Tutorial - Writing Combinational and Sequential Code

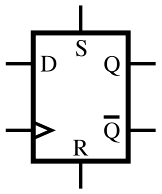
Using VHDL Process or Verilog Always Blocks

This tutorial shows how to write blocks of either VHDL or Verilog that are contained in either a **Process** or an **Always Block** respectively. Processes (in VHDL) and Always Blocks (in Verilog) are fundamental and they need to be well understood. They behave in exactly the same way, so both are introduced here for you, if you are learning just one language right now, pay attention to the examples focused at that particular language. Processes or Always Blocks are used in two main scenarios:

1. To define a block of **combinational** logic
2. To define a block of **sequential** logic

The first scenario is what is commonly seen in textbooks when introducing Processes or Always Blocks to a new student. It is presented here for you, to make you aware of its existence. But in reality, a Process/Always Block used to define a block of combinational logic is seen far less often in "real-world" code than a Process/Always Block used to define sequential logic.

The first question you might be asking yourself is what is the difference between combinational and sequential logic? Combinational (or combinatorial) logic is logic that does not require a clock to operate. The example of the and-gate previously is a combinational example. Sequential logic is logic that does require a clock to operate. The most fundamental building block of sequential logic is the D Flip-Flop (pictured below).



The D Flip-Flop!

If you are unaware of how a D Flip-Flop works, stop reading this immediately! You need to understand how flip-flops are used inside FPGAs before you continue reading. All set? Good.

Combinational process in VHDL:

```vhdl
process (input_1, input_2)
begin
    and_gate <= input_1 and input_2;
end process;
```

Combinational Always Block in Verilog:

```verilog
always @ (input_1 or input_2)
  begin
    and_gate = input_1 & input_2;
  end
```

In the both the VHDL and Verilog code above, input_1 and input_2 are in what is called a **sensitivity list**. The sensitivity list is a list of all of the signals that will cause the Process/Always Block to execute. In the example above, a change on either input_1 or input_2 will cause the Process/Always Block to execute. This process/always block takes the two inputs, perform an "and" operation on them, and stores the result in the signal and_gate. This is the exact same functionality as this code:

```vhdl
-- VHDL:
and_gate <= input_1 and input_2;
```

```verilog
// Verilog:
assign and_gate = input_1 & input_2;
```

Both examples of code serve the same purpose: to assign the signal and_gate. The difference is that one is in a combinational Process/Always Block and the other is not. Therefore since the same result can be achieved without the use of a combinational Process/Always Block, I do not recommend that the beginning digital designer use these statements in this way.

The second way that a Process or Always Block can be used (and the far more interesting example) is to define a block of sequential logic. Again, sequential logic is logic that is clocked.

Using Process/Always Blocks for Sequential Code

Help Me Make Great Content!   Support me on Patreon!   Buy a Go Board!