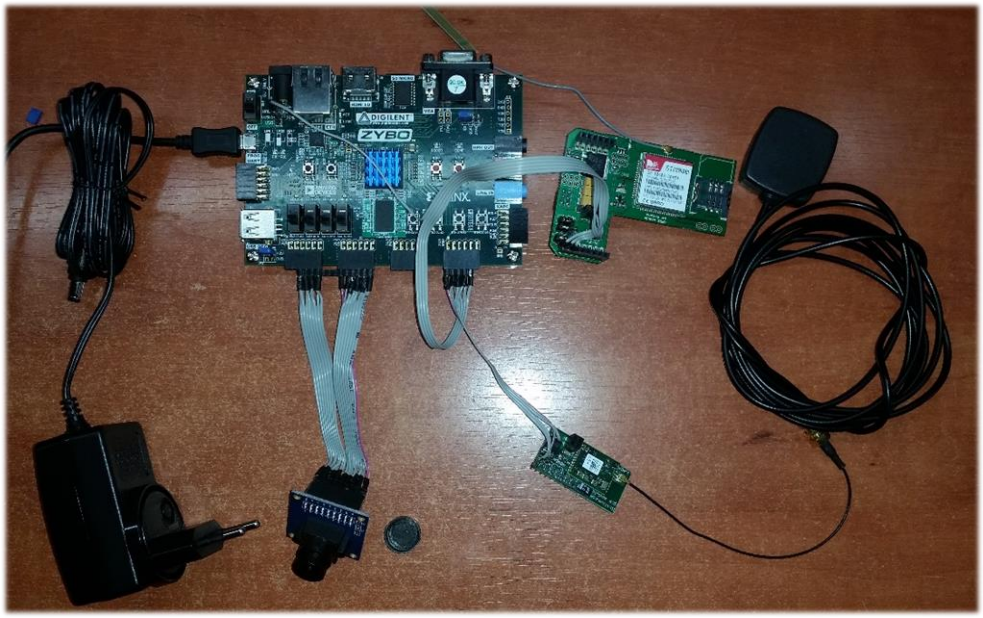
Real-time Traffic Analyzer

University of Debrecen

Team number: XIL-54179



Supervisor

dr. István Oniga PhD
oniga.istvan@inf.unideb.hu

Participants

Renátó Besenczi MSc
netid21@mailbox.unideb.hu
Mihály Szilágyi BSc
szimih@mailbox.unideb.hu

YouTube video: https://youtu.be/ECosPSNb8_Q

Project homepage: https://github.com/rbesenczi/real-time-traffic-analyzer

Board used: Digilent ZYBO
Vivado version: Vivado 2014.4
Date: 06-30-2015

Real-Time Traffic Analyzer

Robocar World Championship - Robocar Cloud

Ed. Real-time Traffic Analyzer, v. 1.0

COLLABORATORS

	TITLE : Real-Time Traffic Analyzer	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Besenczi, Renátó, Szilágyi, Mihály, Ã©s Oniga, István	June 30, 2015	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	06-06-2015	Initial document and system plan	
1.0	06-30-2015	Content extended, XUP compatibility	

Contents

List of Figures

Chapter 1

Introduction

Robocar World Championship is intended to offer a common research platform for developing urban traffic control algorithms and for investigating the relationship between smart cities and robot cars with particular attention to spread of robot cars of the near future. At the heart of this initiative is the Robocar City Emulator. It will enable researchers to test and validate their theories and models.

In the near future, driverless cars are expected to be ubiquitous. However, this fact can be arguable, because the future is uncertain. We should note, for us this system is an initial abstraction with which we can start to build our research and development in the relationship between smart cities and self driving cars.

Basically, the primary aim is to support university level education and research: we intended to establish a platform which can be a de facto standard in the research of the connection between smart cities and self driving cars. (Somewhat similar with the RCSS initiative in the Artifical Intelligence research domain.) The Robocar World Championship (or OOCWC for short) is a competition for routing algorithms which can be interpreted on an exact city (for example New York or London).

Moreover, the most important aim is to support programming classes in education.

Our further aim is to establish an open platform for industial applications. The Robocar City Emulator is an emulator in the aspect of routing algorithms. The emulator uses measured data as input and a route between two points can be considered as output.

Further information about the system can be obtained here

The input of the Robocar City Emulator consists of the measured and estimated data of road loads. We can measure this data in different ways. Besides the crowd sourcing methods (e.g. with a mobile phone application) an automatic way (without the need of human interaction) is plausible. The system discussed below is considered as an automatic measuring system for the input of the Robocar City Emulator.

Competition plays an essential role in the Robocar World Championship initiative. This competition does not only manifest in the level of algorithms, but in the implementation level (e.g. prototypes) of the system. Many versions have already been made, the main branch can be found here. The system discussed below is considered as a branch fork of the main implementation and is not a part of the main, original implementation of the Robocar World Championship.

Chapter 2

Design

2.1 Requirements

System requirements:

- Real-time image processing.
- Real-time data connectivity with a cloud based database solution.
- Accurate positioning.
- Easy installation into cars.

The first three requirements are satisfied. The system can measure traffic density on roads, obtain an accurate position with the GPS, and send aggregated data structure to our Java based TCP server. However, we have not yet designed a console (or box) for the system, so we can't install it into cars. After the testing process finishes, we intented to investigate the possibilities how can such a console be installed into cars.

2.2 Hardware

Because this project is mainly a research project, different kind of implementations are planned to realized. The first type of our hardware design is an ARM based solution. In this type, we use the advantages of the FPGA, mainly for I/O and memory management. The ARM gives us a standard option to perform calculations and image processing. This kind of hardware development can take place on development boards. The main advantage of this type is that we can use an Embedded Linux System, so the high level processing tasks can be developed in a standard Linux/UNIX environment. Another solution is based on a soft-processor. In this type we do not have a physical processor, only an FPGA, so we should add a pre-defined one to our hardware design (e.g. a Xilinx MicroBlaze). In this case we must specify the operation of the processor, however its instruction set is poorly defined. The third type of solution has no processor at all, only a pure FPGA design. The whole process, including the I/O handling, image processing and some basic analytics is performed by an FPGA. Certainly, this solution gives us the fastest processing speed, but its development is more complicated. In this project, we have implemented the ARM based, Embedded System version.

2.2.1 Hardware design

In our project, we needed a device which is much more flexible than a single board PC (e.g. Raspberry Pi). This flexibility gives us the ability to customize every periphery (camera, GPS, etc.) for our requirements. Besides, we are intended to use a Linux based system for running our application. The Zynq SoC have the above mentioned abilities. For the development we used the Xilinx Vivado™ Design Suite WebPACK version 2014.4.

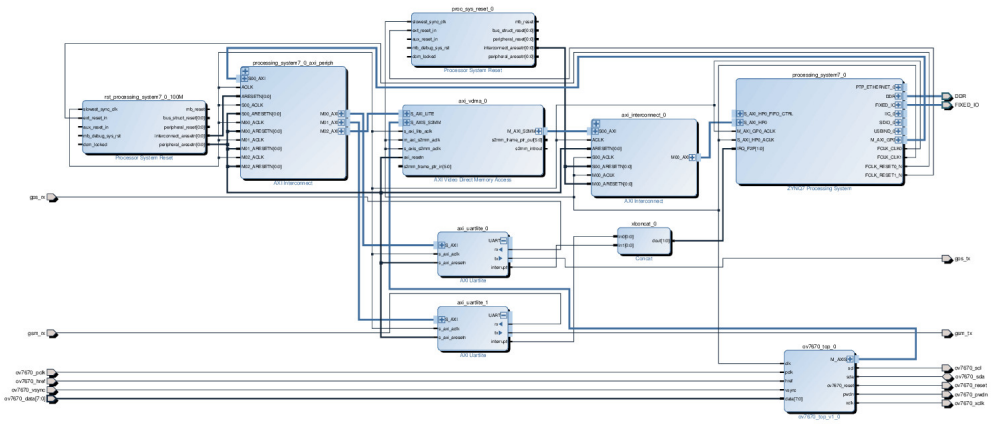


Figure 2.1: Hardware design

The base design is quite simple, but contains our own components. The main component is a ZYNQ Processing System and a Processor System Reset. In the project we connected the devices through the AXI bus. Two main components are the two UartLite serial port. The GPS device is connected to the axi_uartlite_0 serial port which delivers the NMEA data with 4800 baud rate (1 PPS data). We can communicate with the GSM device through the axi_uartlite_1 serial port with 115200 baud rate. The ov7670_top_0 component reads the data from the camera module and transforms it into AXI Stream. The axi_wdma_0 loads the camera data into the memory through the axi_interconnect_0 module.

The communication between the peripherials are handled by the AXI bus. This bus has a 100MHz default clock rate. To deliver the camera data into memory we set a 150MHz secondary clock rate.

We use Video Direct Memory Access to deliver video stream into the memory. The video stream has resolution of 640x480. The VDMA can be utilized easily and thanks for the Linux driver we could easily integrated it into our development. In the beginning of the development process we found very hard to apply VDMA. In order to overcome these difficulties we applied a simple model in which we used a module which creates a TestPattern constant image. More info can be found here.

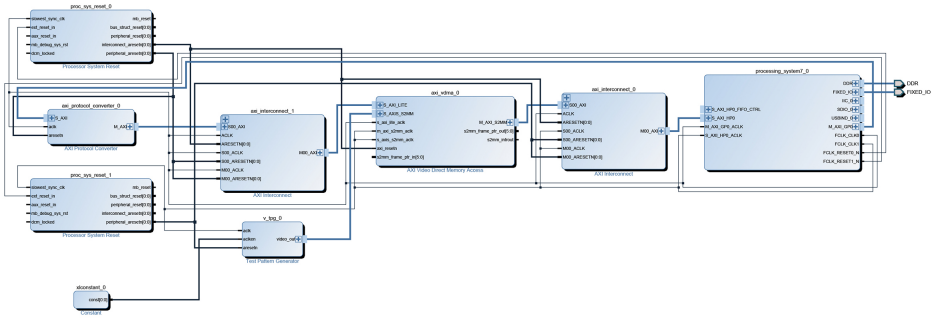


Figure 2.2: VDMA test design

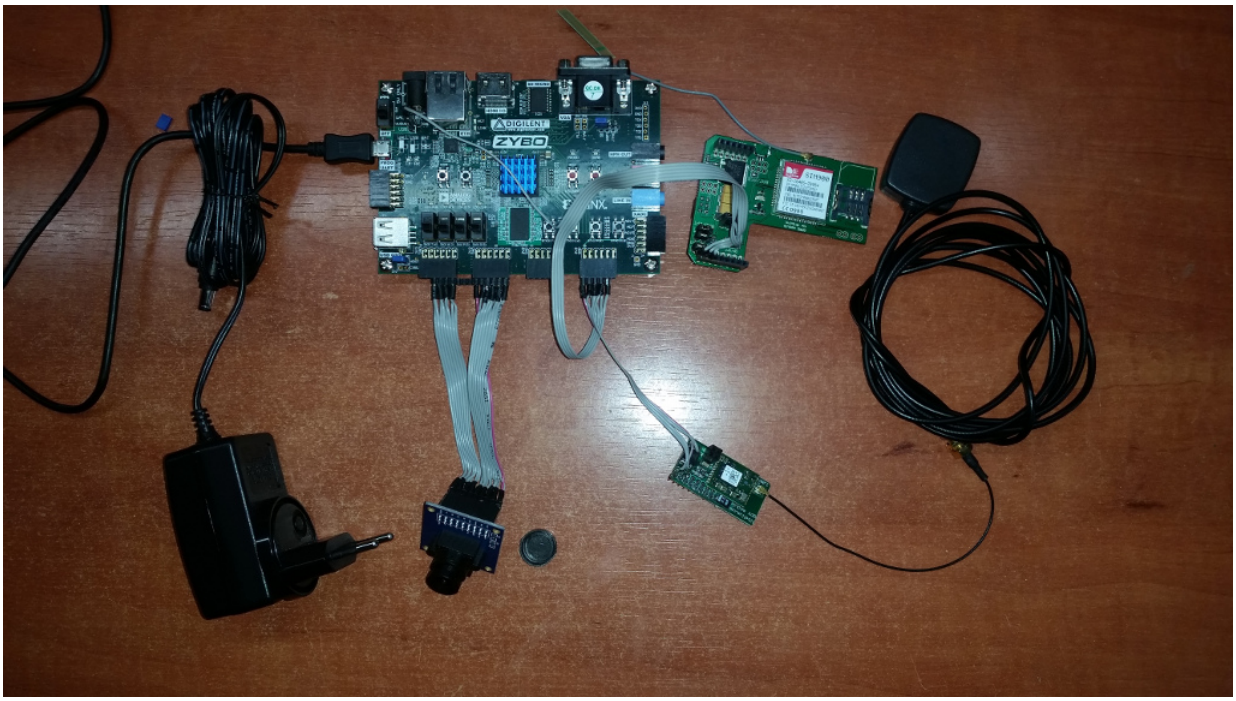


Figure 2.3: The final hardware design

2.2.2 Digilent Zybo

For development we used a Digilent Zybo board. More information can be found here.

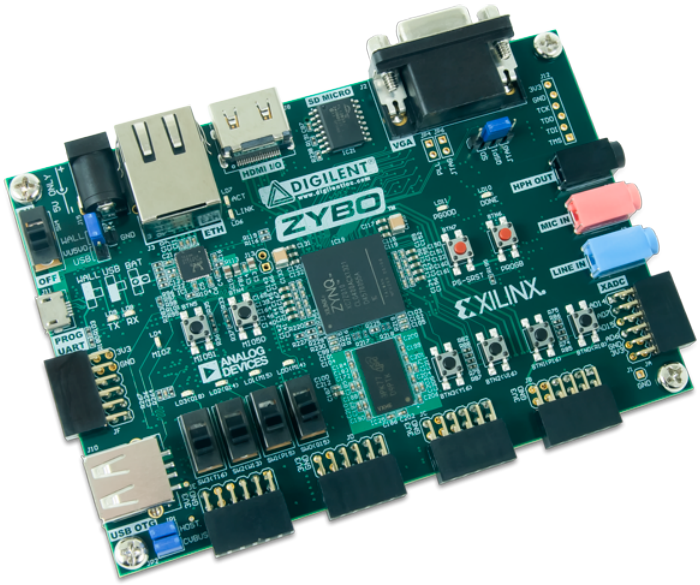


Figure 2.4: Digilent Zybo

2.2.3 Devices

For our purposes the Vincotech's A1080-A GPS reciever seems suitable. It needs minimal external components and communicates on a simple serial port. The module sends NMEA data in 1 PPS time period. This NMEA data is processed by our application running on the Embedded Linux System. Speed, time and position is important for us. The power delivered by the PMOD interface was sufficient for this module.

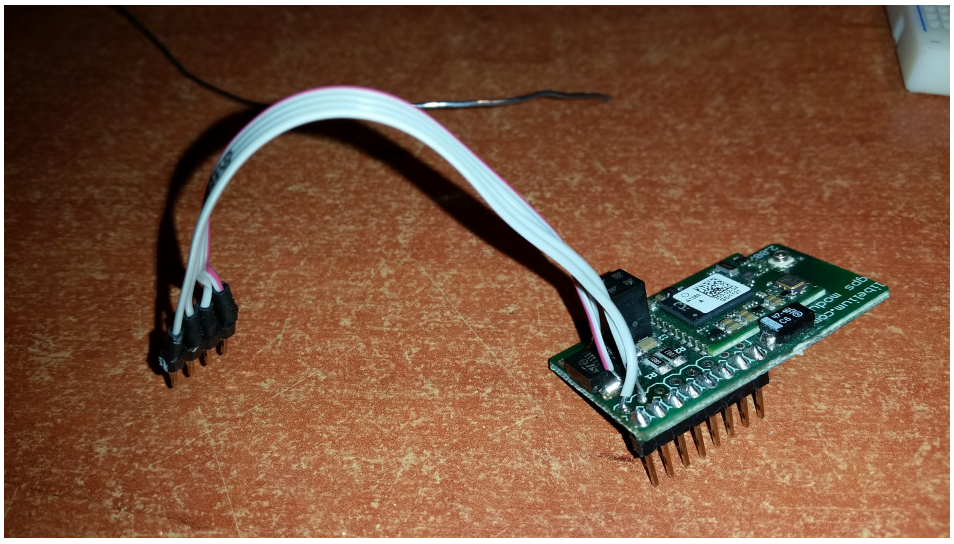


Figure 2.5: GPS

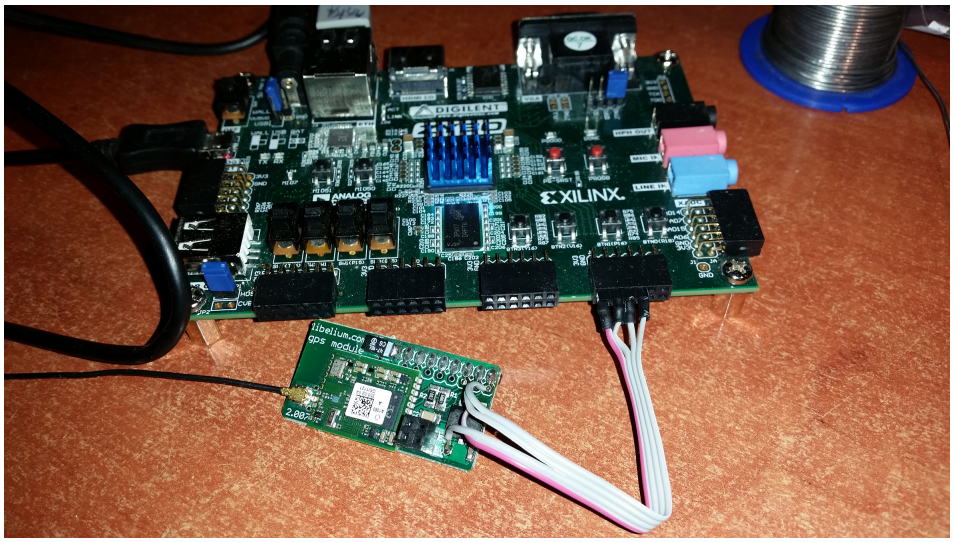


Figure 2.6: GPS

We applied a SIM900 GSM module for GPRS communication. Since this module needs 5V input we could easily connect it to the inner power switch of the ZYBO board. Fortunately, the power plug could handle the module, the initial peak in this kind of setting is usually a critical point. In the future, we must pay attention when applying in cars.

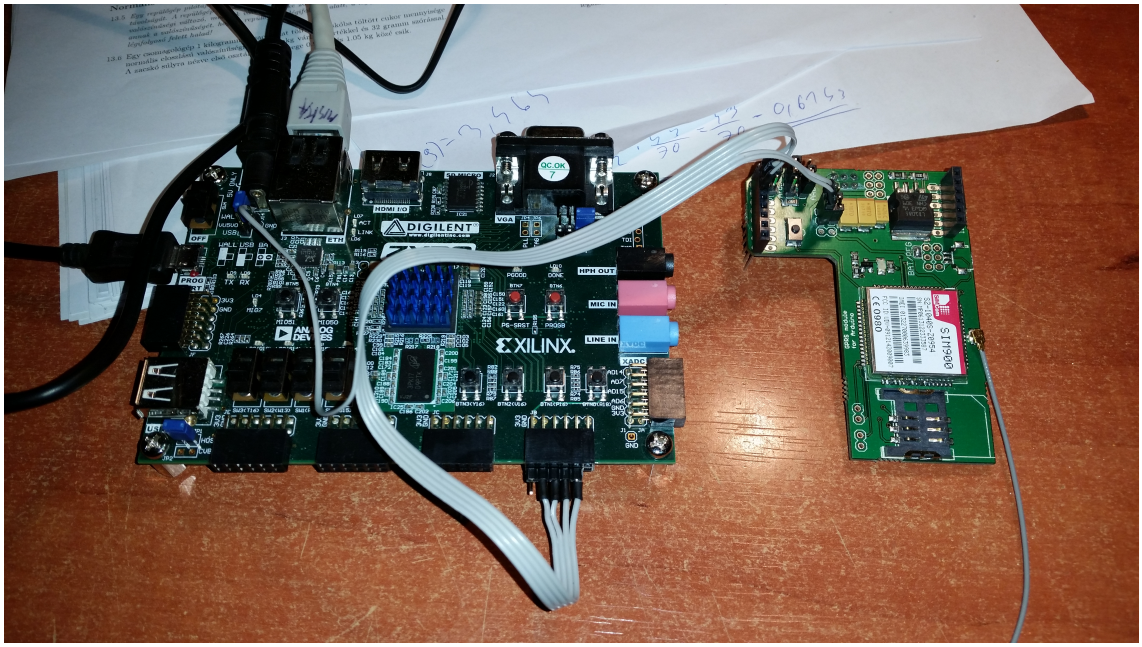


Figure 2.7: GSM

For image acquisition we wanted to use a customizable device rather than a USB webcamera. We have chosen the OV 7670 module because it is easily accessible and its maximum resolution (640x480) is sufficient for our purposes. The main advantage of this module is that we can set its properties, so the video stream can meet our requirements. Thus, the input data can manipulated easier and the processor has a lower load. Currently, we do not perform pre-processing, but we are planning to integrate a HLS-based OpenCV solution.

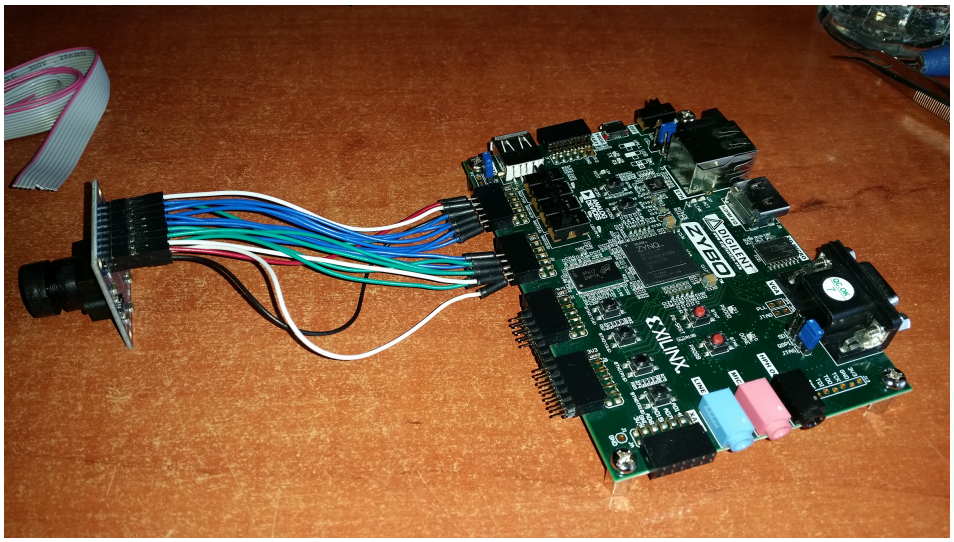


Figure 2.8: Camera module

Our further plan is to use the CAN bus integrated in the ZYNQ SoC. With this module we can obtain further information about the vehicles, for example speed, fuel consumption, etc.

Figure 2.9: CAN bus

2.3 Software

Our software consists of three different modules. The GPS module parses the NMEA data and stores it in a structure. The openCV module performs image processing, which will described later in this chapter. The GSM module initializes the GPRS connection, opens a TCP connection between itself and the server, and sends the information in our predefined Google Protocol Buffers data structure (also described later). The software is implemented in C++, with POSIX Threads, one for each module. Thanks for the Linux system, every peripherials can be read like we read from a file.

2.3.1 Image processing

Our image processing method is an object detection based on Haar-cascade classification. See more information in this paper. The OpenCV library provides a standard solution for this kind of object detection. (For more, follow this link.) This method has great performance in object detection, see: this, or this. We obtained the pre-trained cascade file from School of Computing, University of Utah.

2.3.2 Google Protocol Buffers data structure

We have created our own data structure, which can be found in the src/ subdirectory of the project. For more information, please visit this link.

2.3.3 Java TCP Server

For debugging purposes we have implemented our own TCP Server which can parse and list the data sent by the Real-time Traffic Analyzer.

2.4 Design reuse

The Real-time Traffic Analyzer project is licensed under the GNU GPLv3 and can be found on Github.

Chapter 3

Results

3.1 Challenging issues

For those who develop in Vivado for the first time, like us, is really difficult to see this different perspective of the hardware development. (Especially, after years of experience in the Xilinx ISE.) Besides, after the Spartan 3/6 series, the ZYNQ SoC has its own difficulties. After gaining some experience with the development environment it seems easier. However, the most difficult part was the insufficient descriptions of Linux distributions applied in Embedded Systems. To run such a distribution is quite simple, but modifying it is a combersome task. (For example to attach a peripherial into the device tree of the kernel is really difficult.) We can conclude that after the hard beginning we get familiar with the Vivado and its hardware development process.

3.2 Summary

We have successfully implemented the Automatic Sensor Annotations part of the Robocar World Championship system, although several test should be performed. We also need a performance evaluation. Because we apply the agile development methodology, this first prototype will change many times during the development process.

Chapter 4

Conclusion

4.1 System overview

In this project, we have implemented an automatic data collector for traffic measuring. The system is based on a ZYNQ SoC, which runs an Embedded Linux System. Our software is able to give us traffic density values on roads, accurate time and position, and send to a server application via GPRS. The image processing method is based on a Haar-cascade classifier object detection.

4.2 Future work

Our future plan is to widen the peripherial set for the device. We are planning to attach an LCD touchscreen, mainly for debugging purposes and to inform the driver about some events related to the measurement. After a testing phase and a performance evaluation we will reconsider the whole system and every module (hardware and software). We are also planning a solution to implement the image processing method on FPGA with the HLS.

Chapter 5

References

- Robocar World Championship project homepage: https://github.com/nbatfai/robocar-emulator
- RoboCup: The Robot World Cup Initiative: http://dl.acm.org/citation.cfm?doid=267658.267738
- N. Bátfai, R. Besenczi, A. Mamenyák, M. Ispány, OOCWC: The Robocar World Championship Initiative, The 13th International Conference on Telecommunications, 2015, (accepted manuscript)
- AXI Video DMA (AXI VDMA): http://www.xilinx.com/products/intellectual-property/axi_video_dma.html
- Digilent Zybo: https://www.digilentinc.com/Products/Detail.cfm?Prod=ZYBO
- Vincotech's A1080-A GPS receiver: https://www.libelium.com/forum/libelium_files/GPS%20Receiver%20A1080%20V4.2.pdf
- SIM900 GSM module: http://wm.sim.com/producten.aspx?id=1019
- OV 7670 camera module: http://www.voti.nl/docs/OV7670.pdf
- Google Protocol Buffers: https://developers.google.com/protocol-buffers/
- Viola, P.; Jones, M.: Rapid Object Detection Using a Boosted Cascade of Simple Features, pdf
- Haar Feature-based Cascade Classifier for Object Detection: OpenCV Reference Manual
- Rainer Lienhart, Alexander Kuranov, Vadim Pisarevsky, Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection pdf
- Adam Schmidt and Andrzej Kasinski, The Performance of the Haar Cascade Classifiers Applied to the Face and Eyes Detection pdf
- Real-time Traffic Analyzer Github project homepage https://github.com/rbesenczi/real-time-traffic-analyzer
- YouTube demo video of the project https://youtu.be/ECosPSNb8_Q