Laboratory 4 – Registers and Counters

Objectives: In this laboratory, you will gain experience in designing registers and counters. After completing this laboratory, you should be able to:

- Implement and demonstrate a frequency divider.
- Model and implement parallel and serial load registers.
- Model and implement a counter and show outputs on HEX displays.

Part 1 – Frequency Divider

The DE-10 lite FPGA board has a 50MHz clock (see user's manual), which is too fast to observe visually changes on the LEDs and board displays. In Part 1, you will implement and demonstrate a frequency divider.

As an example, let's say that you need to divide (Figure 1) the clock frequency from 50MHz to 5MHz, which corresponds to a clock period of Tdiv=200ns. A simple approach is to use a counter that toggles the output clock signal every 100ns. Remember that to toggle a signal means if it was a 1, it will be changed to a 0, and if it was a 0, it will be changed to a 1. In the DE-10 lite, the incoming clock is 50MHz, which has a period of Tin=20ns. If you create a counter based on a 50MHz incoming clock, it will increment every 20ns. If the same counter is incremented up to 5 before being reset back to 0, the reset event will occur every 5×20ns=100ns. This will result in an output signal with a HIGH time of 100ns, a LOW time of 100ns, and an overall period of 200ns. In this way, you have created a divided down clock with a frequency of 5 MHz. Figure 1 shows a graphical depiction of this example.

1.1. Download the Clk_divider_1Hz.vhd file posted on the unit webpage and examine the VHDL code, which divides the incoming 50MHz clock down to a 1Hz clock.

1.2. Demonstrate the VHDL code by observing the 1Hz blinking pattern of one the FPGA board LEDs.

1.3. Modify the VHDL code to have a 4Hz blinking pattern of the LED.

1.4. Check the correct functionality of the reset signal using one of the board switches.

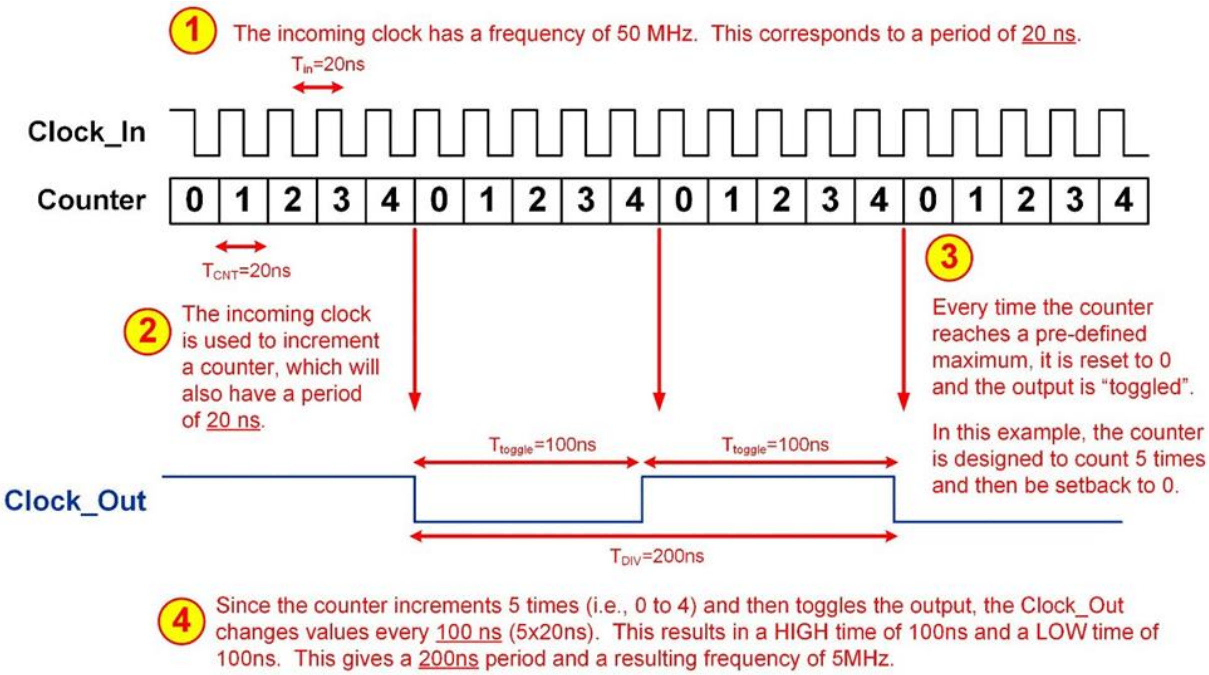

Figure 1 – Graphical Depiction of the Operation of a Clock Divider

Part 2 – Design and implementation of a parallel and serial load register

A register is a group of memory elements (typically edge-triggered flip-flops) which have their clock signal inputs tied together, and thus act as a single memory storage block operating on multiple bits of binary data (typically closely-related data such as bits within a word) at a time. The register stores data and can only change the outputs at the rising edge of the clock signal. In part 2, you will design a 4 bit parallel load, serial load clocked register, with following specifications:

INPUTS:

DATA	a 4 bit data signal.
RESET	an asynchronous reset (sets the outputs to logic '0'). Active low.
PLOAD	parallel load the output to equal the input at the next rising edge of the clock.
S_RIGHT	shift output one bit to the right at the next positive clock edge. Active high.
S_IN	data to shift into the most significant bit during a shift process.
CLK	clock signal. Output is to change on rising edges.

OUTPUTS:

Q	4 bit output signal.

RESET is to have highest priority.
PLOAD is to have a higher priority than S_RIGHT.
S_RIGHT has the lowest priority.

The tasks for Part 2 are as follows:

2.1. Use **behavioral modelling** to design a 4-bit parallel load, serial load clocked register, with the above specifications: Use a **generic statement** in your entity declaration to allow for the change of the size of the register by only changing one parameter value, in one place in your code.

2.2. Simulate your design using Modelsim, with the following sequence (Table 1) of inputs in 100ns time steps.

2.3. Verify your design on your FPGA board by appropriate pin assignment and clock period selection.

Time	CLK	DATA	RESET	PLOAD	S_RIGHT	S_IN
0	10MHz Clock Signal	"0000"	'0'	'0'	'0'	'0'
100ns		"0101"	'1'	'1'	'0'	'0'
200ns		"0000"	'1'	'0'	'0'	'0'
300ns		"1011"	'1'	'1'	'0'	'0'
400ns		"0110"	'1'	'0'	'0'	'0'
500ns		"0110"	'0'	'1'	'0'	'0'
600ns		"0110"	'1'	'1'	'0'	'0'
700ns		"0000"	'1'	'0'	'0'	'0'
800ns		"0000"	'1'	'0'	'0'	'1'
900ns		"0000"	'1'	'0'	'1'	'1'
1000ns		"0000"	'1'	'0'	'1'	'1'
1100ns		"0000"	'1'	'0'	'1'	'0'
1200ns		"0000"	'1'	'0'	'1'	'1'
1300ns		"1001"	'1'	'1'	'1'	'1'
1400ns		"1001"	'0'	'1'	'1'	'1'
1500ns		"1001"	'1'	'0'	'0'	'0'

Table 1 –Input Test patterns

Part 3 – <u>Design and implementation of a parallel and serial load register</u>

The following circuit is a 4-bit synchronous counter which uses four T-type flip-flops. The counter increments its value on each positive edge of the clock if the Enable signal is asserted. The counter is reset to 0 by setting the Clear signal low. You are to implement an **<u>8-bit counter</u>** of this type.

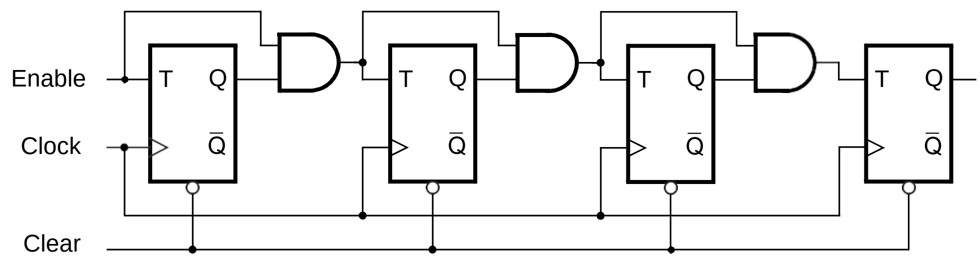

Figure 2: A 4-bit counter.

3.1. Write a VHDL file that defines an 8-bit counter by using the structure depicted in Figure 2. Your code should include a T flip-flop module that is instantiated 8 times to create the counter. Compile the circuit. Use the Quartus II RTL Viewer to see how Quartus II software synthesized your circuit.

3.2. Write a testbench and simulate your circuit to verify its correctness for an 8-bit counter.

3.3. Augment your VHDL file to use the 1Hz frequency clock generated in Part 1 as the Clock input, switches SW1 and SW0 as Enable and Clear inputs, and 7-segment displays HEX1-0 to display the hexadecimal count as your circuit operates. Make the necessary pin assignments needed to implement the circuit on the FPGA board, and compile the circuit.

3.4. Download your circuit into the FPGA chip and test its functionality by operating the implemented switches.