FPGA Design with VHDL

Justus-Liebig-Universität Gießen,
II. Physikalisches Institut

Ming Liu

Dr. Sören Lange

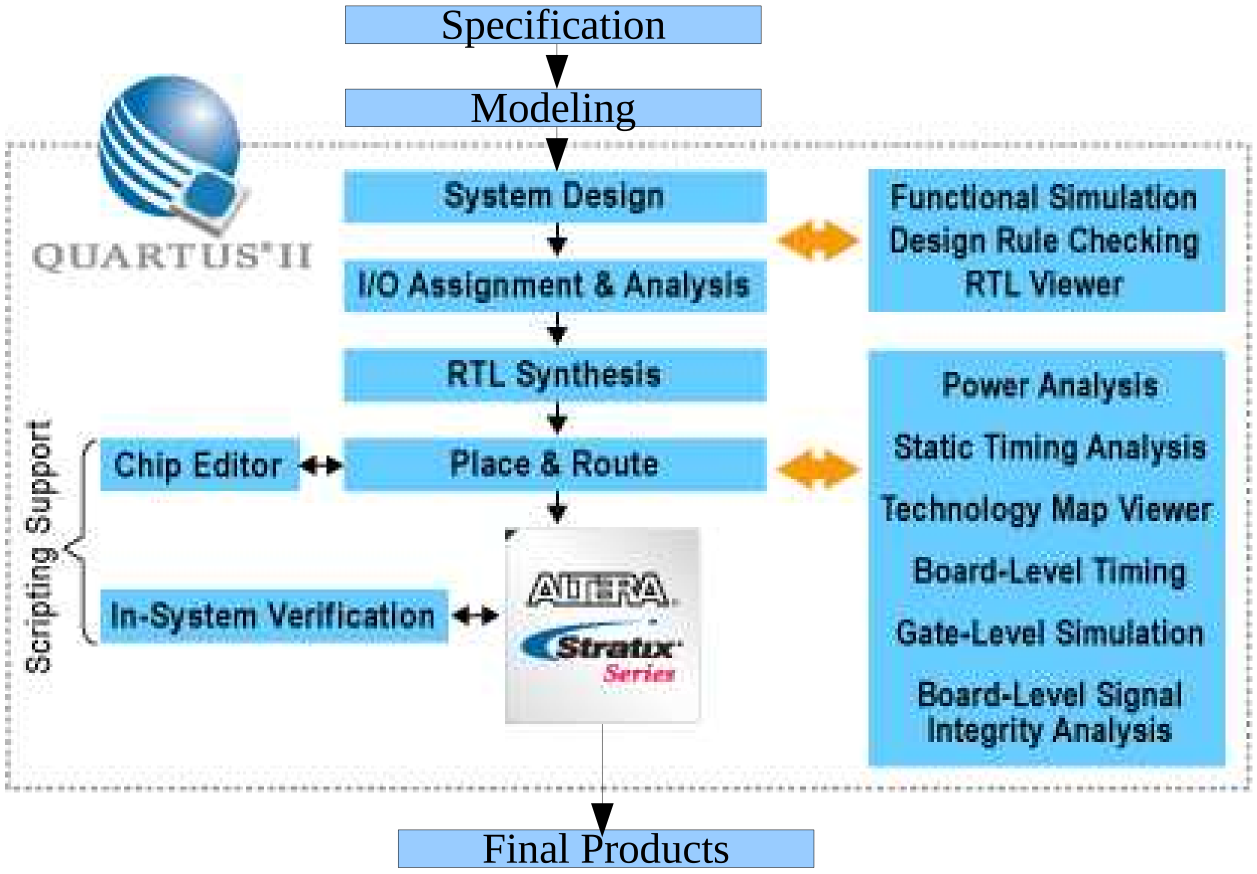
Prof. Dr. Wolfgang Kühn

ming.liu@physik.uni-giessen.de

Lecture 2

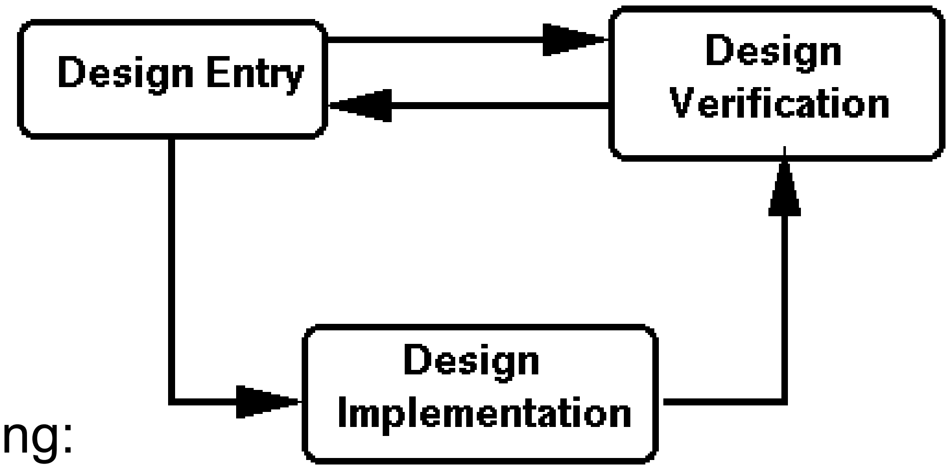
- FPGA Development Flow
- VHDL Basics
  - Concurrent Design
  - Sequential Design
  - Coding Style Issues

FPGA Development Flow



- Specification & modeling (algorithm investigation with high-level languages)
- System design (coding)
  - Hardware description (HDL)
  - Simulation (functional simulation or pre-simulation)
- Constraints (timing, location assignments, ...)
- Synthesis & Implementation (timing simulation or post-simulation)
- System verification & final products

Simplified Development Flow



- Design Entry:
  - Create your design files using:
    - hardware description (Verilog, VHDL)
- Design “implementation” on FPGA:
  - *Synthesis, map, place, and route* to create bit-stream file
  - Divide into CLB-sized pieces, place into blocks, route to blocks
- Design verification:
  - Use Simulator to check functionality
  - check max clock frequency
  - Load into FPGA device (cable connects PC to board)
    - check operation at full speed in real environment.

Related Softwares

- C or SystemC compilers for modeling (for complicated algorithms, can be neglected for simple designs)
- Editors for coding: Emacs, etc..
- Simulation software: Modelsim
- Synthesis & Implementation software: Xilinx ISE
- Other useful tools in the Xilinx software package, such as EDK, Chipscope, FPGA editor, ...

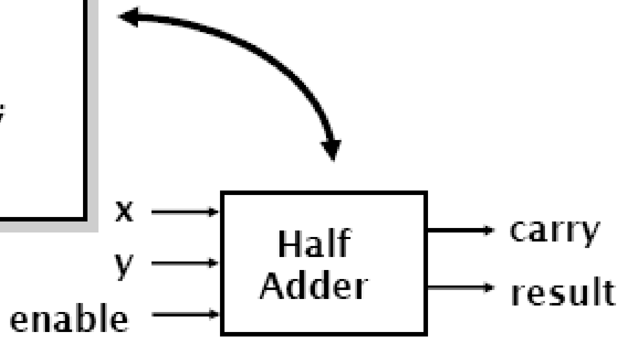
VHDL Basics

- VHSIC Hardware Description Language

  (Very High Speed Integrated Circuit)
- Synthesizable (for implementation) & unsynthesizable subset (for simulation)
- Detailed syntax:

Entity Declaration

```
ENTITY half_adder IS

   PORT( x, y, enable: IN bit;
         carry, result: OUT bit);

END half_adder;
```



• An entity declaration describes the interface of the component
• PORT clause indicates input and output ports
• An entity can be though of as a symbol for a component

VHDL Basics

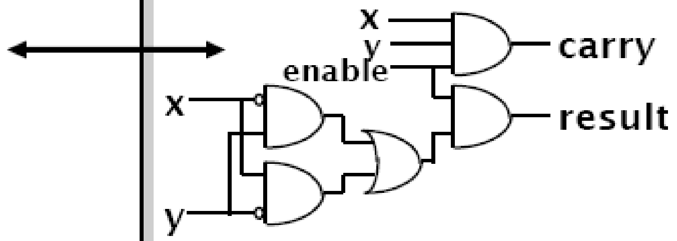
Port Declaration

```
ENTITY test IS
    PORT( name : mode data_type);
END test;
```

- PORT declaration establishes the interface of the object to the outside world
- Three parts
  - Name
  - Mode (in, out)
  - Data type (std_logic, std_logic_vector, ...)

Architecture Declaration

```
ARCHITECTURE behave OF half_adder IS
BEGIN
    PROCESS (enable, x, y)
    BEGIN
      IF (enable = '1') THEN
        result <= x XOR y;
        carry  <= x AND y;
      ELSE
        carry  <=  '0';
        result <=  '0';
      END IF;
    END PROCESS;
END behave;
```



- Architecture declarations describe the operation of the component
- Many architectures may exist for one entity, but only one may be active at a time
- An *architecture* is similar to a *schematic* of the component

VHDL Basics

Generics

- Generics allow the component to be customized upon instantiation
- Generics pass information from the entity to the architecture
- Common uses of generics
  - Customize timing
  - Alter range of subtypes
  - Change size of arrays

Example 1:

```
entity user_logic is
   generic (DATA_WIDTH : integer := 16);
   port(
      data_in : in std_logic_vector(DATA_WIDTH – 1 downto 0);
      ......
   );
end entity user_logic;

architecture arc of user_logic is
   signal counter : std_logic_vector(DATA_WIDTH – 1 downto 0);
......
end arc;
```

Example 2:

- The GENERIC MAP is similar to the PORT MAP in that it maps specific values to generics declared in the component

```
PACKAGE my_stuff IS
   COMPONENT and_gate
     GENERIC ( tplh, tphl : time);
     PORT ( in1, in2 : IN BIT; out1 : OUT BIT);
   END COMPONENT;
END my_stuff;

USE Work.my_stuff.ALL;
ARCHITECTURE test OF test_entity
     SIGNAL S1, S2, S3 : BIT;
BEGIN
   Gate1 : my_stuff.and_gate
     GENERIC MAP (2 ns, 3 ns)
     PORT MAP (S1, S2, S3);
END test;
```

VHDL Basics – Concurrent Design

Signal Declaration

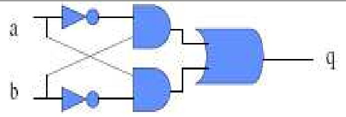
```
SIGNAL signal_name : type_name [:=value];

SIGNAL brdy : BIT;
SIGNAL output : INTEGER := 2;
```

- Signals are used for communication between components
- Signals can be seen as real, physical signals
- Some delay must be incurred in a signal assignment

Signal Assignment

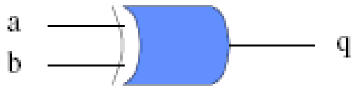
```
ARCHITECTURE signals OF test IS
   SIGNAL a, b, c, out_1, out_2: BIT;
BEGIN
   out_1 <= a NAND b;
   out_2 <= out_1 XOR c;
END signals;
```

- Signals can be assigned either in the behavioral style or in the structural style



```
u1: inverter port map (a,ai);

u2: inverter port map (b,bi);

u3: and_gate port map (ai,b,t3);

u4: and_gate port map (bi,a,t4);

u5: or_gate port map (t3,t4,q);
```

XOR-gate



```
q <= a xor b;
```

VHDL Basics – Concurrent Design

IF- vs CASE-statement Syntax

```
if (a='1') then
      q <= '1';
elsif (b='1') then
      q <= '1';
else
      q <='0';
end if;
```

```
case (a&b) is
      when "00" =>
            q <= '0';
      when others =>
            q <= '1';
end case;
```

FOR- vs WHILE-statement Syntax

```
for i in 0 to 9 loop
      q(i) <= a(i) and b(i);
end loop;
```

For is considered to be a combinational circuit by some synthesis-tools. Thus, it cannot have a wait statement to be synthesised.

```
i:=0;
while (i<9) loop
      q <= a(i) and b(i);
      WAIT ON clk UNTIL clk='1';
end loop;
```

While is considered to be an FSM by some synthesis-tools. Thus, it needs a wait statement to be synthesised

WAIT-statement Syntax

- The wait statement causes the suspension of a process statement or a procedure
- wait [sensitivity_clause] [condition_clause] [timeout_clause] ;
  - sensitivity_clause ::= on signal_name { , signal_name }

    wait on CLOCK;
  - condition_clause ::= until boolean_expression

    wait until Clock = '1';
  - timeout_clause ::= for time_expression

    wait for 150 ns;

VHDL Basics – Concurrent Design

Sensitivity-lists vs Wait-on-statement

```
Summation:
  PROCESS( A, B, Cin)
  BEGIN
    Sum <= A xor B xor Cin;
END PROCESS Summation;
```

=

```
Summation:  PROCESS
  BEGIN
    Sum <= A xor B xor Cin;
        WAIT ON A, B, Cin;
END PROCESS Summation;
```

If you put a sensitivity list in a process, you can't have a wait statement!

If you put a wait statement in a process, you can't have a sensitivity list!

Concurrent Process Equivalents

•All concurrent statements correspond to a process equivalent

```
U0:q <= a xor b after 5 ns;
```

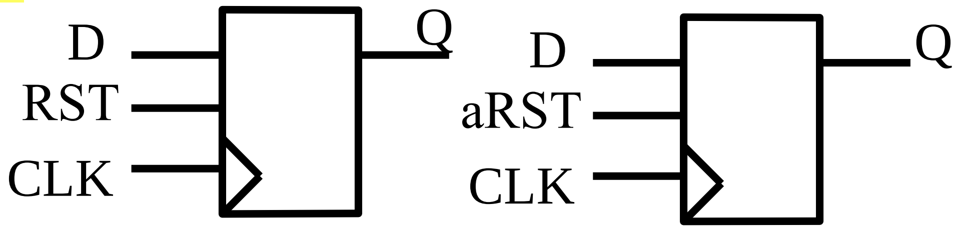
is a short hand notation for

```
U0:process
begin
     q <= a xor b after 5 ns;
     wait on a,b;
end process;
```

VHDL Basics – Sequential Design

Flip-flop

```
process(clk)    -- synthesis might complain that d is not listed
begin
        if (clk='1') and clk'event then
                q<=d;
        end if;
end process;
```



D-flipflop with synchronous reset

```
process(clk)    -- synthesis might complain that neither d nor
                -- reset is listed
begin
        if (clk='1') and clk'event then
                if (reset='1') then
                        q<='0';
                else
                        q<= D;
                end if;
        end if;
end process;
```

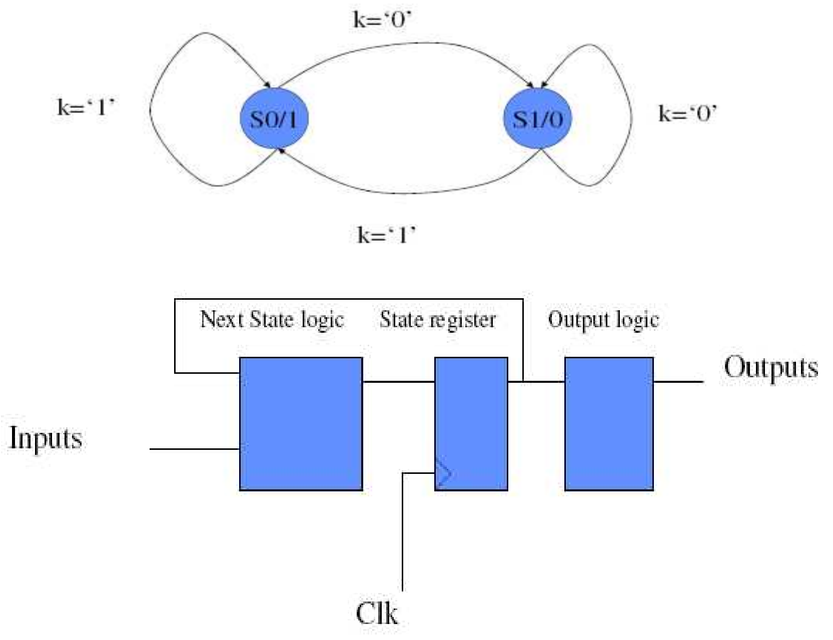
Asynchronous reset

```
process(clk,reset)    -- synthesis might complain that d is not
                      -- listed
begin
        if (reset='1') then
                q<='0';
        elsif (clk='1') and clk'event then
                q<= D;
        end if;
end process;
```

VHDL Basics – Sequential Design

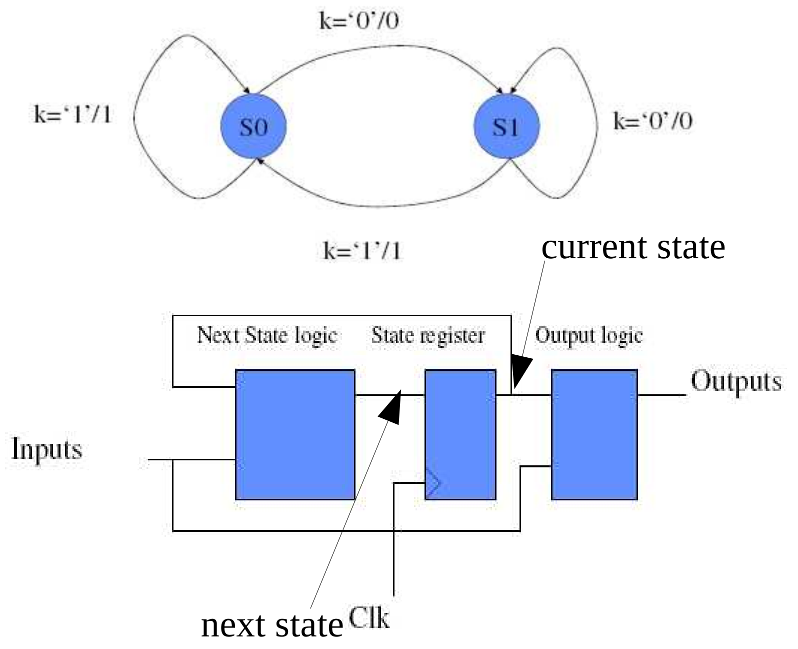
- Finite State Machine (FSM)
  - Mealy Machine & Moore Machine

Moore



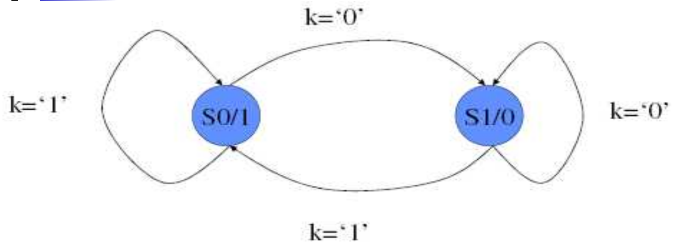
Moore machine: The output has only to do with the current state.

Mealy

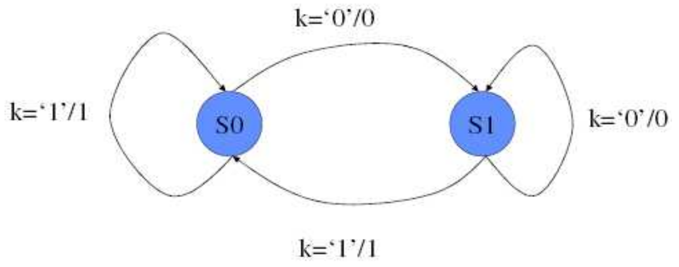


Mealy machine: The output has not only to do with the current state, but also with the input.

VHDL Basics – Sequential Design



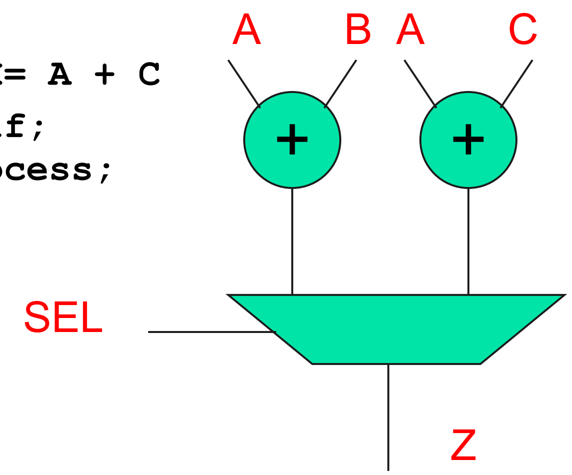
```
Architecture moore of fsm is
            type state_type is (S0,S1);
            signal next_state,pres_state:state_type;
begin
logic:      process(pres_state,k)
            begin
                        next_state <= pres_state;
                        case pres_state is
                                    when S0 =>
                                                q <= '1';
                                                 if (k='0') then
                                                            next_state <= S1;
                                                end if;
                                    when S1 =>
                                                q <= '0';
                                                 if (k='1') then
                                                            next_state <= S1;
                                                end if;
                        end case;
            end process;
regs:        process(clk,reset)
             begin
                        if (reset='1') then -- asynchronous reset
                                    pres_state <= S0;
                        elsif (clk='1') and clk'event then
                                    pres_state <= next_state;
                        end if;
             end process;
end moore;
```

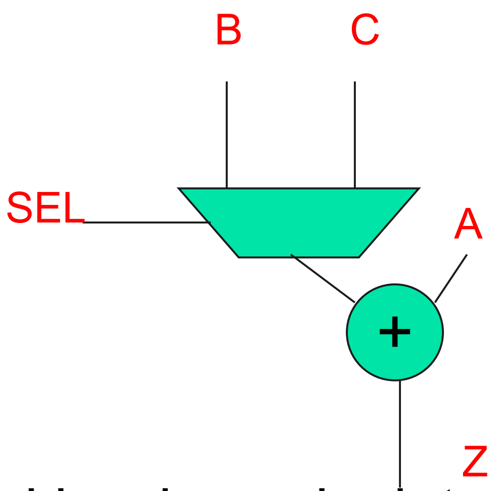


```
Architecture mealy of fsm is
            type state_type is (S0,S1);
            signal next_state,pres_state:state_type;
begin
logic:      process(pres_state,k)
            begin
                        next_state <= pres_state;
                        case pres_state is
                                    when S0 =>
                                                q <= k;
                                                 if (k='0') then
                                                            next_state <= S1;
                                                end if;
                                    when S1 =>
                                                q <= k;
                                                 if (k='1') then
                                                            next_state <= S1;
                                                end if;
                        end case;
            end process;
regs:       ...
end mealy;
```

Coding Style Issues

```
process(A, B, C, SEL)
begin
  if (SEL='1') then
    Z <= A + B;
  else
    Z <= A + C
  end if;
end process;
```



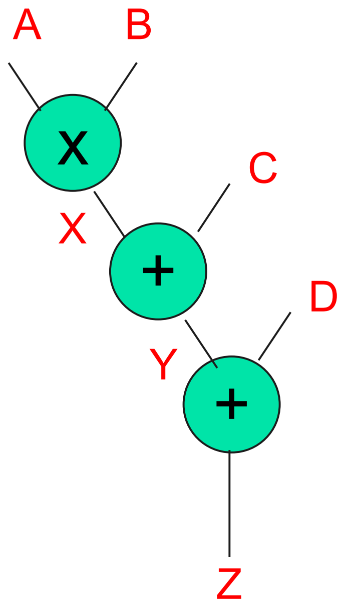


```
process(A, B, C, SEL)
  variable TMP : bit;
begin
  if (SEL='1') then
    TMP := B;
  else
    TMP := C;
  end if;
  Z <= A + TMP;
end process;
```

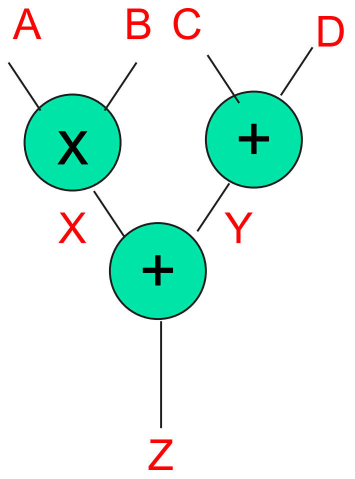
- Structure of initially generated hardware is determined by the VHDL code itself
  - Synthesis optimizes that initially generated hardware, but cannot do dramatic changes
  - Therefore, coding style matters!

Coding Style Issues

```
process(A, B, C, D, X, Y, Z)
begin
  X <= a * b;
  Y <= X + C;
  Z <= Y + D;
end process;
```



```
process(A, B, C, D, X, Y, Z)
begin
  X <= a * b;
  Y <= C + D;
  Z <= X + Y;
end process;
```



CP = Mul + Add + Add

CP = Mul + Add

Coding Style Issues

- Some more words on comparing VHDL with C

C	VHDL
sw description language	hw description language
sequential execution	parallel architecture
single core	multiple processing elements
machine codes on CPUs	logic elements on FPGAs
high clk frequency (GHz)	low clk frequency (MHz)
limited parallelism	massive parallelism
......	......

So don't use the sw programming concepts in VHDL. A good VHDL coding style is to **Think In Hardware!!!**

References

- The VHDL Golden Reference Guide
- Actel HDL Coding Style Guide
- Other VHDL books