# Example 3: Four-Bit Binary Counter

## 1. Objective

The objective of this lab is to design and test a 4-bit binary counter. Aside from learning about the on-board clock signal and push-buttons as well as about frequency dividers, this lab reinforces the design flow steps introduced in the previous labs.

***Note:*** *This document was written initially in the context of using the Atlys FPGA board. Almost everything in here is applicable to the board we use now, either DE2-115, or DE1-SoC, or ZedBoard. As you read, it should be obvious what you need to do specifically to the board you are using…*

## 2. Description

We design a 4-bit binary counter. Our counter has an output "Q" with four bits. During correct operation, the counter starts at "0000" and then binary counts up to output "0001", "0010", "0011", and so on until it outputs "1111", after which it resets to "0000" and starts again. The first implementation of our counter has only one input: a clock signal CK. The clock signal is provided by the external (to the FPGA) clock generator. We use the output Q to drive the first four LEDs on the Atlys board.

The block diagram of the simplest/basic structural implementation of such a binary counter is shown in the next figure. This implementation is known as a ripple counter.

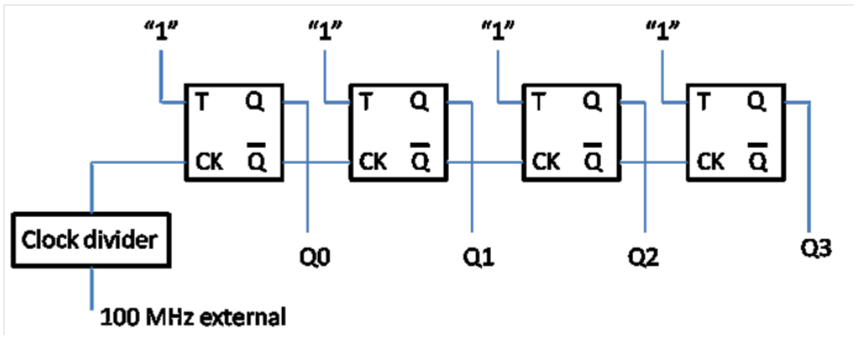

Figure 1 Block diagram of a 4-bit binary counter

***<u>Toggle Flip-Flop</u>***

As shown in the figure above, we use four Toggle Flip-Flops (TFF's). As you remember, the operation of a TFF is as follows: When the "T" input is logic "1", the output "Q" will toggle on each clock transition. When the "T" input is logic "0", the output "Q" will not change.

To start our design, we first create a new project by launching Xilinx ISE WebPack and following the steps discussed in lab 2. Call the new project **fourbit_counter** and select the same location where you created the previous project.

Create and add to the project a first VHDL file called **tff.vhd** with the following content:

```
-- tff.vhd
-- Toggle flip-flop with behavioral description
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my_tff is
  Port (
    T : in  STD_LOGIC;
    clk : in  STD_LOGIC;
    Q, QN : out STD_LOGIC
  );
end my_tff;

architecture My_behavioral of my_tff is

  signal mem : std_logic := '0';

begin

process (clk)
begin
  if T = '0' then null; -- no toggle, so do nothing
  elsif (clk'event and clk = '1') then
    mem <= not mem; -- rising edge of clock and T = 1, toggle stored value
  end if;
end process;

Q  <= mem;
QN <= not mem;

end;
```

## ***Clock Divider***

Our counter uses as a clock a signal generated by the on-board clock generator. This clock generator is a single 100 MHz CMOS oscillator on the Atlys board connected to pin L15 of the Spartan-6 FPGA. Because the frequency of 100 MHz is too high for the human eye to be able to see how the counter output drives the LEDs, we must utilize a clock divider to lower the frequency to about 1 Hz.

Create and add to the project a second VHDL file called **ck_divider.vhd** with the following content:

```
-- clk_divider.vhd
-- This is a clock divider. It takes as input a signal
-- of 50 MHz and generates an output as signal with a frequency
-- of about 1 Hz.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clk_divider is
  Port (
    CLK_IN : in STD_LOGIC;
    CLK_OUT : out STD_LOGIC
  );
end clk_divider;

architecture Behavioral of clk_divider is

constant TIMECONST : integer := 71;
signal count0, count1, count2, count3 : integer range 0 to 1000;
signal D : std_logic := '0';
begin

process (CLK_IN, D)
begin
  if (CLK_IN'event and CLK_IN = '1') then
    count0 <= count0 + 1;
    if count0 = TIMECONST then
      count0 <= 0;
      count1 <= count1 + 1;
    elsif count1 = TIMECONST then
      count1 <= 0;
      count2 <= count2 + 1;
    elsif count2 = TIMECONST then
      count2 <= 0;
      count3 <= count3 + 1;
    elsif count3 = TIMECONST then
      count3 <= 0;
      D <= not D;
    end if;
  end if;
  CLK_OUT <= D;
end process;

end Behavioral;
```

Read the above code to understand its operation. It takes the 50 MHz external clock as input CK_IN and generates an output signal CK_OUT of 1 Hz. The output frequency is adjustable according to the following formula (TIMECONST = 71 in this case in order to get an output frequency of about 1 Hz):

***Output Frequency = 50000000 / ( 2 * (TIMECONST ^ 4) )***

***Note:*** *There are other ways of implementing the TFF or the clock divider. In time, by accumulating more and more experience, you will develop your own VHDL programming style by adopting different coding techniques.*

## ***4-bit Binary Counter***

Finally, let's create a third VHDL file with the top-level description of our fourbit_counter design described in Figure 1. Create and add to the project the third VHDL file called **fourbit_counter.vhd** with the following content:

```
-- fourbit_counter.vhd
-- This is a simple 4-bit (Ripple) binary counter made up
-- of four T flip-flops.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fourbit_counter is
  port (
    clk_50 : in STD_LOGIC;
    Q : out STD_LOGIC_VECTOR (3 downto 0)
  );
end fourbit_counter;


architecture my_structural of fourbit_counter is

component my_tff
  port (
    T : in STD_LOGIC;
    clk : in STD_LOGIC;
    Q, QN : out STD_LOGIC
  );
end component;

component clk_divider
  port (
    CLK_IN : in STD_LOGIC;
    CLK_OUT : out STD_LOGIC
  );
end component;

signal all_T, S0, S1, S2, S3 : STD_LOGIC;
signal clk_1Hz : STD_LOGIC;

begin

-- use signal all_T to drive input T of all T flip-flops to logic '1';
all_T <= '1';

CLOCK: clk_divider port map (CLK_IN => clk_50, CLK_OUT => clk_1Hz);

TFF0: my_tff port map (T => all_T, clk => clk_1Hz, Q => Q(0), QN => S0);
TFF1: my_tff port map (T => all_T, clk => S0, Q => Q(1), QN => S1);
TFF2: my_tff port map (T => all_T, clk => S1, Q => Q(2), QN => S2);
TFF3: my_tff port map (T => all_T, clk => S2, Q => Q(3), QN => S3);

end my_structural;
```

## ***Design Implementation***

At this time, we have coded the entire design and its components. Before continuing to Design Implementation, we first take care of two things:

- Set the fourbit_counter as the top-level design (we need to do this because currently TFF.vhd is the top-level because it was added first to the project). To do that, in the Hierarchy window, Right click on "fourbit_counter – Structural (fourbit_counter.vhd)" and select Set as To-Level.

- Pin assignment. As discussed earlier we use the external clock signal connected to pin **L15** of the Spartan-6 FPGA. So, we assign pin L15 to the input "CK" of our design. Also, we use the output "Q" of our design to drive the first four LEDs of the Atlys board. Now, do the pin assignment as learned in lab 2. After this step, your UCF file should have the following content:

```
# PlanAhead Generated physical constraints
NET "Q[0]" LOC = U18;
NET "Q[1]" LOC = M14;
NET "Q[2]" LOC = N14;
NET "Q[3]" LOC = L14;
NET "CK" LOC = L15;
```

We are now ready to implement the design: in the Processes tab double-click Implement Design (or right-click on Implement Design and select Run).

***<u>Generate the Programming File and Program the FPGA</u>***

Double-click on Generate Programming File in the Process tab. Then, program the FPGA using the Adept software as learned in lab 2. Verify that our counter works correctly.

## 3. Lab assignment

***<u>Lab preparation</u>***

A major problem with the counter implemented in this lab is that the individual flip-flops do not all change state at the same time. Rather, each flip-flop is used to trigger the next one in the series. Thus, in switching from all 1s (count = 15) to all 0s (count wraps back to 0), we don't see a smooth transition. Instead, output Q(0) falls first, changing the apparent count to 14. This triggers output Q(1) to fall, changing the apparent count to 12. This in turn triggers output Q(2), which leaves a count of 8 while triggering output Q(3) to fall. This last action finally leaves us with the correct output count of zero. We say that the change of state "**ripples**" through the counter from one flip-flop to the next. Therefore, this circuit is known as a "**ripple counter**".
This causes no problem if the output is only to be read by human eyes; the ripple effect is too fast for us to see it. However, if the count is to be used as a selector by other digital circuits (such as a multiplexer or demultiplexer), the ripple effect can easily allow signals to get mixed together in an undesirable fashion. To prevent this, we need to devise a method of causing all of the flip-flops to change state at the same moment. That would be known as a "**synchronous counter**" because the flip-flops would be synchronized to operate in unison.

In this lab assignment, you must design a synchronous counter version of our fourbit_counter to arrive to a new block diagram, where all flip-flops are driven by the same clock signal. You should design this counter using the Karnaugh Maps method and utilize JK flip-flops instead of T flip-flops. In addition, the top-level design of the fourbit_counter should have an additional input, "RESET", which when set to logic "1" forces the counter to the initial state "0000". The RESET input should be controlled by one of the pushbuttons of the Atlys board.

***Optional:***

- *Remove entirely the clock divider from the design. Instead of the clock signal of 100 MHz utilize a signal from one of the pushbuttons of the Atlys board. In this case, the counter will advance each time the pushbutton is pressed.*
- *Modify the counter such that it can be told to count up or down.*