Report of ESS LAB

Module Name (German):
Praktikum: Entwurf eingebetteter Systeme

by

Meister Rados

Tutor: M.Sc. Sami Salamin

December 2, 2019

1 Overview

In this lab, a FischerRobot is teached to follow a black line on a table, controlled by reconfigurable Hardware (FPGAs). The hardware configuration, as well as the software later run on a softcore CPU is to be written by the student. This report will point out what exactly was done. The written code is to be found in the appendix (of this email).
The lab is divided into 5 smaller, numerically ordered tasks. It starts off by creating I/O communication for a microcontroller. After that is done and well tested, the microcontroller is teached to run assembly code. A cross-compiler is used to generate assembly code from C code for the microcontroller, which will later contain the control and regulation loop to make the robot follow the black line.

2 Task 0

A useful tutorial sheet is handed out, acting as a small introduction to the software tools used in the lab. It also can be seen as a useful guide to return to, in case it is necessary to look up how a specific action is performed with the software tools.

2.1 Workflow

Everything worked fine and as expected!

2.2 Problems / Comments

- It seems like the programm used to communicate with the FPGA passes values in hexadecimal format. That would have been nice to know and led to minor confusion.
- The tutorial was quiet helpful!

3 Task I

Starting off with I/O, the first chip to be develepped is the Toshiba TC74HC595. A chip used to parallelly output a serially incoming data signal. 4 bits of the 8-bit parallel output are later used to control the databus for the 2 motors that the FischerRobot has mounted.

3.1 Workflow

A datasheet for the Toshiba TC74HC595 is handed out in order to get in touch with the chip to design. There is a schematic diagram of the inner circuit to be found in the datasheet. Rebuilding this circiut is easy since the used software (Xilinx ISE 10.1) provides tools to easily create circuits schematically. Rebuilding the given schema is everything to do. Testing the designed circuit finished this task.

3.2 Problems / Comments

- ISE often closed itself. It leaves a message in the terminal, saying ”SEGFAULT”. This is quiet frustrating.

4 Task II

To fulfil the secound task, it was necessary to rebuild another chip. This time, it’s the Texas Instruments CD74HC597. This chip acts as a serializer. It receives an 8-bit parallel input and can shift these 8 bits serially to its output. This chip is used to shift the parallel output of the two photo-diode-sensors serially to the softcore.

4.1 Workflow

This time, the given datasheet doesn’t provide a schematic diagram of the inner circuit. So it wasn’t easily possible to rebuild it. Looking at the time diagram in the datasheet, the behavior of the chip becomes clear. Using VHDLs process description (which is also well descirbed in the tutorial, task 0) made it possible to implement the behavior of the chip after a few tries.

4.2 Problems / Comments

- Having a close look at the given time diagram, one can see that an impossible behavior is described. The left section of the chip can’t know at which time its values have been read. According to the time diagram, in exact that moment, its data seems to be set to zeroes. Trying to achieve this particular behavior took a couple of weeks, until it was declared irrelevant for this lab. Handing out error-free datasheets avoids such circumstances.

- ISE often closed itself. It leaves a message in the terminal, saying ”SEGFAULT”. This is frustrating.

5 Task III

In the third task, the micorcontroller is placed in the design. Together with that, the whole design, called ”complete” is to be created. The circuits created in the first two tasks are also used in this task.

5.1 Workflow

Creating the schematic design ”complete” is mostly done by creating schematic symbols with the given software tools. Their behavior can mostly be copied from the task sheet. Placing and wiring them finishes the first half of the task. A more difficult thing is to find out how the serializer and the parallelizer are to be placed in order to achieve useful

behavior. The task just says ”think of how to use them” and leaves the student quiet a bit lost unless he/she fully understands the whole ”complete” circuit. A useful wiring can be found by matching the bus sizes from the open pins in the ”complete” circuit with the ones that the serializer and the parallelizer offer. In fact, that way brings up a working design.
The second half consists of creating a simulation environment for the ”complete” circuit. Again, most files are given in the task sheet and can easily be placed and wired. An external program called ”mkrom” generates a hardware description file from assembly code. Another program, called ”as31”, checks if the assembly code has any errors and generates a .ihx file that can be read by the microcontroller.
This assembly code needs to be written by the student. Once it is possible to write values to the databus, depending on what is read be the optical sensors, this task is done.

5.2 Problems / Comments

- The wiring of the circuit ”complete” is really complex and deals with a lot of chips never seen before. It is really hard to understand what they are used for, despite to the good documentation saying what they do.
- The ports of the chips to wire have confusing names. Some are called *_ADDR_* and some *_ADR_*. Name mismatches are hard to be found and lead to errors in later states, such as the simulation.
- These errors aren’t exactly pointed out by the used software. A long desperate search in a big design begins.
- It is necessary to run the simulation for about 25 ms, until the effect of the assembly code is visible. Running the simulation for a much shorter time period (50 ns, which was enough for all tests before) won’t show any results, leading to confusion and frustration.
- The given schematic for the ”complete” circuit has a 16-bit ground bus as an input for the softcore. This can only be recognized by the slighlty thicker wire. This is barely visible, since the resoultion of the image showing the circuit is quiet too low for these details.
- How to generate a 16-bit ground bus isn’t described in the task sheets.
- The given schematic for the ”complete_top” circuit has an address bus with 17 bit width, connected to the address bus coming from the inner circuit with just 16 bit width. The image presented in the task sheet has a lonely GND symbol near to the wider bus, but is not connected to it. Connecting this GND symbol to the wider bus seemed useful and worked. That would be good to know since this fairly undescribed image of an incomplete schematic is confusing.
- The order of the motorbus bits is ordered reversed in the task sheet.
- The licence for the simulation tool expired by 2017-11-30.
- ISE often closed itself. It leaves a message in the terminal, saying ”SEGFAULT”. This is really frustrating.

6 Task IV

This task introduces Esterel. Esterel is a programming language used for modelling behavior of outputs based on given inputs. To complete this task, writing a control and regulation loop for the FischerRobot is required. It should read the values from the optical sensord and, based on them, control the motors in order to follow the black line.

6.1 Workflow

Deciding not to use Esterel and doing the whole task 4 in C Code allowes to skip this task.

6.2 Problems / Comments

- I've never heard of that language. And since it isn't actually needed for this lab, why is it in here?

7 Task V

In the last task, a C program needs to be written, which translates the Esterel functions to C code from which assembly code can be generated.

7.1 Workflow

Since task IV was left out, the procedures for controlling the motors and reading the sensors needed to be done in this taks.
Copying the required C commands for the cross-compiler was done quickly. Writing the procedures for controlling the motors and reading the sensors was easy, too. Building the control and regulation loop was fun, since everything worked as expected at the first try. Building trivial search strategies wasn't that easy, because the microcontroller seems not to be capable of waiting for a specific amount of time. Including time.h wasn't successfull. Counting up to a large number worked as a successfull way to skirt around this issue. The robot will now make a $360°$ turn in that direction it last saw the line. IF it doesn't find it again, it will move a bit forward and repeat looking for the line around it. Repeatedly not finding the line will it cause to move in larger steps forward, making it possible to increase searching room.

7.2 Problems / Comments

- That was the most fun part. Liked it really much!

8 Sources

The sources in the appendix of this email are:

- complete.sch
  The schematic file of the whole circuit
- complete_top.rbt
  The programming file for the FPGA
- DFFR.vhd
  A D-Flip-Flop with asynchronous reset function
- IC74595.sch
  A schematic file of the parallelizer
- IC74597.vhd
  VHDL description of the used serializer
- robot.c
  The C file containing sensor/actor control and regulation loop
- robot.ihx
  The generated assembly file for the softcore
- Tri_State_Logic.vhd
  VHDL description of the tri-state logic used for the parallelizer.