Development of Advanced Driver Assistance Systems using LabVIEW and a Car Simulator

Benedikt Janßen, Philipp Wehner, Diana Göhringer, Michael Hübner
Chair for Embedded Systems for Information Technology and Application-Specific Multi-Core Architectures Group
Faculty for Electrical Engineering and Information Technology
Ruhr-University Bochum, Germany

{benedikt.janssen, philipp.wehner, diana.goehringer, michael.huebner}@rub.de

Abstract

Innovative approaches are required to face current academic challenges in engineering, such as increasing withdrawals from studies and a small number of female students. This article presents a remote lab exercise in the automotive domain targeting embedded system architectures and their programming. In the automotive industry, Electronic Control Units (ECUs) become more and more important. Advanced Driver Assistant Systems (ADAS) can help to increase safety and comfort. The efficient development of ADAS is therefore of high importance and mandatory for industrial competitiveness. Using the presented remote lab, students in electrical engineering and computer science can be introduced to this important topic in an early phase of their career. The exercise targets the development of an ECU that exemplary supports the driver in an everyday scenario. The development environment consists of National Instruments LabVIEW software connected to a car simulator.

• Keywords: **Applied computing → Education → Interactive learning environments • Applied computing → Education → Distance learning • Applied computing → Education → E-learning.**

Advanced Driver Assistance Systems, Automotive Electronic Control Units, LabVIEW, E-Learning, Car Simulator

1. Introduction

ELLI, as abbreviation for „Exzellentes Lehren und Lernen in den Ingenieurswissenschaften“ – „Excellent Teaching and Learning in Engineering“ combines the expertise of three universities in the German state North Rhine-Westphalia: RWTH Aachen University, Ruhr-University Bochum and the Technical University Dortmund. The goal of ELLI is to increase the quality of teaching and to improve the conditions for studying. With a focus on remote-learning environments, support of mobility and internationalization, professional competence and student lifecycle, ELLI targets innovative approaches regarding current academic challenges. It tries to decrease withdrawals from studies and to support the enrollment of female students.

This article describes a remote lab exercise, as part of ELLI. Students get the chance to develop Advanced Driver Assistance Systems (ADAS) using a LabVIEW-based environment. LabVIEW was selected as base system, as it can be integrated to the ELLI web interface. A server enables the parameterization of the LabVIEW Virtual Instrument (VI) across an Ethernet network. Students can access the remote labs at reserved times and accomplish their tasks from any computer on the internet.

The exercise targets the development of an ECU that exemplary supports the driver in an everyday scenario. The LabVIEW design software is hereby connected to a car simulator. The ECU consists of a virtual System-on-Chip (SoC) [1]. SoCs basically combine two views on an embedded system. On the one hand processors can be used to execute software while on the other hand peripheral modules can process sensor data and control actuators in parallel. The architecture of embedded systems and the fact that hardware structures get more and more complex play important roles in this context. One has therefore to distinguish between functionality on the software and on the hardware layer. A so called Hardware/Software Co-Design approach combines the development of the software with the simultaneous development of hardware. In the remote lab, the students can use a simple accumulator architecture to access peripheral modules via memory-mapping.

The paper is structured in the following manner: Section 2 presents related work in the area of E-Learning, remote labs and virtual system platforms. Section 3 shows the experimental setup including the LabVIEW application and the remaining features of the exercise. After presenting the user interface in Section 4, the planned experiments are shown in Section 5. The paper is concluded in Section 6.

2. Related Work

A LabVIEW-based remote laboratory is presented in [2]. It allows controlling a multi-pipe fluid flow experiment. Students can develop their own algorithms, using engineering software, and observe the results. A testbed is available that can be used to obtain parameters. The Chalmers University of Technology uses a LabVIEW-based remote Lab in the context of the automotive industry [3]. Students can use a web interface to control an electric drives system and to understand its design, the modeling and the assessment. The web course was successfully evaluated by the students. Kalúz et al. [4] present a hardware/software architecture for automatic-control remote laboratories. The novel approach is based on programmable logic controllers and industrial network routers. A web page with on-the-fly content generation, based on JavaScript, is presented. The authors evaluate their approach with the example of a thermo-optical education system and other remote laboratories.

Besides the simulation of a SoC in LabVIEW, alternatives exist that can be used to provide a virtual system. The open-source machine emulator and virtualizer QEMU [5] uses a dynamic translation of processor instructions of the guest system to instructions for the host system. QEMU achieves a high performance and can be extended with user-defined peripherals.

These peripherals could handle the connection with the car simulator and provide all the required functionality that exists besides the presented accumulator architecture in LabVIEW. A use case for these peripheral extensions is shown in [6], demonstrating the usage of QEMU for the emulation of Networks-on-Chip (NoCs). Gem5 is a modular simulator for different CPU processor models and memory systems [7]. Different layers of abstraction exist that can be used to find a tradeoff between simulation accuracy and performance. The available Instruction Set Architectures are ARM, ALPHA, MIPS, Power, SPARC and x86. It is furthermore possible to execute an Operating System (OS) on the virtual processors. Due to the flexible structure of Gem5, future versions of the presented remote lab could also utilize this simulator to enable a fine-grained view on the Processing System (PS). The Virtual System Platform (VSP), provided by Cadence [8], can be used to analyze hardware and software behavior at an early point in the development cycle. It is possible to extend the functionality of the simulator by custom made components. The Cadence VSP therefore supports the approach that is presented in this paper: External peripheral components can be implemented in virtual programmable logic, while accurate processor models execute functionality in software.

For this exercise, LabVIEW is used as it provides a flexible development environment. Getting started with LabVIEW is an easy task, even for Bachelor students. In future versions, the exercise can be extended with the aforementioned simulators.

3. Experiment Setup

The students, who connect to the experiment remotely, have access to a user interface that is written in LabVIEW. It is split into a programmer window, to develop an assembler application for the processor of the ECU, and a simulator window, to execute the assembler code. After the start of the experiment, the students enter their application into the programmer window. During the simulation process, commands are sent to the driving simulator and simulation data of the car is received. Moreover, current information about the ECU state, such as its data memory, is displayed. The overall setup is shown in Figure 1. The LabVIEW application (LabVIEW VI) running on the host computer uses a DLL to open a network socket and to connect to a local server. The server is necessary as the driving simulator supports only one incoming connection. A second connection can be opened by a supervisor who can control the simulated environment during the lab exercise. The protocol is based on TCP/IP and data transfers are constructed as strings, which can contain commands for the simulator or data from it. Inside the DLL, the data is parsed into the correct string format and subsequently send over a local network socket.

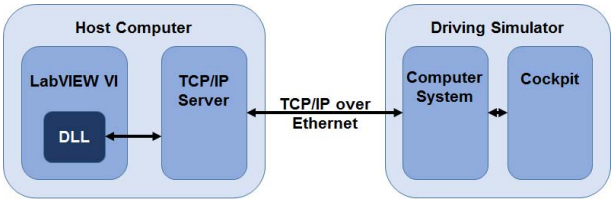

Figure 1: Setup of the Experiment

3.1 LabVIEW Application

The LabVIEW VI consists of two finite state machines (FSM). These FSM implement the programmer and the simulator. Their interfaces are described in Section 4.

The programmer is used to write the assembler code of the ECU's processor into the program memory. The corresponding FSM is shown in Figure 2. It is possible to store new instructions and to set the address of the program memory, in order to write to arbitrary addresses. After the code is stored in the program memory, the students can start the simulator and the data is passed to the next FSM.

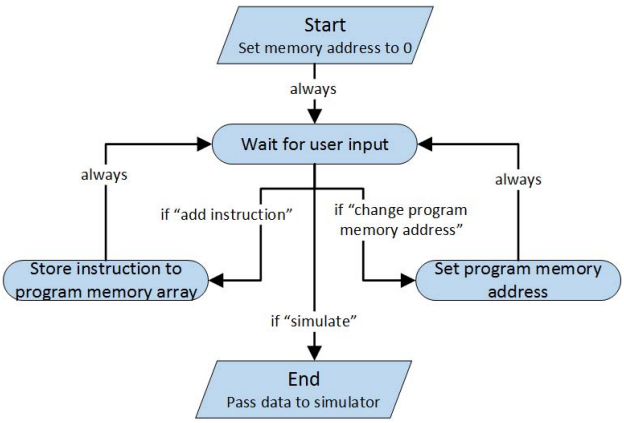

Figure 2: Programmer State Machine

The simulator FSM communicates with the car simulator in every cycle in order to update the sensor data of the car simulation. Moreover, the user can choose to execute the next instruction or to change the state of one of the cockpit inputs and outputs, such as the cruise control and the radio.

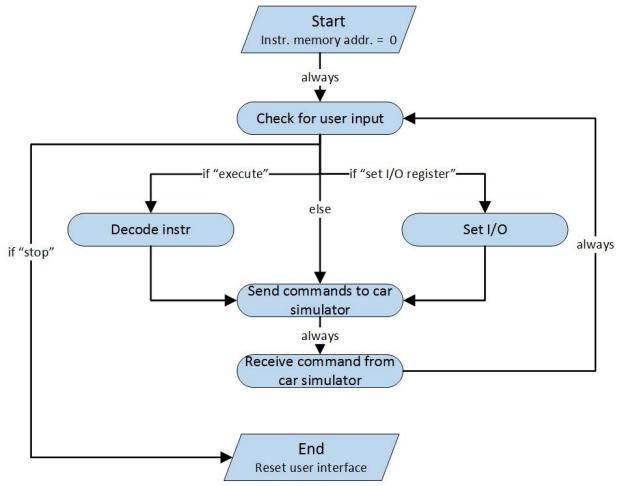

Figure 3: Simulator State Machine

3.2 ECU Processor Architecture

In order to teach the concept of memory-mapping, to show students the meaning of registers, and to be compliant with the offered lectures Computer Architecture and Hardware/Software

Codesign at the Ruhr-University Bochum, the accumulator architecture has been chosen as the basis for this exercise. Furthermore, the accumulator machine is an excellent entry in the topic of processor architecture and therefore of great importance from the didactic point of view. The structure of the chosen accumulator machine is depicted in Figure 4.

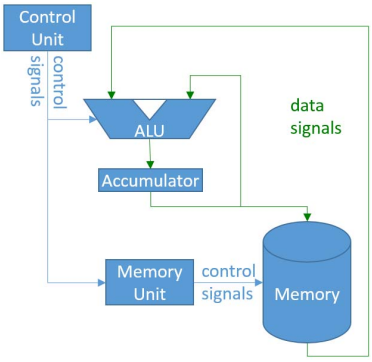

Figure 4: Accumulator Architecture

The accumulator architecture is a very basic architecture compared to today's x86 architecture. The result of an operation is always written into the accumulator register. Commands can access the memory directly and operations with two operands fetch one from the accumulator register and the other one from memory. A list of available commands is given in Table 1.

Table 1: Available Commands

Command	Operand	Description
Load	[Addr]	Load content from memory into the accumulator register
Store	[Addr]	Store content from the accumulator into memory
Add	[Addr]	Replace content of the accumulator by the sum of the accumulator value and the value at memory address [Addr]
Sub	[Addr]	Replace content of the accumulator by the substraction of the accumulator value and the value at memory address [Addr]
Mult	[Addr]	Replace content of the accumulator by the multiplication of the accumulator value and the value at memory address [Addr]
BGT	[Addr]	Executes the following command if the value of the value at memory address [Addr] is greater than the accumulator content's value
BST	[Addr]	Executes the following command if the value of the value at memory address [Addr] is smaller than the accumulator content's value
BEQ	[Addr]	Executes the following command if the value of the value at memory address [Addr] is equal to the accumulator content's value
JMP	[Addr]	Jumps to address [Addr]
NOP		No operation will be executed by the processor

Moreover, the student will learn about resource usage of processors implementation in a theoretic part of the lab exercise. For this, a basic architecture, such as the accumulator, shows a significant difference in necessary resources compared to today's microprocessors.

On the other side, it is a good example to teach students the downside of having only few, here one, internal registers. Together with the theoretical part, they learn the costs of the memory access that is necessary in every command listed in Table 1.

3.3 Processor Peripherals

The peripheral modules of the processor can be accessed via memory-mapped I/O. Therefore, students will learn that certain memory addresses do not refer to the data memory of the processor but to registers of processor peripherals. The memory address mapping is listed in Table 2. A description of the internals behind the memory-mapped I/O is given in the theoretic part.

Table 2: Memory Address Mapping

Address	Function	Read/ Write	Description
0 - 15	Memory	Rd/Wr	Data memory block
16	Car reset	Rd/Wr	Reset car
17	Brake	Rd/Wr	Brake on/off
18	Memory	Rd/Wr	
19	Light	Rd/Wr	Light on/off
20	Radio	Rd/Wr	Radio on/off
21	Handbrake	Rd/Wr	Handbrake on/off
22	Cruise control	Rd/Wr	Cruise control on/off
23	unavailable	%	Read and writes are not allowed
24	Ignition	Rd/Wr	Ignition on/off
25	Steer angle	Rd/Wr	Steering wheel position in degree: 0 to 360
26	Brake pedal	Rd/Wr	Brake pedal push state in percent: 0 to 100
27	Accelerator pedal	Rd/Wr	Accelerator pedal push state in percent: 0 to 100
28-36	Memory	Rd/Wr	Data memory block
37	Seatbelt sensor	Rd	Seatbelt state open (0) / close (1)
38	Cruise control	Rd	Cruise control button pressed (1) / released (0)
39-63	Memory	Rd/Wr	Data memory block

3.4 Car Simulator

A car simulator allows the testing of automotive components in a safe but accurate environment. The cockpit is hereby equipped with a driver seat, a steering wheel, pedals for brake and acceleration, and functionality that is provided by software.

For the purpose of the presented remote experiment, the car simulator „Trainer F12PT-3L40", manufactured by Foerst GmbH [9], is used. It is shown in Figure 5 and consists of a single-seated car cockpit with three monitors for visualization. The simulator includes all relevant electrical and electronic devices that can be used to extend its functionality, such as interfaces and control units. Sensors consecutively measure information that is processed by the control software. The communication of the car simulator with external components is enabled by an integrated computer system via an Ethernet interface. The Ethernet interface allows the remote control of the simulated environment and the simulated vehicle. This interface also allows reading out values from the virtual car represented by the car simulator. The car simulator is therefore suitable for the development of ADAS, as it is flexible and accurate. Via the Ethernet interface, ECUs can send relevant control information to the car simulator that reacts on the incoming data. Also sensor data is transmitted to the sender on request. The car simulator hereby acts as server and listens to incoming TCP/IP connections on a predefined port. Requests to the simulator are capsuled in a message container.

Figure 5: Foerst Trainer F12PT-3L40

All in all, more than 150 functions are available that can be used to manipulate the driving behavior, weather conditions and vehicle parameters. Also critical driving situations can be instantiated by respective functions.

Conceivable is also the analysis of the driving environment by specialized image processing algorithms. This can exemplary include the traffic signs or lane detection. The car can automatically react in the case of an emergency or inform the driver appropriately.

Within the driving environment, each car is identifiable by a unique id. It is furthermore possible to calculate the distance between two cars. With this information, an adaptive cruise control system can be implemented that keeps the distance to a car in front of the driver's car constant.

The Ethernet interface can moreover be used to connect the car simulator to external development environments, such as Matlab/Simulink or LabView. These environments can improve the design of control units and help to analyze the results on a graphical user interface (GUI).

The car simulator is hence a flexible tool for the development of ADAS and other areas of research, right up to car to car communication technology.

4. Graphical User Interface

The user interface is accessed remotely through the website of the ELLI project. The access to the LabVIEW VI is enabled through a framework which is based on the Microsoft Internet Information Server. In this section we are going to present the two views of the graphical user interface (GUI), the programmer and the simulator view. The webserver framework is not part of this article.

4.1 Programmer View

The programmer view is used for adding instructions and their arguments to the program memory of the ECU processor. It is depicted in Figure 6. During start or reset, the program memory address is set to the first address, 0. The address can be changed by the address field and the "Set" button. These two elements are labeled with '1'. The students can use this to insert a subroutine at a specific memory location. The content in the program memory cannot be moved. Therefore, the developer needs to schedule the program memory layout before loading the program. By doing so, the students are forced to plan the size of their program.

The overall memory size is given on the lower left side of the programming window, labeled with '3'. Instructions can be loaded into the memory by the inputs labeled with '2'. The available commands correspond to the ones listed in Table 1. The input 'Value' is used as operand value.

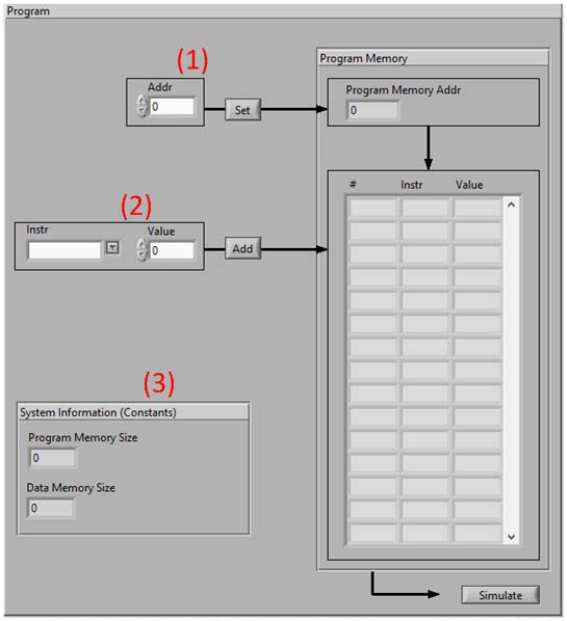

Figure 6: Programmer View

When the program is written to the program memory, the simulation can be started by pressing the "Simulate" button on the lower right side of the programmer view window.

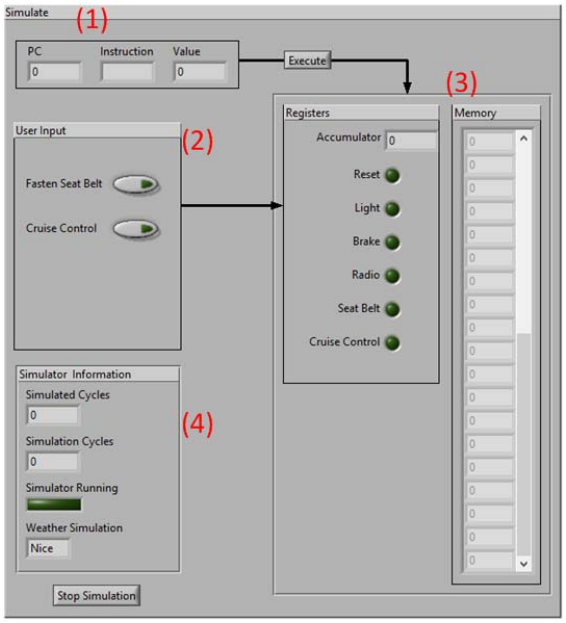

Figure 7: Accumulator Architecture Simulator View

4.2 Simulator View

During the simulation the current program counter (PC), instruction and its operand are shown in the section labeled with '1' in Figure 7. The user can simulate input signals such as closing the seatbelt or turning the cruise control on or off by the buttons in the "User Input" section labeled with '2'. The memory

content is shown in the section labeled with '3'. This section also shows the accumulator register content, as well as the state of some of the registers which are mapped in the memory space. The "Simulation Information" section, labeled with '4', shows the current state of the simulation.

The simulated cycles correspond to the cycles of the processor which executes one instruction per cycle. The simulation cycles are the cycles the simulator FSM is executed. Moreover, the current state of the weather inside the simulated environment is indicated.

In order to stop the simulation, the "Stop Simulation" button can be used.

5. Experiment

The remote experiment implemented within the ELLI project targets the implementation of a basic ECU program, which controls parts of the car after the power on process. Moreover, it monitors user inputs to activate the cruise control and triggers the cruise control peripheral module.

Therefore, the students need to implement the following sequence:

First they should check whether the driver's seatbelt is closed or not. If this is the case, the motor can be started. Afterwards, the weather should be checked. If it is foggy, rainy, twilight or night, the lights should be switched on. Otherwise, they should be switched off. After checking the weather conditions and before the driver starts driving, the radio should be turned on and the electronic handbrake must be released.

If the cruise control button is pressed while driving, the program should activate the cruise control peripheral module in order to keep the current velocity.

6. Conclusion and Outlook

This article presents a remote laboratory in the domain of automotive ECU development and embedded systems. A car simulator is connected to a LabVIEW environment, realizing an accumulator architecture. Students can remotely access the LabVIEW VI to manipulate the driving scenario based on custom algorithms. The exercise is part of the joint project "ELLI". The project targets the improvement of teaching and innovative approaches to face current academic challenges.

In the future, students will have access to the remote laboratory during the semester. A meaningful evaluation will also be part of all ELLI exercises. The students will therefore get the chance to further improve the remote laboratory, which will help to reach the goals of the overall project.

If the feedback of the students is positive and the evaluation is successful, it is also planned to extend the exercise by adding more experiments with more complex processor models and NoC based architectures. One possible extension is the integration of MPSoCSim [10], which is a current research project within our groups. The web interface of the remote laboratory can be extended to enable the parameterization of the NoC simulator. This enabled more advanced experiments with more advanced driver assistance systems.

7. Acknowledgements

The project "ELLI" is funded by the Federal Ministry of Education and Research and has a duration from 2011 to 2016.

8. References

[1] P. Marwedel. "Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems". 2nd Edition. Springer 2010. ISBN 978-94-007-0256-1

[2] P. K. Imbrie, S. Raghavan. "Work In Progress - A Remote e-Laboratory for Student Investigation, Manipulation and Learning". Proceedings Frontiers in Education 35th Annual Conference, Indianopolis, IN, 2005, pp. F3J ff.

[3] S. T. Lundmark, A. Rabiei, T. Abdulahovic, S. Lundberg, T. Thiringer, M. Alatalo, E. A. Grunditz, C. Du-Bar. "Experiences from a distance course in electric drives including on-line labs and tutorials". XXth International Conference on Electrical Machines (ICEM), Marseille, 2012, pp. 3050-3055.

[4] M. Kalúz, J. García-Zubía, M. Fikar, L. Čirka. "A Flexible and Configurable Architecture for Automatic Control Remote Laboratories". IEEE Transactions on Learning Technologies, vol. 8, no. 3, pp. 299-310, July-Sept. 2015.

[5] QEMU: Open Source Processor Emulator, http://wiki.qemu.org

[6] P. Wehner, D. Göhringer. "Parallel and Distributed Simulation of networked Multi-Core Systems". In: 2014 International Symposium on System-on-Chip (SoC). Tampere, Finland, Oct. 2014, pp. 1-5.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood. "The Gem5 Simulator". In: SIGARCH Comput. Archit. News 39.2 (Aug. 2011), pp. 1-7.

[8] Cadence Design Systems, Inc. "Virtual System Platform – An open, connected, and scalable virtual prototyping solution". Available at: https://www.cadence.com

[9] FOERST GmbH, www.fahrsimulatoren.eu/

[10] P. Wehner, J. Rettkowski, T. Kleinschmidt, D. Göhringer. "MPSoCSim: An extended OVP Simulator for Modeling and Evaluation of Network-on-Chip based heterogeneous MPSoCs". In Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XV), Samos, Greece, July 2015, pp. 390-395.