Accelerating Systems with Programmable Logic Components – 1DT109

2023 HT1 – HT2

Lab 1 - Interfacing with a Serial Keyboard

Lab rule.

To pass a lab, you have to book a time with a TA with one of the two lab sessions for each lab, and show your solution together with your partner to the TA. Note that the lab is mandatory, meaning that you need to pass all labs in order to do the project in P2. Ideally, you should demonstrate your solution by the end of each lab session 2. However, we will not make this a hard deadline. But we indeed require that you should have every lab checked before P2.

The main purpose of the lab session is for you to get help from the TAs, to deepen your understanding of Verilog, and to have fun. To make the most out of the lab sessions, please get well prepared for the lab tasks and be clear about your problem. Please don't ask questions such as how and what to do for a lab task. Before each lab, the lab handout lecture answers questions exactly for such a purpose.

Before P2 starts, we will have a special lab session where we all sit down and explain the solution together. Everyone should have the lab checked before this lecture. Otherwise, the corresponding lab is considered failed.

Overview.

The purpose of this lab is to get you familiar with Verilog design methodology from reading specification, then design, verification, and finally waveform checking.

In this lab you are going to design a circuit and interface it with an external serial keyboard. We will only be using the Xilinx **simulator** for this lab (no real FPGA or real keyboard is required).

You can design at any level of abstraction; you are particularly encouraged to use RTL-level modelling. Finally, you are required to verify your design using the provided testbench.

Requirements.

The know-how gained when doing the exercise, and the lectures on Verilog.

Tools.

Xilinx Vivado Design Suite

Intended Learning outcomes.

- Mastering problem analysis and design partitioning.
- Real-world RTL design example (interfacing with an external device).

Assessment.

- Your design needs to produce the expected result using the provided test bench.
- Both students must understand all parts of the solution by themselves (this will be checked), and both students must be present during the demonstration. If you cannot both be present during any of the lab sessions for demonstration and have a valid reason, get in touch with the teaching assistants to book another time slot.

1 Background: Simplified Serial Keyboard Protocol

You can read below details about the serial keyboard communication protocol:

- http://www.burtonsys.com/ps2_chapweske.htm
- https://en.wikipedia.org/wiki/PS/2_port

For this lab, however, we assume a simplified version of the protocol: a uni-directional communication from keyboard to the host. We also assume that only a single character is sent from the keyboard when a key is released, i.e., holding down a key does not send any characters (in the real protocol, separate characters are sent per key press and release, i.e., scan and break codes).

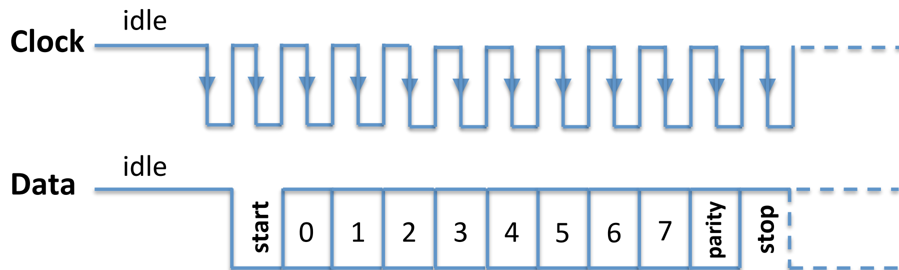


Figure 1 Simplified serial keyboard communication protocol.

Communication is performed using two signals: clock and data. When keyboard is idle, both the clock and the data signals have the logic value "high" (logic "1"). Before sending a character, keyboard starts generating the keyboard clock signal. Keyboard then initiates the communication by sending the "start" bit. This is done by keeping the data value "low" (logic "0") for one clock cycle (see above figure).

N.B. In the lab, we assume that the keyboard always changes the data values on the **rising edge** of the **keyboard clock**. Therefore, the host should read the valid keyboard data on the **falling edge** of the **keyboard clock**. Each data value remains valid for one keyboard clock cycle.

Keyboard then sends the ASCII code of the character (8 bits) starting with **least significant bit (LSB)**, followed by a parity bit used for error detection/correction. Finally, the keyboard sends a stop bit (logic 1) that signals the end of the current transmission.

We are not using the parity bit for this lab; therefore, parity bit and stop bit are discarded after they are received (but they should still be received by the host).

Your task is to design a module for a **host system module** that **receives** characters from a keyboard and displays them. Your module should contain the ports in Figure 2.

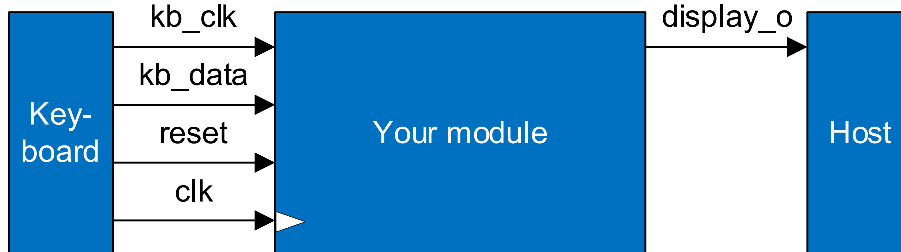


Figure 2 Module inputs and outputs.

Note the differences between the "kb_clk" and "clk" signals. "kb_clk" is the **keyboard clock** used for synchronizing the data transfer from the keyboard, whereas the "clk" signal is the **host system clock** used as the reference clock in your module to implement the sequential part of your logic.

2 Module Design

Although you are welcome to come up with your own design, you are strongly encouraged to follow the example design in Figure 3.

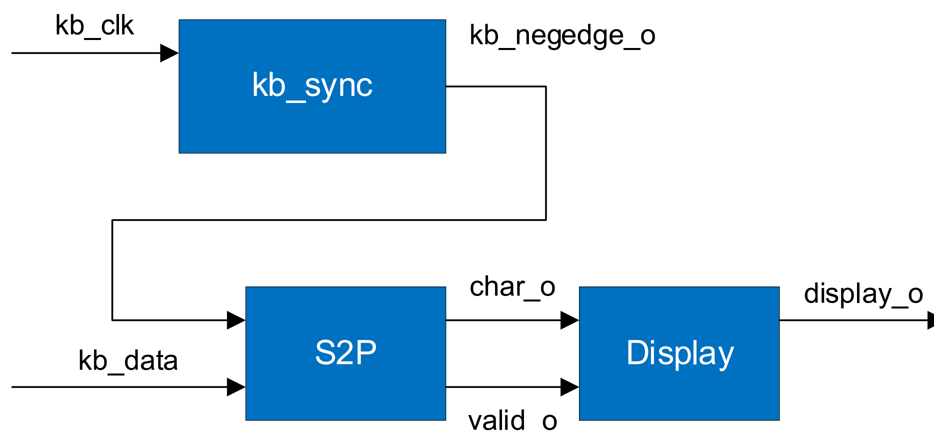


Figure 3 Example design with submodules (clock and reset signals omitted).

2.1 The kb_sync Module

Detects the falling edge of the "kb_clk" signal. This can be done by sampling the "kb_clk" signal on every rising edge of the system clock "clk" and compare the value with the previous sample. "kb_negedge_o" is asserted whenever "previous_value == 1'b1" and "current_value == 1'b0." We assume that the frequency of "clk" is much

higher than that of “kb_clk_i”. Think about why driving the “kb_sync” module directly with “kb_clk” is not a good idea.

2.2 S2P Module

Communication between the keyboard and the host is a serial communication. The Serial-to-Parallel (S2P) serially receives the bits (kb_data) from the keyboard. S2P module receives the “start” bit (refer Figure 1) that marks the beginning of the data transmission, followed by 8 data bits and a parity. Finally, the “stop” bit marks the end of the transmission. S2P discards the “parity” and “stop” bits and extracts the ASCII code of the pressed key and sends the character (b bits) to the display module. When the character is ready, the “valid_o” signal should be asserted for one clock cycle, which notifies the “Display” module to read the valid character.

S2P can be implemented using a **shift-register** and a **finite state machine**. Shift-register is used to receive the bits serially (from LSB to MSB) and to extract the 8 bits that correspond to the ASCII code. The sequence of reading the bits serially can be controlled using a finite state machine.

> Tips! A finite state machine is a memory that holds the current state of the system and transitions into new states according to the events (changes in input values). You can then decide which operations should be performed in each state (i.e., how output signals should change in each state). State machines can be used to solve a variety of problems, e.g., to generate the control signals needed to move data along a data path (for instance to generate the control signals in each pipeline stage). You can read about state machines both in the lecture slide and online sources, as well as the example in the Appendix.

2.3 Display

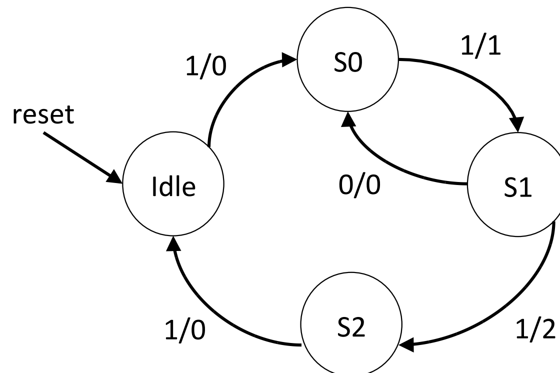
Display module reads a new character when the “valid_o” signal is asserted. In essence, display module can be simplified as a register.

2.4 Your Task

Complete the design and validate it with the provided test bench. Add the test bench files to your project by choosing “Add Sources → Add or create simulation sources” in Vivado. The test bench provides the clock and reset generators, as well as the keyboard emulator. The keyboard emulator sends out “Accelerator – 1DT109” (mind the spaces “ ”). Instantiate your design in the test bench (DUT) and verify that it

receives and displays the message. You may need to declare wires to connect the components.

Appendix – A state machine example



Assume the four states shown in the figure. Events (i.e., input values) are evaluated on the rising clock edges. Based on the current state and the event, the system either transitions into a new state on the rising clock edge or remains in the current state.

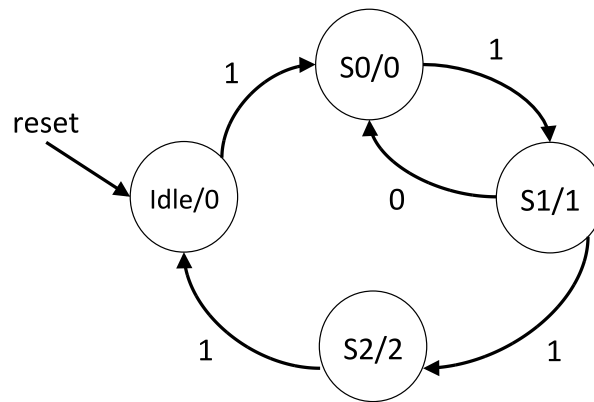
Besides state changes, the output values may also change upon changes in the input values. For instance, input value "1" in state "idle" results in transition into state "S0" and the output will be set to value "0". Likewise, input value "1" in state "S1" results in transition into state "S2" and the output will be set to value "2".

A state machine can be designed using two different approaches depending on how outputs react to changes in the inputs.

- **Mealy Machine**. outputs react immediately to the changes in the inputs without waiting for the next rising clock edge, i.e., outputs depend on the current state and the input values.
- **Moore Machine.** outputs do not immediately react to the changes in the inputs; outputs wait for the rising clock edge and change upon state transitions, i.e., outputs only depend on the current state.

The state machine shown above represents a Mealy machine, since outputs immediately change when input values change. The first value on each arc represents the input value, and the second value represents the immediate change of the output value (input-value/output-value;) however, the states change on the rising edge of the clock.

The Moore equivalent of the above machine is as follows.



Note that Moore machine may require more states. However, Moore machine removes the glitches from the output values and does not result in race conditions when communicating with other machines. An example Verilog representation of the machines is given as attachments moore.v and mealy.v, respectively.