- output

```vhdl
process(a_reg, b_reg, result_reg)
variable result : std_logic_vector(0 to 31);
begin
  result := result_reg;
  if (a_reg < b_reg) then
    result := foo(a_reg, b_reg);
  end if; -- any use of result will use its new value
  result_next <= result;
end process;
```

Use a Record To Contain All Output Signals

```vhdl
process(r_reg)
variable r : my_record_type;
begin
  r := r_reg;
  case r.state is
  when S0 => r.a := foo(r.a);
    r.state <= S1;
  when others => null;
  end case;
  r_next <= r;
end process;
```

Synchronous Reset

```vhdl
process(a_reg, b_reg)
variable r : my_record_type;
begin
  r := r_reg;
  if reset = '0' then
    r.state := S1;
    r.a = (others => '0');
  end if;
  r_next <= r;
end process;
```