Basic VHDL for FPGA Design

Minimal Getting Started Guide

Brian Woods

University of North Carolina at Charlotte

(Standalone Lecture)

Concepts

VHDL for FPGA Design

- Quick introduction to language for FPGA design
- This does NOT...
  - describe the whole language
  - describe all of its uses
  - discuss simulation
- Just the minimum to write your first FPGA core

VHDL

- VHDL is a **Hardware Description Language** (HDL)
- Lots of others exist...
  - Verilog
  - SystemC
  - SystemVerilog
  - BlueSpec
  - JHDL

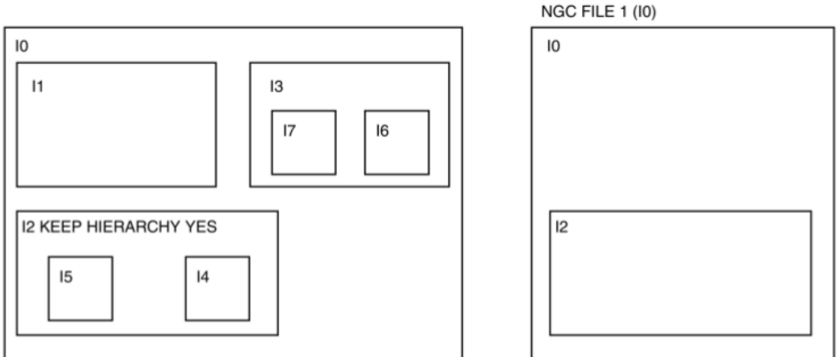
VHDL Basics

- VHDL is *NOT* a software language
- VHDL is ***NOT*** a software language
- *Not all legal VHDL can be synthesized*, only a subset
- Verbose and strongly typed
- Statements are parallel with the exception of inside processes
- Looks a lot like ADA ... but that probably doesn't help you

Terms

- **entity** — interface of a hardware building block
- **top-level entity** — blocks are organized in a hierarchy with the top-level being the root

Keep Hierarchy Diagram



Terms

- $\boxed{\textbf{\textit{architecture}}}$ — used to describe the behavior of an entity
- $\boxed{\textbf{\textit{configuration}}}$ — there can be more than one architecture per entity; a configuration binds one an component instance to an entity-architecture pair
- $\boxed{\textbf{\textit{package}}}$ — a collection of data types and function/procedure

Libraries

- Entity, architectures, and packages are *compilable* units in VHDL
- A library is a storage location for compiled units
- Libraries explicitly creates a namespace for compiled units
  the `use` command imports the namespace
- If you don't specify a library; the default is called work

Constraints

- Top-level entities usually have inputs and outputs; i.e.
  - clock
  - reset
  - application inputs
  - application outputs
- Constraints are used for a lot of things, but first, we use them to associate top-level I/O with external pins on FPGA chip

Coding Styles

- $\boxed{\textbf{\textit{data flow}}}$ — assignment statements
- $\boxed{\textbf{\textit{structural}}}$ — instantiate components
- $\boxed{\textbf{\textit{behavioral}}}$ — sequential semantics describes *what* the hardware should do

Data Flow Example

```
n1 <= (a1 and b1) or c1;
m1 <= (c1 or d1) and a1;
z1 <= n1 or not(m1);
```

Stuctural Example

```
inst1: comp1 port map(a_internal => a_external1, b => b1, c => c1);
inst2: comp1 port map(a_internal => a_external2, b => b2, c => c2);
inst3: combine port map(c1 => c1, c2 => c2, out1 => out1);
```

Behavioral Example

```
out1 <=
    "0001" when (in1 = "00") else
    "0010" when (in1 = "01") else
    "0100" when (in1 = "10") else
    "1000";
```

Language

Language Basics

- Lexigraphical symbols include
  - Keywords (ARCHITECTURE, IF, PORT, etc.)
  - Literals (0, 1, 2, 'Z', "1010")
  - Operators ($+$, $-$, AND, $<=$)
  - Whitespace (space, tab, newline)
- VHDL is *NOT* case sensitive but probably should be!

Typical File

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

entity fsm_2 is
  port ( clk, reset, x1 : IN std_logic;
         outp           : OUT std_logic);
end entity;

architecture beh1 of fsm_2 is
   variables
   signals
   component declaration
begin
   assignments
   instantiations
   processes
end beh1;
```

More Terms

- $\boxed{\textit{signals}}$ — define wires in a design
  - only one driver per signal
  - used with assignment statement $<=$
- Conventional language constructs
  - $\boxed{\textit{variables}}$
  - $\boxed{\textit{assignment}}$
  - $\boxed{\textit{if/then/else}}$ - $\boxed{\textit{when/else}}$ - $\boxed{\textit{case}}$ - $\boxed{\textit{for-loop}}$
  - $\boxed{\textit{functions}}$ and $\boxed{\textit{procedures}}$

HDL Specific

inside an architecture begin/end

- `'Event` — true when signal changes
- `<=` — parallel assignment
- *`name : comp`* — creates an instance of *comp* named *name*
- Process begin/end — creates sequentially interpreted block of code
- Sensitivity list

if then else example

```
ifexample: process (in1, in2)
begin
  if (in1='1' and in2='1') then
    out1 <= '0';
  elsif (in1='1' or in2='1') then
    out1 <= '1';
  else
    out1 <= '0';
  end if;
end process ifexample;
```

When Else Example

```
out1 <= "01" when (in1='1' and in2='1') else
        "01" when (in1='1' and in2='0') else
        "10" when (in1='0' and in2='1') else
        "00";
```

Case Example

```
caseexample:process (in3bit)
begin
  case int3bit is
    when "001"  => out1 <= '1';
    when "010"  => out1 <= '1';
    when "100"  => out1 <= '1';
    when others => out1 <= '0';
  end case;
end process caseexample;
```

For (Generate) Example

```
-- for generates need to be named
addergen: for i in 0 to 3 generate
begin
  nbitadder: adder port map(
    in0  => a(i),
    in1  => b(i),
    cin  => carry(i),
    out  => y(0),
    cout => carry(i+1)
  );
end generate addergen;
carry(0) <= cin;
cout     <= carry(3+1);
```

Components and Instantiation

- Way to use other entities/architectures
- Must declare a component before instantiation
- Declare a component and instantiation in the architecture
  - Component goes before the begin
  - Instantiation goes after
- Can instantiate it multiple time
- Each instantiation must have a unique name

Component/Instantiation Example

```
architecture example of slides
  component adder is
  port(in0, in1, cin: in std_logic;
       out, cout: out std_logic)
  end component;
begin
adder1: adder port map(
  in0  => a,
  in1  => b,
  cin  => cin,
  out  => y,
  cout => cout
);
...
end example;
```

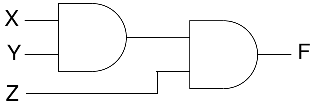
Idioms

Coding Idioms

- If you write behavioral code using idioms or templates, the synthesis tool will infer *macros* or optimized netlists for the technology
- For example,
    - a small code change, big change in hardware
    - resulting code uses resources more efficiently

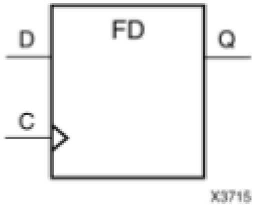
Three-Input AND Gate

```
library ieee;
use ieee.std_logic_1164.all;
entity and_3 is
  port(X, Y, Z : in std_logic;
       F : out std_logic);
end and_3;
architecture imp of and_1 is
begin
  F <= X AND Y AND Z;
end imp;
```



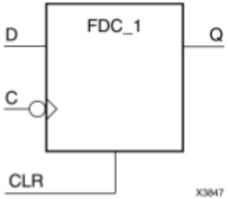
Flip-Flop With Positive-Edge Clock

```
library ieee;
use ieee.std_logic_1164.all;
entity registers_1 is
  port(C, D : in std_logic;
       Q : out std_logic);
end registers_1;
architecture archi of registers_1 is
begin
  process (C)
  begin
    if (C'event and C='1') then
      Q <= D;
    end if;
  end process;
end archi;
```



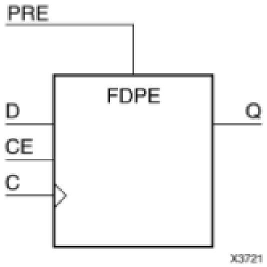
FF W/Neg-Edge CLK and Async RST

```
library ieee;
use ieee.std_logic_1164.all;
entity registers_2 is
  port(C, D, CLR : in std_logic;
       Q         : out std_logic);
end registers_2;
architecture archi of registers_2 is
begin
  process (C, CLR)
  begin
    if(CLR = '1')then
      Q <= '0';
    elsif (C'event and C='0')then
      Q <= D;
    end if;
  end process;
end archi;
```



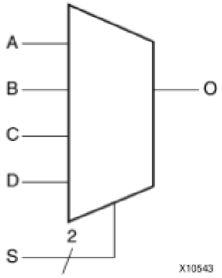
4-Bit Register

```
library ieee; use ieee.std_logic_1164.all;
entity registers_5 is
  port(C, CE, PRE : in std_logic;
       D          : in std_logic_vector (3 downto 0);
       Q          : out std_logic_vector (3 downto 0)
end registers_5;
architecture archi of registers_5 is
  begin
  process (C, PRE)
  begin
    if (PRE='1') then
      Q <= "1111";
    elsif (C'event and C='1')then
      if (CE='1') then
        Q <= D;
      end if;
    end if;
  end process;
end archi;
```



4-to-1 1b MUX

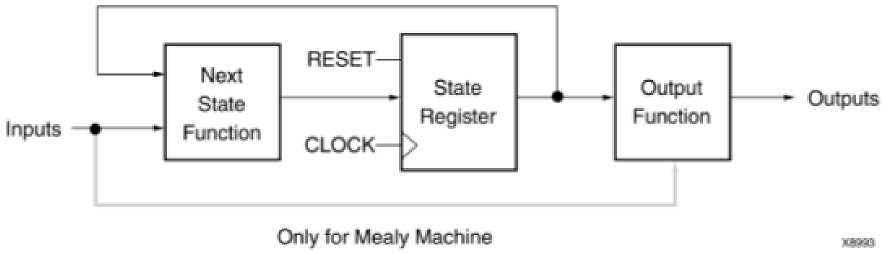
```
library ieee; use ieee.std_logic_1164.all;
entity multiplexers_1 is
  port (a, b, c, d : in std_logic;
        s : in std_logic_vector (1 downto 0);
        o : out std_logic);
end multiplexers_1;
architecture archi of multiplexers_1 is
  begin
  process (a, b, c, d, s)
  begin
    if (s = "00") then o <= a;
    elsif (s = "01") then o <= b;
    elsif (s = "10") then o <= c;
    else o <= d;
    end if;
  end process;
end archi;
```



UNC CHARLOTTE

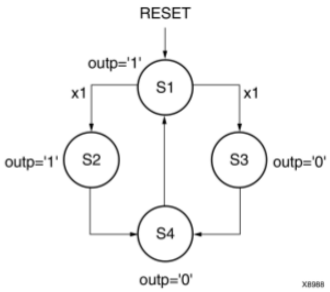
State Machines

- In theory, composed of
  - next state combinational function
  - state memory (FF or RAM)
  - output combinational function
- Moore-type: outputs only depend on current state
- Mealy-type: outputs depends on inputs and state



State Machines

- With VHDL, can be coded with...
  - one process — compact but error prone
  - two processes — probably best choice
  - three processes — longer not necessarily clearer
- Example:



UNC CHARLOTTE

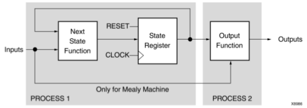
Two Process State Machine

```
library ieee; use ieee.std_logic_1164.all;
entity fsm_2 is
  port ( clk, reset, x1 : IN std_logic;
         outp           : OUT std_logic);
end entity;
architecture beh1 of fsm_2 is
  type state_type is (s1,s2,s3,s4);
  signal state, next_state: state_type ;
begin

   process 1 (next slide)

   process 2 (slide after next)

end beh1;
```



Process1

```
process1: process (clk,reset)
begin
  if (reset ='1') then state <=s1;
  elsif (clk='1' and clk'Event) then
    case state is
      when s1 =>
        if x1='1' then
          state <= s2;
        else
          state <= s3;
        end if;
      when s2 => state <= s4;
      when s3 => state <= s4;
      when s4 => state <= s1;
    end case;
  end if;
end process process1;
```

Process2

```
process2 : process (state)
begin
  case state is
    when s1 => outp <= '1';
    when s2 => outp <= '1';
    when s3 => outp <= '0';
    when s4 => outp <= '0';
  end case;
end process process2;
```

Process1 Ver 2

```
process1 : process (state, x1)
begin
  case state is
    when s1 =>
      if x1='1' then;
        next_state <= s2;
      else
        next_state <= s3;
      end if;
      outp <= '1';
    when s2 =>
      next_state <= s4;
      outp <= '1';
    when s3 =>
      next_state <= s4;
      outp <= '0';
    when s4 =>
      next_state <= s1;
      outp <= '0';
  end case;
end process process1;
```

Process2 Ver 2

```
process2: process (clk,reset)
begin
  if (reset ='1') then;
    state <= s1;
  elsif (clk='1' and clk'Event) then
    state <= next_state;
  end if;
end process process2;
```

State Machines Options

- Lots of possible implementations that are controlled with constraints
  - state: what kind of memory
  - next state: what kind of encoding
- Synthesizer will “guess” but designer might need to provide hints (speed versus space, etc.)

Other Inferred Macros

- Decoders, Priority Decoders
- Counters, Accumulators
- Various Shift Registers
- Tristate Gates
- Adders/Subtractors/Comparators
- Pipelined Multipliers/Dividers
- RAMs and ROMs

Writing VHDL

Before Writing

- Does the design need to be partitioned?
- Best to keep it conceptually simple per entity
- Once simplified enough, visual the hardware
- How can VHDL make realizing that easier?
  - For generates
  - Buses/arrays
  - etc
- How to write/structure it where it's easily readable and reusable

Writing

- Assign ports
- Start with skeleton comments
- Fillout code
- Use of idioms
- Periodically save and check syntax

Writing Example

```
--carry out, high when all three inputs are high
--           or two are high
co <= (a and b) or (a and ci) or (b and ci);

-- output, high when 1 or 3 inputs are high
o  <= a xor b xor ci;
```

Testbenches

- Used to exercise circuit for debugging/testing
- Should go through a reasonable amount of cases
- Extremely useful in large systems
- *Make sure to write hardware synthesizable VHDL*, other wise simulation and implementation results could differ

Writing a Testbench

- First initize any registers in the test bench
- Decide how to best cycle through the input values
- Determine what internal signals are needed
- Write some form of process(es) that cycle the inputs
- Determining time increments
  - for combinational systems, `wait for` statements
  - for clocked systems, use the clock

Fixed Point

Fixed Point Arithmetic

- Assume 12.4 means 12 bits before the binary point and 4 after
- Multiplication
  - Is additive on each side
  - $32 * 32 = 64$
  - $16 * 8 = 24$
  - $12.6 * 4.4 = 16.10$
  - $16 * 8.8 = 24.8$
  - $(16 * 2^8) * 8.8 = (16 * 8.8) * 2^8 = 24.8 * 2^8 = 32$

Fixed Point Arithmetic

- Assume 12.4 means 12 bits before the binary point and 4 after
- Addition
  - Add one in general
  - $32 + 32 = 33$
  - $16 + 8.8 = 17.8$
  - $16.0 + 0.16(positiveonly) = 16.16$
  - $16.0 + 1.16 = 17.16$

Further Reading

- Xilinx `http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/xst.pdf`
- Synth Works math tricks `http://www.synthworks.com/papers/vhdl_math_tricks_mapld_2003.pdf`
- *VHDL: Program by Example* Douglas Perry 4th
- *Designers Guide* Peter Ashendon
- Krzysztof Kuchcinski `http://fileadmin.cs.lth.se/cs/Education/EDAN15/Lectures/Lecture4.pdf`
- Xilinx `ftp://ftp.xilinx.com/pub/documentation/misc/examples_v9.zip`