Digital Design and Computer Architecture LU

Lab Exercise 1

Robert Najvirt, Thomas Polzer
{rnajvirt, tpolzer}@ecs.tuwien.ac.at
Department of Computer Engineering
University of Technology Vienna

Vienna, October 10, 2016

1 Overview

The first lab exercise will make you acquainted with the tools used in this lab and design your first FPGA system with VHDL. A basic FPGA design flow consists of a simulator and tools for synthesis and place & route. The simulator is used for verifying and debugging the functionality and the timing of the circuits. The synthesis tools translate the behavioral and/or structural description into a gate-level netlist. This netlist can then be mapped to the FPGA's logic cells. Finally the produced bitstream file is used to configure the FPGA.

Note that this document is not the complete assignment. Please view the protocol template for all required measurements, screenshots and questions to be answered.

2 Required Reading

- Design flow tutorial
- VHDL modeling slides
- State machine slides
- IP core documentation
- Logic Analyzer Resources

3 Task Description

In the first task, you will need to connect provided modules using a structural description to yield a functional system. In the second task, you will use simulations to verify and debug the design. In the third task, you will enhance the system by connecting another component to it that you will design yourself using behavioral description. In the fourth task, you will verify the running design using logic analyzer measurements.

Task 1: Structural modeling Your task is to write a top-level structural VHDL description of a system.

Figure 1 and Figure 2 show how the components need to be connected. The description must be done in VHDL and should contain only structural

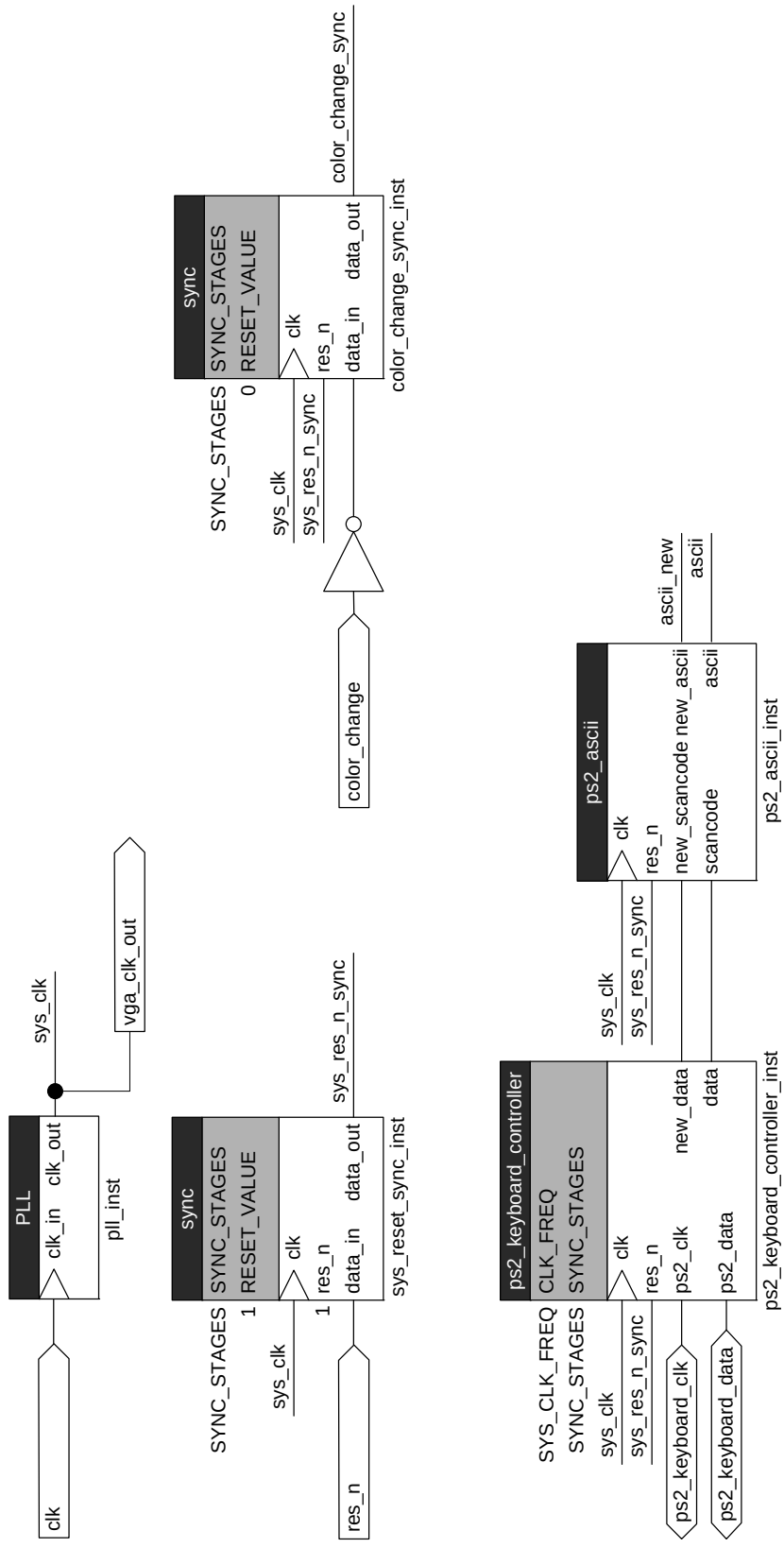

Figure 1: Structural system description (part1).

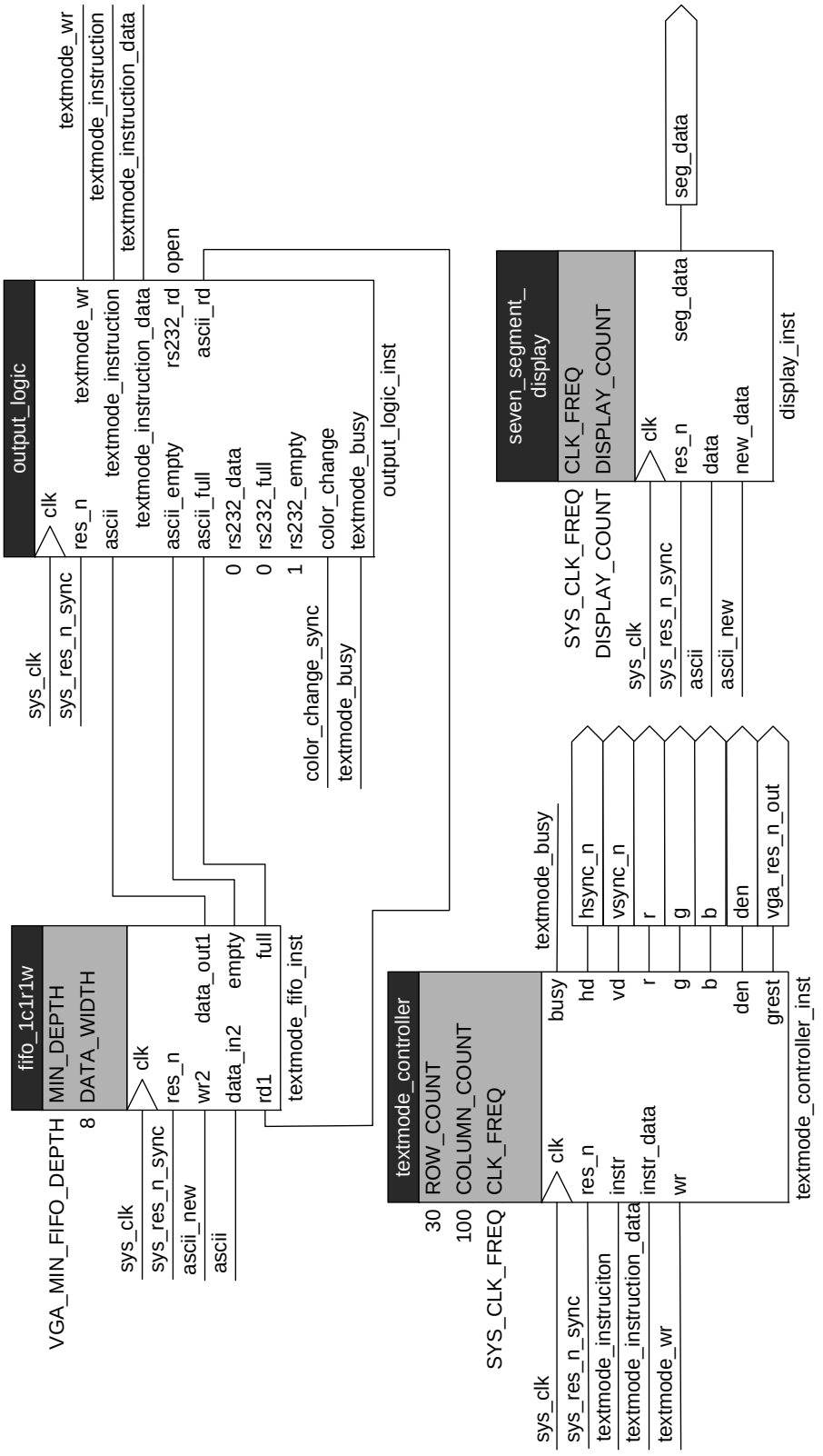

Figure 2: Structural system description (part2).

primitives (component instantiations and concurrent signal assignments). All information needed to wire the IP cores together is contained in the figures.

Add all needed IP cores and your top-level description to a Quartus project. The PLL displayed in the figure is not supplied. You need to generate it using the corresponding wizard within Quartus (see lab documentation). The frequency of the system clock (PLL output) should be 25 MHz.

The generics used in the figures are defined in Table 1. Make sure you use constants in VHDL. Do not set the values directly in the generic map.

Constant	Value
SYS_CLK_FREQ	25000000
SYNC_STAGES	2
VGA_MIN_FIFO_DEPTH	10
DISPLAY_COUNT	2

Table 1: Used generics.

After the PLL is created, add the necessary pin assignments and synthesize the project. Download the resulting bit stream to the FPGA board and make sure, that the system works. Do not forget to take screenshots for the lab protocol.

Task 2: Simulation In this task, you will perform simulations of the system from Task 1 at two stages of the design. Firstly the VHDL files describing the system will be simulated directly to show the modeled behavior, secondly a post-layout simulation that uses files derived from the synthesis and place & route processes will reveal behavior and timing of the system very close to that of running hardware.

Behavioral simulation Perform a behavioral simulation of the system. An appropriate testbench is provided in the simulation directory. Add all signals of the top-level entity to the waveform window and run the simulation long enough to trace the propagation of the three characters input to the PS2 interface through to the *seven_segment_display* and the *textmode_controller*. Take screenshots showing the propagation of the characters through the system including the *sys_clk, ps2_keyboard*, ascii*, seg_data, textmode_instruction*, textmode_wr* signals and the signals at the *ascii** inputs of *output_logic* and include these screenshots in your lab protocol.

Furthermore, measure the time intervals requested (in the protocol template) with the help of markers. Take a screenshot showing the marker pair for one of the measurements and include it in your protocol.

In a second step use the testbenches provided for the *ps2_ascii* component to check it for bugs. If you find one, fix it and describe how it affects the design. The testbenches as well as a README file, explaining how to use them, are located in the *ps2_ascii* directory.

Post-layout simulation Use the netlist file (.vho) and the timing file (.sdo), which were generated during the previous task, for performing a post-layout simulation[1]. The testbench file used in the behavioral simulation can also be employed for post-layout simulation.

The timing file provides information on the real physical signal delays. Therefore, signals do not switch instantaneously after the clock edge, in constrast to a behavioral simulation. Every single bit of a signal vector switches individually depending on the propagation and routing delays of the corresponding circuitry. Run the simulation long enough in order to take a screenshot of the switching of *seg_data* to display the first character. Zoom into the waveform until you can see the diffenrent delays of the signals and use two markers to measure the duration between the first and the last bit toggling.

Task 3: Behavioral Modeling Your task is to implement the receiver state machine for a serial port. The generics of the serial port interface are described in Table 2 and the signals in Table 3.

Name	Functionality
CLK_FREQ	Actual clock frequency of the *clk* signal given in Hz.
$BAUD\_RATE$	The baud rate used for the serial port.
$SYNC\_STAGES$	Number of stages used in the input synchronizer.
TX_FIFO_DEPTH	Number of elements which can be stored within the transmitter FIFO.
RX_FIFO_DEPTH	Number of elements which can be stored within the receiver FIFO.

Table 2: Serial port generics description.

The interface protocol of the serial port is the same as for the FIFO buffers described in the IP core documentation. The transmitter port uses the write port of the FIFO, while the receiver port uses its read port.

[1]Depending on the settings, the Quartus timing analyzer might produce two sets of vho and sdo files: one with fast and one with slow timings. For this exercise use the conservative (slow) timing estimates.

Name	Direction	Signal width	Functionality
clk	in	1	Global clock signal.
res_n	in	1	Global reset signal (low active, not internally synchronized).
tx_data	in	8	Data which should be transmitted to the host.
tx_wr	in	1	If 1, a new data byte is present on *tx_data* and should be copied to the internal buffer.
tx_free	out	1	If 1, the internal buffer is not full and a new data byte may be written into the buffer.
rx_data	out	8	The byte which was last read from the receiver FIFO.
rx_rd	in	1	If 1, the next byte is read from the receiver FIFO if it is available. The byte is available at the *rx_data* port. If no byte is available, the value of the *rx_data* port is undefined.
rx_data_full	out	1	If 1, the receiver FIFO buffer is full and characters may have been lost because they could not be stored in the buffer.
rx_data_empty	out	1	If 1, the receiver FIFO is empty, otherwise received bytes are available in the buffer.
rx	in	1	The receive signal of the UART (Host → Device).
tx	out	1	The transmit signal of the UART (Device → Host).

Table 3: Serial port signal description.

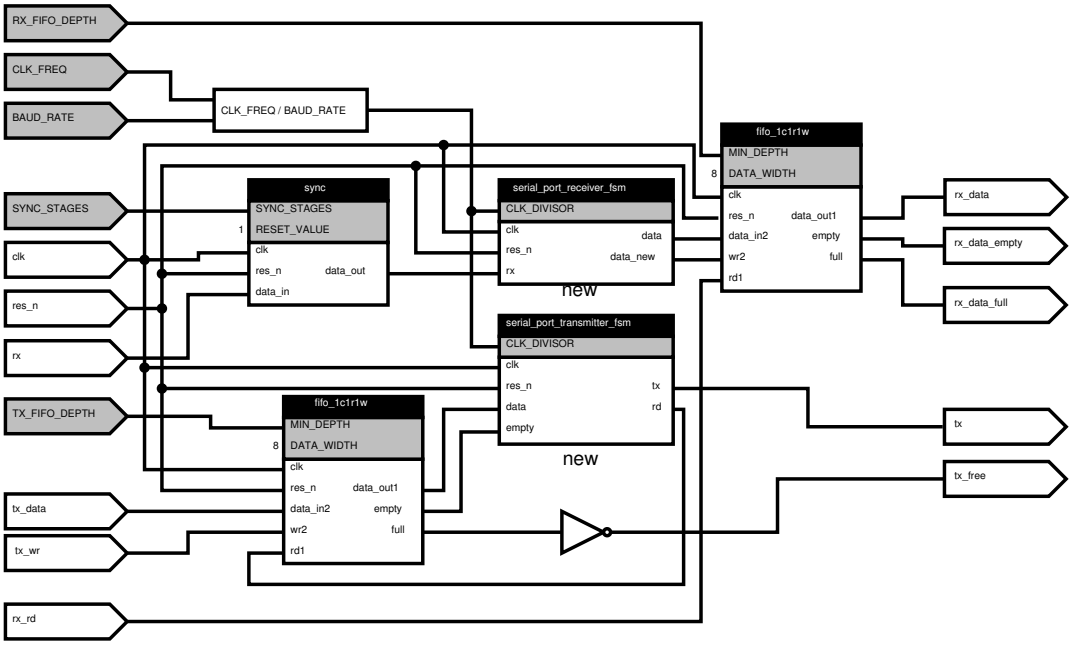

Figure 3: Serial port internal circuitry.

The *tx* and *rx* signals use the standard UART protocol with 8 data bits, 1 stop bits and no parity. The used BAUD rate is configured using the corresponding generics.

The serial port consists of five components, namely a synchronizer, two FIFO buffers and the two state machines which implement the receiver and the transmitter. Figure 3 shows how these components are connected together and how the input and output signals are assigned.

The operation of the receiver FSM is described by its state chart (see Figure 4), its state transition table (see Table 4) and its output behavior (see Table 5).

Start by implementing the receiver state machine. We recommend to follow the three process method for the implementation. Create behavioral simulations to validate the correctness of your implementation.

Afterwards create a structural VHDL design which connects all components (synchronizer, FIFO buffers and the state machines) corresponding to Figure 3. You should also use a behavioral simulation to validate its functionality. For the protocol, take a screenshot of a simulation showing the reception of a whole UART frame and include markers measuring the duration of the frame including the start-bit. Include the receiver and transmitter FSM's states and all signals connecting the *serial_port_receiver_fsm* and *serial_port_transmitter_fsm* as in Figure 3 in the in the screenshot.

State	Condition	Next state
$IDLE$	$rx = 1$	$WAIT\_START\_BIT$
$WAIT\_START\_BIT$	$rx = 0$	$GOTO\_MIDDLE\_OF\_START\_BIT$
$GOTO\_MIDDLE\_OF\_START\_BIT$	$clk\_cnt = \frac{CLK\_DIVISOR}{2} - 2$	$MIDDLE\_OF\_START\_BIT$
$MIDDLE\_OF\_START\_BIT$	always	$WAIT\_DATA\_BIT$
$WAIT\_DATA\_BIT$	$clk\_cnt = CLK\_DIVISOR - 2$	$MIDDLE\_OF\_DATA\_BIT$
$MIDDLE\_OF\_DATA\_BIT$	$bit\_cnt < 7$ $bit\_cnt = 7$	$WAIT\_DATA\_BIT$ $WAIT\_STOP\_BIT$
$WAIT\_STOP\_BIT$	$clk\_cnt = CLK\_DIVISOR - 2$	$MIDDLE\_OF\_STOP\_BIT$
$MIDDLE\_OF\_STOP\_BIT$	$rx = 0$ $rx = 1$	$IDLE$ $WAIT\_START\_BIT$

Table 4: Serial port receiver FSM state transition table.

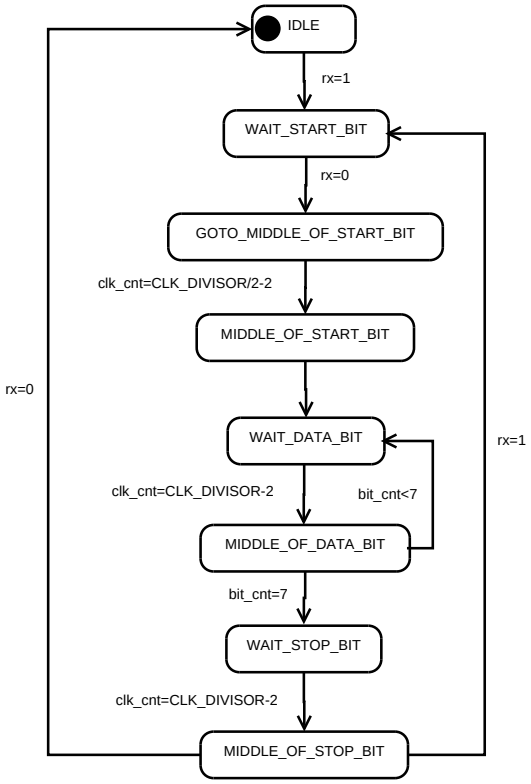

Figure 4: Serial port receiver state machine.

To demonstrate the correct functionality of your serial port design when really implemented in hardware, add the serial port component to your implementation of the first task. Therefore the receiver part of your serial port should be connected to the output logic, while the transmitter part should be used to transmit the ASCII codes entered using the keyboard to the PC (output of the PS/2 to ASCII converter). For simplicity, the tx_free port is left unconnected.

For the rx and tx signals you have to create new ports in your top level entity and map them to the corresponding FPGA pins.

After compilation, find the resource usage of the searial module including all submodules and add it to the protocol.

Task 4: Measurement In the VHDL system description, add copies of the *hsync_n, vsync_n, r(7), g(7), b(7), rx, tx* signals to the top level entity and map them to approriate pins on the FPGA board's extension connector. Connect the logic analyzer to the chosen pins and trace the signals. Measure the requested time intervals (specified in the protocol template):

State	Outputs
Default	clk_cnt keeps its data bit_cnt keeps its data $data\_int$ keeps its data $data\_new = 0$ $data\_out$ keeps its data
$IDLE$	
$WAIT\_START\_BIT$	$bit\_cnt = 0$ $clk\_cnt = 0$
$GOTO\_MIDDLE\_OF\_START\_BIT$	$clk\_cnt = clk\_cnt + 1$
$MIDDLE\_OF\_START\_BIT$	$clk\_cnt = 0$
$WAIT\_DATA\_BIT$	$clk\_cnt = clk\_cnt + 1$
$MIDDLE\_OF\_DATA\_BIT$	$clk\_cnt = 0$ $bit\_cnt = bit\_cnt + 1$ $data\_int =$ $rx\&data\_int(7\ downto\ 1)$
$WAIT\_STOP\_BIT$	$clk\_cnt = clk\_cnt + 1$
$MIDDLE\_OF\_STOP\_BIT$	$data\_new = 1$ $data\_out = data\_int$

Table 5: Serial port receiver FSM output table.

- The length of a whole UART frame including the start bit sent from the FPGA to the computer. Include markers for measuring the duration in the screenshot. Transmit an appropriate character such that you can perform this measurement.
- The hsync to hsync interval in the seventh visible pixel row (not character line). Include the *hsync*, *vsync*, *r(7)*, *g(7)*, *b(7)* signals and markers for measuring the time interval in the screenshot. Send characters to the display before making the measurement so that the line is not all black.

Include the trigger setting for the latter measurement in the protocol and do not forget to answer the questions therein.

4 Submission Specification

Your results are handed in via myTI. The deadline is **October 25*th* 2016, 23:59**. Upload a ZIP or TAR.GZ file containing the following information:

- Your lab protocol as PDF
- The source code of all IP cores
- The source code of the PLL
- The SDC file containing the clock definition
- The source code of your top-level module
- The source code of your state machines
- The testbench for your UART
- Your Quartus project (don't forget a cleanup!) or a TCL script creating the project (including the pin mappings!)

Make sure the submitted Quartus project is compilable. All submissions which can not be compiled will be graded with zero points!