"FPGA-Programming" Exercise Sheet IV:

Counter

Goal of this exercise is to learn, how to describe and utilize counters with VHDL.

As target hardware always choose FPGA chip Cyclone V SoC 5CSEMA5F31C6

1) *Synchronous counter with T-flip-flops*

Following figure shows a synchronous 4-bit counter based on four toggle flop-flops elements.

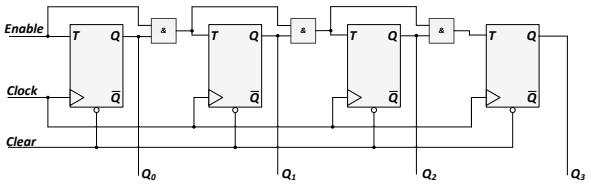

Figure 1.1: Circuit of a synchronous 4-bit counter

The counter increments its value on each positive edge of the clock if the *Enable* signal is high. The counter is reset to zero on the next positive clock edge by a synchronous *low-active Clear* input.

Goal of this part is to first implement a 8-bit version of the counter. Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_TFFcounter**.
ii) Include the VHDL sub-level entity **e_hex7seg** of exercise sheet III).
iii) Describe the VHDL sub-level entity of a T-flip-flop (**e_ToggleFF**).
Use a *sequential process* with synchronous low-active reset.
iv) Create the VHDL top-level entity (**e_my_TFFcounter**) of the 8-bit synchronous T-FF counter (**a_myTFFcounter_1**). Therefore 8 instances of the toggle flip-flop component should be used. In the design use $KEY[0]$ for the *Clock* input, use switch $SW[1]$ for the *Enable* input and switch $SW[0]$ for the *Reset* input. Furthermore use the 7-segment displays $HEX1$ and $HEX0$ in order to display the counter value in a hexadecimal fashion and $LEDR$ to display the counter value in a binary fashion.
v) Import the necessary pin assignments and compile the circuit. From the **Compilation Report** learn the number of resulting logic elements. Perform a functional simulation by creating a **test bench** and verify the correct functionality.
vi) Download the counter circuit to the FPGA device and test the functionality.
vii) Also implement a 4-bit version (**a_myTFFcounter_2**) of the circuit and use the **RTL Viewer** to examine how Quartus synthesises the circuit. What are the differences in comparison with above figure?

2) *Synchronous counter using high-level statements*

Another way to specify a counter is by using a register (by using a **std_logic_vector** signal) and cyclically adding 1 to its vale.

This can be accomplished using the following VHDL statement:

```
Q <= Q + 1;
```

Goal of this part is to first implement a 16-bit version of the synchronous counter by using the syntax stated above. Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_counter**.

ii) Include the VHDL sub-level entity **e_hex7seg** of part 1).

iii) Create a VHDL top-level entity (**e_my_counter**) for the 16-bit synchronous counter. Include all functions of **std_logic_unsigned** from the **ieee** library and describe the architecture of the circuit named **a_my_counter_1** with high-level syntax.

Use $KEY[0]$ as *Clock* input, switch $SW[1]$ as *Enable* input and switch $SW[0]$ as synchronous *Reset* input. Use the four 7-segment displays $HEX3$ to $HEX0$ to display the counter value in a hexadecimal fashion.

iv) Import the necessary pin assignments and compile the circuit. From the **Compilation Report** learn the number of resulting logic elements. Perform a functional simulation by creating a **test bench** and verify the correct functionality.

v) Download the counter circuit to the FPGA device and test the functionality.

vi) Also implement a 4-bit version (**a_mycounter_2**) of the circuit and use the **RTL Viewer** to examine how Quartus synthesises the circuit. What are the differences compared to the design of part 1) ?

3) *Synchronous counter using a LPM module*

Another way to describe a synchronous counter is by using a LPM module, which should be demonstrated within the part. A tutorial for using LPM modules in Quartus 17.0 can be found following the link:

`ftp://ftp.altera.com/up/pub/Intel_Material/17.0/Tutorials/VHDL/Using_Library_Modules.pdf`

For the realisation of the counter using a LPM module perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_LPMcounter**. As LPM module use **lpm_counter** for the implementation of the 16-bit counter. Name the LPM module **e_lpm_count**.

ii) Parametrize the LPM module in a way to exhibit a *enable* input and a synchronous *reset* (*clear*) input.

iii) In the design use push-button $KEY[0]$ as *Clock* input, switch $SW[1]$ as *Enable* input and switch $SW[0]$ as synchronous *Reset* input. Furthermore use the four 7-segment displays $HEX3$ to $HEX0$ in order to display the counter value in a hexadecimal fashion.

iv) Import the necessary pin assignments and compile the circuit. From the **Compilation Report** learn the number of resulting logic elements. Perform a functional simulation by creating a **test bench** and verify the correct functionality.

v) Download the counter circuit to the FPGA device and test the functionality.

4) *Running digits*

Goal of this part is to design a circuit that successively flashes the digits 0 through 9 on the 7-segment display $HEX0$. Each digit should be displayed for about one second. Use a counter to determine the approximately one-second intervals. As clock input signal of the counter the 50 MHz clock signal of the development board (**CLOCK_50**) must be used. No other clock signals should be derived in the design. All flip-flops of the design should be clocked directly by the same 50 MHz clock signal.

Following figure shows a small part of the design which should be created. The figure illustrates how a large bit-width counter can be used to produce an *enable* signal for a smaller counter. The rate at which the smaller counter increments can be controlled by choosing an appropriate number of bits in the larger counter.

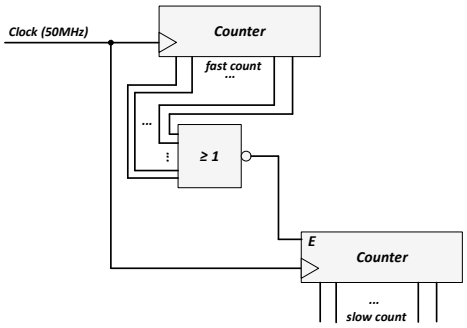

Figure 4.1: Possibility for the realisation of a slow counter

Implement and test this circuit. Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_slowclock**.
ii) Create a VHDL sub-level entity (**e_bcd7seg**), which uses a *combinatorial process* named **p_decode** that decodes a 4-bit vector as BCD number for driving a 7-segment display.
iii) Describe the functionality (**a_my_slowclock_1**) within the VHDL top-level entity. For both, the implementation of the fast (**p_slow_clock**) and the slow (**p_4bit_cnt**) counter use a *sequential process*. Determine the minimal bit-width of the counter value of the fast counter in order to display each number for round about one second.
iv) Import the necessary pin assignments, compile and download the circuit and test its functionality.
v) Furthermore create a second architecture description (**a_my_slowclock_2**) in which the behaviour will be described in a way that each number will be shown **exactly** for one second. Furthermore add a synchronous low-active reset ($KEY[0]$) to both counter and modify the enable condition in a way that the number 0 after reset will also be displayed for one second. Compile, download and test the circuit again.

5) *Rotating word*

Within this part the word **dE1** should rotate on the three 7-segment displays $HEX2$ to $HEX0$ from right to left in one-second intervals. Following table clarifies the rotation:

Clock cycle	Display		
0	d	E	1
1	E	1	d
2	1	d	E

Different possibilities for the realisation exist. One possibility is to re-use the VHDL code of exercise sheet I) part 5). Only small changes are necessary by replacing both switches by a 2-bit wide counter, which will be incremented at one-second intervals.

For the description of the circuit, perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_rotatingDE1**.

ii) Use the description of the fast counter of part 4) to provide an *enable* signal of precisely one-second interval. Implement a slow 2-bit wide counter (**p_2bit_cnt**) which only uses three of the four possible states. Use push-button $KEY[0]$ as synchronous reset for both counter.

iii) Include the VHDL entities **e_char7seg** and **e_2bit3to1mux** from exercise sheet I) part 5).

iv) Import the necessary pin assignments.

v) Compile, download and test the functionality of the circuit.

6) *Running word*

Create an extended version of the circuit of part 5) in a way that the word **dE1** runns through all six 7-segment displays from right to left. Following table illustrates the pattern:

Clock cycle	Display					
0				d	E	1
1			d	E	1	
2		d	E	1		
3	d	E	1			
4	E	1				d
5	1				d	E

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_runningDE1**.

ii) Describe the behaviour (**a_my_runningDE1_1**). Use the circuit of the previous part as basis and extend the functionality accordingly. Use the entities **e_char7seg** and **e_2bit6to1mux** of exercise sheet I) part 6). Use push-button $KEY[0]$ as synchronous reset for both counters.

iii) Import the necessary pin assignments. Compile, download and test the functionality of the circuit.

iv) Also create a further architecture description (**a_my_runningDE1_2**) which extends the functionality of the circuit by a direction signal. Use switch $SW[0]$ for the determination of the direction. If switch $SW[0] = 0$ than the pattern of the table above should be the case. If switch $SW[0] = 1$ than the direction should be the opposite. Hint: Perform the required changes in the slow counter **p_2bit_cnt**.

v) Compile, download and test the functionality of the circuit with direction selection extension.