Laboratory Exercise 7 - Week 1
Counters

The purpose of this exercise is to build and use counters. The designed circuits are to be implemented on an Intel FPGA DE0-CV Board.

Part I

Consider the circuit in Figure 1. It 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 high. The counter is reset to 0 on the next positive clock edge if the asynchronous Clear input is low. You are to implement an 8-bit counter of this type.

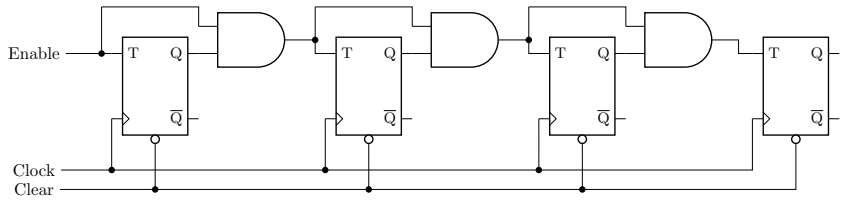

Figure 1: A 4-bit synchronous counter using T-type flip-flops.

perform the following steps:

1. Write a VHDL file that defines an 8-bit counter by using the structure depicted in Figure 1. Your code should include a T flip-flop entity that is instantiated 8 times to create the counter.
2. Compile the circuit. How many logic elements (LEs) are used to implement your circuit?
3. Simulate your circuit to verify its correctness.
4. Augment your VHDL file to use the pushbutton KEY0 as the Clock input and switches SW1 and SW0 as Enable and Clear inputs, and 7-segment displays HEX1-0 to display the hexadecimal count as your circuit operates.
5. Download your circuit into the FPGA chip and test its functionality.

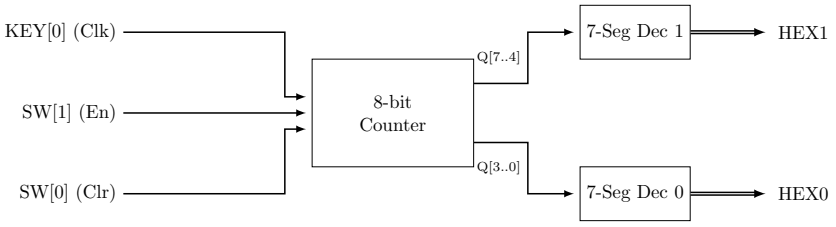

Figure 2: Top-level block diagram of the 8-bit counter.

The code for the T flip-flop entity is shown in Figure 3. For the top-level entity, refer to the block diagram in Figure 2.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY t_ff IS
    PORT ( T, Clk, Clear : IN STD_LOGIC;
           Q : OUT STD_LOGIC);
END t_ff;

ARCHITECTURE Behavior OF t_ff IS
    SIGNAL Q_int : STD_LOGIC;
BEGIN
    PROCESS ( Clk, Clear )
    BEGIN
        IF Clear = '0' THEN
            Q_int <= '0';
        ELSIF RISING_EDGE(Clk) THEN
            IF T = '1' THEN
                Q_int <= NOT Q_int;
            END IF;
        END IF;
    END PROCESS;
    Q <= Q_int;
END Behavior;
```

Figure 3: T flip-flop with asynchronous clear.

Part II

Another way to specify a counter is by using a register and adding 1 to its value. This can be accomplished using the `ieee.numeric_std` library and the `unsigned` type.

1. Write a VHDL file for a 16-bit counter using the following statement: `Q <= Q + 1;`.
2. Compile the circuit and determine the number of LEs needed.
3. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Part I.
4. Implement the counter on your DE0-CV board, using the displays HEX3-0 to show the counter value.

Figure 4 shows how to convert data between different types using `numeric_std` library.

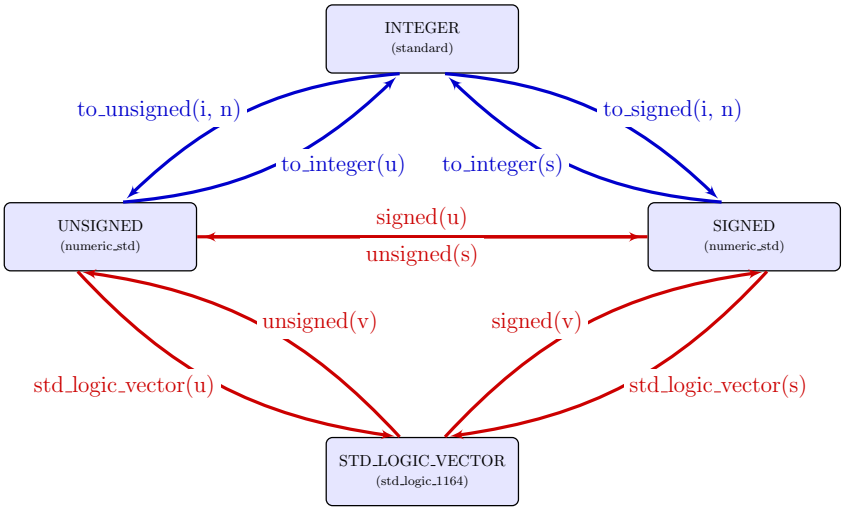

Figure 4: Data type conversions in `ieee.numeric_std`.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY part2 IS
    PORT ( Clk, Reset, Enable : IN STD_LOGIC;
           Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END part2;

ARCHITECTURE Behavior OF part2 IS
    SIGNAL count_val : UNSIGNED(15 DOWNTO 0);
BEGIN
    PROCESS ( Clk )
    BEGIN
        IF RISING_EDGE(Clk) THEN
            IF Reset = '0' THEN
                count_val <= (OTHERS => '0');
            ELSIF Enable = '1' THEN
                count_val <= count_val + 1;
            END IF;
        END IF;
    END PROCESS;
    Q <= STD_LOGIC_VECTOR(count_val);
END Behavior;
```

Figure 5: 16-bit counter using `numeric_std`.

An implementation example of the counter using `numeric_std` is provided in Figure 5.

Part III

Use an LPM from the Library of Parameterized modules to implement a 16-bit counter. Choose the LPM options to be consistent with the above design, i.e. with enable and asynchronous clear.

1. Create a VHDL file that instantiates `lpm_counter`.
2. Compare this version with the previous designs in terms of LE usage.
3. Verify using simulation.

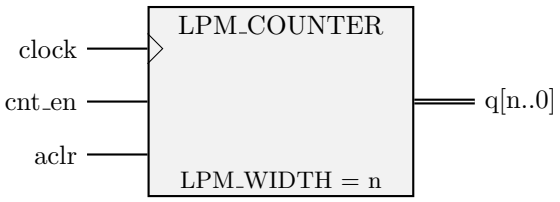

Figure 6: LPM Counter Block.

Figure 6 shows the LPM counter block representation.

Updated By: R. Sutthaweekul
Release Date: 2026-01-02