Exercise 11:

a) Pseudorandom Number Generator

This module creates a sequence of pseudo-random numbers.

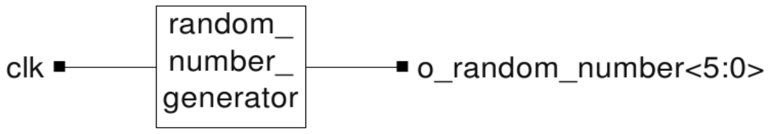

Figure 1: random_number_generator

Port declaration:

● *clk*	: in std_logic
● *o_random_number*	: out std_logic_vector (5 downto 0)

This is done using a linear feedback shift register. Such a register with a length of n bits creates a sequence of numbers between 1 and 2^n-1, where each number appears only once before the entire sequence is repeated. (For detailed information on linear feedback shift registers, see [1]). The polynomial for 6 bits is:

x(1) <= x(6) XOR x(5)

NOTE: 0 is not a valid value!

This random number generator is later used in the Pong game to create a random number to be utilized for the ball movement. It will determine the initial direction and speed of the ball when the player serves. Additonally, it will be used to change the angle of the edge of the paddle every time the ball hits it.

Implement the randon number generator in VHDL, simulate its behavior and test it with the given testbench.

1 http://en.wikipedia.org/wiki/Linear_feedback_shift_register

b) Event Trigger

This module generates the so-called event signal for the ball and paddle movement. These signals are then used by the modules *ball_movement* and *paddle_movement*. Each time these signals become active, there is a possibility for the ball or paddles to move. This is done to divide the system clock and slow down the ball and paddle movement to a value the user can react to.
The time between two events can be changed by the generics *G_PADDLE_EVENT_INTERVAL* and *G_BALL_EVENT_INTERVAL*. These generics define the number of clock cycles between two events. When this time has elapsed, the corresponding signal must become active for one clock cycle.

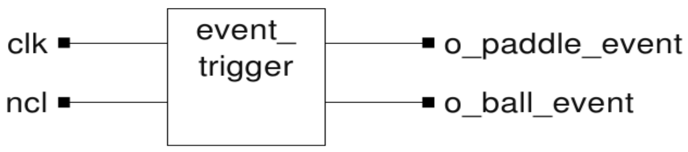

Figure 2: event_trigger

Port declaration:

● *clk*	: in std_logic
● *reset*	: in std_logic
● *o_paddle_event*	: out std_logic
● *o_ball_event*	: out std_logic

Generic declaration:

● *G_PADDLE_EVENT_INTERVAL*	: integer
● *G_BALL_EVENT_INTERVAL*	: integer

This module should then be tested with your own testbench.

c) Input Decoder

The input decoder scans the PS/2 data input of the FPGA board and reads data from the keyboard. Because the PS/2 port runs at its own clock frequency, there is already a process given which samples the PS/2 clock each system clock cycle and generates a signal to signify a negative PS/2 clock edge. This signal is called *new_ps2_data*. When it becomes active, the value of the input i*_ps2_data* is valid.
Now your task will be to implement the rest of the input decoder. It should support the following features:

- recognition for pressing a key and setting the corresponding output
- recognition for releasing a key and disabling the corresponding output
- a time-out function for the case that a scan-code is not completely transmitted
- a detection if both movement keys for one player are pressed simultaneously to prevent the up and down outputs of one player from being active at the same time.

A communication from the FPGA board to the keyboard is not necessary. Received data which can not be associated with the pressing or releasing of a key can be ignored. All outputs should be active as long as the corresponding keys are pressed.

For information on the PS/2 protocol, see the FPGA evaluation board manual[2].

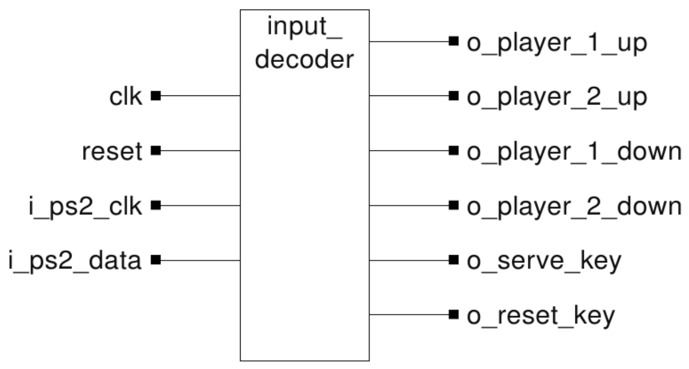

Figure 3: input_decoder

Port declaration:

clk	: in std_logic
reset	: in std_logic
i_ps2_clk	: in std_logic
i_ps2_data	: in std_logic
o_player_1_up	: out std_logic
o_player_2_up	: out std_logic
o_player_1_down	: out std_logic
o_player_2_down	: out std_logic
o_serve_key	: out std_logic
o_reset_key	: out std_logic

2 http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-rm.pdf