"FPGA-Programming" Exercise Sheet V:

Timers and Real-time Clock

The purpose of this exercise is to study the use of clocks in timed circuits.

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

1) *Generic modulo-k counter-module*

   Create a modulo-k counter-module, by extending the design of the 16-bit synchronous counter of exercise sheet IV) part 2) by using two additional parameters. The parametrisable bit-width of the counter should be n. The n-bit counter should count from 0 up to $k-1$. If the counter has reached the value of $k-1$ than the next value will be 0. To the entity add an additional output named **sl_rollover**, which will be set to 1 in the clock cycle where the count value is equal to $k-1$.

   Perform following steps:

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

   ii) Describe the VHDL sub-level entity (**e_modulo_counter**) of the parametrisable n-bit wide modulo-k counter. For the definition of these two parameters use the keyword **generic**. Set the default values to $n = 4$ and $k = 15$. Describe the functionality by a positive-edge triggered *sequential process* with a asynchronous low-active reset (**sl_reset_n**). As outputs the entity should have the counter values (**slv_Q**) and a rollover signal (**sl_rollover**), which indicates the counter value $k-1$.

   iii) Write the top-level entity named **e_my_modcnt**, which instantiates the component **e_modulo_counter** with parameters set to $n = 8$ and $k = 20$. The circuit should use push-button $KEY[1]$ as manual clock input. The current counter value should be displayed on $LEDR[7-0]$. The **rollover** signal should be displayed on $LEDR[9]$.

   iv) Import the necessary pin assignments, compile the circuit, perform a functional simulation and test the circuit in hardware.

2) 3-*digit BCD-counter*

Use the circuit of part 1) and implement a 3-digit BCD counter. Therefore use three instances of the modulo-k counter circuit. Display the contents of the counter on the 7-segment displays, $HEX2$ to $HEX0$. Connect all of the counters of the circuit to the 50 MHz clock signal and make sure that the BCD-counter will be incremented at one-second intervals. Use push-button $KEY[0]$ as asynchronous reset in order to reset the BCD counter to zero.

Perform following steps:

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

ii) First, describe a VHDL sub-level entity named **e_bcd7seg** in which a BCD to 7-segment decoder should be described by using a *conditional signal assignment*.

iii) Write a parametrisable VHDL sub-level entity named **e_modulo_counter_er** for the realisation of a synchronous n-bit wide modulo-k counter. The circuit should feature a asynchronous low-active reset **reset_n** and a synchronous enable **enable**. The circuit should exhibit the same outputs like the one of part 1).

iv) Create the top-level entity (**e_my_BCDcounter**) and describe the requested functionality. Instantiate a modulo-k counter named **I_slow_clock** in which the enable input is always set. The rollover signal of the counter is **sl_one_sec_en_int**. Furthermore instantiate three modulo-k counter for providing the three BCD digits. Define the values of the parameter n and k. Assure that all flip-flops of the design will be clocked by the same clock signal **CLOCK_50**. Think about how to wire the enable-inputs and the rollover-outputs.

v) Import the necessary pin assignments, compile the circuit and test the functionality in hardware.

3) *Real-time clock*

Design and implement a circuit, that acts as a real-time clock. It should display the minutes (from 00 to 59) on the 7-segment displays $HEX[5]$ and $HEX[4]$, the seconds (from 00 to 59) on $HEX[3]$ and $HEX[2]$ and the hundredths of a second (from 00 to 99) on $HEX[1]$ and $HEX[0]$. Use switches $SW[7-0]$ to preset the minute part of the time displayed by the clock when $KEY[1]$ is pressed. Stop the clock whenever $KEY[0]$ is pressed and continue the clock when $KEY[0]$ is released.

Perform following steps:

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

ii) Include the entity of the 7-segment decoder (**e_bcd7seg**) from part 2).

iii) Include the entity named **e_modulo_counter_er** from part 2).

iv) Describe a parametrisable VHDL sub-level entity named **e_modulo_counter_sler** which implements a synchronous n-bit wide modulo-k counter. The circuit should exhibit a synchronous enable (**sl_enable**). If the enable signal is set, the behaviour of the circuit featuring a synchronous reset and a synchronous load signal, should be described by using a priority encoder. Therefore the circuit exhibits two additional input signals, namely a control signal **sl_load** for presetting the input data given by **slv_d_in**. Output signals of the circuit are the counter value **slv_Q** and the signal **sl_rollover**, which indicates the counter value $k-1$.

v) Create the top-level entity (**e_my_RTC**) and describe the requested functionality. Use the component **e_modulo_counter_er**, once for providing pulses in hundredths of a second intervals, and for creating the BCD digits of the ones and tens for displaying the hundredths of a second, and the seconds.
Use the component **e_modulo_counter_sler** for the creation the BCD digits of the ones and tens for displaying the minutes. Think about how to wire the enable and reset inputs and how to use the rollover outputs signals.

vi) Import the necessary pin assignments, compile the circuit and test it in hardware.

4) *Morse Code*

An early method of telegraph communication was based on the Morse Code. Following figure illustrates the international Morse-Code.

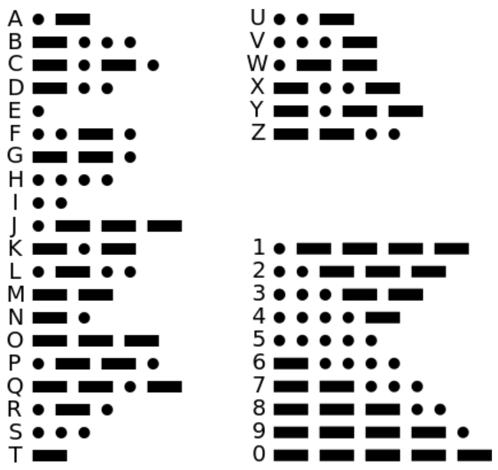

Figure 4.1: Morse Code

This code uses patterns of short an long pulses to represent a message. Each letter is represented as a sequence of dots (a short pulse), and dashes (a long pulse). Here a dot represents one time unit, a dash represents three time units. The gap between two parts of the same character is one time unit, the gap between the characters of a word is three time units, the gap between two words is seven time units.

Goal of this part is to design a circuit, that takes as input one of the *first eight* characters of the Morse-Alphabet and displays the Morse-Code of this character on the red LEDs. Therefore the circuit should use switches $SW[2-0]$, as well as buttons $KEY[1-0]$ as inputs. If $KEY[1]$ is pressed, the circuit should display the Morse-Code of the character which has been set by the switches $SW[2-0]$ ($000 \hat{=} A$, $001 \hat{=} B, \ldots$). $KEY[0]$ acts as asynchronous reset input. Following figure shows the high-level diagram of the circuit.

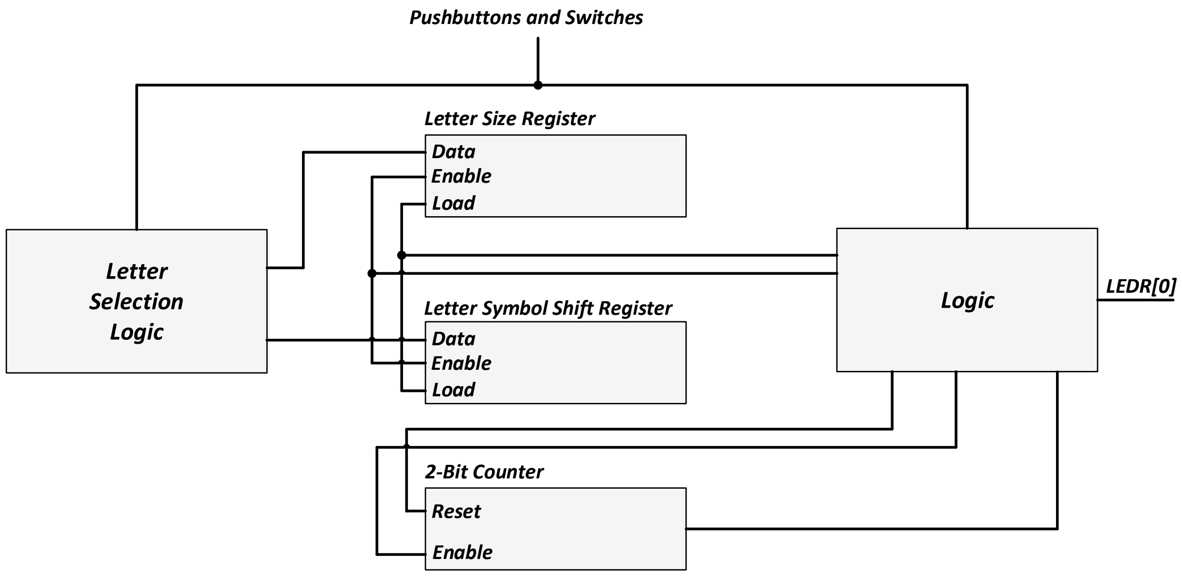

Figure 4.2: High-level illustration of the Morse-Code circuit

Perform following steps:

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

ii) Describe a parametrisable VHDL sub-level entity named **e_modulo_counter_ser** which implements a synchronous n-bit wide modulo-k counter. The circuit should exhibit a synchronous enable (**sl_enable**). If the enable signal is set, the behaviour of the modulo-k counter with synchronous reset should be described. Output signals of the circuit are the counter value **slv_Q** and the signal **sl_rollover**, which indicates the counter value $k - 1$.

iii) Create the top-level entity (**e_my_morse**) and describe the requested functionality. For the specification of the characters and the length of the characters use the VHDL keyword **constant**. Use the component **e_modulo_counter_ser** to describe a 0.5 seconds counter named **I_half_sec** whose enable signal is always set. Use the component **e_modulo_counter_ser** to realise a 2-bit counter named **I_pulse_counter** which counts the amount of elapsed 0.5 second intervals.

iv) Code a *sequential process* with asynchronous reset which controls the signal vectors **slv_send_data_int** and **slv_data_size_int**.
The signal vector **slv_send_data_int** takes the character **slv_chosen_letter_int** and the signal vector **slv_data_size_int** takes the size of the chosen letter in Morse pulses **slv_letter_size_int**. In the reset condition both signal vectors should be reset. Otherwise, at each positive clock-edge it should be checked, if push-button $KEY[1]$ is pressed. New values will only be adopted, if the last chosen character has been

completely displayed. If a new character has been chosen and the reset signal of the 2-bit wide counter is set, than the vector **slv_send_data_int** should be shifted to the right by one bit and the signal vector **slv_data_size_int** should be decremented.

v) Think about how the enable signal **sl_pulse_cnt_en_int** and the reset signal of the 2-bit counter **sl_reset_pulse_counter_int** have to be described.
The signal vector for choosing characters **slv_chosen_letter_int** and the signal vectors holding the length of the character **slv_letter_size_int** should be determined by a *conditional signal assignment*.
Furthermore determine the boolean expression for driving $LEDR[0]$ which is depending on the count value of the 2-bit counter **slv_pulse_cycle_int** and the vector **slv_data_size_int**.

vi) Import the necessary pin assignments, compile the circuit and test it in hardware.