



Tutorial Session 2 – Hierarchical Design and Introduction to VHDL

Introduction

The goal of this tutorial session is to build upon last weeks intro to Altera's MAX+plus II design tool. First, we will introduce hierarchical design, that is, the use of smaller parts in a larger design. The example we will cover is building a 4-bit ring counter out of D Flip-Flops. Then we will progress to covering the basic aspects of VHDL. VHDL is one of the industry's most used hardware design languages. Learning this language now will give you a jump start if you choose to pursue a career in digital design.

Part 1: D Flip-Flop

1. Open MAX+plus II and open a new schematic. Save this as `dff.gdf`. Now click **File => Project => Name** and name the project `ring_counter`.
2. Open Lab 1 to Figure 3, which describes a D Flip-Flop in elemental form.
3. Using the techniques learned during the first session; input the gates into your blank schematic.
4. Label your circuit inputs as shown in the figure.
5. Verify the functionality of the flip-flop by running a simulation. To do this save and compile your project, open a new waveform.
6. Add all of the inputs and outputs as discussed last week. Set PRESET and CLEAR high, and CLOCK to the default clock input.
7. Set D high between 300ns and 500ns, run the simulation, and verify that Q goes high during that interval and QBAR goes low.
8. Now we will make a symbol for our flip-flop. The Altera software can pretty much automate this process. To do this, first click **File => Create Default Symbol**.
9. Now click **File => Edit Symbol**, and you will see the symbol that was automatically created. Here, you can make any changes as desired. Note that the default symbol was saved as `dff.sym`.
10. We now have a functional model for the D Flip-Flop. In the next section, we will integrate this design in a ring counter.

Part 2: Ring Counter

1. Open a new graphic editor file and save it as `ring_counter.gdf`. Make sure to also add this file to the project.
2. Let's begin by adding in **four** of our previously designed **flip-flops**. These should be available to you in the library named after your working directory.

- Place these horizontally across the screen, leaving some space between each one.
- Now we need to add in two inputs. These are the **OSCLOCK** and the **START** inputs. Place an **AND** gate after the inputs.
 - Place **6** outputs to the right side of the screen. Name these **ECLK, CLK, T3, T2, T1, and T0**.
 - Add an inverter near the outputs. Right click on the inverter and click **Rotate => 270**.
 - Connect up the output of the **AND** gate in step 3 to each of the **CLOCK** inputs on the **flip-flops**. Also connect it to the **NOT** gate of step 5 and the **CLK** output.
 - Connect the output of the **NOT** gate to the **ECLK** output.
 - Add in three **inverters** and a three-input **AND** gate near the outputs. Connect the outputs of the inverters to the input of the AND gate.
 - Connect the **Q outputs** of the **four flip-flops** to the **T3, T2, T1, and T0** outputs. Make sure the left most flip-flop is connected to T0, the next to T1, etc.
 - Now connect each **Q** output of the flip-flops to the **D** inputs on the flip-flop to the immediate right. This does not include the far right flip-flop. Do not connect this one back to the input of the flip-flop on the far left.
 - Connect the same **three Q outputs** to the **3 inverter** inputs in step 8.
 - Make a connection from the **output** of the **AND** gate in step 8 to the **D input** of the far left **flip-flop**.
 - Finally, add an input labeled **CLEAR** at the top left of the diagram, and connect it to all of the **CLEAR** and **PRESET** inputs.
 - Now to simulate the design. Add all pins as done previously today and during last session. Make sure to set **OSCLOCK** to the **default clock signal**, and to set **START high**. Also, make **CLEAR high** for the entire simulation except for the first clock cycle, where you are to make it **low**. Run the simulation.
 - You should see each successive input go high (T0, T1, T2, T3, T0, T1, ...). If you cannot see all this happen, make sure that the length of the simulation is 10us.
 - The final step is to make a symbol for your design. Do this using the same method as described in Part 1.
 - It might be good here to note that you can view your hierarchy. Click **MAX+plus II => Hierarchy Display**. You should be able to see the ring counter with the 4 flip-flops to the right.

Part 3: VHDL Basics (read only)

- VHDL is a programming language that has been designed and optimized for describing the behavior of digital systems.
- VHDL takes advantage of hierarchical design. For this reason, you will learn how to make the previous ring counter design using VHDL. This will be completed over the next couple of sessions. Today, you will be describing the D Flip-Flop.

3. Although VHDL is a very extensive language, and you are not expected to master it in this course, it is good to get an overview and begin using it now, as it will give you a jump start in the industry.

Part 4: D Flip-Flop in VHDL

1. Open a new text editor file. To do this, click **File => New** and then click **Text Editor File** and click **OK**.
2. A comment in VHDL is signified by adding two dashes, followed by the comment. Add your name to the top of the code, followed by CHD tutorial, as shown below:

```
-- John Doe – CHD Tutorial 2
```

3. Next add in the following lines:

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

The above lines are similar to include statements in C and C++.

4. Now is time to add the major declarations. The first is the **entity** declaration. Within this is the description of the input and output pins for the design. Below is the entity declaration for the D Flip-Flop:

```
entity dff_vhdl is  
port(  
    clock : in  std_logic;  
    d     : in  std_logic;  
    preset : in  std_logic;  
    clear  : in  std_logic;  
    q      : out std_logic;  
    q_bar  : out std_logic  
);  
end dff_vhdl;
```

A few keys to pay attention to above include the fact that the last declared port does not have a semicolon after it, and that similar inputs/outputs may be grouped together if they are of the same type.

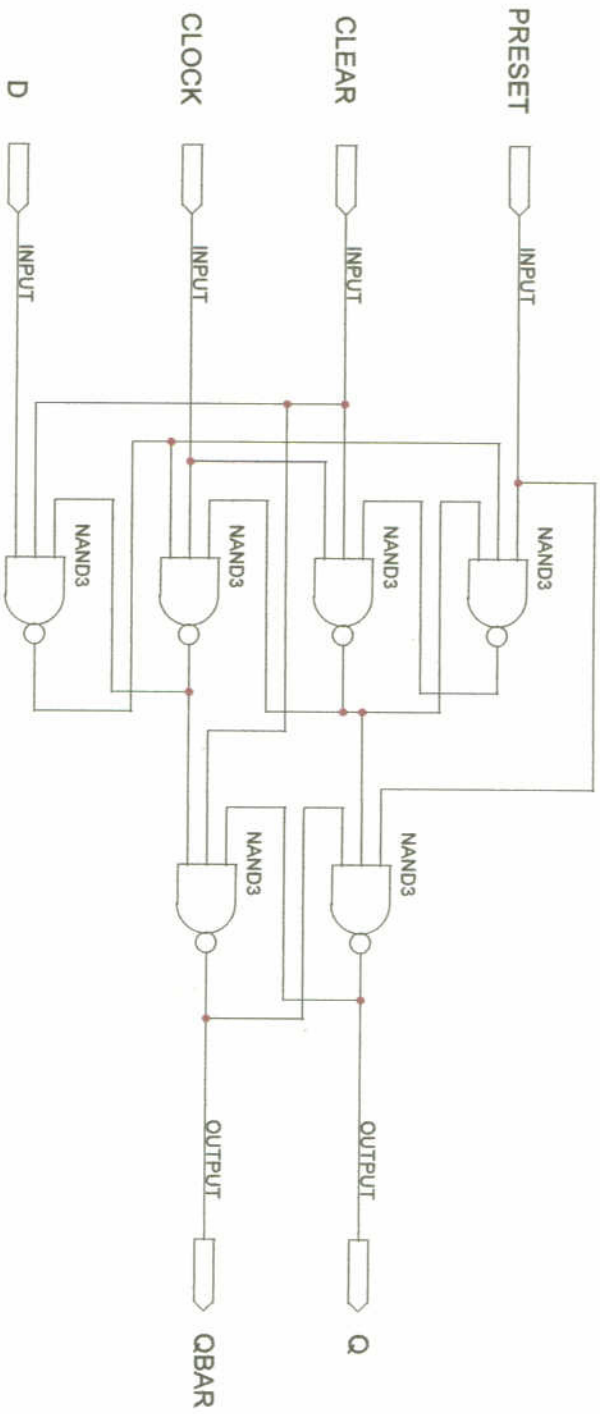
5. The next major part of a VHDL design is the architecture design. There may be multiple architectures for one entity, so that you can choose which to use. There are three types of ways to describe a component in VHDL. These will be described for you in a later tutorial.
6. The architecture description we will use is described below

```
architecture dff_1 of dff_vhdl is  
begin  
    reg: process(clear,clock)
```

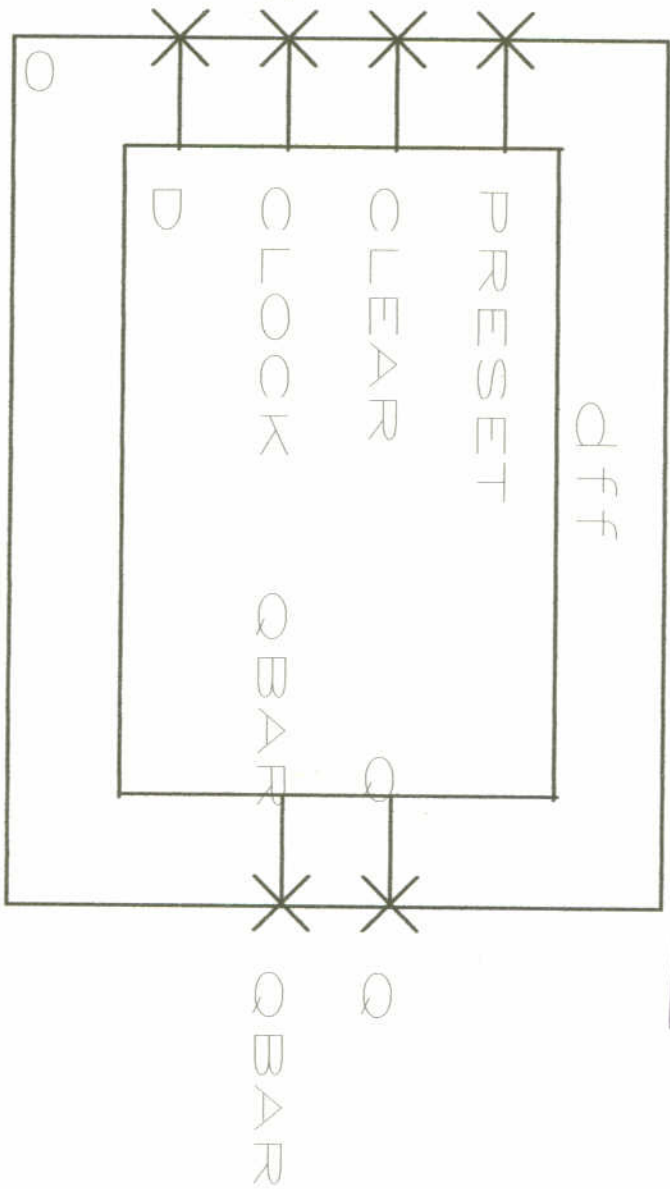
```
variable Qreg: std_logic;
begin
    if rising_edge(clock) then
        if preset = '1' and clear = '1' then
            Qreg := d;
        end if;
    end if;
    q <= Qreg;
    q_bar <= not Qreg;
end process;
end dff_1;
```

7. Keep in mind that you are not expected to understand this now. This is just an introduction to the language.
8. The next step is to make a symbol from your file. First, save your file as **dff_vhdl.vhd**. Next, follow the same procedure to create a symbol as you did for the graphic design.
9. Next session, we will continue to develop on the VHDL ideas in place here. As a reference, you should refer to <http://www.acc-eda.com/vhdlref/>. This is a very good site for learning the basics of VHDL, and it is recommended that you check this out sometime before lab three, and it will be a breeze.

Part 1 - DFF Schematic



Part 1 - DFF Symbol

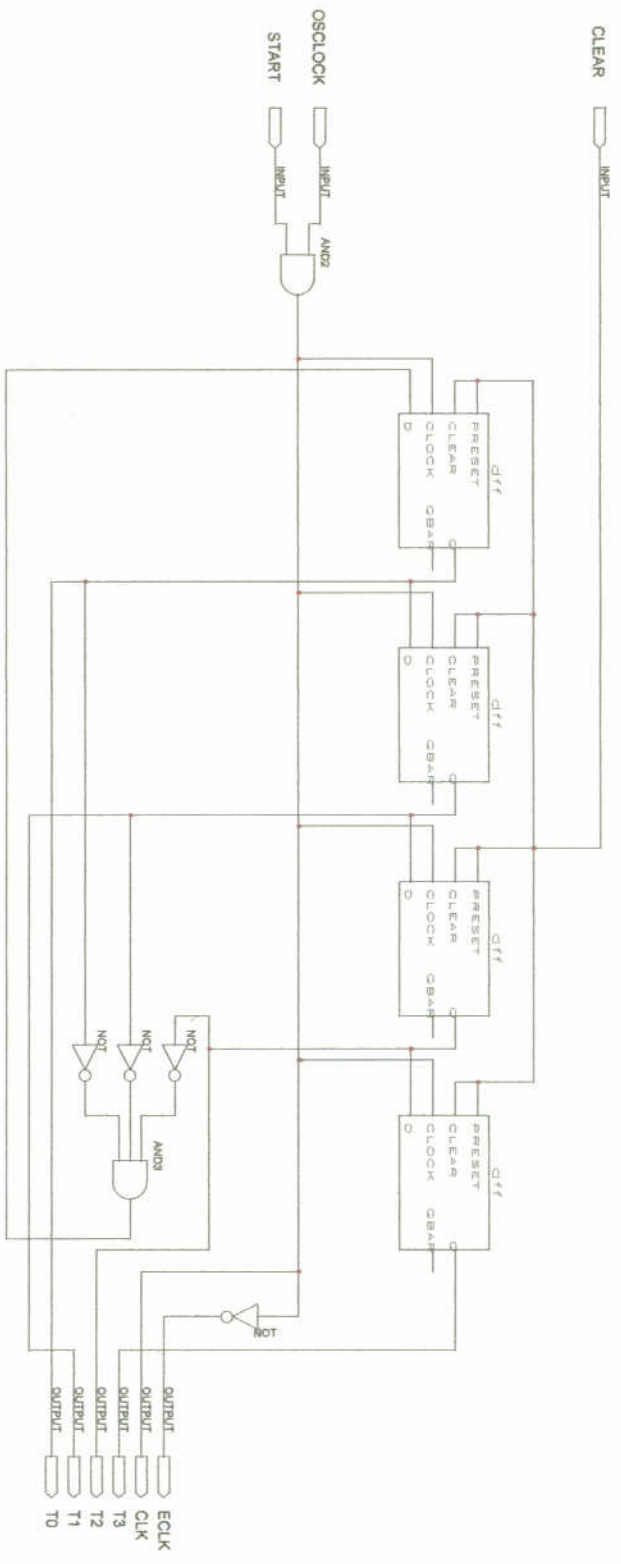


Part 1 - DFF Simulation

- Name:
- [] PRESET
- [] D
- [] CLOCK
- [] CLEAR
- [] QBAR
- [] Q



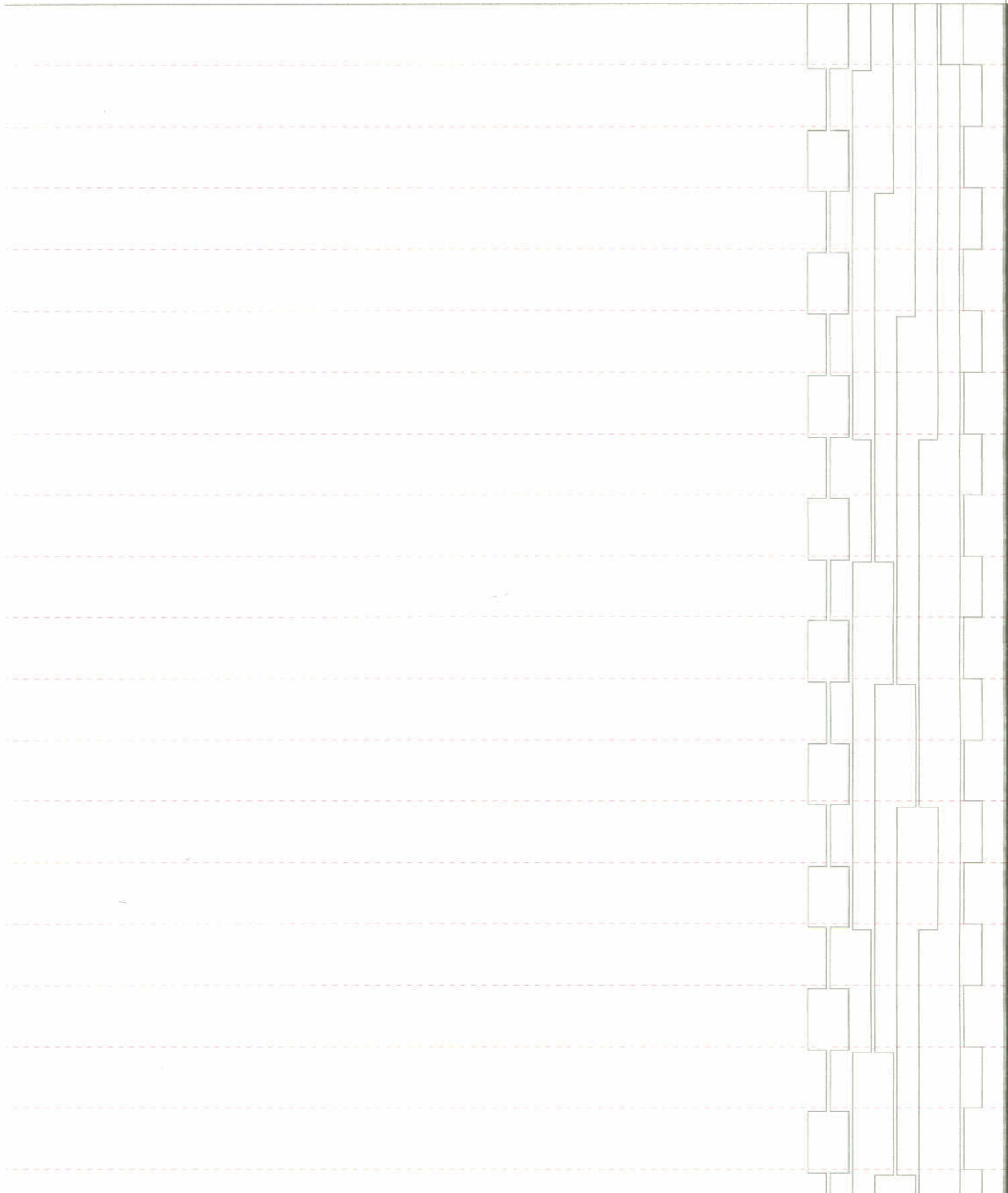
Part 2 - Ring Counter Schematic



Part 2 - Ring Counter Simulation

Name:

- II) START
- III) OSCLOCK
- III) CLEAR
- IOJ T3
- IOJ T2
- IOJ T1
- IOJ T0
- IOJ ECLK
- IOJ CLK



Part 4 - DFF VHDL CODE

```
-- John Doe - CHD Tutorial 2
library ieee;
use ieee.std_logic_1164.all;

entity dff_vhdl is
port (
    clock : in std_logic;
    d      : in std_logic;
    preset : in std_logic;
    clear  : in std_logic;
    q      : out std_logic;
    q_bar  : out std_logic
);
end dff_vhdl;

architecture dff_1 of dff_vhdl is
begin
    reg: process(clear, clock)
        variable Qreg: std_logic;
    begin
        if rising_edge(clock) then
            if preset = '1' and clear = '1' then
                Qreg := d;
            end if;
        end if;
    end process;
    q <= Qreg;
    q_bar <= not Qreg;
end dff_1;
```

Part 4 - DFF VHDL Symbol

