Department of Computer Engieneering

Hardware/Software Codesign II
Summer Term 2015

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

Lab 3

Robot Control

1 Introduction

After a serial interface for the Digilent Nexys 3 board was implemented in the previous lab, this interface is now used to communicate with the workstation. The goal is to implement a control software for the robot on the board.

2 General

As there is only a limited number of suitable I/O-ports on the Nexys 3 board, the serial interface from the previous lab is used to communicate with the workstation. The following steps have to be realised:

1. control signals are transmitted from the workstation to the FPGA-board
2. incoming control signals must be checked for validity
3. valid control signals have to be processed, invalid ones need to be discarded
4. the resulting robot position information is sent to the workstation
5. the current state of the robot is to be outputted on the seven segment display

The robot has two elements on which a grabber is attached. Three motors are used to position the grabber, one to rotate the whole robot on its socket, one to rotate the grabber, and finally one to open and close the grabber.

It should be noted that the robot does not know which actual position it had when it was switched on but it just assumes that it was fully erected. The six angle values are 0 in that initial (i.e. switching-on) position. In this lab one should take care that when the robot is activated, it is in a valid initial position.

3 Protocols

This section introduces the protocols that are used in this project. It is vital that these protocols are satisfied. Figure 1 shows the dataflow over the complete project. Points 1 to 3 are relevant to this lab.

3.1 Workstation → Board

The board shall check the inputs made by the user. Therefore the command that was inputted at the workstation is sent to the FPGA-board (Figure 1, point 1). The inputs are made as differences, like “one further” or “one back”. Figure 2 shows the exact data format.

The motor number, which is encoded as binary number, determins the motor that is to be moved. If the F-Bit is set, the angle of the motor is to be incremented by 1°. If the B-Bit is set, it is to be decremented by 1°.

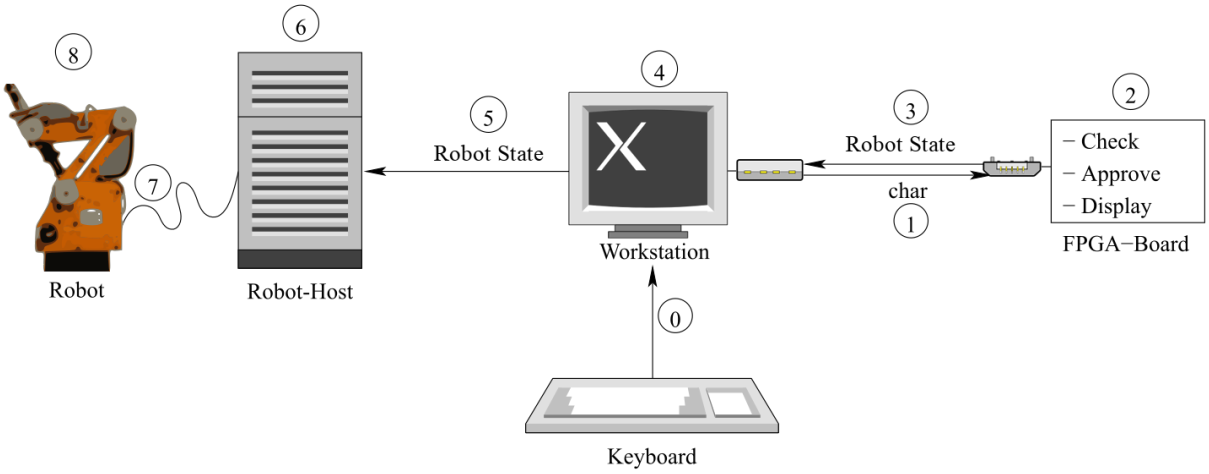

Figure 1: Data Flow within the Project

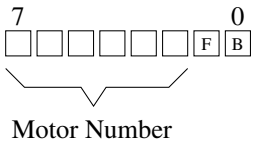

Figure 2: Data Frame for Workstation → FPGA-Board transmission

If both bits (F-bit and B-bit) are set, the board should reset the assumed position to the initial position. The motor number has no meaning then.

Such reset is reasonable since physical access to control components cannot always be ensured which means that sometimes the reset button cannoct be pressed. **Attention: In the workstation application (see Section 5), the i-key must never be pressed when the robot is not in the legal initial position!** The parts of the robot could collide with each other and damage the robot.

Hint: *For the motor number, the values "`000001`" to "`000110`" will correspond to motor number 1 to 6. All other values are invalid.*

3.2 Board → Workstation

The board checks (Figure 1, point 2) whether the incoming commands are valid, i.e. the robot does not collide with itself, its socket or the underground. Then it sends back (Figure 1, point 3) the robot state (angles of all motors) as given in Listing 1. The state contains the current (target) position of the robot. This data structure is always returned. It contains, in the case of a valid command, the updated position or, in the case of an invalid command, the unchanged postion.

Listing 1: Robot State

```
// Number of Engines
#define ENGINE_COUNT    6

// Angles of Engines
float engines[ENGINE_COUNT];
```

Hint: *The value of `engines[0]` corresponds to motor number 1, `engines[1]` corresponds to motor number 2 and so on.*

This data is also used in all following processing steps on the workstation (Figure 1, point 4) and finally also for the communication with the robot server (Figure 1, points 5 and 6) and the robot (Figure 1, points 7 and 8).

The serial interface transmits the data byte for byte. The robot state contains floating point values consisting of more than one byte. The 32-bit floating point formats that are used on the MicroBlaze on the FPGA board and on the workstation are the same one (IEEE 754 [2]).

Since the MicroBlaze stores data in the big-endian-format but the workstation (x86) uses the little endian-format, a conversion is needed. The application for the workstation performs the conversion between these two formats. The angle values of the motors are expected, beginning with motor 1 in ascending order, each starting with the most significant byte first.

Hint: *The C-standard does not define the large data types meaning that these data types are compiler and architecture dependent. Is an application to be executed on several architectures or compilable by different compilers, it is important not to depent on the actual sizes of these data types. If in doubt, the size of a data type can be retrieved using the* `sizeof()`*-function. The return value is the size of the given data type or variable in byte. E.g. on an x86-processor* `sizeof(char)` *results in* 1.

3.3 Workstation → Robot

The workstation takes the command that the board deems to be valid and forwards it over another server to the control device of the robot. For this it uses the same structure (Listing 1) that is used for the transmission from the board to the workstation.

The control device for the robot uses a micro-controller to convert the incoming command to control signals for the motors that in turn move the robot into the requested position. This control also is responsible to keep the robot in that position. It expects the angle values of the motors beginning with motor 1 in accending order, each starting with the most significant byte first.

4 Seven Segment Display

A seven segment display, which will be used to output information to the user, is provided on the board. A hardware description, which drives the seven segment display is already given and has not to be implemented during the lab. The display can be accessed by the software using the *General Purpose Registers* (actually via ***GPO1***). Its width is 13 bit. The bits `11 downto 0` are displayed as a hexadecimal value on digit 1 to 3 and bit `12` is used to enable the minus sign on digit 4. An example is depicted in Figure 3.

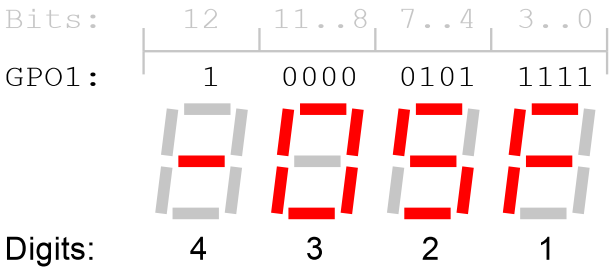

Figure 3: Example data displayed on the seven segment display

5 Software - Workstation

The application that is executed on the workstation is also provided. It allows to manipulate the robot state (defined by 6 engines) by keyboard inputs and sends the new values to the board. Once it received proofed values from the board, the application is able to send the position to the server (connected to the robot). This application contains a further collision control to make sure that errors in the implementation (on FPGA board) will not destroy the robot. The application can be executed by starting the batch file "`robo.bat`".

Table 1 lists the keys with which the application can be controlled. The control application can be configured by command line parameters that are illustrated by Table 2. Normally it is not necessary to change the standard settings that are given in the batch file.

`1-6`	Select the controlled motor
`+`	Increment angle of selected motor by 1°
`-`	Decrement angle of selected motor by 1°
`i`	Reset the position of the robot.
`q`	Quit

Table 1: Handling of the Robot Control Application

`-d | --debug`	deactivate the GUI, network and control component of the software and show the data received from the FPGA-board as text.
`-h | --help`	output a short help
`-n | --no-network`	deactivate the network component, so that no server must be running, with which the application would try to connect otherwise.
`-o <Serial Port>`	open serial port for communication with the FPGA-board. The ports are called `/dev/ttyS<Num>`. The default value is `/dev/ttyS0`.
`-p <Server-Port>`	server port where the robot server is listening, default is 8080.
`-rs | --robot-solid`	start program for permanently installed robot
`-rp | --robot-portable`	start program for robot on the movable cabinet (default)
`-s <Server-IP>`	server ip where the robot server is listening, default is 127.0.0.1
`-t | --test`	connect to the locally executed emulation of the software environment found on the development board (the emulation has to be started before)

Table 2: Command Line Parameters of the Robot Control Application

6 Tasks

6.1 Implementation of the Protocol

Implement a software that handles the protocol between workstation and FPGA board. This software shall interpret the control signals received from the workstation and return the robot state (see Listing 1) with new values for the motors to the workstation. In the prepared project from Lab II, in file `state.c`, the function `updateState()` has to be extended accordingly. The parameters of this function are the engine number (`engineNr`) and the command (`cmd`; see F-bit and B-bit in Figure 2), that has to be performed.

After the robot state is updated, the new values have to be sent back to the workstation. In file `serial_out.c`, the function `sendByte()` should already be finished from Lab II. Now the function `sendData()` can be implemented to send back the data structure from Listing 1. As described in Section 3.2, all engine values (32 bit) have to be sent in ascending order, each starting with the most significant byte first (see Figure 4).

Hint: *The MircoBlaze Micro Controller is able to handle only one interrupt at the same time. This means an interrupt service routine has to be processed completely before the next interrupt can be handled. As one interrupt is needed for receiving data and another for sending data, the send back operation can not be executed during the receiving interrupt service routine (`isrRxReq()`). Therefore a global variable `sendBack` (see `robo_control.h`) exists. This variable is evaluated in the main loop to call the function `sendData()` when data has to be sent.*

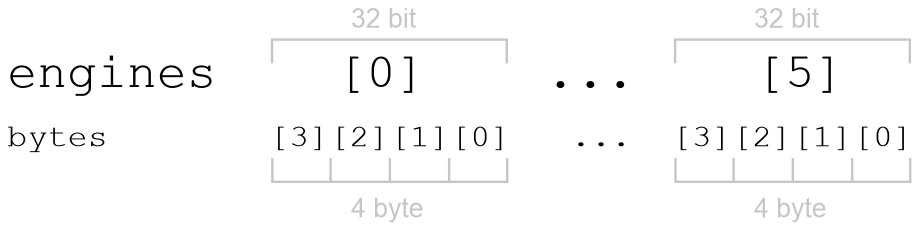

Figure 4: Engine and byte order when sending the robot state

6.2 Display

The seven segment display on the board shall show the active state (all motor angles) of the robot. As the display has only 4 digits, it can show only one angle at the same time. To change the shown motor, the switches (`SW2, SW1, SW0`) shall be used. The value "`000`" on the switch corresponds to `engines[0]` (→ motor number 1; see Section 3.2), the value "`001`" to `engines[1]` and so on.

Hint: *Holding down button* `BTNU (A8)` *will show the motor number on the display instead of its value. A motor number higher than 6 will result in outputting "`---`" as value.*

The function `updateSevenSegment()` is called in every iteration of the main loop and shall be used to update the seven segment display. It can be found in the file `seven_segment.c`. The value of the switches can be read from a *General Purpose Register* (actually ***GPI1***). The width of the register is 3. The corresponding angle shall be written to ***GPO1*** (see Section 4).

Hint: *The seven segment display will display data on* ***GPO1*** *as a hexadecimal value. To make it human readable, the value shall be transformed into a decimal value. Therefore the data for each digit of the seven segment display has to be generated (see Section 4, Figure 3).*

6.3 Remarks

C99/C++: Stick to the language part of standard C, but do not use libraries from the C standard. Limit your dependencies to the function interfaces that are given in the lab tutorials.

During this lab, there is not enough time to solve the tasks, you should therefore prepare the tasks at home and discuss only minor problems with the tutor.

7 Questions

1. The interrupt service routine needs time to process incoming interrupt requests. How can this time be minimised? Which adjustments would be necessary to your source code, if you not already did it?
2. Try to estimate the time that passes between inputting a command and the point when the reply to it is completely received at the workstation.
3. What could be changed to reduce that time without changing the hardware?

References

[1] *Linux Programmer's Manual*, 1989-2006. Sections 1 und 2.

[2] *IEEE 754*, November 2006. http://en.wikipedia.org/wiki/IEEE_floating_point.