

Laboratory 2: Counters

1: Introduction

1.1: Aim

The purpose of this laboratory is to introduce you to synchronous logic design, and the use of counters.

1.2: Learning outcomes

At the end of this laboratory you should be able to:

- Use VHDL to implement synchronous (clocked) designs
- Use ModelSim-Altera to simulate synchronous designs
- Build counter based circuits

2: Clocks, registers and processes

With synchronous circuits, the system has memory, and is controlled by a clock. We use a `process` statement to control the state of the system on clock edges.

Create a new project called counter, and enter the following VHDL code which implements a basic 4-bit counter:

```
counter.vhd
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.numeric_std.all;
4
5 entity counter is
6     port (clock : in std_logic;
7           count : out unsigned(3 downto 0) );
8 end counter;
9
10 architecture cnt of counter is
11     signal i_count : unsigned(3 downto 0) := (others => '0'); -- Internal counter
12 begin
13     process (clock) is
14     begin
15         if rising_edge(clock) then
16             i_count <= i_count + 1; -- Increment the counter every clock cycle
17         end if;
18     end process;
19
20     count <= i_count; -- Connect internal counter register to count output port
21 end architecture cnt;
```

We cannot toggle `count` directly since `count` is an output it can only appear as a result of an assignment (on the left hand side of `<=`, not on the right hand side). Therefore we have created an internal signal, `i_count`, which we will use for the counter, and assign the result to `count` (line 20). We have set the initial value of `i_count` to 0.

Start Analysis and Synthesis, and sketch the circuit as analysed in the **RTL Viewer**:



Enter the following pin numbers using the **pin planner**:

- `clock`: R24
- `count(3 downto 0)`: E24, E25, E22, E21

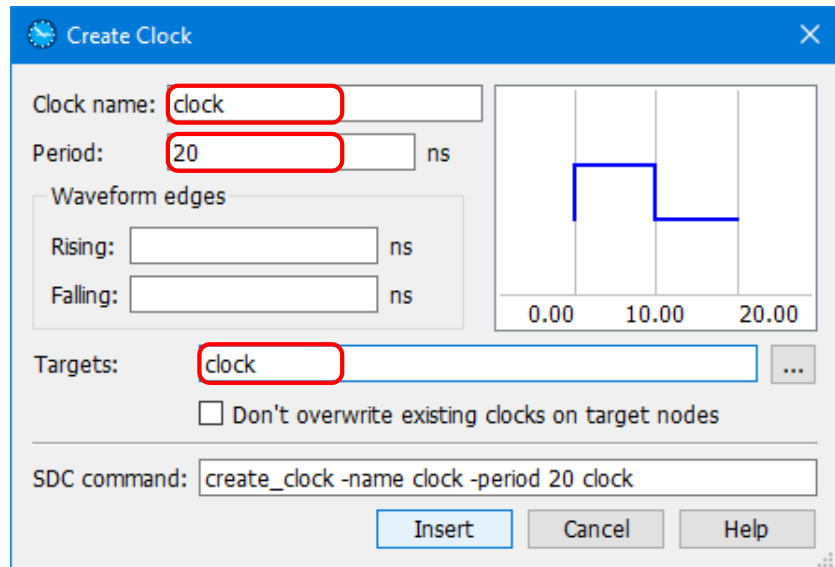
Laboratory 2: Counters

We next need to set clock constraints. These are used to ensure that the propagation delay of the logic is less than the clock period.

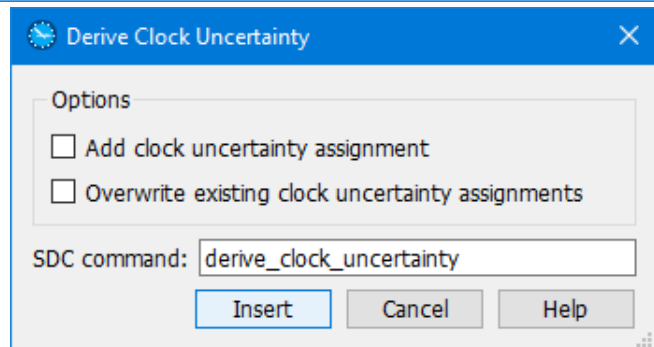
Create a new Synopsis Design Constraints file: **File » New...** then in the dialog select **Synopsis Design Constraints File** (near the bottom).

Then select **Edit » Insert Constraint » Create Clock...** to create the clock constraint:

Set **Clock name** to clock, the **Period** to 20 and **Targets** to clock, and then click **Insert** to indicate that the clock will operate at up to 50 MHz.





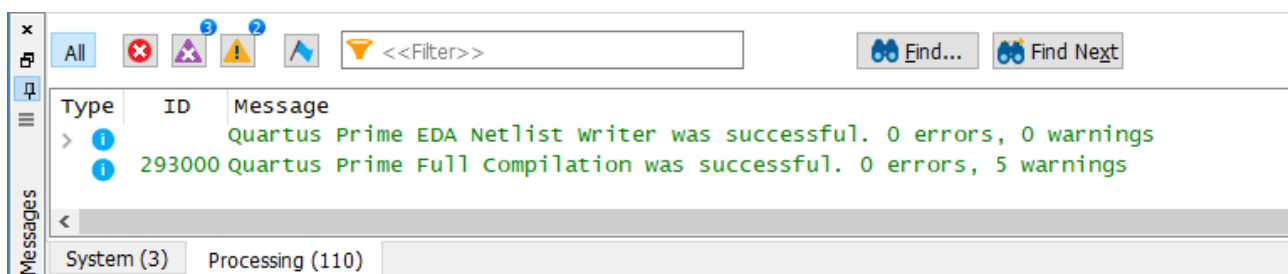
Then on the following line, **Edit » Insert Constraint » Derive Clock Uncertainty...** and click the **Insert** button.



Save the file as `counter.sdc`. This will automatically add the constraints file to the project.

Do a full compile of your design.

In the **Messages** panel, if there are critical warnings, click on  to display only those errors. Fix the cause of any errors or critical warnings and recompile if necessary. (If the **Messages** panel filter bar is not visible it can be enabled by clicking  and selecting **Filter Bar**).



From the compilation report, determine the resources required by your design (**Fitter » Resource Section**)

4 input:

3 input:

≤ 2 input:

Registers:

Finally, download your design onto the FPGA, and press KEY3 to see the LEDs count. (Note that you may have some key-bounce).

Laboratory 2: Counters

3: Making the counter more useful

One of the limitations of this counter is that it only counts to 16, and we have limited control of the counting. We want to add the following features to the counter:

- To be able to change number of bits of the counter.
- To be able to specify the modulo of the counter.
- To be able to reset the counter asynchronously.
- To be able to enable the counter so that we can chain several counters together in series.

Modify the entity for the counter to the following:

```
counter.vhd
5 entity counter is
6   generic (bits : positive := 4;          -- Size of the counter
7           modulo : positive := 10);      -- Counter modulo
8   port (clock : in std_logic;
9         reset : in std_logic := '0';
10        enable : in std_logic := '1';
11        count : out unsigned(bits-1 downto 0);
12        carry : out std_logic);
13 begin
14   assert modulo <= 2**bits
15     report "Not enough bits for the size counter"
16     severity error;
17 end counter;
```

By default, the counter will be a modulo 10 counter. The `assert` just checks that we have enough bits for the specified modulo, and will cause a compiler error if we do not specify enough bits when we instantiate it.

Edit your architecture to make the following changes:

Change the number of bits of your internal counter register (to the size specified by the generic).

Add an asynchronous reset to your design (to reset the counter to 0 when `reset` is '1'). Note – you will need to add the reset signal to the process sensitivity list.

Only increment the counter when the `enable` signal is '1'.

Modify the counter to make it a modulo counter. When the internal count is `modulo-1` then the next count should be 0, rather than incrementing.

Provide a `carry` output signal when the internal count is `modulo-1` and the counter is enabled.

Why does the `carry` output need to include the `enable` input?

How is the `enable` signal implemented in the design? (Hint use the **RTL Viewer**)

What are the resources now used by the counter?

4 input:

3 input:


<=2 input:

Registers:

3.1: Testing your design

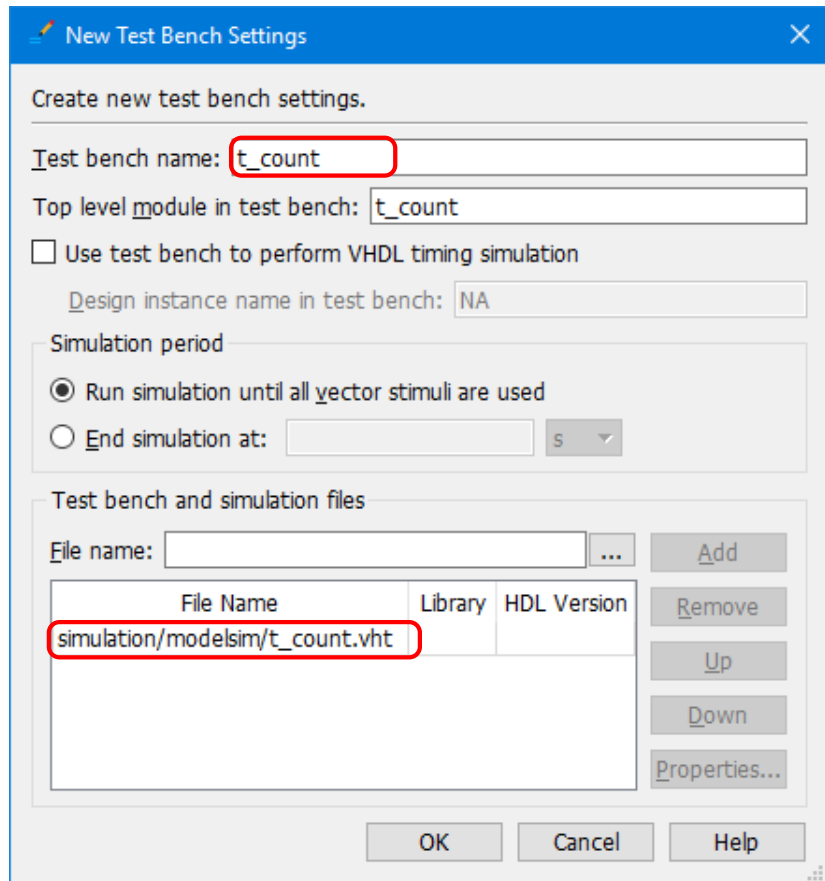
Copy the provided test bench `t_count.vht` to the simulator directory (<project>/simulation/modelsim/).

Laboratory 2: Counters

Within **Assignments » Settings...** (or ) under Category, select **EDA Tool Settings » Simulation**,

then in the NativeLink settings panel, select **Compile test bench**

Click **Test Benches...** then **New...** and set up the Test Bench as follows:



New Test Bench Settings

Create new test bench settings.

Test bench name: t_count

Top level module in test bench: t_count

☐ Use test bench to perform VHDL timing simulation

Design instance name in test bench: NA

Simulation period

☒ Run simulation until all vector stimuli are used

☐ End simulation at: s

Test bench and simulation files

File name: ...

File Name	Library	HDL Version
simulation/modelsim/t_count.vht		

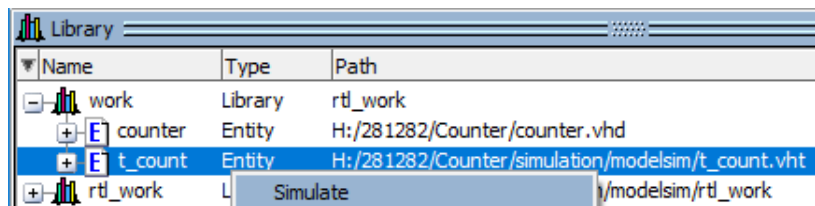
Perform a functional simulation (**Tools » Run Simulation Tool » RTL Simulation**).

3.1.1: Manual simulation setup

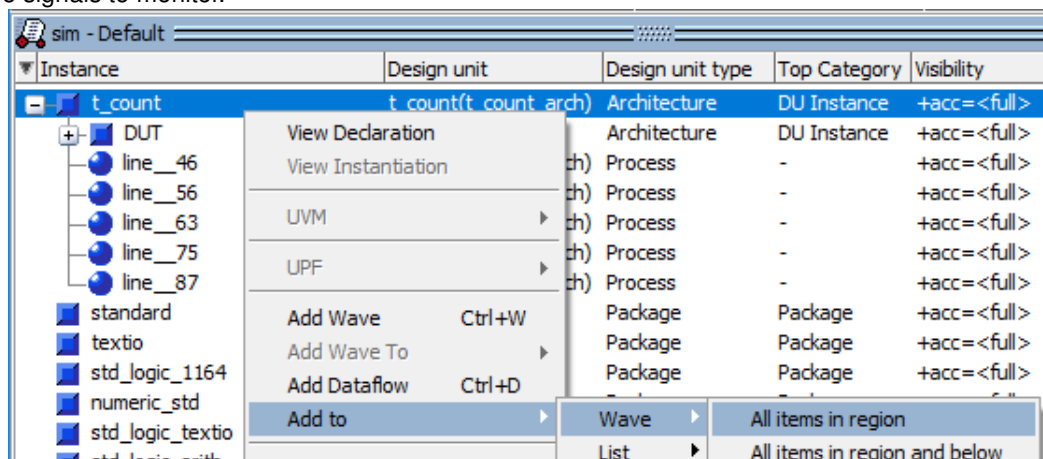
Alternatively, you could select **None** in the NativeLink settings, and load the test bench manually within ModelSim.

Within ModelSim, select **Compile » Compile...**, select the t_count.vht test bench file, and press **Compile** to compile the test bench into the project. Click **Done** when completed.

In the **Library** panel, expand work and right-click on t_count, and select **Simulate** to start the simulation. The **sim** panel will then appear:



In the **sim** panel, Right click on t_count, and select **Add to » Wave » All items in region** (as shown below) to select the signals to monitor.



Finally **Simulate » Run » Run -All** (or the toolbar button ) to run the simulation to the end.

Laboratory 2: Counters


3.2: Checking the results


If your design was correct, there should be no error messages within the **Transcript** panel.

If there are any errors:

Correct your design and make sure that you **save the file**.

Within the Library panel, under work, right click on counter and select **Recompile**.

Then select **Simulate » Restart...** (or the  button) to reload the file and restart the simulation.

Run -all (either type the **run -all** in the transcript window, or select **Simulate » Run » Run -all** menu, or use the  toolbar button) to rerun the simulation.

Once your design is working correctly, have your simulation checked by a supervisor

When does it start counting? Why then?

Explain the signals at 280 ns. Why does carry go high for the cycle before the counter resets to 0?

What is happening at 480 ns? Why does the carry go to '0'?

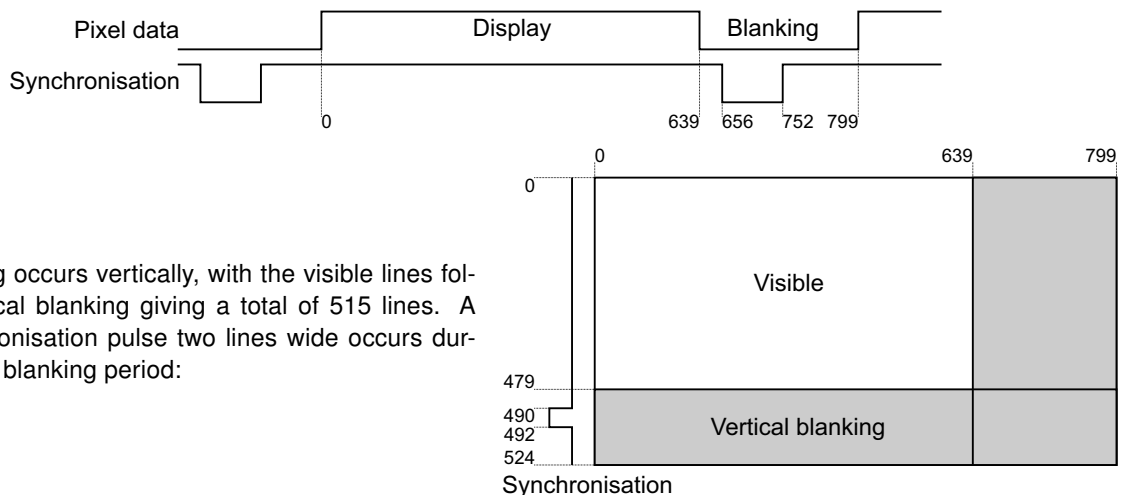
Explain what happens at 940 ns. Is this synchronous or asynchronous?

When you have finished, you may close ModelSim.

4: VGA display

We will now make use of the counters for driving a VGA display. The image on the display is formed by a series of scan lines. For this we will use two counters – one to count which pixel we are currently displaying, and a second counter for the row we are currently on.

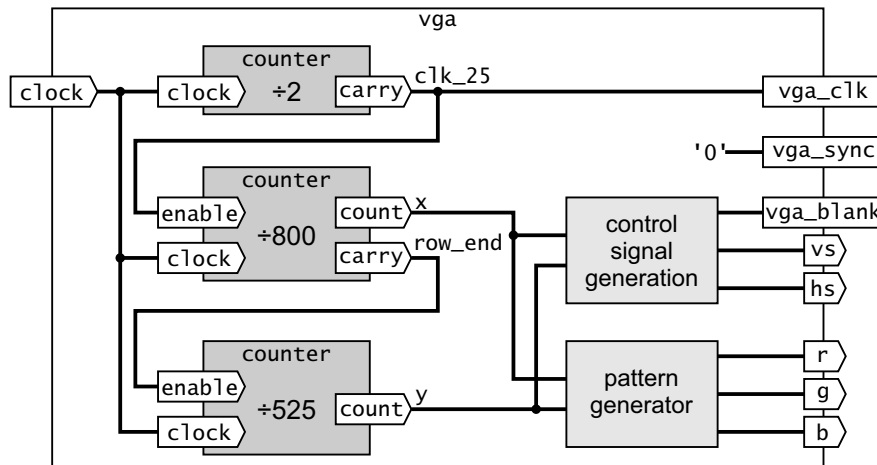
For a 640×480 display, the timing is shown here. The total line length is 800 pixels, with a blanking period after displaying the pixel data. During the blanking period, we need to provide a synchronisation pulse.



A similar timing occurs vertically, with the visible lines followed by vertical blanking giving a total of 515 lines. A vertical synchronisation pulse two lines wide occurs during the vertical blanking period:

Laboratory 2: Counters

VGA resolution operates with a pixel clock of 25 MHz. We have available on the FPGA board a 50 MHz clock, so we can use a simple divide by 2 to generate the pixel clock.



We can use combinatorial logic to generate the VGA control signals (blanking and synchronisation signals) from the counter outputs.

- `vga_sync` is only required for the D/A converter on the DE2-115 development board, and should be set to `'0'`.
- `vga_blank` should be high (`'1'`) during the visible region, and go low (`'0'`) during each of the horizontal and vertical blanking periods.
- `hs` and `vs` are horizontal and vertical synchronisation signals sent directly to the monitor. They should normally be high (`'1'`), and go low (`'0'`) for the synchronisation pulses as indicated by the timing above.

The pattern generator generates a colour for each pixel based on the position on the display. A simple pattern generator is provided.

Copy `vga.vhd` into your project directory, and add to your project (within the **Files** view of the **Project Navigator** pane, right-click on Files).

Once added, right-click on `vga.vhd` and **Set as Top-Level Entity** to make this the main file.

Select **Assignments » Remove Assignments...**, and select **Pin, Location & Routing Assignments** before clicking **OK**. This removes the old pin-numbers used by the counter.

Within `vga.vhd`, add the logic required to instantiate the three counters. For example to instantiate the first counter you could add within the `vga` architecture the following code:

```
1 pix_clock: entity work.counter
2   generic map (modulo => 2, bits => 1)
3   port map (clock => clock, carry => clk_25);
```

Remember that instantiation is like plugging in a chip. Here `pix_clock` is the label given to this instance. The generic mapping sets the parameters, and the port mapping is how the component is wired to the rest of the circuitry. The port mapping is always `<port> maps to (=) <signal>`.

Derive the VGA control signals, and wire them to the corresponding output ports:

`vga_clock`
`vga_blank`
`hs` and `vs`

Check that your design compiles correctly (**Analysis and Elaboration**).

Set the pin numbers undefined pins.

Recompile the design

Connect a VGA monitor to the development board, and download your design onto the FPGA to test.

Have your design checked by a supervisor.

4.1: Additional patterns

If you have time, modify the pattern generator to generate a different test pattern. Patterns you might like to try:

```
pattern.vhd
1 r <= not x(7 downto 0) when y(6) = '0' else (others => '0');
2 g <= not x(7 downto 0) when y(5) = '0' else (others => '0');
3 b <= not x(7 downto 0) when (y(6) xor y(5)) = '0' else (others => '0');
```

Laboratory 2: Counters

or

	pattern.vhd
1	<code>r <= x(7 downto 0) xor y(7 downto 0);</code>
2	<code>g <= x(8 downto 1) xnor y(8 downto 1);</code>
3	<code>b <= x(8 downto 1) or y(8 downto 1);</code>

or generate your own pattern.

5: Summary

How do you control a synchronous design with a clock in VHDL?

What signals need to be in the sensitivity list of a process in synchronous design?

The design here used an asynchronous reset. How would you implement a synchronous reset?