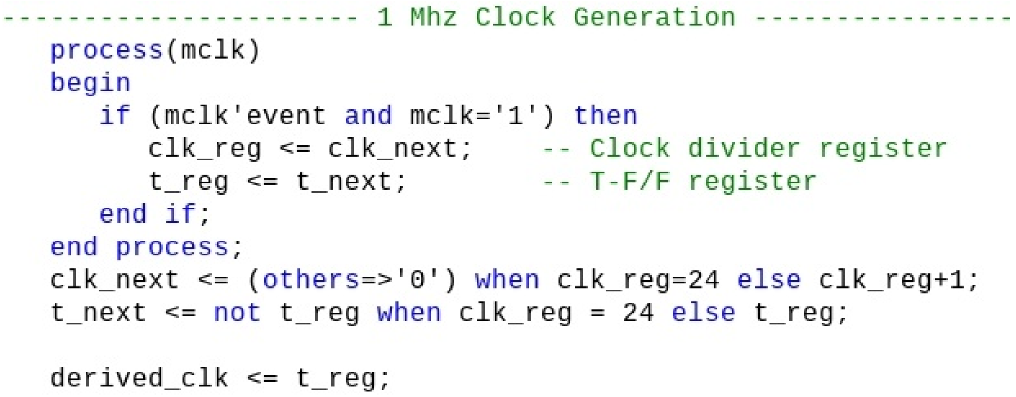
Creating and using a derived clock in a Spartan 6 FPGA

Section 9.1.3 of the textbook is titled “Misuse of derived clocks”. In that section you learn that standard practice is to use the master clock as the clock signal to both fast and slow parts of a circuit and then use a slower derived signal as an enable to those clocked components that need to be “clocked” at a slower rate. This is good practice and is recommended. However, the enable signals end up being pulses with a duration equal to one period of the master clock. With our 50Mhz mclk that means 20 ns long pulses.

Finding 20 ns pulses with the Digilent Analog Discovery logic analyzer is problematic. The ENGR-433 lab circuits are not running high clock rates nor attempting to meet tight timing requirements and thus using a slower clock has an advantage. Thus for this class using a derived clock that has 50% duty cycle (square wave) as input to clocked components, and also using non-clock wiring in the FPGA to distribute a derived clock signal, is acceptable.

Here is a derived clock circuit, in this case creating a 1Mhz square wave, that should work reliably. Counters and other circuits run by this clock operate correctly and synthesis warnings are not produced..



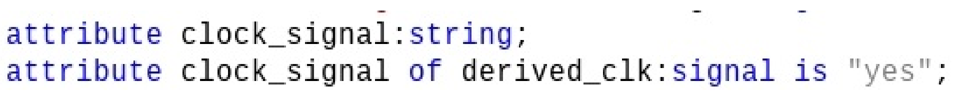
```
---------------------- 1 Mhz Clock Generation ----------------
   process(mclk)
   begin
      if (mclk'event and mclk='1') then
         clk_reg <= clk_next;    -- Clock divider register
         t_reg <= t_next;        -- T-F/F register
      end if;
   end process;
   clk_next <= (others=>'0') when clk_reg=24 else clk_reg+1;
   t_next <= not t_reg when clk_reg = 24 else t_reg;

   derived_clk <= t_reg;
```

Only use t_next to update t_reg. **Do NOT use t_next as a clock** signal.

(clk_reg, clk_next are data type unsigned;   t_reg and t_next are std_logic type)

In this circuit the master clock is divided down to twice the desired frequency which is then used to run a T-F/F that divides by two and creates a square wave with desired frequency. The output of the T-F/F is the derived clock.

To make the synthesizer happy and avoid errors or warnings, place these statements ahead of the begin statement in your architecture (along with the various internal signal definitions):

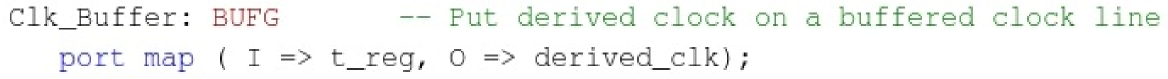


```
   attribute clock_signal:string;
   attribute clock_signal of derived_clk:signal is "yes";
```

Note that the word “derived_clk” in the statement above is the name of the clock signal and you can make it whatever you use in the clock generator circuit (so long as it is not a keyword).

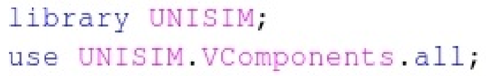
Distributing a derived clock using the global clock lines

If a derived clock will be driving many clocked components it may be good to use a global clock line to distribute it. To place a derived clock onto a global clock line a Bufg component is manually instantiated in your design. For example, the t_reg signal (see prior page) is connected to the input of the Bufg component and the output is then derived_clk:



```
Clk_Buffer: BUFG       -- Put derived clock on a buffered clock line
   port map ( I => t_reg, O => derived_clk);
```

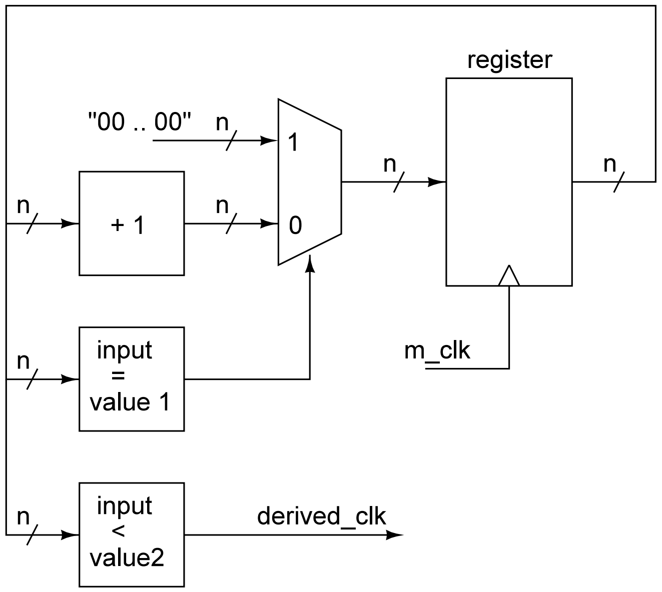
To use this part from the Xilinx library you must have these lines at the top of your VHDL file:



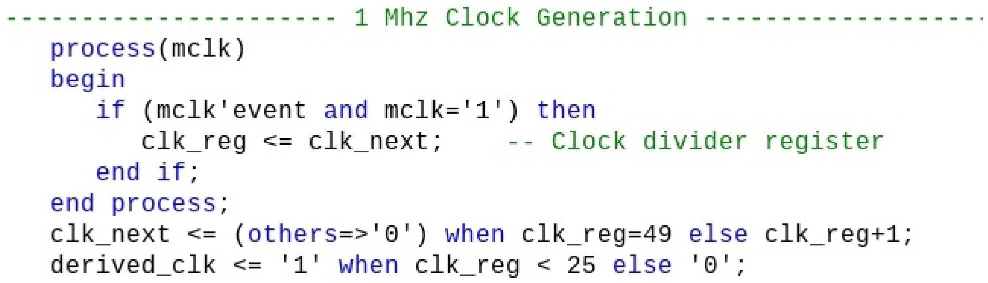
```
library UNISIM;
use UNISIM.VComponents.all;
```

Example of and explanation of a derived clock that **has problems**.

Consider the circuit shown below and VHDL statements that create this circuit:



Note that the derived clock is produced by combinational logic. Combinational logic can have glitches due to dynamic or static hazards etc. Thus if the derived clock connects to an edge triggered circuit, that circuit may see multiple edges when you expect that it will only see one edge. For example, an edge triggered counter may appear to increment on both edges of the derived clock if one of the clock edges has a glitch associated with it.



```
---------------------- 1 Mhz Clock Generation ------------------
   process(mclk)
   begin
      if (mclk'event and mclk='1') then
         clk_reg <= clk_next;    -- Clock divider register
      end if;
   end process;
   clk_next <= (others=>'0') when clk_reg=49 else clk_reg+1;
   derived_clk <= '1' when clk_reg < 25 else '0';
```

So Don't use the last statement in this clock description (the rest of it is ok).