Assignment 3

[Working directory: **busarbiter**]

In this task you will prove the correctness of the implementation of an *arbiter* in a bus system.

In order to get started, change the working directory in OneSpin to **busarbiter**. Open the "Read Verilog" dialog. **First, select and read-in in the package file busarbiter-package.sv.** Then read-in the remaining SystemVerilog source files.

Design Specification

Fig. 1 shows the bus system with three masters, one slave and the arbiter to be verified. Masters may request bus access using their individual request signal. Only one master at a time is granted access by the arbiter. The bus transaction begins with the grant signal. The master deasserts its request signal after receiving the grant signal. The slave signals completion of the transaction by asserting its **ack** signal for the duration of one clock cycle. (After assertion, the **ack** signal goes low for at least one clock cycle.) The grant signal stays stable during the whole transaction. Another transaction can begin in the immediate clock cycle after the **ack** signal was asserted.

The arbiter follows a *static priority* arbitration policy. Master 0 has the highest priority, Master 2 the lowest. Whenever more than one master requests bus access, the one with the highest priority wins arbitration.

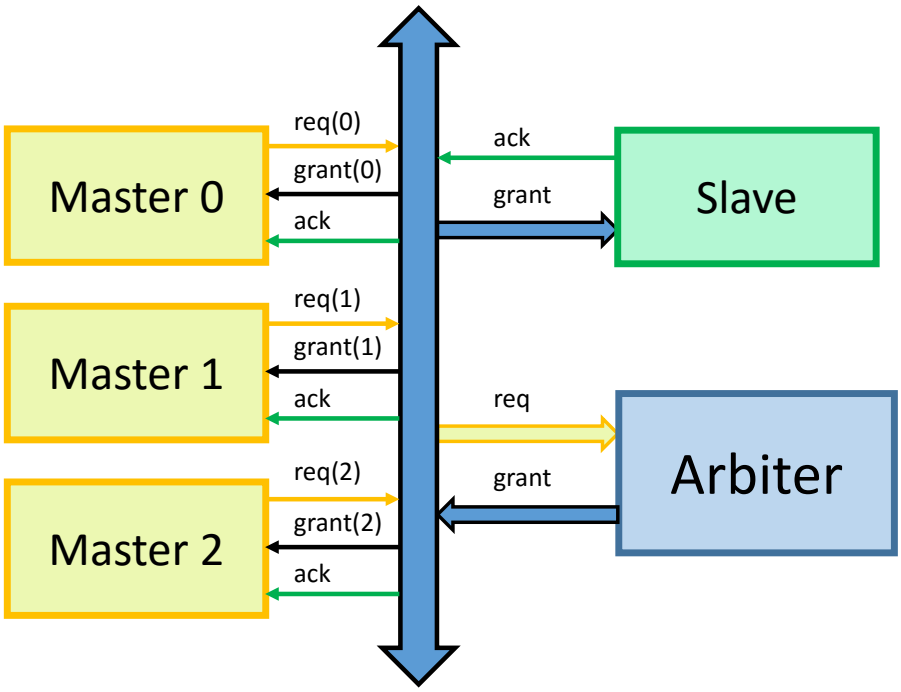

Figure 1: Bus system with three masters, one slave and an arbiter

Arbiter input signals:

clk:	**std_logic**	– clock signal
reset:	**std_logic**	– reset signal (active high)
bus_req:	**std_logic_vector(2 downto 0)**	– requests by masters
ack:	**std_logic**	– from slave, signaling completion of transaction

Arbiter output signals:

bus_grant:	**std_logic_vector(2 downto 0)**	– grant signals, one for each master.

Task 1

Write a property **p_reset** that verifies the reset behavior of the arbiter. What are the outputs of the circuit after the reset?

The file **busarbiter.sva** may help you set up the proof module.

Task 2

Write a property **p_at_most_one_grant** that proves that at any point in time, not more than one master is being granted access to the bus.

Task 3

As stated above, a bus transaction begins with the assertion of a grant signal. Write a property **p_grant_stable** that proves that during a transaction the grant signal remains stable.

(Hint: Do not use SVA constructs other than **$rose(), $fell(), $stable() and implies**.)

Task 4

Write a property **p_arbitration_master0** that verifies correct arbitration when Master 0 requests and wins arbitration. In the same way, write two properties **p_arbitration_master1** and **p_arbitration_master2** verifying correct arbitration when Masters 1 and 2 request and win arbitration, respectively.

Task 5

Write a property **p_grant_master1** proving that if Master 1 is being granted then Master 1 has requested and Master 0 has not requested.

(Hint: Use the SVA construct **implies** to formulate an implication between two sequences.)

Analogously, write a property **p_grant_master2** verifying the correct preconditions for Master 2 being granted access.