Department of Computer Engieneering

Hardware/Software Codesign II
Summer Term 2015

Prof. Dr. Wolfram Hardt, Dipl.-Inf. Michael Nagler

Lab 2

Finite State Machine

1 Introduction

In the following three labs, we will work on a robot control for a robot that we have at this chair. Figure 1 illustrates the complete scenario of the project.

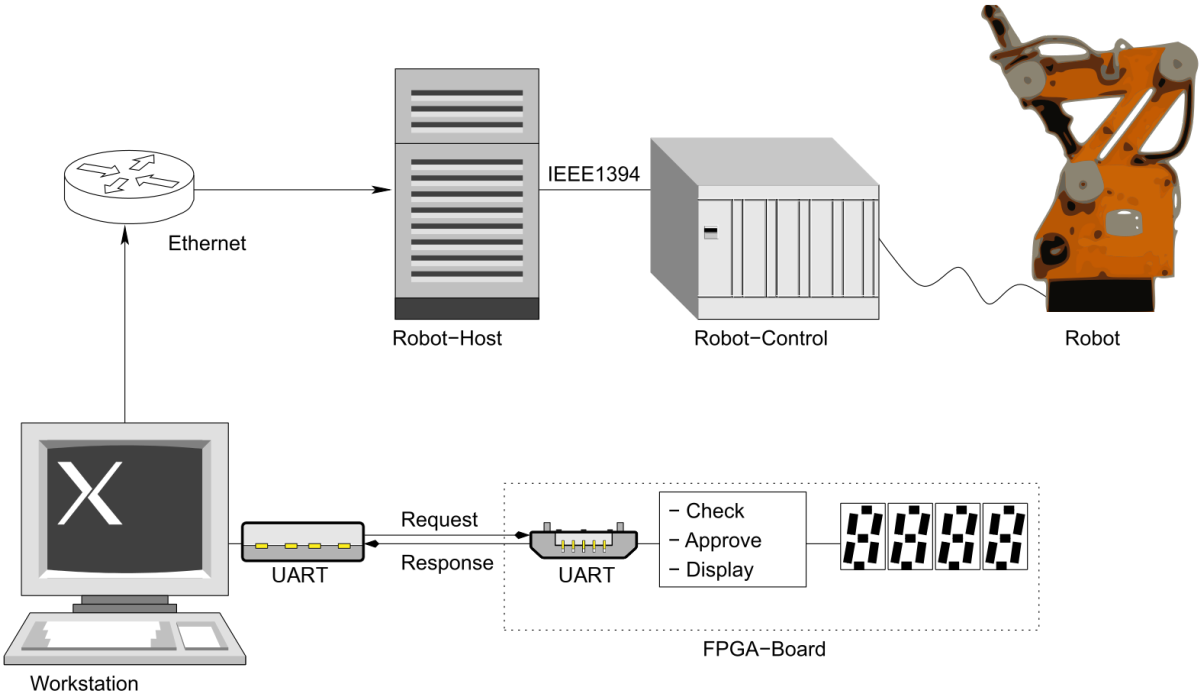

Figure 1: Control Set-Up

In this practical course, we will exclusivly deal with the implementation of the components on the FPGA-board. The software that runs on the PC is completely provided and is available as source code. It also contains a collision check, which is not provided in source code, because in Lab IV it is to be developed. The collision check has to prevent that the robot will make movements where it turns in a way that it collides with itself or the table on which it is positioned.

During this Lab, we first implement the finite state machine (FSM) that handles the protocol of the serial interface. This will be an integral part of the communication between the system on the FPGA and the workstation (see Figure 1).

2 Universal asynchronous receiver/transmitter (UART)

2.1 General Interface Characteristics

The UART interface is a serial and asynchronous interface, which is often used in conjunction with communication standards like RS-232. All pin-names, we use in this lab, are referred to from the view of the FPGA-board if not stated otherwise.

The transmission is sequential, i.e. the bits are transmitted one after the other over a single transmission line. The transmission begins with the least significant bit and ends with the most significant bit. The standard level on the line is logical '1'.

The transmission starts with one or two start bits, during which the line is LOW ('0'). After that five to eight data bits, one optional parity bit and one or two stop bits follow. During the stop bits the line is on HIGH-level ('1'). Figure 2 shows the signal interaction between the FPGA and the UART interface.

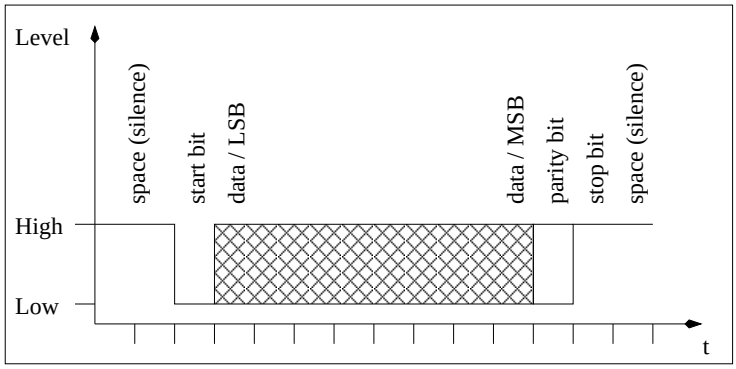

Figure 2: Signal Interaction between FPGA and Line Driver

To have a good sampling result, we sample the signal in the middle of the time interval, in which it has to be available. The clock of the communication can be configured, but in the Lab we will work with $9.6kHz$.

Please mind that both communication partners need to have the same parameters for the interface, otherwise they will sample wrong data at wrong moments.

When two communication partners are connected the receive and send lines are connected, i.e. $tx_{PC} \rightarrow rx_{FPGA}$ and $tx_{FPGA} \rightarrow rx_{PC}$.

2.2 Requirements from the Software Linkage

The next Lab deals with the software component of the robot control, which will communicate using the serial interface for the FPGA-board that we create in this Lab. In order to be compatible to the software part the hardware implementation of the serial interface should adhere to the structure depicted in Figure 3.

Send and receive follow this protocol:

Receive: When data is received, the hardware triggers an interrupt. The interrupt service routine can read the data from the *General Purpose Register* ***GPI1***. In order to acknowledge the received data and to set back the hardware component, the software writes a '1' to *General Purpose Register* ***GPO3*** and immediately writes a '0' again. As long as this process is not finish, the hardware is not supposed to load any further data into the receive buffer since the data might become corrupted (new data would be lost)!

Send: The software writes the data that is to be sent to *General Purpose Register* ***GPO2***. It also sets bit 8 to '1' (Bit 8: `data_ready` and Bit 7 downto 0: `data`). When this is done, the hardware

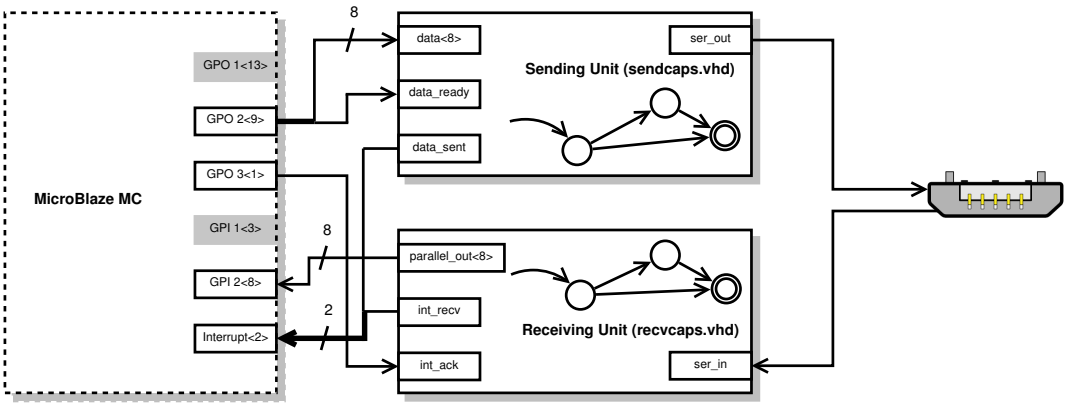

Figure 3: Structure of the serial components and their connection to the micro controller

component transmits the data. When all data is transmitted, the hardware component raises an interrupt (`data_sent`). In the interrupt service routine the software sets back bit 8 to '0', which acknowledges the transmission confirmation. The FSM for the send part goes back to its initial state and is ready to send further data.

In the end the signal interaction should look like the examples in Figure 4 and Figure 5.

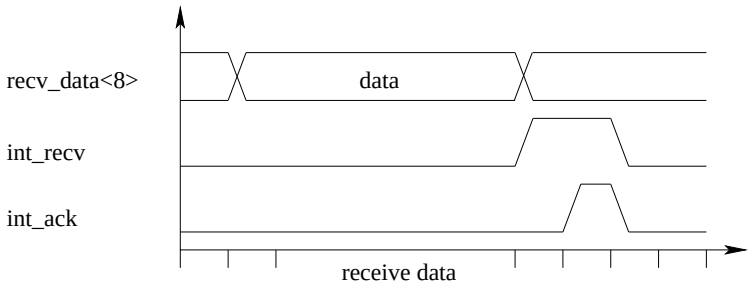

Figure 4: Signal Interaction of the Serial IP-Component when Receiving

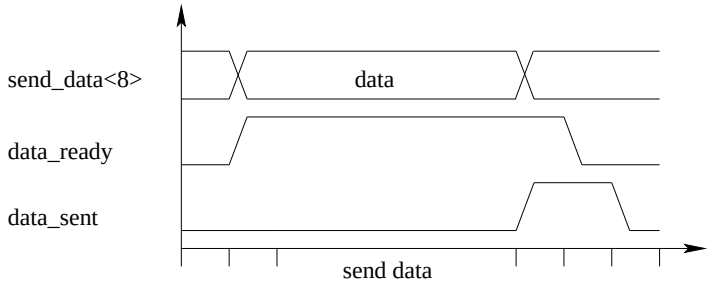

Figure 5: Signal Interaction of the Serial IP-Component when Sending

3 Tasks

3.1 Prepared Hardware

A complete project (`RoboControl/`), in which all instantiations of the hardware components are done, is prepared (Figure 6). It contains:

- a MicroBlaze micro controller with all needed general purpose registers and interrupts.
- a component to output the received data on the LEDs
- a component to output data on the seven segment display of the Nexys 3 board.
- a component that provides a prepared frame for the serial interface, which just needs to be filled in

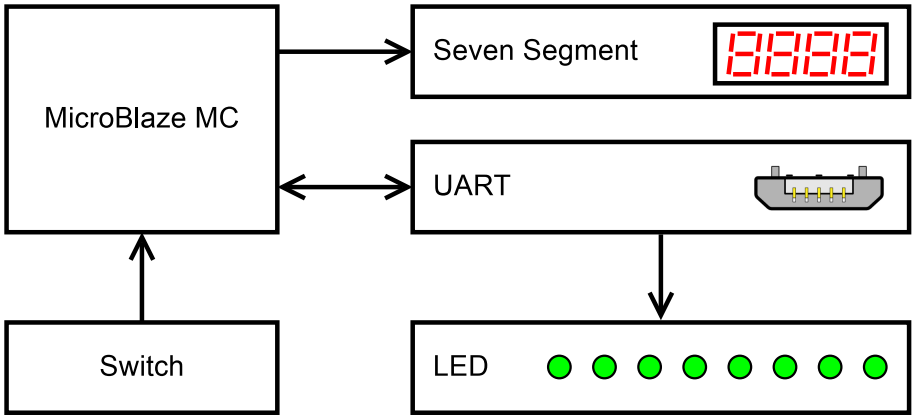

Figure 6: Block Diagram of the Prepared Hardware-System

Hint: *The Spartan6-FPGA provides a clock frequency of* $100MHz$*. This is much too high for the UART interface with* $9.6kBaud$*. Therefore it must be devided. However, it is recommended not to provide multiple clocks, since they are only a for clock wires. This is because the system clock has high demands concerning the signal skew. Even if a tool provides such possibility, this leads to many design errors. For this reason you should only use the divided clock signal as an enable signal.*

3.2 Prepared Software

For the software part there is a prepared workspace for the SDK with generated *Board Support Package* and *Application Project*. The given software provides the complete architecture of the final application. There are empty functions that have to be filled in during the next labs. It also initialises the interrupt system and executes the main loop of the application.

3.3 Send Operation

Implement the send operation, that sends data from the board to the work-station. This operation is generally the simpler one since the recovery of the transmission clock is done by the work-station hardware. This unit is to be developed according to the given information. It is also recommended to divide the implementation into data path and control unit. The transmission parameters for the serial interface are:

baud rate:	9600
data bits:	8
partiy:	no
stop bits:	1

The entities given in Listing 1 and 2 are already wired in the hardware project and only need to be filled with the VHDL-code for the serial interface core.

Listing 1: Entity for the Send Unit

```
entity SendCaps is
port
(
    clk          : in  std_logic;                     -- Clock
    res          : in  std_logic;                     -- Sync. Reset (high active!!)
    data         : in  std_logic_vector(7 downto 0); -- Data
    data_ready   : in  std_logic;                     -- Data Ready
    ser_out      : out std_logic;                     -- Serial Data (TX-line)
    data_sent    : out std_logic                      -- Data Sent
);
end entity SendCaps;
```

Listing 2: Entity for the Receive Unit

```
entity RecvCaps is
port
(
    clk             : in std_logic;                      -- Clock
    res             : in std_logic;                      -- Sync. Reset (high active!!)
    parallel_out    : out std_logic_vector(7 downto 0); -- Data
    serial_in       : in  std_logic;                     -- Serial Data (RX-line)
    int_ack         : in  std_logic;                     -- Interrupt Acknowledge
    int_recv        : out std_logic                      -- Interrupt Request
);
end entity RecvCaps;
```

3.4 Receive Operation

Now implement the receiving part of the serial interface. Basicly one could chose different transmission parameters. But we choose the same as for the send operation since on most work-station hardware it cannot be configured independently.

3.5 Simulation

Implement and simulate the design before coming to Lab II! The units `RecvCaps` and `SendCaps` can be tested with a prepared testbench. Therefore open the simulation view in the *Project Navigator* (click on `Simulation` on top of the *Hierarchy View* on the left side of the window). Then select `serial_testbench` in the *Hierarchy View* and run `ISim Simulator` → `Simulate Behavioural Model` in the window below.

After this the simulation is opened in the *ISim Simulator*. If you want to analyse more signals than that are originally shown, you can add them using drag'n'drop. The actual simulation is started with „`Simulation` → `Run All`“. You can observe the outputs of the testbench in the lower part of the window. If the testbench finds inconsistencies or deviations from the desired behaviour, it prints corresponding messages there.

The window for the signal traces shows the current simulation time. If the simulation takes longer then `40 ms`, the simulation did not come to a conclusive end.

3.6 Test

To be able to test the hardware part, the software part, that provides the send and receive functionalities, has to be implemented. Therefore the interrupts have to be handled and the send request has to be forwarded to the hardware component. The project already contains function prototypes which need to be filled in. The interrupt service routines are located in the file `robo_control.c` and the function, that writes the data, that has to be sent, is located in the file `serial_out.c`.

Implement these functions so that the received data is sent back to the workstation. This will test the receiving unit and the sending unit as well as the software part to drive them. This implemented functionality will be needed in Lab III.

As *General Purpose Register* ***GPO1*** is directly connected to the seven segment display, it can be used

to output a value to it, which might help when debugging the system.

3.7 Remarks

As in the previous lab, the time for the implementation during the lecture is limited. Therefore you should have a well prepared solution when you turn up in the lesson.

In order to create a VHDL-description that can be synthesized and that additionally works as anticipated, it is indispensible to use the 3-process or 2-process notation.

4 Questions

1. What are the functions of the start bits and stop bits?
2. Why is such a low clock frequency used on the serial line?
3. Why it is better to sample the bits in the middle of their time interval?
4. Above, big and little-endian was mentioned, what does it mean?
5. You implemented the send function and the receive function independently. Which advantages does it bring in comparison of an implementation with a single FSM?
6. What are other possibilities to acknowledge received data? Can similar techniques be used for the sending unit?

References

[1] *EIA-232*, August 2006. http://en.wikipedia.org/wiki/RS-232.

[2] Hardt, Wolfram: *Hardware/Software Codesign 1*. Vorlesung, 2005.

[3] Hardt, Wolfram: *Hardware/Software Codesign 2*. Vorlesung, 2006.

[4] Löb, Hans-Peter: *Integration eines prototypischen Realtime Media Access Controllers in eine PowerPC basierte Hardware-Umgebung*. Studienarbeit, April 2005.