Simple VHDL Tutorial

September 26, 2007

1 Objective

Currently, this tutorial is designed to explain how to create a process in VHDL. It isn't a complete tutorial, but it is a draft and I will be updating this tutorial with VHDL hints to help with each of the labs. As of now, this tutorial is for Lab 2.

This assumes you have used the **Create and Import Peripheral Wizard** to create a hardware core. For the purposes of this example I will assume the core is named: *mycore_v1_00_a* and it has the base range of *0x70000000 - 0x7000FFFF*.

2 User Logic

Open the core's *user_logic.vhd* file. Notice a few things:

```
signal slv_reg0   : std_logic_vector(0 to C_DWIDTH-1);
signal slv_reg1   : std_logic_vector(0 to C_DWIDTH-1);
signal slv_reg2   : std_logic_vector(0 to C_DWIDTH-1);
signal slv_reg3   : std_logic_vector(0 to C_DWIDTH-1);
```

Xilinx's wizard sets the addressing of the hardware core to begin with the slave registers. So this means slv_reg0 is a register with the address of 0x70000000 (the base address of our core) and since each register is 32-bits wide it means slv_reg1 is 0x70000004, slv_reg2 is 0x70000008 etc ...

The Wizard creates a few Processes:

- SLAVE_REG_WRITE_PROC

  This process stores a 32-bit value from the bus to a particular register.

- SLAVE_REG_READ_PROC

  This process outputs a 32-bit value from the particular register to the bus.

So if we want to Read from slv_reg0 the SLAVE_REG_READ_PROC is putting the slv_reg0's data onto the bus. Alternatively, if we wanted to write data to slv_reg1 the SLAVE_REG_WRITE_PROC is storing whatever data is on the bus in to slv_reg1.

Now we are going to use this core as an accelorator to off-load a calculation from the PowerPC, allowing it to perform some other calculation. To start we will create a simple process to add 5 to slv_reg0 and store the result in slv_reg3. We will rely on the PowerPC to specify the operand to slv_reg0 and read back the result.

Using the SLAVE_REG_READ_PROC process as an example, let's write our own process after the line:

```
end process SLAVE_REG_READ_PROC;

-- Our New Process ---
ADD_5_PROC : process( slv_reg0 ) is
begin
  -- add 5 to slv_reg0 and store result in slv_reg3
  slv_reg3 <= slv_reg0 + 5;
end process ADD_5_PROC;
```

So what does all of this mean?

1. **ADD_5_PROC** is the name of the Process
2. Everything between the parentheses **process (...)** is the sensitivity list

   The sensitivity list tells the process which signals to "watch" and whenever one of them changes the process updates and performs the calculation

   process(slv_reg0) means our process is going to "watch" slv_reg0 and whenever it changes it will recalculate
3. **slv_reg3 <= slv_reg0 + 5;** is our calculation
4. **end process ADD_5_PROC;** ends the process

Common VHDL syntax mistakes:

- missing a ";" at the end of a line (just like C)
- using = instead of <=
- missing signals in the sensitivity list (every signal on the right side of a <= should be in the sensitivity list)

Assigning values to signals, use <= You can only assign values to a signal (register) in one process. So if you looked at your user_logic.vhd file you will notice that after you added the above process there are **two** processes that are assigning a value to slv_reg3. Look in SLAVE_REG_WRITE_PROC and see if you can find all of the places that say:

```
slv_reg3 <= ...
```

In hardware this is literally trying to drive the register with two different sources. Hardware doesn't like to have multiple sources trying to change its state... it likes one. So to fix this, right now we will just comment out all of the **slv_reg3 <= ...** in the **SLAVE_REG_WRITE_PROC**. Comments two or more dashed lines: –

```
-- This is my comment
```

Now slv_reg3 is only being assigned a value in our new process **ADD_5_PROC**. A good question to ask is whether it is possible have multiple processes read the value from a register at one time. The answer to this is yes, of course. Unlike a write, a read just connects the register to all components reading it. This is why we can have **slv_reg0** appear in **SLAVE_REG_READ_PROC** and **ADD_5_PROC**.

In lab two we do not require you to have an intimate knowledge of VHDL. Using what is described here you will have all the necessary knowledge to create your own process to solve lab two's problem.