Lab Protocol

Code mobility in Networked Embedded System

NES

Group 4

abstract: The lab protocof contains the final project documentation. We present the introductory part of the project and all necessary organization details in the chapter 1. The requirements are stated in chapter 2. Chapter 3 provides the reader with unambiguous specification, Implementation details are represented in chapter 4.

January 29, 2013

Igor Pelesić, Matrikelnumber 0006828
Konstantin Selyunin, Matrikelnumber 1228206
Miljenko Jakovljević, Matrikelnumber 0426673

Contents

1 Project Outline

1.1 Organization

The roles and responsibilities for the project are represented as follows:

- **Project manager:** Konstantin Selyunin [S]
  - Defining tasks
  - Internal organization
  - Control meeting deadlines
  - Agent assember language: Development and Implementation
  - Adaptation of drivers
- **System architect:** Igor Pelesić [P]
  - Defining and reviewing technical aspects
  - Designing communication protocol
  - Adaptation of drivers
  - Platform: Design and Implementation
- **Zigbee communication:** Miljenko Jakovljević [J]
  - Designing board-to-board communication using zigbee
  - Presentation for workshop 1: communication part

1.2 Project Description

The purpose of the project is to design, implement and evaluate code mobility platform on Embedded system engineering board [2]. Our goal is to develop the system that allows users to build and execute simple agent program on top of hardware ESE platform. To achieve the goal we have developed three layered software: agent layer, platform layer, communication layer. Our goal is to show that code mobility concepts that are successfully used on much higher abstraction level are applicable for the embedded applications. During the project we have developed and implemented infrastructure that allows developer of agent program do not be aware of the hardware services presented on the given platform.

1.3 Definitions, Acronyms, and Abbreviations

By *code mobility* we mean the capability of code to change the location where it is executed.

Strong code mobility is the ability to allow migration of both code and execution state to the destination, *weak* code mobility allows code transfer but it does not involve the transfer of the execution state.

Platform is a component that provides corresponding hardware services to

1.4 Background

The research has been done to use code mobility in distributed environment [1] and various application has been developed including [3] web application platform that allows people without major programming experience to develop the application as work-flow specification in graphical form. The use of code mobility is to "move the knowledge close to the resources" [4] and enable higher flexibility of accessing remote resources.

1.5 Workpackages

In the following section we describe workpackage deliverables for our project:

1.5.1 WP1 Documentation

1.5.1.1 Requirements and Specification

Before the development and implementation, clear requirements should be defined. In this deliverable we define user roles, global requirements for the project, functional and non-functional requirements.

1.5.1.2 Presentation for Workshop 1

In the first workshop we introduce to the audience the general overview of our project and specification. For this deliverable we have done self-contained presentation, which inroduce all necessary concepts, our goals and approach. The goal for preparing the presentation is to convey a message of our project to audience, assuming no prior knowledge of code mobility concepts. We introduce milestones and time plan, as well as project management concepts to achieve the goal.

1.5.1.3 Presentation for Workshop 2

In the second workshop we present the results of our work. We do the test application for using the code mobility on the board. We discuss our major design decisions that have beed made during the design and implementation phases.

1.5.1.4 Lab protocol

The lab protocol will consist of outline of our project, the requirements and specification for the project. In addition it contains precision description of Agent language, low-level assember-like language that support code mobility syntax. Desciption of the API and structure of our software.

Workpackage 1. Documentation			
Responsible:	Konstantin Selyunin, Igor Pelesić, Miljenko Jakovljević	Start date:	07.11.2012
Deliverables:	D1.1 Requirements and Specification D1.2 Presentation for Workshop 1 D1.3 Presentation for Workshop 2 D1.4 Lab protocol	Finish date: Estimated Effort: Interdependencies:	28.01.2013 180 hours all

1.5.2 WP2 Adaptation of drivers

The platform will provide access to hardware for mobile agents. During this deliverable drivers for the following peripherials should be adapted or otherwise implemented:

1. **Bargraph:** Port A of nodes 0 and 1 is connected to the led bargraph. The driver should display encoded in binary number a value from the range `0 ... 255` on the bargraph.

2. **Heater:** Two heating resistors on the node 2 could be controlled by PWM signal. Driver that provide setting a duty cycle should be implemented. To control PWM PIN of the microcontroller timers should be configured and appropriate mode of the PWM should be selected.

3. **Cooler:** The cooling fan is also controlled by PWM signal. The same aproach as for the heating should be used here. Controlling the speed of the fan should be done by setting up the duty cycle of the PWM signal.

4. **Temperature sensor:** Three temperature sensors are connected to the bottom of the sink with I2C interface. The driver should read data from all sensors and return the average.

5. **Led matrix display:** Led matrix display with 6 segments of 5 by 7 each is connected to the node 3. The driver should provide API for writing single character and arrays of characters to the led matrix.

6. **TFT display:** Node 2 is connected to 640 by 360 TFT display. The driver should provide the following capabilities: set the cursor to the position on the display, set font and background colors and print arrays of characters on the display.

Workpackage 2. Adaptation of drivers			
Responsible:	Igor Pelesić , Konstantin Selyunin	Start date:	15.11.2012
Deliverables:	D2.1 driver implementation	Finish date:	12.12.2012
		Estimated Effort:	50 hours
		Interdependencies:	

1.5.3 WP3 Agent language tool

To design mobile agents special language that supports constructs for mobility is required. In this deliverable we design and implement the low-level assember-like language. The Agent language should provide access to the hardware as well as have syntax for expressing code-mobility concepts.

Workpackage 3. Agent language tools			
Responsible:	Konstantin Selyunin	Start date:	06.12.2012
Deliverables:	D3.1 agent language tool	Finish date:	21.12.2012
		Estimated Effort:	40 hours
		Interdependencies:	D1.1

1.5.4 WP4 Platform Communication

Protocol needs to provide environment for communication between platforms and transfering code. During this deliverable communication protocol that fulfil aforementioned requirements should be implemented. The main purpose of the project is to implement main code mobility concepts so we do not restrict ourselves to fulfil real-time requirements. CSMA/CA protocol will suit for our purpose, so we propose to implement communication using this protocol. One of the main goals for possible future work is to make agents and message transfer real-time.

Workpackage 4. Communication			
Responsible:	Igor Pelesić, Miljenko Jakovljević	Start date:	10.12.2012
Deliverables:	D4.1 CSMA/CA communication protocol	Finish date:	21.12.2012
		Estimated Effort:	60 hours
		Interdependencies:	D1.1

1.5.5 WP5 Platform

Platform supports concurrent execution of mobile agents as well as provides means for transfering agent code and messages. The main challenges in this deliverable are to implement priority based scheduler, execution layer and communication layer. It is of paramount importance that each platform support only hardware that is physically connected to dedicated μC, to save memory. It should be done during compile time.

Workpackage 5. Platform			
Responsible:	Igor Pelesić	Start date:	21.12.2012
Deliverables:	D5.1 Platform. Scheduler	Finish date:	15.01.2013
	D5.2 Platform. Execution layer	Estimated Effort:	120 hours
	D5.3 Platform. Communication layer	Interdependencies:	

1.6 Milestones and timeplan

For successful completion of our project, the following deadlines should be met:

- 22.11.2012 Clear defined requirements and specification
- 06.12.2012 Workshop 1: presentation of project outline, specification and requirements. Discussion of challenges, possible fallacies and pitfalls.
- 15.12.2012 Avaliability of Agent language tool
- 21.12.2012 Completion of communication protocol (D4.1)
- 23.01.2013 Avaliability of platform (D5.1), completion of implementation work.
- 29.01.2013 Documentation of the work in the lab protocol
- 29.01.2013 Demo application for workshop 2.
- 31.01.2013 Workshop2: Presentation of results. (D1.3)

1.7 Gantt diagramm

To represent interdependencies between tasks and sequence of execution of all tasks in our project we use Gantt diagram.

Figure 1.1: Gantt Diagram of the Project

2 Requirements

This section lists all requirements that should meet the project with respect to user roles of code mobility application. Defining requirements in a rigorous way will help us to exercise realistic validation scenarios.

2.1 User roles

R_UR_1 Application Developers (Tasks: Create control application in agent language, debug, test, prepare deployment packages)

R_UR_2 Application Consumers (Tasks: Deploy control application on target system, fill valuable bug reports)

R_UR_3 Plattform Developers (Tasks: Maintenance, Extensions, Porting to another target board)

R_UR_4 Application Designers (Tasks: Design control application)

2.2 Global Requirements:

2.2.1 Application Development requirements:

R_AD_1 The App Developer should be enabled to instantiate up to 4 agents on a single node, which are running concurrently.

R_AD_2 The App Developer should be allowed to configure the execution scheduling of the agents via a prioriatization of the agents.

R_AD_3 The platform should provide a simple agent programming language to the App Developer in which the agents of an application can be developed.

R_AD_4 The agent language should provide the App Developer with the possibility to reproduce its code on another node or on another board.

R_AD_5 The agent language should provide the App Developer with the possibility to communicate with another agent on the same board.

R_AD_6 The agent language should provide the App Developer with the possibility to access the node hardware.

R_AD_7 The agent language should provide the App Developer with the possibility to implement loops.

R_AD_8 The agent language should provide the App Developer with the possibility to compare variables.

R_AD_9 The agent language should provide the App Developer with the possibility to perform addition, subtraction, multiplication and division on variables.

R_AD_10 The agent language should provide the App Developer with the possibility to perform delays in the execution of code.

R_AD_11 The platform should allow debugging of agents executions.

R_AD_12 The platform should provide means for the creation of easily installable deployment packages.

2.2.2 Application Consumers requirements:

R_AC_1 The platform should provide means to deploy the agent software on the target boards easily.

R_AC_2 A tracing mechanism should be provided in order to ease the process of fault detection and to allow valuable bug descriptions.

2.2.3 Application Designers requirements:

R_A_DES_1 A description of the platform possibilities and limitations should be provided.

R_A_DES_2 The platform should provide means for reducing the overall complexity of a system, by allowing encapsulation of different tasks.

R_A_DES_3 The platform should provide configurable inter agent communication facilities.

R_A_DES_4 The platform should provide means to enable standby scenarios by allowing dynamical code reproduction.

R_A_DES_5 The platform should provide means for strong mobility, where an agent and its execution state are transferred to a new node or board and the execution on the new destination is started from the memorized state.

R_A_DES_6 A description of a platform should provide a list of all available services

2.3 Non-functional requirements

R_NF_1 The platform should be open to extensions i.e adding new hardware.

R_NF_2 The agent language should be extendable.

R_NF_3 Scalability

R_NF_4 Documentation

R_NF_5 A platform tracing mechanism should be provided which allows for more efficient bug-fixing.

2.4 Low-Level Requirements

2.4.1 Communication protocol

R_LL_CP_1 Protocol must provide means to avoid collisions on the bus

R_LL_CP_2 Protocol must provide means to check correctness of the data sent

2.4.2 Drivers

R_LL_DRV_1 Drivers shall deliver access for the platform to hardware by means of API

R_LL_DRV_2 The cooler driver must provide means to set up the duty cycle of the fan in range 0 (turn off) to 100 (full speed).

R_LL_DRV_3 The heater driver must provide functions to set the dissipated power of the heating resistors in range: 0 (turn off) to 100 (max power dissipation).

R_LL_DRV_4 Temperature driver must provide means to read temperature from all three sensors with precision of 1/8 of degree Celcius.

R_LL_DRV_5 Let matrix driver must provide means to display char arrays on the led indicators.

R_LL_DRV_6 TFT display driver must provide means to set background color of a display, position cursor to the desired location, set the font and background color and print array of characters on the display

3 Specification and design

3.1 General

The following figure depicts the general outline of the code mobility project.

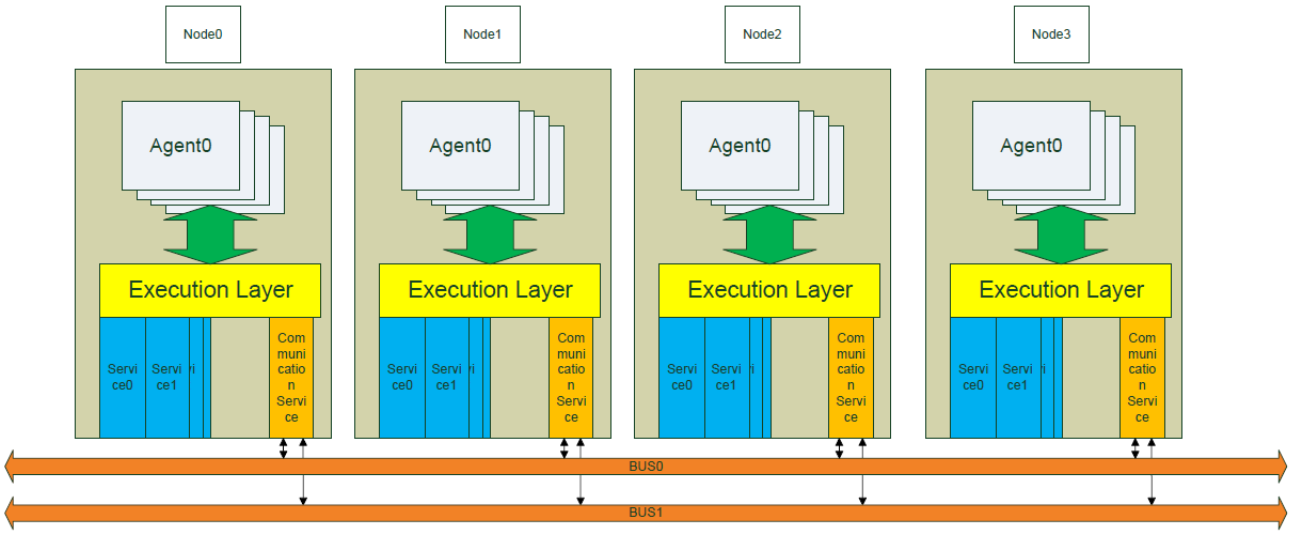

Figure 3.1: Overview

On each of the 4 nodes, which can be found on the ESE board, a virtualization platform will be deployed. This virtualization platform will be able to execute up to 4 agents concurrently. The agents will be programmed in a simplistic assembler like agent language. On the platform there will be an execution layer which is able to execute the agent language. The agents will be able to access the hardware attached to a node via services which are provided by the virtualization platform. Additionally the agents can reproduce themselves to another node or even board. Within the platform a scheduler will be responsible for providing execution time to each of the agents according to their priority.

3.2 Virtualization Platform

The main task of the virtualization platform is to interprete the agent language commands of the agents and to provide them access to the hardware attached to a node via well defined interfaces. Additionally the platform should allow the concurrent execution of the agents. Therefore some basic means for code and data protection for the agent memory is required. This is achieved by assigning each of the agents an own memory segment and not allowing any other agent to access any other memory but its own. If some collaboration between the agents is required this must be requested via the communication service. The metadata of an agent as its code and memory segment will be stored in a structure that is shown in the figure below.

Every agent has a unique id within the virtualization platform. Additionally a priority and a status for the scheduler are stored. Assigning these values for an agent lies within the scope of an agent developer. Reproducing an agent on a virtualization platform where the agent's id

```
typedef struct {

        uint8_t id;
        uint8_t priority;
        agent_status_t status;

        uint32_t status_flag;
        uint16_t pc;

        int16_t regs[REG_MAX];

        uint16_t code_len;
        uint16_t regstr_len [STR_REG_MAX];

        uint16_t* code;

        char** reg_str;

        volatile char* rec_msg_content;
        volatile uint16_t rec_msg_len;

} agent_t;
```

Figure 3.2: Agent structure

is already used will result in a denial of the reproduction by the platform. Every agent has 13 numerical general purpose registers used for the execution of the agent language. Additonally there are 3 char general purpose registers. The result of every agent language command will be written to the accumulator. There is also a program counter which is used for the execution of the agent and the numerical agent language representation is stored as well. The agent structure also contains a buffer for receiving messages from other agents.
In order to reproduce the agent on another board or node the agent's structure needs to be serialized and transmitted via the communication layer.
Additionally the virtualization platform has to provide the agent developer with some means to deploy the agent executable to the virtualization platform, during compilation of the platform. During the initialization of the platform all deployed agents should be instantiated on the given platform.

3.3 Execution Layer

The execution layer is responsible for the execution of an agent which is written in the agent language and later translated to agent opcodes. The agent language provides means for:

- storing values to the general purpose registers
- comparing the contents of the general purpose registers
- performing basic mathematical functions like addition, subtraction, multiplication and division
- a jump operation

- reproduction and cloning functions
- sleep, delay and terminate functions
- functions to access the hardware attached to a node

If a function of the agent language returns a value, this value will be stored in the accumulator, where it can be used later on for further operations e.g. comparison etc.
The basic workflow of the execution layer as soon it is called by the scheduler is to read the next agent language opcode (all agent opcodes have a fixed length) as identified by the program counter, to decode it and to perform the function which is described by the opcode. Eventually the program counter value is changed and the control is returned back to the scheduler.

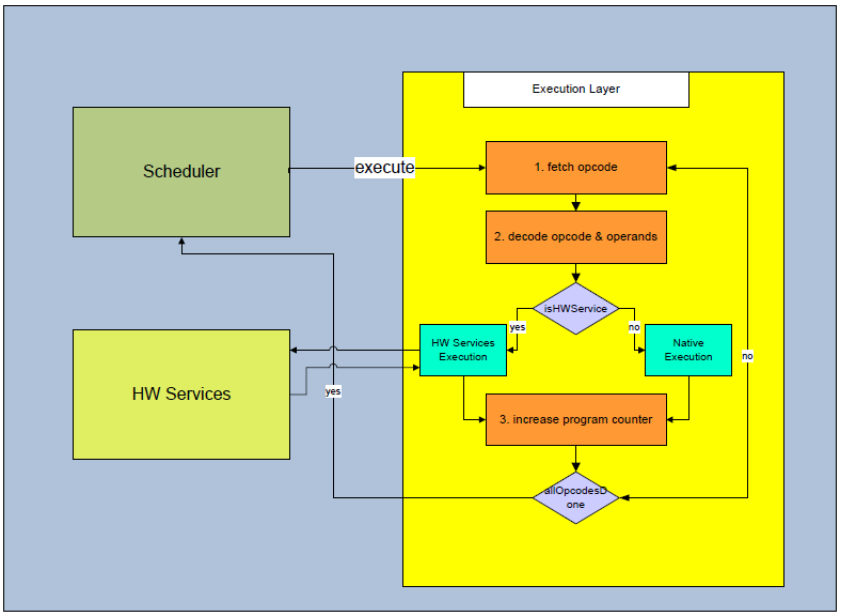

Figure 3.3: Execution Layer

The execution layer is called by the method execute which takes the following input parameters:

- Pointer to a specific agent structure
- Number of opcodes to execute

3.4 Hardware Services

The virtualization platform provides access to the hardware attached to a node via according hardware device drivers. The methods of the device drivers are made public to the execution layer which in turns allows the agents to access these methods. As the hardware supported by a node differs from node to node the virtualization platform should be able to discover during its initialization which hardware is supported on the node where it's running.
This will be achieved by defining a global set of function pointers within the virtualization platform. This set should contain all possible methods of all available device drivers. During initialization the platform will assign the according function pointer to a method provided by the device driver if the device is supported, otherwise the according function pointer will stay null.

```
/**
        Function : execute_agent
                excutes the next opcodes of an agent
        Returns :
                the amount of successfully executed opcoded
        Parameters :
                agent : agent status structure
                opcode: amount of opcodes to be executed
**/
uint8_t execute_agent(agent_t *agent, uint8_t opcode_size);
```

Figure 3.4: Executing agent language opcode

```
typedef struct {

        void (*set_bargraph)(uint8_t value);

        uint32_t (*clk_get_time)(void);

        void (*set_cooler)(uint8_t duty_cycle);

        void (*DISPLAY_drawBg)(uint16_t rgb);

        void (*DISPLAY_drawDot)(uint8_t row, uint8_t col,
                                uint16_t rgb, uint8_t grid);

        void (*DISPLAY_draw_char)(uint8_t x, uint8_t y,
                                uint16_t font_color, uint16_t bg_color,
                                uint8_t pixel_size, char c);

        void (*heater_set)(uint8_t duty_cycle);

        void(*button0_callback)(void);
        void(*button1_callback)(void);

        uint16_t (*therm_get_temp)(uint8_t name);

        void (*dotmatrix_send)(char *data);

} drivers_t;
```

Figure 3.5: Device drivers methods

The platform detects all the supported drivers on a specific node by inspection of the linked drivers. All drivers linked will be initialized during the setup of a platform and the methods provided by the drivers will be stored in the global function pointer table. Choosing this approach we would reach some form of modularity which would allow us to exchange the device drivers without necessity to change the platform code.
If an interaction with a device driver is blocking, then the calling agent will be put to status blocking unless there is an answer from the device driver.

3.4.1 Device drivers

3.4.1.1 Cooler

The device driver for the cooler should be initialized with a function:

- `void init_cooler(void)` This function should configure the timer and set PWM mode. After executing init function cooler should stay off.
- `void set_cooler(uint8_t duty_cycle)` This function sets the duty cycle of the PWM-signal, which controls the speed of the fan and the cooling effect.
- `required components` 1 timer for PWM signal

3.4.1.2 Heater

The device driver for the heating registers should be initialized with a function:

- `void heater_init(void)`

  This function should configure the timer and set PWM mode. After executing init function heater should stay off.
- `void heater_set(uint8_t duty_cycle)`

  This function sets the duty cycle of the PWM-signal, which controls the dissipated power (0 - no heating, 100 - max power dissipation)
- `required components` 1 timer for PWM signal

3.4.1.3 Temperature sensor

The temperature sensor driver should be initialized with the following function:

- `void therm_init(void)`

  This function should initialize temperature sensors connected to I2C bus.
- `uint16_t therm_get_temp(uint8_t name)`

  This function returns the value of the temperature in degrees Celcius.

3.4.1.4 Led matrix

The led matrix driver should be initialized with the following function:

- `void init_dotmatrix(void)`
- `void dotmatrix_send(char *data)`

  Using this function we send the first six characters to the led matrix.

3.4.1.5 Bargraph

With the following function we initialize LED bargraph, connected to the port A of nodes 0 or 1

- `void bargraph_init(void)`
- `void set_bargraph(uint8_t value)`

  This function is used to display the corresponding `value` on the bargraph.

3.4.1.6 TFT display

The following function is used to initialize TFT display that is connected to the node 2 of the ESE board:

- `void DISPLAY_init(void)`
- `void DISPLAY_drawBg(uint16_t rgb)`

  This function is used to draw the background of the display. RGB color could be defined using the following macro: `RGB(R[0..255], G[0..255], B[0..255])`.
- `void DISPLAY_string(uint8_t x, uint8_t y, uint16_t font_color, uint16_t bg_color, uint8_t pixel_size, char *string)`

  The following function is used to display char array on the display, starting from the position `x, y` with corresponding RGB values of font and background. The size of the font could be changed by setting the size of the basic drawing pixel.

3.5 Communication Layer

The agents should be able to communicate with other agents on the same node or on the same board. Therefore the agent language provides means to request the sending or receiving of a message.
The sending function is blocking the further execution of the agent until the message is sent. When an agent wants to send a message this message is proceeded to the communications service which takes care of the actual transmission. While the sending procedure is ongoing the further execution of the sending agent is blocked. As soon as the communication service signalizes a successful message transmission or a failure the result of the sending function is written to the accumulator and the agent will be made available for further execution.
When an agent sends a message to another agent, the receiving platform stores the content to the receiver agent structure. The receiving agent is able to retrieve the last message from its buffer. However only one message can be stored within the receiving agent structure and the next message will overwrite the content and possible the id of the last message.
The communication service provides no guarantees that sending of a message will succeed; it works on a best effort approach. Therefore the agent developer has to make sure by reading the return value of a sending operation whether the message was successfully sent or not and should initialize a retransmission in case of failure.
Every message sent should be identified by an id, in order to allow the transmission of messages with different semantics.
The receiver of a message should be identified via the node number (0..3) where the receiving agent is currently expected to be running and the receiver agent id. As the ids of agents are within the scope of the agent developer she has to make sure, that the correct receiving agent is addressed. Additionally a multicast message could be supported by allowing omitting the node address which should result in sending the message to all agents identified by the provided id.

3.6 Scheduler

The main task of the scheduler which is part of the virtualization platform is to identify the next agent to be executed and to utilize the execution layer to perform the execution of the according agent. The decision which agent to be chosen should be made on a static priority based scheduling policy.
Every agent is assigned a priority (0..254) by the agent developer which is stored within the agent structure. The highest priority is 254 and the lowest priority is 0. Based on the priorities

of the currently running agents the scheduler creates a static list by which the order of the agent execution is defined. The scheduler instructs the execution layer to execute exactly priority + 1 opcodes for a given agent. Eventually the control returns to the scheduler and the next agent from the list is picked. The list is iterated cyclically.

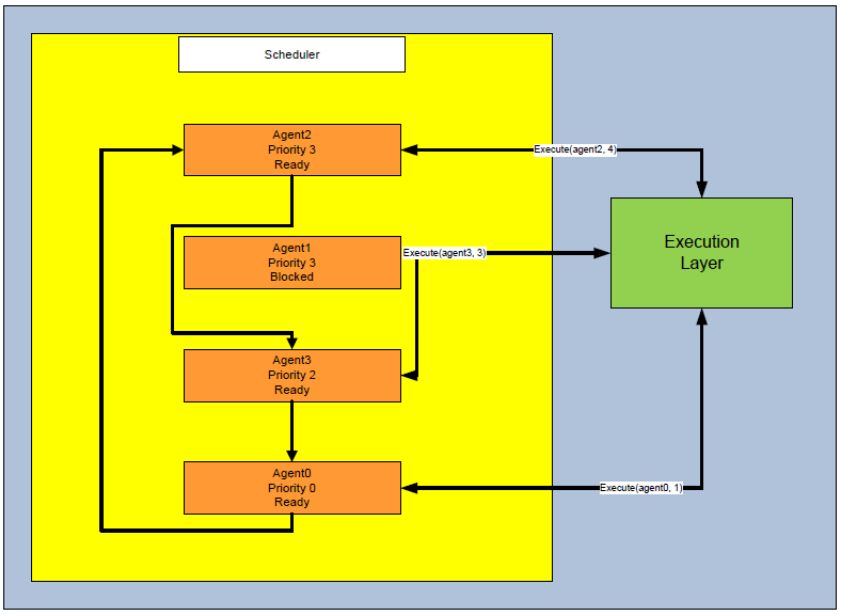

Figure 3.6: Scheduler

If an agent which is to be scheduled next is currently blocked, then its execution should be omitted.
As an agent can reproduce it self to another node or board or clone itself within the same platform the scheduling list requires adaptation as soon as new agent is deployed on a platform. Whenever a given platform is the destination end point of a reproduction respectively cloning operation the scheduler needs to update its scheduling list before proceeding with further executions.

3.7 Agent language

3.7.1 Agent language (Assembler level)

To develop a mobile agent *Agent language* will be used. Language is tied to agent internal structure and support necessary operation for code mobility and message exchange. While writing the program for agent user should not be aware of hardware services presented on a given platform, but have common knowledge about all available services and what operations are allowed to do with the services (have list of services and available operations).

It is the responsibility of the platform to provide required service to the agent (perform measurement, IO operation) or to manifest an error if the service is not available on the given platform. All current variables are allowed to store only in registers of agent structure (Fig. 3.2).

We propose to use the following principle: every opcode should be 16 bits long, that will lead to more simple procedure during decoding of the executable on the platform side. Another principle is that `reg_0` is used as `accumulator`: the results of all computations, comparisons an messages received by the agent will be put to this register. This will lead to more compact code on the platform side.

The language supports the following groups of operations: arithmetic, control flow, code mobility, message exchange, access to hardware services. Every agent has 16 registers in its internal structure: 13 for holding 16-bit numerical values and 3 for holding character strings.

To achieve the following twofold goal: keep the length of the every opcode 16 bits as well provide a capability to directly write values to the 16-bit registers we propose to split every 16-bit register into `high` and `low` parts, that will be used in `ldl` and `ldh` commands. Pictorially we represent it as follows.

The following table reporesents registers of agent structure as well as their corresponding addresses.

Addressing registers of agent structure			
General purpose registers		**Character registers**	
Register	`rrrr`	Char Register	`rrrr`
`reg_0`	0000	`reg_str_0`	1101
`reg_1`	0001	`reg_str_1`	1110
`reg_2`	0010	`reg_str_2`	1111
`reg_3`	0011		
`reg_4`	0100		
`reg_5`	0101		
`reg_6`	0110		
`reg_7`	0111		
`reg_8`	1000		
`reg_9`	1001		
`reg_10`	1010		
`reg_11`	1011		
`reg_12`	1100		

3.7.1.1 Arithmetic operations of agent assembly language

Addition

Add the content of `reg_d` and `reg_r` (or `value`) and put the result into `reg_0`.

`add reg_d, reg_r`

Syntax	Operands	Program counter	Flags
`add reg_d, reg_r`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `reg_0` $\leq$ `reg_r` $\leq$ `reg_12`	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_d + reg_r`

16-bit opcode:

		reg_d	reg_r
0000	0011	dddd	rrrr

`add reg_r, value`

Syntax	Operands	Program counter	Flags
`add reg_r, value`	`reg_0` $\leq$ `reg_r` $\leq$ `reg_12`, `0x00` $\leq$ `value` $\leq$ `0xFF`	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_r + value`

16-bit opcode:

	reg_r	value	
0011	rrrr	vvvv	vvvv

Subtraction

Subtract `reg_s` (or `value`) from `reg_m` and put the result into `reg_0`.

`sub reg_m, reg_s`

Syntax	Operands	Program counter	Flags
`sub reg_m, reg_s`	`reg_0` $\leq$ `reg_m` $\leq$ `reg_12`, `reg_0` $\leq$ `reg_s` $\leq$ `reg_12`,	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_m - reg_s`

16-bit opcode:

		reg_m	reg_s
0000	0110	mmmm	ssss

`sub reg_m, value`

Syntax	Operands	Program counter	Flags
`sub reg_m, value`	`reg_0` $\leq$ `reg_m` $\leq$ `reg_12`, `0x00` $\leq$ `value` $\leq$ `0xFF`	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_m - value`

16-bit opcode:

	reg_m	value	
0110	mmmm	vvvv	vvvv

Division

Divide `reg1` by `reg2` (or value) and put the result into `reg_0`.

`div reg_d, reg_r`

Syntax	Operands	Program counter	Flags
`div reg_d, reg_r`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `reg_0` $\leq$ `reg_r` $\leq$ `reg_12`,	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_d / reg_r`

16-bit opcode:

		reg_d	reg_r
0000	1001	dddd	rrrr

`div reg_d, value`

Syntax	Operands	Program counter	Flags
`div reg_d, value`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `0x00` $\leq$ `value` $\leq$ `0xFF`	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_d / value`

16-bit opcode:

	reg_d	value	
1001	dddd	vvvv	vvvv

Multiplication

Multiply `reg1` and `reg2` (or `value`) and put the result into `reg_0`.

`mul reg_d, reg_r`

Syntax	Operands	Program counter	Flags
`mul reg_d, reg_r`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `reg_0` $\leq$ `reg_r` $\leq$ `reg_12`	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_d * reg_r`

16-bit opcode:

		reg_d	reg_r
0000	1100	dddd	rrrr

`mul reg1, value`

Syntax	Operands	Program counter	Flags
`mul reg_d, value`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `0x00` $\leq$ `value` $\leq$ `0xFF`	`PC = PC + 1`	`C`

Operation

`reg_0` $\leftarrow$ `reg_d * value`

16-bit opcode:

	reg_d	value	
1100	dddd	vvvv	vvvv

3.7.1.2 Control flow operations and comparison in agent assembly language

Jump if greater

Jump to `offset` in code segment of agent structure if the value of `reg_0`. is 1.

`jmpgr offset`

Syntax	Operands	Program counter
`jmpgr offset`	`-128` $\leq$ `offset` $\leq$ `+127`	`PC = PC + offset +1` if `reg_0 = 1`, `PC = PC + 1` otherwise

16-bit opcode:

		offset	
1111	0011	vvvv	vvvv

Jump if equal

Jump to `offset` in code segment of agent structure if the value of `reg_0` is 0.

`jmpeq offset`

Syntax	Operands	Program counter
`jmpeq offset`	`-128` $\leq$ `offset` $\leq$ `+127`	`PC = PC + offset +1` if `reg_0 = 0`, `PC = PC + 1` otherwise

16-bit opcode:

		offset	
1111	0110	vvvv	vvvv

Jump if less

Jump to `offset` in code segment of agent structure if the value of `reg_0` is -1.

`jmpls offset`

Syntax	Operands	Program counter
`jmpls offset`	`-128` $\leq$ `offset` $\leq$ `+127`	`PC = PC + offset +1` if `reg_0 = -1`, `PC = PC + 1` otherwise

16-bit opcode:

		offset	
1111	1100	vvvv	vvvv

Comparison

Compare `reg1` and `reg2` (or `value`).

`compare reg_d, reg_r`

Syntax	Operands	Program counter
`compare reg_d, reg_r`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `reg_0` $\leq$ `reg_r` $\leq$ `reg_12`	`PC = PC + 1`

Operation

`reg_0` $\leftarrow$ `1` if (reg_d - reg_r $>$ 0)
`reg_0` $\leftarrow$ `0` if (reg_d - reg_r $=$ 0)
`reg_0` $\leftarrow$ `-1` if (reg_d - reg_r $<$ 0)

16-bit opcode:

		reg_d	reg_r
0000	1010	dddd	rrrr

`compare reg_d, value`

Syntax	Operands	Program counter
`compare reg_d, value`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `0x00` $\leq$ `value` $\leq$ `0xFF`	`PC = PC + 1`

Operation

`reg_0` $\leftarrow$ `1` if (reg_d - value $>$ 0)
`reg_0` $\leftarrow$ `0` if (reg_d - value $=$ 0)
`reg_0` $\leftarrow$ `-1` if (reg_d - value $<$ 0)

16-bit opcode:

	reg_r	value	
1010	rrrr	vvvv	vvvv

3.7.1.3 Code mobility operations of agent assembly language

Move code

Move agent structure to platform that possess required service

`move service`

Syntax	Operands	Program counter
`move service`	`service_0` $\leq$ `service` $\leq$ `service_255`	`PC = PC + 1`

Operation

Serialize and transmit agent structure to the platform that possess required service

16-bit opcode:

	reg_r	service	
1111	0001	ssss	ssss

Clone code
Replicate agent structure on the given platform
`clone`

Syntax	Program counter
`clone`	`PC = PC + 1`

16-bit opcode:

1111	0010	0000	0000

Die
Destroy agent structure and free corresponding memory
`die`

16-bit opcode:

1111	0100	0000	0000

3.7.1.4 Message exchange

Send Message exchange between agents
`sendmsg reg, agent, platform`

Syntax	Operands	Program counter

`sendmsg reg, agent, platform`	`platform_0` $\leq$ `platform` $\leq$ `platform_3`
	`agent_0` $\leq$ `agent` $\leq$ `agent_3`
	`reg_0` $\leq$ `reg` $\leq$ `reg_12`
	`reg_str_0` $\leq$ `reg` $\leq$ `reg_str_2`
		`PC = PC + 1`

Operation
Send value of the register `reg` to the agent `aa` on the platform `pp`
16-bit opcode:

		register	agent	platform
1111	1000	rrrr	aa	pp

Receive
Pull message from `platform` to `register`.
`pullmsg reg`

Syntax	Operation	Program counter
`pullmsg`	`reg_0` $\leq$ `reg` $\leq$ `reg_12` `reg_str_0` $\leq$ `reg` $\leq$ `reg_str_2`	`PC = PC + 1`

Operation
`reg` $\leftarrow$ message
16-bit opcode:

			rrrr
1111	1010	0000	0000

3.7.1.5 Store, move and wait operations

Store

Store `value` in `h`-part of `reg_d`

`ldh reg_d, value`

Syntax	Operands	Program counter
`ldh reg_d, value`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `0x00` $\leq$ `value` $\leq$ `0xFF`	`PC = PC + 1`

Operation

`reg_d_h` $\leftarrow$ `value`

16-bit opcode:

	reg_d	value	
1101	dddd	vvvv	vvvv

Store `value` in `l`-part of `reg_d`

`ldl reg_d, value`

Syntax	Operands	Program counter
`ldl reg_d, value`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `0x00` $\leq$ `value` $\leq$ `0xFF`	`PC = PC + 1`

Operation

`reg_d_l` $\leftarrow$ `value`

16-bit opcode:

	reg_d	value	
0100	dddd	vvvv	vvvv

Push `char` value in the `str_reg`

`storecr reg_str, char`

Syntax	Operands	Program counter
`storecr reg_str, char`	`reg_str_0` $\leq$ `reg_str` $\leq$ `reg_str_2`	`PC = PC + 1`

Operation

`reg_str` $\Leftarrow$ `value`

16-bit opcode:

	reg_str	value	
1011	rrrr	vvvv	vvvv

Clear the `str_reg`

`clr reg_str`

Syntax	Operands	Program counter
`clr str_reg`	`reg_str_0` $\leq$ `reg_str` $\leq$ `reg_str_2`	`PC = PC + 1`

Operation

`clear str_reg`

16-bit opcode:

			rrrr
0000	0010	0000	rrrr

Move

Move value from reg_r to reg_d

`mv reg_d, reg_r`

Syntax	Operands	Program counter
`mv reg_d, reg_r`	`reg_0` $\leq$ `reg_d` $\leq$ `reg_12`, `reg_0` $\leq$ `reg_r` $\leq$ `reg_12`	`PC = PC + 1`

Operation

`reg_d` $\leftarrow$ `reg_r`

16-bit opcode:

		reg_d	reg_r
0000	1101	dddd	rrrr

Wait

Wait for ms

`wait delay_ms`

Syntax	Operands	Program counter
`wait delay_ms`	`0` $\leq$ `delay_ms` $\leq$ `0xFF`	`PC = PC + 1`

16-bit opcode:

		delay	
0000	0101	dddd	dddd

Assign priority value

Assign priority of the agent to value in range 0..3

`priority value`

Syntax	Operands	Program counter
`priority value`	`0` $\leq$ `value` $\leq$ `3`	`PC = PC + 1`

Operation

`priority` $\leftarrow$ `value`

16-bit opcode:

		priority	
0000	1000	pppp	pppp

3.7.1.6 Access to hardware services

Set

Set service to reg or value

`setservice service_id, reg`

Syntax	Operands	Program counter
`setservice service_id, reg`	`service_0` $\leq$ `service_id` $\leq$ `service_255`, `reg_0` $\leq$ `reg` $\leq$ `reg_12`	`PC = PC + 1`

16-bit opcode:

	reg	service_id	
0111	rrrr	ssss	ssss

Get
Put corresponding value from the service to the `reg_0`.

```
getservice service_id
```

Syntax	Operand	Program counter
`getservice service_id`	service_0 ≤ service_id ≤ service_255	`PC = PC + 1`

16-bit opcode:

		service_id	
0000	0111	ssss	ssss

3.8 Communication Architecture

The communication architecture is designed to support communication between nodes on the same development board as well as between boards.

3.8.1 Hardware

The communication on the board is carried out over two serial bus channels. One of them is to be used for a distributed control application running on nodes 0-3. Another bus is dedicated for code mobility between nodes 0-4.

Access to the bus is controlled by separate UART modules on each micro-controller. The bit rate is constrained by the maximum value of 2 Mbps according to the manual.

Node 4 functions as a gateway to another board. It is a bridge between the local and the wireless zigbee network.

3.8.2 Code Mobility

Code mobility between nodes includes local mobility on the same board and remote mobility between different boards. Executable agents generally have larger volume than control data. Sending at regular time intervals is not assumed, thus communication is aperiodic. A simple protocol based on message acknowledgment can be used.

There are two use cases: a) local mobility: destination is one of the nodes 0-3. b) remote mobility: destination is the gateway node 4. The gateway is to contain a zigbee stack implementation to enable access to the personal area network.

3.8.3 Addressing Scheme

Simple local addressing requires unique identifiers for each node. For remote communication, board addresses have to be compatible with the configuration of the zigbee network. Since, each node will have a static number of agent execution environments, the address has to contain its identifier as as well.

3.8.4 Communication Interface

The interface for accessing the communication system is given below in Figures 3.7 through 3.9.

```
struct frame{
        unsigned dst_node:4;            //destination node
        unsigned dst_board:4;           //destination board
        unsigned dst_agent:4;           //destination agent
        frame_id_t frame_id;            //source of a message
        unsigned frame_length:16;       //length of frame payload in
            bytes
        unsigned index:16;              //index for buffering
        struct frame *next_frame;       //next frame to be sent
        char *data;                     //payload
};
```

Figure 3.7: Message Structure

```
/**
   Function: recv_handler
             Reassembles a complete frame from received packets
   Parameters: msg_length       length of the current packet
               msg_body         payload of the current packet
 */
void recv_handler(uint8_t msg_length, uint8_t *msg_body);
```

Figure 3.8: Message Receiving

```
/**
   Function: send_message
             Sends message over communication interface
   Parameters: frame
               frame to be sent
   Returns: amount of sent packets

*/
uint8_t send_message(frame_t frame);
```

Figure 3.9: Message Sending

4 Implementation

4.1 Platform

4.1.1 Initialization

The platform initialization depends on the settings of a nodes makefile and on the data provided by the Application developer. The settings of a nodes makefile influence the set of hardware services available to the specific platform. When a node is linked to some hardware drivers supported by the platform e.g. bargraph, the according makefile will compile the platform code with a C preprocessor setting -DBARGRAPH. The platform will only support those drivers for which C preprocessor defines where made, by inspecting the defines and only assigning the driver function pointers supported. This allows for simple adaptation and extension of the platform by changing the drivers linked to the platform. Additionally by choosing this approach a smaller size of the executable is achieved.

```
# put platform specific hardware drivers to be supported by this node
OBJ_ESEL_MDEP_\$(MNAME)_y += protocol0 bargraph
```

Figure 4.1: Platform makefile

The makefile snippet from the figure shown above will result in a compilation of the platform with the setting -DPROTOCOL0 -DBARGRAPH.

The initialization code of the platform checks for these defines and only registers and initializes those drivers supported as shown in the figure below.

```
#ifdef BARGRAPH
        bargraph_init();
        platform.drivers.set_bargraph = set_bargraph;
#endif

#ifdef PROTOCOL0
         protocol_init(platform.id, recv_handler);
#endif
```

Figure 4.2: Platform drivers initialization

Additionally the agents to be executed need to be initialized on the platform by the application developer. This is achieved by providing C macros to the application developer which need to be filled with proper data. The C macros offered by the platform are shown in the figure below.

The AGENT_INIT macro needs to be defined by the application developer in order to instantiate an agent. As its input parameters it requires the agent id, agent priority and a binary string

```
#include "platform.h"

PLATFORM_CONFIGURATION()
{
  AGENTS_CONFIGURATION(){
      // agent id, agent prio, agent_code
      AGENT_INIT(0x02, 0x02, 00000111000000001111110
      00110100010000010100011110010000000000000001111
      1001111111011),
  },

  BOARD_ID(0x00)
};
```

Figure 4.3: Platform agent initialization

representing the agent code, which is delivered by the platform assembler tool (asm_agent). During platform initialization the binary string is converted to a binary representation in order to reduce the actual code size. Up to 4 agents can be initialized. All configured agents are assigned the status ready.

Additionally the application developer is able to initialize the board id, required for inter board communication via the BOARD_ID macro.

4.1.2 Execution

After successfull platform intialization the scheduler iterates through the configured agents i.e. those with status ready and forwards them to the execution layer to be executed via the method execute_agent shown in figure 4.4 on page 30.

The execution layer fetches the next opcode for the considered agent, decodes the according and finally executes the specific opcode. Eventually the program counter is increased and the next opcode gets executed. The execution of an agent is stopped as soon as the desired amount of opcode has been executed or if an agent was put to a different status than ready.

The decoding of the agent opcodes is performed by analyzing the 8 bit opcode header of the total 16 bit opcode as exemplarily shown in figure 4.5 on page 31.

Finally the opcode gets executed and agent configuration structure is updated as shown in figure 4.6 on page 31.

After all opcodes of an agent have been executed or the according agent was stopped the scheduler looks for the next agent with status ready to be executed.

4.1.3 Communication

The communication layer provides means to send and receive messages via the USART serial bus. The lower level implementation of the CSMA/CA protocol allows up to 15 bytes of payload to be transferred with a single message. Due to this limitation an upper layer protocol is introduced which allows greater messages to be exchanged between nodes.

```
uint8_t execute_agent(agent_t *agent, uint8_t opcode_size) {

  uint8_t opcodes_done = 0;

  while (opcodes_done < opcode_size) {
    //1. fetch next opcode
    uint16_t opcode = agent->code[agent->pc];

    //2. decode opcode
    opcode_t dec_opcode = decode_opcode(opcode);

    //3. execute opcode
    execute_opcode(agent, dec_opcode);

    //4. increase program counter
    if (agent->status == ready) {
      if (agent->pc < agent->code_len - 1 || agent->pc == 0xffff) {
        agent->pc += 1;
      } else {
        agent->status = stopped;
        break;
      }
      opcodes_done += 1;
    } else {
      return opcodes_done;
    }

  }

  return opcodes_done;
}
```

Figure 4.4: Platform agent execution

This protocol works with frames, where a frame is split into a sufficient amount of packets which are transmitted via the serial bus sequentially. In order to increase data throughput 2 types of packages were introduced: start packages and data packages.

The start packages always initialize the sending of a new frame and contain all the necessary data to succesfully address the destination of the packet and inform the receiver about specific frame settings i.e. frame id and frame length. Figure 4.1.3 on page 32 shows the layout of start packages.

The data packages are only used when a frame payload is greater than the 15 byte which can be sent within a single packet. These data packages identify the frame to which the belong and are able to transmit more payload data within a package. Figure 4.2 on page 32 shows the layout of data packages.

```
uint8_t nibble1 = NIBBLE1(opcode);
uint8_t nibble2 = NIBBLE2(opcode);

switch (nibble1) {
//0000
case 0:
  switch (nibble2) {

  //clr reg_str
  //0000 0010 0000 rrrr
  case 2:
    result.id = CLEAR;
    result.reg1 = NIBBLE4(opcode);
    break;

  //add reg_d, reg_r
  //0000 0011 dddd rrrr
  case 3:
     result.id = ADD_R;
     result.reg1 = NIBBLE3(opcode);
     result.reg2 = NIBBLE4(opcode);
    break;
```

Figure 4.5: Platform agent opcode decoding

```
case JMP_G:
  PRINTF("jmpgr offset:%d\n", opcode.value);
  if (agent->regs[REG_ACC]==1) {
    agent->pc = agent->pc + opcode.value;
  }
  break;

case JMP_E:
  PRINTF("jmpeq offset:%d\n", opcode.value);
  if (agent->regs[REG_ACC]==0){
    agent->pc = agent->pc + opcode.value;
  }
  break;

case JMP_L:
  PRINTF("jmpls offset:%d\n", opcode.value);
  if (agent->regs[REG_ACC]==-1){
    agent->pc = agent->pc + opcode.value;
  }
  break;
```

Figure 4.6: Platform agent opcode execution

dst_node	packet len
start_type	src board
src_node	frame id
packet id hi	
packet id low	
dst board	dst agent
frame length hi	
frame length low	
data	
...	
crc	

Table 4.1: Start Package

dst_node	packet len
start_type	src board
src_node	frame id
packet id hi	
packet id low	
data	
...	
crc	

Table 4.2: Data Package

The receiving platform of the communication reassembles the received packets into a single frame prior to informing the according agent about this event.

4.1.4 Code Mobility

In order to provide means for code mobility a localization service is introduced which allows identifying the hardware supported by a specific node. This is achieved by a static array storing the addresses of the nodes supporting a specific hardware as shown in figure 4.7 on page 32. This localization is only valid for the current ESE board and requires adaptation when porting the platform to another board.

```
uint8_t service_locations[MAX_SERVICE][MAX_NODES] = {
        {NODE0_ID, NODE1_ID, INVALID, INVALID},   //BARGRAPH
        {NODE1_ID, INVALID, INVALID, INVALID},    //THERMOMETER
        {NODE1_ID, INVALID, INVALID, INVALID},    //COOLER
        {NODE1_ID, INVALID, INVALID, INVALID},    //HEATER
        {NODE3_ID, INVALID, INVALID, INVALID},    //LED
        {NODE2_ID, INVALID, INVALID, INVALID},    //LCD
        {NODE0_ID, NODE1_ID, NODE2_ID, NODE3_ID} //BUTTONS
};
```

Figure 4.7: Service localization

After the address of the receiving board has been identified, the agent is serialized via the serialiaze_agent method. A frame containing a code mobility message is marked by a code mobility header and trailer (0x55).

When a complete frame has been received by a platform it checks whether this is a data message or code mobility message by inspecting the first(header) and last(trailer) byte of the received message. If a code mobility message was received the platform deserializes the agent, increments its program counter by 1 and instantiate this very agent within the platform so its considered for execution during the next scheduling round.

```
if (GET_MOBILITY_HEADER(current->data) == MOBILITY_BYTE &&
        GET_MOBILITY_END(current->data, current->frame_length - 1)
        == MOBILITY_BYTE){
  //code mobility message

  for (i = 0; i < AGENT_MAX; i++){
    if (platform.agents[i].status == stopped){
      agent_t agent = deserialize_agent(current->data);
      agent.id = id;
      agent.regs[REG_ACC] = 0;
      agent.pc+= 1;
      platform.agents[i] = agent;
      break;
    }
  }
```

Figure 4.8: Agent deserialization

In order to allow to distinguish whether the agent was moved or is the initiatior of the moving, the receving platform writes a 0 to accumulator of the received agent, whereas the sending platform writes the amount of sent packets to the accumulator of the sending agent.

4.2 Agent language assembler tool

Agent language assemler tool provides the means to convert agent program with `.ma` extension (which stands for "mobile agent") into binary code, that could be executed on the platform. After the compilation it generates two files: listing of the program for the debugging purposes and binary file with `.bin` entension.

For the implementation of Agent language assembler tool we use Python programming language.

The implementation of Agent language assembler tool is basically two pass assember, block diagram of which is shown in in the figure below.

This tool performs two passes over source file. In the first pass it reads the entire source file, looking for labels in the source code and identifying opcodes. All labels, mnemonics, operands are collected and are put to the symbol table. No instructions are assembled during this pass and symbol table contain labels, mnemonics and operands. As every instruction has fixed size it is clear enough how to determine the offset in the branching instruction. But we should point out that during the execution of each instruction programm counter increments by 1, so the `final_offset = relative_offset - 1`.

One of the most important issues is to correctly assemble labels of the branching instructions. There can be two problems with labels: *multiple-defined labels* and *invalid labels*. Example of multiple-defined labels error is as follows:

```
GOTO: add reg_0, reg_4
      ...
      ...
GOTO: setservice temp, reg_0
      ...
      ...
      pullmsg reg_4
      compare reg_4, reg_0
      jmpls GOTO
```

Figure 4.9: Multiple labels error

5 Validation

We have performed validation on different levels for different part of our project. The main components on which validation was performed are the agent language tool, virtualization platform, communication protocol, as well as the final validation of the overall system.

5.1 Agent language tool

Validation of the agent language is based on the unit testing methodology, since inputs and outputs for this part can be clearly and unambiguously defined.

5.2 Platform validation

The platform was validated by series of tests, based on the use cases in our specification (see sec.3 and Table 5.2).

Id	Test Description	Correct Condition	Result
1	Platform initialization	Platforms display status clear. The execution platform is running.	OK
2	Agent execution	Scheduler allocates execution resources.	OK
3	Messaging	Agents exchange messages.	OK
4	Agent migration	Sending an agent between platforms. Resuming execution.	OK
5	Load survivability	High data load induced. System works under stress.	OK

Table 5.1: Use Cases for Platform Validation

5.3 Validation of communication protocol

The communication protocol was the most elusive part to validate. Based on extensive testing in different use scenarios we could establish a safe margin for maximum of data transmission rate. For the validation of the protocol we used tools for serial communication between microcontroller and PC, such as `xxd`.

5.4 Overall system validation

In order to test the services provided by the aforementioned software components together we have implemented application for controlling temperature. First of all we test the interaction between agent language tool and execution platform by compiling the platform for x86 target applying our debug platform.

The second step was to perform functional validation of the platform by executing the application code. The code itself in C-format has been previously validated on microcontroller simulator for the target environment.

6 Results, future plans and expenditure of work

6.1 Platform

As the size of flash memory in μC is enough limited (128 kB), one of our goals was to reduce the size of firmware as small as possible, in order to keep the platform extendable for adding new features. Every platform that corresponds to a node on the ESE board supports only dedicated hardware drivers, this approach is achieved during compilation time.

6.2 Communication

We have implemented CSMA/CA protocol which allows up to 15 bytes of payload in a single message. To transfer payload more that 15 bytes we introduce frames which provide us with higher level of abstraction. We transfer agent code using the aforementioned frames. Before moving the code, the platform serialize agent and then send it as messages over the communication medium.

As a result we have communication protocol that supports transferring agent code and messages between agents from one board to another.

During the implementation phase we spent two times more then it was planned trying to make board to board communication using zigbee and tiny os.

What we have figured out is the following: it is possible to transfer the data from one board to another using tinyOS and zigbee stack, it works great. The main pitfall here is the communication between zigbee node and serial bus. TinyOS is a modular software and provides means to access hardware by adding components to the kernel and access the hardware by means of interfaces of this components.

For accessing the serial port the following components could be used: `UARTByte, UARTStream, ...` which provide different level of abstraction: from sending a single byte to arrays via UART, but none of this components send anything to the serial in practice. TinyOS website provides information that in most of the cases components are platform specific, so we come to conclusion that probably serial communication using tinyOS does not work due to incompatibility of the platform.

6.3 Agent language

We have developed the Agent language, defined executable opcodes and implemented assembler tool for converting agent program into binary code.

The main decision here was to keep the size of all opcodes constant, so that it lead to more simple assemble procedure and more simple decoding on the platform side.

As it is of prior importance to provide user with the feedback about the program written, we manifest all the possible errors in the agent program. Agent language tool generates listing file that shows the results of assemble and binary executable code of agent that could be inserted in a desired platform.

According to the specification, for mobile agent developer it is of no importance by what means to access the hardware. She could write, for instance,

```
getservice temp
```

and while executing this command platform will access temperature sensors on the ESE board via I2C interface, compute average of all three sensors and put the resulting value to the `reg_0` of agent. By using this approach we go one level up in the abstraction ladder hierarchy.

But there is always space for improvement. As one of the future goals one can consider to implement high level language, that will be compiled into the designed assembler language.

6.4 Drivers

While some drivers have already been implemented and provided with ESE library, to use all the required drivers in our project it is required to attach the drivers to the platform.

During this step some side condition has occurred: for example, because of simulataneous using of the same timer by two drivers the platform has been restarted.

6.5 Time expenditure

We started our work on October, 29th and end three month later, on 29th of January. Because of the problems with zigbee implementation and porting of the protocol to bus1, memory leaks, non-reproducable the whole project needed longer time then estimated.

According to the table of planned and real dates the implementation phase last one month longer then it was planned.

It is necessary to take into account that essentially from that month 2 weeks was Christmas holidays and due to the illness of our colleague [J] it is required more time that it was supposed to.

It is worth to mention that despite the force-majeure we succeeded to complete our project on time.

Planned and real dates			
	planned dates	real end date	
preliminary tasks (assigning roles, gathering data, setting up environment)	29.10.2012 - 10.11.2012	8.11.2012	-2 days
defining requirements, and specification (ALAT, communication architecture, platform)	11.11.2012 - 30.11.2012	29.11.2012	-1 days
presentation for workshop 1	30.11.2012 - 5.12.2012	5.12.2012	0 days
implementation phase (total)	6.12.2012 - 21.12.2012	21.01.2013	+30 days
implementation phase (Agent language tool)	6.12.2012 - 12.12.2012	17.01.2013	+5 days
implementation phase (communication protocol)	6.12.2012 - 11.12.2012	10.01.2013	-1 days
implementation phase (refinement of drivers)	03.12.2012 - 07.12.2012	07.12.2012	-1 day
implementation phase (platform, total)	08.12.2012 - 27.12.2012	23.01.2013	+26 days
implementation phase (platform, scheduling)	08.12.2012 - 15.12.2012	15.12.2012	0 days
implementation phase (platform, hardware middle layer)	15.12.2012 - 21.12.2012	10.01.2013	+20 days
implementation phase (platform, communication layer)	22.12.2012 - 27.12.2012	23.01.2013	+26 days
validation phase	13.01.2013 - 24.01.2013	26.01.2013	+2 days
workshop 2 presentation	23.01.2013 - 29.01.2013	29.01.2013	0 days
lab protocol documentation	12.11.2012 - 23.01.2013	29.01.2013	+5 days

Spent time per workpackage			
workpackage	planned time	spent time	difference
D1.1 Requirements and Specification	60	58	-2
D1.2 Presentation for Workshop 1	20	24	+4
D1.3 Presentation for Workshop 2	20	20	0
D1.4 Lab protocol	80	90	+10
D2.1 driver implementation	50	40	-10
D3.1 agent language tool	40	43	+3
D4.1 CSMA/CA communication protocol	60	57	-3
D5.1 Platform. Scheduler	20	14	-6
D5.2 Platform. Execution layer	30	43	+13
D5.3 Platform. Communication layer	70	113	+43

7 Conclusion

In our project we have focused on implementing and validating code mobility environment on the ESE board. As a testing application we have developed a distributed control application, where all actions are executed by agents, that reside on the each platform of the ESE board, Testing application demonstrates all the capabilities of developed environment: after the start of the application mobile agent is transfered from node 0 to node 1, where then this agent fetch temperature value of heating element from the platform, compares it with setpoint and adjust it by setting cooling service. Meanwhile the agents on the nodes 2 and 3 are responsible for accessing TFT display and led matrix.

The testing application illustrates the following implementation results: the ability to transfer code from one node to another including state and values of all registers, transferring messages between agents, execute several agents on the current platform.

For project management we used e-mail, personal meetings and skype to get the work done. We installed and deployed to the public cloud Heroku open source bugtracker (redmine). This web application could be used for composing Gantt diagram, defining and assigning tasks and documenting current results of the project, as well as planning future work. Redmine is great helper for project administrator since it is possible to assign roles and tasks for each members and monitor the current working process online.

On the whole, the course was extremely helpful for various reasons. First of all, we master our skills in development of embedded system applications and communication between nodes. We get acquainted ourselves with new code mobility concept, implemented simple assembler-like language, sufficient for writing useful mobile agents. The most hard and time consuming part of the project was implementing virtualization platform that support scheduler, provides access to hardware services, and capable of transferring agents between platforms.

In the related work we used for inspiration, code mobility applications were implemented mostly in Java. As it is known, Java offers many high level abstractions with its rich set of library functions. By moving to the domain of embedded systems, many otherwise hidden nuts and bolts of low-level programming become apparent. Among these are the issues such as race conditions, bugs in interrupt programming, non-reproducible Heisenberg bugs, memory leaks, &c. Although Java is progressively gaining in popularity even in the embedded community, a programmer could never experience the thrill associated with these issues by remaining completely on the higher level of abstraction.

It is worth to mention that working in a group was extremely helpful for mastering project management skills. Defining plan and meeting the deadlines is worth to practice for real day-to-day experience. Despite all the difficulties with the third guy [J], we succeeded to get work done in time.

The workshop days was of great advantage for us to master our presentation skills as well as to discuss possible fallacies and pitfalls of our project and intended solution.

Bibliography

[1] E. Bartocci, D. Cacciagrano, N. Cannata, F. Corradini, E. Merelli, L. Milanesi, and P. Romano. An agent-based multilayer architecture for bioinformatics grids. *EEE transactions on Nanobioscience*, 6(2):142–148, 2007.

[2] Jürgen Galler. A modular software package for the embedded systems engineering board. Bachelor thesis, Faculty of Informatics, A-1040 Wien Karlsplatz 13, 2011.

[3] Flavio De Paoli, Antonella Di Stefano, Andrea Omicini, and Corrado Santoro, editors. *Proceedings of the 7th WOA 2006 Workshop, From Objects to Agents (Dagli Oggetti Agli Agenti), Catania, Italy, September 26-27, 2006*, volume 204 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[4] Paolo Romano, Ezio Bartocci, Guglielmo Bertolini, Flavio De Paoli, Domenico Marra, Giancarlo Mauri, Emanuela Merelli, and Luciano Milanesi. Biowep: a workflow enactment portal for bioinformatics applications. *BMC Bioinformatics*, 8(S-1), 2007.

A Source code

A.1 Platform

A.1.1 platform.h

```
1  /*
2   * platform.h
3   *
4   * Created on: Dec 8, 2012
5   * Author: igor
6   */
7
8  #ifndef PLATFORM_H_
9  #define PLATFORM_H_
10
11 #include <stdio.h>
12 #include <string.h>
13 #include <stdlib.h>
14
15 #include "global.h"
16 #include "bargraph.h"
17 #include "thermometer.h"
18 #include "cooler.h"
19 #include "heater.h"
20 #include "DISPLAY.h"
21 #include "protocol0.h"
22 #include "ledmatrix.h"
23 #include "pushbutton.h"
24
25 #define AGENT_MAX 4
26 #define OPCODE_LEN 16
27 #define STR_REG_MAX 3
28 #define REG_MAX 13
29
30 #define MAX_SERVICE 7
31 #define MAX_NODES 4
32 #define INVALID 0xff
33
34 #define NODE0_ID 0x04
35 #define NODE1_ID 0x03
36 #define NODE2_ID 0x02
37 #define NODE3_ID 0x01
38
39 #define PLATFORM_CONFIGURATION() \
40 platform_config_t platform_config =
41
42 #define AGENTS_CONFIGURATION() \
43 .agents_conf =
44
45 #define AGENT_INIT(agentid, agentprio, agentcode) \
```

```
46 { .id=agentid, \
47 .active = 1, \
48 .prio = agentprio, \
49 .code = #agentcode }
50
51 #define PLATFORM_ID(id) \
52 .platform_id = id
53
54 #define BOARD_ID(id) \
55 .board_id = id
56
57 typedef struct {
58
59   uint8_t id;
60   uint8_t active;
61   uint8_t prio;
62   char *code;
63
64 } agent_config_t;
65
66 typedef struct {
67   agent_config_t agents_conf[AGENT_MAX];
68   //uint8_t platform_id;
69   uint8_t board_id;
70   uint8_t frame_id;
71 } platform_config_t;
72
73 extern platform_config_t platform_config;
74
75 typedef struct {
76
77   void (*set_bargraph)(uint8_t value);
78
79   uint32_t (*clk_get_time)(void);
80
81   void (*set_cooler)(uint8_t duty_cycle);
82
83   void (*DISPLAY_string)(uint8_t x, uint8_t y, uint16_t font_color, ↩
       uint16_t bg_color, uint8_t pixel_size, char *string);
84   void (*DISPLAY_drawBg)(uint16_t rgb);
85
86   void (*heater_set)(uint8_t duty_cycle);
87
88   void(*button0_callback)(void);
89   void(*button1_callback)(void);
90
91   uint16_t (*therm_get_temp)(uint8_t name);
92
93   void (*dotmatrix_send)(char *data);
94
95 } drivers_t;
96
97 enum agent_status {
98   stopped, ready, blocked
99 };
100 typedef enum agent_status agent_status_t;
101
102 #define REG_ACC 0
```

```
103 #define OVERFLOW 32
104 #define OVERFLOW_MASK 0x80000000
105 #define ERROR 0x000000FF
106 #define SET_ERROR(flag, errno)  (flag |= (errno & ERROR))
107
108 typedef struct {
109
110   uint8_t id;
111   uint8_t priority;
112   agent_status_t status;
113
114   uint32_t status_flag;
115   uint16_t pc;
116
117   int16_t regs[REG_MAX];
118
119   uint16_t code_len;
120   uint16_t regstr_len [STR_REG_MAX];
121
122   uint16_t* code;
123
124   char** reg_str;
125
126   volatile char* rec_msg_content;
127   volatile uint16_t rec_msg_len;
128
129   volatile uint8_t sem;
130 } agent_t;
131
132 typedef struct {
133   volatile agent_t agents[4];
134   drivers_t drivers;
135   uint8_t id;
136 } platform_t;
137
138 extern volatile platform_t platform;
139 extern uint8_t service_locations[MAX_SERVICE][MAX_NODES];
140 extern volatile uint8_t button0_pressed;
141 extern volatile uint8_t button1_pressed;
142
143 void init_drivers(void);
144 void init_agents(void);
145 void reset_agent(uint8_t id);
146 uint8_t clone_agent(agent_t *agent);
147 void platform_init(void);
148 void run_platform(void);
149
150 void buttoncallback0(void);
151 void buttoncallback1(void);
152
153
154 #endif /* PLATFORM_H_ */
```

A.1.2 platform.c

```
1 /*
2 * platform.c
```

```
3  *
4  * Created on: Dec 8, 2012
5  * Author: igor
6  */
7
8  #include "platform.h"
9  #include "hw_layer.h"
10 #include "scheduler.h"
11 #include "comm_layer.h"
12
13 #include "util/delay.h"
14
15 volatile platform_t platform;
16 volatile uint8_t button0_pressed;
17 volatile uint8_t button1_pressed;
18
19 uint8_t service_locations[MAX_SERVICE][MAX_NODES] = {
20       {NODE0_ID, NODE1_ID, INVALID, INVALID},   //BARGRAPH
21       {NODE1_ID, INVALID, INVALID, INVALID},    //THERMOMETER
22       {NODE1_ID, INVALID, INVALID, INVALID},    //COOLER
23       {NODE1_ID, INVALID, INVALID, INVALID},    //HEATER
24       {NODE3_ID, INVALID, INVALID, INVALID},    //LED
25       {NODE2_ID, INVALID, INVALID, INVALID},    //LCD
26       {NODE0_ID, NODE1_ID, NODE2_ID, NODE3_ID} //BUTTONS
27 };
28
29 void init_drivers(void){
30
31 #ifdef BARGRAPH
32   bargraph_init();
33   platform.drivers.set_bargraph = set_bargraph;
34 #endif
35
36 #ifdef PROTOCOL0
37    protocol_init(platform.id, recv_handler);
38 #endif
39
40 #ifdef TIMER2
41
42 #endif
43
44 #ifdef CLOCK
45
46 #endif
47
48 #ifdef HEATER
49   heater_init();
50   platform.drivers.heater_set = heater_set;
51 #endif
52
53
54 #ifdef DISPLAY
55   DISPLAY_init();
56   platform.drivers.DISPLAY_string = DISPLAY_string;
57   platform.drivers.DISPLAY_drawBg = DISPLAY_drawBg;
58 #endif
59
60 #ifdef THERMOMETER
```

```
61    therm_init();
62    platform.drivers.therm_get_temp = therm_get_temp;
63  #endif
64
65
66  #ifdef PUSHBUTTON
67    platform.drivers.button0_callback = buttoncallback0;
68    platform.drivers.button1_callback = buttoncallback1;
69    init_pushbutton0(platform.drivers.button0_callback);
70    init_pushbutton1(platform.drivers.button1_callback);
71    button0_pressed = 0;
72    button1_pressed = 0;
73  #endif
74
75  #ifdef COOLER
76    init_cooler();
77    platform.drivers.set_cooler = set_cooler;
78    //set_cooler(50);
79  #endif
80
81  #ifdef LEDMATRIX
82    init_dotmatrix();
83    platform.drivers.dotmatrix_send = dotmatrix_send;
84
85  #endif
86
87  }
88
89
90  void buttoncallback0(void){
91    button0_pressed = 1;
92  }
93
94  void buttoncallback1(void){
95    button1_pressed = 1;
96  }
97
98  void init_agents(){
99
100   uint8_t i = 0;
101
102   for (i=0; i < AGENT_MAX; i++) {
103
104     if (platform_config.agents_conf[i].active == 1) {
105       uint8_t id = platform_config.agents_conf[i].id;
106       platform.agents[id].id = id;
107       platform.agents[id].status = ready;
108       platform.agents[id].priority = platform_config.agents_conf[i].prio;
109
110       if (platform.agents[id].reg_str == 0) {
111         platform.agents[id].reg_str = (char**) malloc(STR_REG_MAX * ↩
              sizeof(char*));
112         platform.agents[id].reg_str[0] = (char*) malloc(1);
113         platform.agents[id].reg_str[1] = (char*) malloc(1);
114         platform.agents[id].reg_str[2] = (char*) malloc(1);
115       }
116
117       size_t len = strlen(platform_config.agents_conf[i].code);
```

```
118         platform.agents[id].code = (uint16_t*) malloc((len / OPCODE_LEN) * ↩
                sizeof(uint16_t));
119         uint16_t ind = 0;
120         char opcode[OPCODE_LEN];
121         while (ind < len / OPCODE_LEN) {
122           strncpy(opcode,(platform_config.agents_conf[i].code + (ind * ↩
                OPCODE_LEN)), OPCODE_LEN);
123           platform.agents[id].code[ind] = strtol(opcode, NULL, 2);
124           ind += 1;
125         }
126         platform.agents[id].code_len = ind;
127       }
128     }
129
130 }
131
132 void reset_agent(uint8_t id){
133
134   agent_t *agent = (agent_t*)&(platform.agents[id]);
135   agent->id = 0;
136   agent->status = stopped;
137   agent->priority = 0;
138   memset(agent->regs, 0, REG_MAX * sizeof(int16_t));
139
140   if (agent->reg_str != 0){
141
142     if (agent->reg_str[0] != 0) {
143       free(agent->reg_str[0]);
144       agent->reg_str[0] = 0;
145     }
146
147     if (agent->reg_str[1] != 0) {
148       free(agent->reg_str[1]);
149       agent->reg_str[1] = 0;
150     }
151
152     if (agent->reg_str[2] != 0) {
153       free(agent->reg_str[2]);
154       agent->reg_str[2] = 0;
155     }
156
157     free(agent->reg_str);
158     agent->reg_str = 0;
159
160   }
161
162   memset(agent->regstr_len, 0, sizeof(uint16_t) * STR_REG_MAX);
163
164   if (agent->code != 0){
165     free(agent->code);
166     agent->code = 0;
167   }
168   agent->code_len = 0;
169   agent->pc = 0;
170   agent->status_flag = 0;
171
172   if (agent->rec_msg_content != 0) {
173     free(agent->rec_msg_content);
```

```
174       agent->rec_msg_content = 0;
175       agent->rec_msg_len = 0;
176     }
177
178   }
179
180   uint8_t clone_agent(agent_t *agent){
181     uint8_t i = 0;
182     uint8_t result = 1;
183
184     for (i = 0; i < AGENT_MAX; i++){
185       if (platform.agents[i].status == stopped){
186         reset_agent(i);
187
188         platform.agents[i].id = i;
189         platform.agents[i].status = ready;
190         platform.agents[i].priority = agent->priority;
191
192         if (platform.agents[i].reg_str == 0) {
193           platform.agents[i].reg_str = (char**) malloc(STR_REG_MAX * sizeof(char*));
194           platform.agents[i].reg_str[0] = (char*) malloc(agent->regstr_len[0]);
195           platform.agents[i].reg_str[1] = (char*) malloc(agent->regstr_len[1]);
196           platform.agents[i].reg_str[2] = (char*) malloc(agent->regstr_len[2]);
197         }
198
199         memcpy(platform.agents[i].reg_str[0], agent->reg_str[0], agent->regstr_len[0]);
200         platform.agents[i].regstr_len[0] = agent->regstr_len[0];
201
202         memcpy(platform.agents[i].reg_str[1], agent->reg_str[1], agent->regstr_len[1]);
203         platform.agents[i].regstr_len[1] = agent->regstr_len[1];
204
205         memcpy(platform.agents[i].reg_str[2], agent->reg_str[2], agent->regstr_len[2]);
206         platform.agents[i].regstr_len[2] = agent->regstr_len[2];
207
208         platform.agents[i].code_len = agent->code_len;
209         platform.agents[i].code = (uint16_t*) malloc( agent->code_len * sizeof(uint16_t));
210         memcpy(platform.agents[i].code, agent->code, agent->code_len * sizeof(uint16_t));
211
212         platform.agents[i].pc = agent->pc + 1;
213         platform.agents[i].status_flag = agent->status_flag;
214
215         memcpy(platform.agents[i].regs, agent->regs, REG_MAX * sizeof(int16_t));
216         platform.agents[i].regs[REG_ACC] = 0;
217         result = 0;
218
219         if ( agent->rec_msg_content!= NULL){
220           memcpy(platform.agents[i].rec_msg_content, agent->rec_msg_content, agent->rec_msg_len);
```

```
221           platform.agents[i].rec_msg_len = agent->rec_msg_len;
222         }
223         break;
224       }
225     }
226
227     return result;
228 }
229
230 /*
231 * Initialize the platform with the provided configuration and
232 * setup all requested drivers.
233  */
234 void platform_init(void) {
235
236 #ifdef NODE0
237     platform.id = NODE0_ID;
238 #elif NODE1
239     platform.id = NODE1_ID;
240 #elif NODE2
241     platform.id = NODE2_ID;
242 #elif NODE3
243     platform.id = NODE3_ID;
244 #endif
245
246     init_drivers();
247     init_agents();
248     sei();
249 }
250
251 void run_platform(void) {
252
253     schedule_next();
254 }
255
256 int main(void) {
257
258
259     platform_init();
260
261
262     while (1){
263       run_platform();
264     }
265
266     return 1;
267 }
```

A.1.3 scheduler.h

```
1 /*
2 * scheduler.h
3 *
4 * Created on: Dec 10, 2012
5 * Author: igor
6 */
7
```

```
8  #ifndef SCHEDULER_H_
9  #define SCHEDULER_H_
10
11 #include "platform.h"
12 #include "exe_layer.h"
13
14 extern volatile uint8_t last_agent;
15
16 void schedule_next(void);
17
18 #endif
```

A.1.4 scheduler.c

```
1  /*
2   * scheduler.c
3   *
4   * Created on: Dec 10, 2012
5   * Author: igor
6   */
7  #include "scheduler.h"
8
9  volatile uint8_t next_agent_id = 0;
10
11 void schedule_next(void) {
12
13   uint8_t first_agent_id = next_agent_id;
14   uint8_t all_blocked = 0;
15
16   // consider next agent
17   agent_t *next_agent = (agent_t*) &(platform.agents[next_agent_id]);
18
19   // search for an unblocked agent
20   while (next_agent->status != ready) {
21     if (next_agent_id < 3) {
22       next_agent_id += 1;
23     } else {
24       next_agent_id = 0;
25     }
26
27     if (first_agent_id == next_agent_id){
28       all_blocked = 1;
29       break;
30     }
31
32     next_agent = (agent_t*) &(platform.agents[next_agent_id]);
33   }
34
35   if (all_blocked){
36     return;
37   }
38
39   // execute the next opcodes for the agent
40   uint8_t opcodes_done = execute_agent(next_agent, next_agent->priority +↩
         1);
41
42   // schedule next agent
```

```
43    next_agent_id += 1;
44  }
```

A.1.5 hw_layer.h

```
1  /*
2  * hw_layer.h
3  *
4  * Created on: Dec 9, 2012
5  * Author: igor
6  */
7
8  #ifndef HW_LAYER_H_
9  #define HW_LAYER_H_
10
11 struct block_dev_access_data{
12 uint8_t id;
13 struct block_dev_acess_data* next;
14 };
15
16 typedef struct block_dev_access_data block_dev_access_data_t;
17
18 typedef struct block_dev_access_list{
19 uint8_t size;
20 block_dev_access_data_t* first;
21 block_dev_access_data_t* last;
22 } block_dev_access_list_t;
23
24 #endif /* HW_LAYER_H_ */
```

A.1.6 exe_layer.h

```
1  /*
2  * exe_layer.h
3  *
4  * Created on: Dec 10, 2012
5  * Author: igor
6  */
7
8  #ifndef EXE_LAYER_H_
9  #define EXE_LAYER_H_
10
11 #include "platform.h"
12
13 //functions
14 #define SETSERVICE 0
15 #define GETSERVICE 1
16 #define STORE_L 2
17 #define ADD_R 3
18 #define ADD_V 4
19 #define SUB_R 5
20 #define SUB_V 6
21 #define DIV_R 7
22 #define DIV_V 8
23 #define MUL_R 9
24 #define MUL_V 10
```

```
25 #define JMP_G 11
26 #define JMP_E 12
27 #define JMP_L 13
28 #define CMP_R 14
29 #define CMP_V 15
30 #define MOVE  16
31 #define CLONE 17
32 #define DIE   18
33 #define SEND  19
34 #define RECV  20
35 #define STORE_H 21
36 #define STORE_C 22
37 #define MV      23
38 #define WAIT    24
39 #define PRIO    25
40 #define CLEAR 26
41 #define CONV  27
42
43 //services
44 #define SERVICE_BARGRAPH 0
45 #define SERVICE_THERMOMETER 1
46 #define SERVICE_COOLER 2
47 #define SERVICE_HEATER 3
48 #define SERVICE_LED 4
49 #define SERVICE_LCD 5
50 #define SERVICE_BUTTON0 6
51 #define SERVICE_BUTTON1 7
52
53 #define ERROR_NO_SERVICE_PRESENT 1
54
55 #define MAX_LCD_ROWS 8
56
57 #define N1_MASK 0xF000
58 #define N2_MASK 0x0F00
59 #define N3_MASK 0x00F0
60 #define N4_MASK 0x000F
61 #define B1_MASK 0xFF00
62 #define B2_MASK 0x00FF
63 #define HN4_MASK 0x000C
64 #define LN4_MASK 0x0003
65
66 #define NIBBLE1(opcode) ((opcode & N1_MASK) >> 12)
67 #define NIBBLE2(opcode) ((opcode & N2_MASK) >> 8)
68 #define NIBBLE3(opcode) ((opcode & N3_MASK) >> 4)
69 #define NIBBLE4(opcode)  (opcode & N4_MASK)
70 #define BYTE1(opcode)   ((opcode & B1_MASK) >> 8)
71 #define BYTE2(opcode)    (opcode & B2_MASK)
72 #define HNIBBLE4(opcode)((opcode & HN4_MASK) >> 2)
73 #define LNIBBLE4(opcode) (opcode & LN4_MASK)
74
75 #define BYTE_SIGN 0x80
76 #define NEG_SIGN 0xff00
77 #define POS_SIGN 0x00ff
78
79 #define TEMP_MASK 0xfff8
80 #define REG_STR_MASK 0x07
81
82 typedef struct {
```

```
83  uint8_t id;
84  uint8_t reg1;
85  uint8_t reg2;
86  int16_t value;
87  uint8_t node_id;
88  uint8_t agent_id;
89  } opcode_t;
90
91  int16_t get_signed_value(uint8_t value);
92  uint8_t execute_agent(agent_t *agent, uint8_t opcode_size);
93  opcode_t decode_opcode(uint16_t opcode);
94  void execute_opcode(agent_t *agent, opcode_t opcode);
95
96  #ifdef X86
97  #define PRINTF printf
98  #else
99  #define PRINTF //
100 #endif
101
102
103 #endif
```

A.1.7 exe_layer.c

```
1   /*
2    * exe_layer.c
3    *
4    * Created on: Dec 10, 2012
5    * Author: igor
6    */
7   #include "exe_layer.h"
8   #include "comm_layer.h"
9
10  uint8_t execute_agent(agent_t *agent, uint8_t opcode_size) {
11
12    uint8_t opcodes_done = 0;
13
14    while (opcodes_done < opcode_size) {
15      //1. fetch next opcode
16      uint16_t opcode = agent->code[agent->pc];
17
18      //2. decode opcode
19      opcode_t dec_opcode = decode_opcode(opcode);
20
21      //3. execute opcode
22      execute_opcode(agent, dec_opcode);
23
24      //4. increase program counter
25      if (agent->status == ready) {
26        if (agent->pc < agent->code_len - 1 || agent->pc == 0xffff) {
27          agent->pc += 1;
28        } else {
29          agent->status = stopped;
30          break;
31        }
32        opcodes_done += 1;
33      } else {
```

```
34        return opcodes_done;
35      }
36
37    }
38
39    return opcodes_done;
40  }
41
42  opcode_t decode_opcode(uint16_t opcode) {
43
44    opcode_t result;
45    uint8_t temp;
46
47    uint8_t nibble1 = NIBBLE1(opcode);
48    uint8_t nibble2 = NIBBLE2(opcode);
49
50    switch (nibble1) {
51    //0000
52    case 0:
53      switch (nibble2) {
54
55      //clr reg_str
56      //0000 0010 0000 rrrr
57      case 2:
58        result.id = CLEAR;
59        result.reg1 = NIBBLE4(opcode);
60        break;
61
62      //add reg_d, reg_r
63      //0000 0011 dddd rrrr
64      case 3:
65         result.id = ADD_R;
66         result.reg1 = NIBBLE3(opcode);
67         result.reg2 = NIBBLE4(opcode);
68        break;
69
70        //wait delay_ms
71        //0000 0101 dddd dddd
72      case 5:
73        result.id = WAIT;
74        result.value = BYTE2(opcode);
75        break;
76
77        //sub reg_m, reg_s
78        //0000 0110 mmmm ssss
79      case 6:
80        result.id = SUB_R;
81        result.reg1 = NIBBLE3(opcode);
82        result.reg2 = NIBBLE4(opcode);
83        break;
84
85        //getservice
86        //0000 0111 dddd dddd
87      case 7:
88        result.id = GETSERVICE;
89        result.value = BYTE2(opcode);
90        break;
91
```

```
92        //priority value
93        //0000 1000 pppp pppp
94      case 8:
95        result.id = PRIO;
96        result.value = BYTE2(opcode);
97        break;
98
99        //div reg_d, reg_r
100       //0000 1001 dddd rrrr
101     case 9:
102       result.id = DIV_R;
103       result.reg1 = NIBBLE3(opcode);
104       result.reg2 = NIBBLE4(opcode);
105       break;
106
107       //compare reg_d, reg_r
108       //0000 1010 dddd rrrr
109     case 10:
110       result.id = CMP_R;
111       result.reg1 = NIBBLE3(opcode);
112       result.reg2 = NIBBLE4(opcode);
113       break;
114
115       //mul reg_d, reg_r
116       //0000 1100 dddd rrrr
117     case 12:
118       result.id = MUL_R;
119       result.reg1 = NIBBLE3(opcode);
120       result.reg2 = NIBBLE4(opcode);
121       break;
122
123       //mv reg_d, reg_r
124       //0000 1101 dddd rrrr
125     case 13:
126       result.id = MV;
127       result.reg1 = NIBBLE3(opcode);
128       result.reg2 = NIBBLE4(opcode);
129       break;
130
131   }
132   break;
133   //1111
134 case 15:
135   switch (nibble2) {
136   //move service
137   //1111 0001 ssss ssss
138   case 1:
139     result.id = MOVE;
140     result.value = BYTE2(opcode);
141     break;
142
143     //clone
144     //1111 0010 0000 0000
145   case 2:
146     result.id = CLONE;
147     break;
148
149     //jmpgr offset
```

```
150       //1111 0011 vvvv vvvv
151     case 3:
152       result.id = JMP_G;
153       result.value = get_signed_value(BYTE2(opcode));
154       break;
155
156       //die
157       //1111 0100 0000 0000
158     case 4:
159       result.id = DIE;
160       break;
161
162       //jmpeq offset
163       //1111 0110 vvvv vvvv
164     case 6:
165       result.id = JMP_E;
166       result.value = get_signed_value(BYTE2(opcode));
167       break;
168
169       //sendmsg reg_d, agent, platform
170       //1111 1000 rrrr aa pp
171     case 8:
172       result.id = SEND;
173       result.reg1 = NIBBLE3(opcode);
174       result.agent_id = HNIBBLE4(opcode);
175       result.node_id = LNIBBLE4(opcode);
176       break;
177
178       //pullmsg
179       //1111 1010 0000 rrrr
180     case 10:
181       result.id = RECV;
182       result.reg1 = NIBBLE4(opcode);
183       break;
184
185       //jmpls offset
186       //1111 1100 vvvv vvvv
187     case 12:
188       result.id = JMP_L;
189       result.value = get_signed_value(BYTE2(opcode));
190       break;
191
192       //convert reg_str, reg_m
193       //1111 1111 ssss mmmm
194     case 15:
195       result.id = CONV;
196       result.reg1 = NIBBLE3(opcode);
197       result.reg2 = NIBBLE4(opcode);
198       break;
199
200     }
201     break;
202
203     //add reg_r, value
204     //0011 rrrr vvvv vvvv
205   case 3:
206     result.id = ADD_V;
207     result.reg1 = NIBBLE2(opcode);
```

```
208       result.value = get_signed_value(BYTE2(opcode));
209       break;
210
211       //ldh reg_d, value
212       //0100 dddd vvvv vvvv
213     case 4:
214       result.id = STORE_L;
215       result.reg1 = NIBBLE2(opcode);
216       result.value = get_signed_value(BYTE2(opcode));
217       break;
218
219
220       //sub reg_m, value
221       //0110 mmmm vvvv vvvv
222     case 6:
223       result.id = SUB_V;
224       result.reg1 = NIBBLE2(opcode);
225       result.value = get_signed_value(BYTE2(opcode));
226       break;
227
228       //setservice service, reg
229       //0111 rrrr ssss ssss
230     case 7:
231       result.id = SETSERVICE;
232       result.reg1 = NIBBLE2(opcode);
233       result.value = get_signed_value(BYTE2(opcode));
234       break;
235
236       //div reg_d, value
237       //1001 dddd vvvv vvvv
238     case 9:
239       result.id = DIV_V;
240       result.reg1 = NIBBLE2(opcode);
241       result.value = get_signed_value(BYTE2(opcode));
242       break;
243
244       //compare reg_d, value
245       //1010 rrrr vvvv vvvv
246     case 10:
247       result.id=CMP_V;
248       result.reg1=NIBBLE2(opcode);
249       result.value = get_signed_value(BYTE2(opcode));
250       break;
251
252       //storecr reg_str, char
253       //1011 rrrr vvvv vvvv
254     case 11:
255       result.id = STORE_C;
256       result.reg1 = NIBBLE2(opcode);
257       result.value = BYTE2(opcode);
258       break;
259
260       //mul reg1, value
261       //1100 dddd vvvv vvvv
262     case 12:
263       result.id = MUL_V;
264       result.reg1 = NIBBLE2(opcode);
265       result.value = get_signed_value(BYTE2(opcode));
```

```
266       break;
267
268       //ldh reg_d, value
269       //1101 dddd vvvv vvvv
270     case 13:
271       result.id = STORE_H;
272       result.reg1 = NIBBLE2(opcode);
273       result.value = BYTE2(opcode);
274       break;
275
276     default:
277       break;
278     }
279
280     return result;
281   }
282
283   void execute_opcode(agent_t *agent, opcode_t opcode) {
284     uint16_t tmp = 0;
285     int16_t sgn_tmp = 0;
286     //frame_t frame;
287
288
289     switch (opcode.id) {
290     case SETSERVICE:
291       PRINTF("setservice service_id: %d, reg: %d\n", opcode.value, opcode.↩
          reg1);
292
293       switch (opcode.value) {
294
295       case SERVICE_BARGRAPH:
296         if (platform.drivers.set_bargraph != NULL) {
297           platform.drivers.set_bargraph((agent->regs[opcode.reg1] & 0x00ff)↩
              );
298         } else {
299           SET_ERROR(agent->status_flag, ERROR_NO_SERVICE_PRESENT);
300         }
301         break;
302
303       case SERVICE_LED:
304         if (platform.drivers.dotmatrix_send != NULL) {
305           _delay_ms(50);
306           if (opcode.reg1 >= REG_MAX) {
307             opcode.reg1 = opcode.reg1 - REG_MAX;
308             if (agent->regstr_len[opcode.reg1] != 0) {
309               platform.drivers.dotmatrix_send(agent->reg_str[opcode.reg1]);
310             }
311           }
312         }
313         break;
314
315       case SERVICE_COOLER:
316         if (platform.drivers.set_cooler != NULL){
317           platform.drivers.set_cooler((agent->regs[opcode.reg1] & 0x00ff));
318         }
319         break;
320
321       case SERVICE_HEATER:
```

```
322         if (platform.drivers.heater_set != NULL){
323           platform.drivers.heater_set((agent->regs[opcode.reg1] & 0x00ff));
324         }
325         break;
326 
327       case SERVICE_LCD:
328         {
329 
330           if (platform.drivers.DISPLAY_string != NULL){
331             //ldl reg_0, row_nr 0 clear
332             //setservice lcd, str_reg_0
333             uint8_t row;
334             if (agent->regs[REG_ACC] == 0){
335               //clear string
336               uint8_t i;
337               for (i = 1; i < MAX_LCD_ROWS + 1; i++){
338                 row = 150 - (( i - 1 ) * 20) ;
339                 platform.drivers.DISPLAY_string(20, row, RGB(30,238,30), ↩
                      RGB(0,0,0), 2, "                    ");
340               }
341             } else {
342               if (agent->regs[REG_ACC] < MAX_LCD_ROWS + 1) {
343                 //calculate row
344                 row = 150 - (( agent->regs[REG_ACC] - 1 ) * 20) ;
345                 if (opcode.reg1 >= REG_MAX) {
346                   opcode.reg1 = opcode.reg1 - REG_MAX;
347                   platform.drivers.DISPLAY_string(20, row, RGB(30,238,30), ↩
                        RGB(0,0,0), 2, agent->reg_str[opcode.reg1]);
348                 } else {
349                   char buf[] = "          ";
350                   sprintf(buf,  "%d", agent->regs[opcode.reg1]);
351                   platform.drivers.DISPLAY_string(20, row, RGB(30,238,30), ↩
                        RGB(0,0,0), 2, buf);
352                 }
353               }
354 
355             }
356 
357           }
358         }
359         break;
360       default:
361         break;
362 
363       }
364       break;
365       case GETSERVICE:
366         PRINTF("getservice service_id: %d\n", opcode.value);
367         switch (opcode.value){
368         case SERVICE_THERMOMETER:
369           //_delay_ms(5000);
370           if (platform.drivers.therm_get_temp != NULL){
371 
372             tmp = (platform.drivers.therm_get_temp(THERMOMETER1) >>5);
373             tmp += (platform.drivers.therm_get_temp(THERMOMETER2) >>5);
374             tmp += (platform.drivers.therm_get_temp(THERMOMETER3) >>5);
375             tmp /= 3;
376             agent->regs[REG_ACC] = tmp;
```

```
377
378             agent->regstr_len[0] = 0;
379             free(agent->reg_str[0]);
380
381             agent->reg_str[0] = malloc (6);
382             agent->regstr_len[0] = 6;
383
384             uint16_t after = (tmp & 0x0007);
385             after *= 125;
386
387             uint16_t before = ((tmp & 0xfff8) >> 3);
388             sprintf(agent->reg_str[REG_ACC], "%d.%03d", before, after);
389
390           } else {
391             SET_ERROR(agent->status_flag, ERROR_NO_SERVICE_PRESENT);
392           }
393           break;
394         case SERVICE_BUTTON0:
395           {
396             agent->regs[REG_ACC] = button0_pressed;
397             button0_pressed = 0;
398           }
399           break;
400         case SERVICE_BUTTON1:
401           {
402             agent->regs[REG_ACC] = button1_pressed;
403             button1_pressed = 0;
404           }
405           break;
406         default:
407           break;
408         }
409
410         break;
411
412     case STORE_L:
413       PRINTF ("ldl reg_d:%d, value:%d\n", opcode.reg1, opcode.value);
414       agent->regs[opcode.reg1] = opcode.value;
415       break;
416
417     case ADD_R:
418       PRINTF("add reg_d: %d , reg_r: %d\n", opcode.reg1, opcode.reg2);
419       agent->regs[REG_ACC] = agent->regs[opcode.reg1] + agent->regs[opcode.↩
          reg2];
420       break;
421
422     case ADD_V:
423       PRINTF("add reg_r:%d, value:%d\n", opcode.reg1, opcode.value);
424       agent->regs[REG_ACC] = agent->regs[opcode.reg1] + opcode.value;
425       break;
426
427     case SUB_R:
428       PRINTF("sub reg_m:%d, reg_s:%d\n", opcode.reg1, opcode.reg2);
429       agent->regs[REG_ACC] = agent->regs[opcode.reg1] - agent->regs[opcode.↩
          reg2];
430       break;
431
432     case SUB_V:
```

```
433       PRINTF("sub reg_m:%d, value:%d\n", opcode.reg1, opcode.value);
434       agent->regs[REG_ACC] = agent->regs[opcode.reg1] - opcode.value;
435       break;
436
437     case DIV_R:
438       PRINTF("div reg_d:%d, reg_r:%d\n", opcode.reg1, opcode.reg2);
439       agent->regs[REG_ACC] = agent->regs[opcode.reg1] / agent->regs[opcode.↩
          reg2];
440       break;
441
442     case DIV_V:
443       PRINTF("div reg_d:%d, value:%d\n", opcode.reg1, opcode.value);
444       agent->regs[REG_ACC] = agent->regs[opcode.reg1] / opcode.value;
445       break;
446
447     case MUL_R:
448       PRINTF("mul reg_d:%d, reg_r:%d\n", opcode.reg1, opcode.reg2);
449       agent->regs[REG_ACC] = agent->regs[opcode.reg1] * agent->regs[opcode.↩
          reg2];
450       break;
451
452     case MUL_V:
453       PRINTF("mul reg1:%d, value:%d\n", opcode.reg1, opcode.value);
454       agent->regs[REG_ACC] = agent->regs[opcode.reg1] * opcode.value;
455       break;
456
457     case JMP_G:
458       PRINTF("jmpgr offset:%d\n", opcode.value);
459       if (agent->regs[REG_ACC]==1) {
460         agent->pc = agent->pc + opcode.value;
461       }
462       break;
463
464     case JMP_E:
465       PRINTF("jmpeq offset:%d\n", opcode.value);
466       if (agent->regs[REG_ACC]==0){
467         agent->pc = agent->pc + opcode.value;
468       }
469       break;
470
471     case JMP_L:
472       PRINTF("jmpls offset:%d\n", opcode.value);
473       if (agent->regs[REG_ACC]==-1){
474         agent->pc = agent->pc + opcode.value;
475       }
476       break;
477
478     case CMP_R:
479       PRINTF("compare reg_d:%d, reg_r:%d\n", opcode.reg1, opcode.reg2);
480
481       if (agent->regs[opcode.reg1] > agent->regs[opcode.reg2]){
482         agent->regs[REG_ACC] = 1;
483       } else if (agent->regs[opcode.reg1] == agent->regs[opcode.reg2]){
484         agent->regs[REG_ACC] = 0;
485       } else if (agent->regs[opcode.reg1] < agent->regs[opcode.reg2]){
486         agent->regs[REG_ACC] = -1;
487       }
488       break;
```

```
489
490     case CMP_V:
491       PRINTF("compare reg_d:%d, value:%d\n", opcode.reg1, opcode.value);
492
493       if (agent->regs[opcode.reg1] > opcode.value){
494         agent->regs[REG_ACC] = 1;
495       } else if (agent->regs[opcode.reg1] == opcode.value){
496         agent->regs[REG_ACC] = 0;
497       } else if (agent->regs[opcode.reg1] < opcode.value){
498         agent->regs[REG_ACC] = -1;
499       }
500       break;
501
502     case MOVE:
503       PRINTF("move service:%d\n", opcode.value);
504       {
505         // find dst
506         uint8_t i;
507         uint8_t dst_node;
508
509         for (i=0; i < MAX_NODES; i++){
510           if (service_locations[opcode.value][i] != INVALID){
511             if (dst_node != platform.id){
512               dst_node = service_locations[opcode.value][i];
513               break;
514             }
515           }
516         }
517
518         //prepare frame
519         frame_t frame;
520         frame.dst_node = dst_node;
521         frame.dst_agent = 0;
522         frame.frame_id.id = platform_config.frame_id;
523         frame.frame_id.src_board = platform_config.board_id;
524         frame.dst_board = platform_config.board_id;
525         frame.frame_id.src_node = platform.id;
526         frame.index = 0;
527
528         uint16_t len;
529         frame.data = serialize_agent(*agent, &len);
530         frame.frame_length = len;
531
532         platform_config.frame_id += 1;
533         agent->regs[REG_ACC] = send_message(frame);
534         free(frame.data);
535       }
536       break;
537
538     case CLONE:
539       PRINTF("clone\n");
540       agent->regs[REG_ACC] = clone_agent(agent);
541       break;
542
543     case DIE:
544       PRINTF("die\n");
545       agent->status = stopped;
546       break;
```

```
547
548     case SEND:
549       PRINTF("sendmsg reg:%d, agent:%d, platform:%d\n", opcode.reg1, opcode↩
              .agent_id, opcode.node_id);
550       {
551         frame_t frame;
552         frame.dst_node = opcode.node_id;
553         frame.dst_agent = opcode.agent_id;
554         frame.frame_id.id = platform_config.frame_id;
555         frame.frame_id.src_board = platform_config.board_id;
556         frame.dst_board = platform_config.board_id;
557         frame.frame_id.src_node = platform.id;
558         frame.index = 0;
559         if (opcode.reg1 >= REG_MAX) {
560           opcode.reg1 = opcode.reg1 - REG_MAX;
561           frame.frame_length = agent->regstr_len[opcode.reg1];
562           frame.data = (char*) malloc (frame.frame_length);
563           memcpy(frame.data, agent->reg_str[opcode.reg1], frame.↩
                frame_length);
564         } else {
565           frame.frame_length = sizeof(int16_t);
566           frame.data = (char*) malloc (frame.frame_length);
567           memcpy(frame.data, &(agent->regs[opcode.reg1]), frame.↩
                frame_length);
568         }
569         platform_config.frame_id += 1;
570         agent->regs[REG_ACC] = send_message(frame);
571         free(frame.data);
572       }
573       break;
574
575     case RECV:
576       PRINTF("pullmsg reg:%d\n", opcode.reg1);
577       if (agent->rec_msg_len != 0){
578
579         if (opcode.reg1 >= REG_MAX) {
580           opcode.reg1 = opcode.reg1 - REG_MAX;
581
582           if (agent->regstr_len[opcode.reg1] != 0){
583             free(agent->reg_str[opcode.reg1]);
584           }
585
586           agent->reg_str[opcode.reg1] = agent->rec_msg_content;
587           agent->regstr_len[opcode.reg1] = agent->rec_msg_len;
588
589         } else {
590           agent->regs[opcode.reg1] = agent->rec_msg_content[0];
591           free(agent->rec_msg_content);
592           agent->rec_msg_len = 0;
593         }
594
595         agent->rec_msg_content = 0;
596         agent->rec_msg_len = 0;
597         agent->regs[REG_ACC] = 0;
598
599       } else {
600         agent->regs[REG_ACC] = -1;
601       }
```

```
602
603       break;
604
605     case STORE_H:
606       PRINTF("ldh reg_d:%d, value:%d\n", opcode.reg1, opcode.value);
607       sgn_tmp = agent->regs[opcode.reg1];
608       tmp =  (opcode.value << 8);
609       sgn_tmp = (sgn_tmp & 0x00FF) | tmp;
610       agent->regs[opcode.reg1] = sgn_tmp;
611       break;
612
613     case STORE_C:
614       PRINTF("storecr reg_str:%d, char:%d\n", opcode.reg1, opcode.value);
615       opcode.reg1 = (opcode.reg1 - REG_MAX);
616       agent->reg_str[opcode.reg1] = (char*) realloc (agent->reg_str[opcode.↩
             reg1], agent->regstr_len[opcode.reg1] + 1);
617       agent->reg_str[opcode.reg1][agent->regstr_len[opcode.reg1]] = opcode.↩
             value;
618       agent->reg_str[opcode.reg1][agent->regstr_len[opcode.reg1]+1] = '\0';
619
620       agent->regstr_len[opcode.reg1]+= 1;
621       break;
622
623     case MV:
624       PRINTF("mv reg_d:%d, reg_r:%d\n", opcode.reg1, opcode.reg2);
625
626       if (opcode.reg1 >= REG_MAX && opcode.reg2 >= REG_MAX){
627         //both str
628         opcode.reg1 = opcode.reg1 - REG_MAX;
629         opcode.reg2 = opcode.reg2 - REG_MAX;
630         realloc(agent->reg_str[opcode.reg1], agent->regstr_len[opcode.reg2↩
               ]+1);
631         memcpy(agent->reg_str[opcode.reg1], agent->reg_str[opcode.reg2], ↩
               agent->regstr_len[opcode.reg2]+1);
632         agent->regstr_len[opcode.reg1] = agent->regstr_len[opcode.reg2];
633
634       } else if (opcode.reg1 >= REG_MAX) {
635         //dst str
636         opcode.reg1 = (opcode.reg1 - REG_MAX);
637         agent->reg_str[opcode.reg1] = (char*) realloc (agent->reg_str[↩
               opcode.reg1], agent->regstr_len[opcode.reg1] + 1);
638         agent->reg_str[opcode.reg1][agent->regstr_len[opcode.reg1]] = agent↩
               ->regs[opcode.reg2] & 0x00ff;
639         agent->reg_str[opcode.reg1][agent->regstr_len[opcode.reg1]+1] = '\0↩
               ';
640         agent->regstr_len[opcode.reg1]+= 1;
641
642       } else if (opcode.reg2 >= REG_MAX) {
643         //src str
644         opcode.reg2 = (opcode.reg2 - REG_MAX);
645         agent->regs[opcode.reg1] = agent->reg_str[opcode.reg2][0];
646       } else {
647         agent->regs[opcode.reg1] = agent->regs[opcode.reg2];
648       }
649       break;
650
651     case WAIT:
652       PRINTF("wait delay_ms:%d\n", opcode.value);
```

```
653       _delay_ms(opcode.value*10);
654       break;
655
656     case PRIO:
657       PRINTF("priority value:%d\n", opcode.value);
658       agent->priority = opcode.value;
659       break;
660
661     case CLEAR:
662       PRINTF("clr reg_str:%d\n", opcode.reg1);
663       opcode.reg1 = (opcode.reg1 - REG_MAX);
664       memset(agent->reg_str[opcode.reg1], 0, agent->regstr_len[opcode.reg1↩
          ]+1);
665       agent->reg_str[opcode.reg1] = (char*)realloc(agent->reg_str[opcode.↩
          reg1], 1);
666       agent->reg_str[opcode.reg1][0] = '\0';
667       agent->regstr_len[opcode.reg1] = 0;
668       break;
669
670     case CONV:
671       PRINTF("convert reg_str:%d, reg_%d\n", opcode.reg1, opcode.reg2);
672       opcode.reg1 = (opcode.reg1 - REG_MAX);
673       agent->regstr_len[opcode.reg1] = 10;
674       agent->reg_str[opcode.reg1] = (char*)realloc(agent->reg_str[opcode.↩
          reg1], 10+1);
675       sprintf(agent->reg_str[opcode.reg1], "%d.000", agent->regs[opcode.↩
          reg2]);
676       break;
677
678
679     default:
680       break;
681
682     }
683
684 }
685
686 int16_t get_signed_value(uint8_t value) {
687   int16_t result = 0;
688
689   if ((value & BYTE_SIGN) != 0) {
690     result = value | NEG_SIGN;
691   } else {
692     result = value & POS_SIGN;
693   }
694   return result;
695 }
```

A.1.8 comm_layer.h

```
1  /*
2   * comm_layer.h
3   *
4   *  Created on: 21.12.2012
5   *      Author: igor
6   */
7
```

```
8  #include "platform.h"
9  #include "protocol0.h"
10
11 #ifndef COMM_LAYER_H_
12 #define COMM_LAYER_H_
13
14 #define START_PACKET 0x00
15 #define DATA_PACKET  0x01
16
17 #define START_PACKET_LEN 0x07
18 #define DATA_PACKET_LEN 0x04
19 #define PACKET_LEN 0x0f
20
21 /*
22  * |  dst_node  | packet len  |
23  * |  start_type  |   src board |
24  * |  src_node  | frame id  |
25  * |    packet id hi      |
26  * |    packet id low     |
27  * |  dst board | dst agent |
28  * |    frame length hi     |
29  * |    frame length low    |
30  * |      data      |
31  * |      ...     |
32  * |      crc     |
33  *
34  *
35  * |  dst_node  | packet len  |
36  * |  data_type |   src board |
37  * |  src_node  | frame id  |
38  * |    packet id hi      |
39  * |    packet id low     |
40  * |      data      |
41  * |      ...     |
42  * |      crc     |
43  *
44  */
45
46 #define MASK 0x0f
47 #define SET_HEADER_NODE(header, node) (header |= (node & MASK) << 4)
48 #define SET_HEADER_LEN(header, len) (header |= (len & MASK))
49
50 #define SET_PAYLOAD_TYPE(payload, type) (payload[0] |= (type & MASK) << ↩
      4)
51 #define SET_PAYLOAD_SRC_BOARD(payload, board) (payload[0] |= (board & ↩
     MASK))
52
53 #define SET_PAYLOAD_SRC_NODE(payload, node) (payload[1] |= (node & MASK) ↩
     << 4)
54 #define SET_PAYLOAD_FRAME_ID(payload, id) (payload[1] |= (id & MASK))
55
56 #define SET_PAYLOAD_PACK_ID(payload, id) \
57   payload[2] = (( id & 0xff00 ) >> 8); \
58   payload[3] = ( id & 0x00ff)
59
60 #define SET_PAYLOAD_DST_BOARD(payload, board) (payload[4] |= (board & ↩
     MASK) << 4)
61 #define SET_PAYLOAD_DST_AGENT(payload, agent) (payload[4] |= (agent & ↩
```

```
    MASK) )
62
63 #define SET_PAYLOAD_FRAME_LEN(payload, len) \
64   payload[5] = ((len & 0xff00) >> 8); \
65   payload[6] = ( len & 0x00ff)
66
67
68 typedef struct {
69   unsigned src_board:4;
70   unsigned src_node:4;
71   unsigned id:4;
72 } frame_id_t;
73
74 typedef struct {
75   uint8_t header;
76   uint8_t payload[PACKET_LEN];
77 } packet_t;
78
79 struct frame{
80   unsigned dst_node:4;
81   unsigned dst_board:4;
82   unsigned dst_agent:4;
83   frame_id_t frame_id;
84   unsigned frame_length:16;
85   unsigned index:16;
86   struct frame *next_frame;
87   char *data;
88 };
89
90 typedef struct frame frame_t;
91
92 typedef struct {
93   uint16_t size;
94   frame_t *first;
95   frame_t *last;
96 }frame_list_t;
97
98 extern frame_list_t frm_list;
99
100 uint8_t send_message(frame_t frame);
101 //uint8_t send_msg(uint8_t message_header, uint8_t *msg_body);
102
103 #define HI_MASK 0xf0
104 #define LO_MASK 0x0f
105
106 #define GET_DST_NODE(header) (header & HI_MASK) >> 4)
107 #define GET_MSG_LEN(header) ((header & LO_MASK))
108
109 #define GET_PAYLOAD_TYPE(payload) ((payload[0]& HI_MASK) >> 4)
110 #define GET_PAYLOAD_SRC_BOARD(payload) (payload[0] & LO_MASK)
111
112 #define GET_PAYLOAD_SRC_NODE(payload) ((payload[1] & HI_MASK) >> 4)
113 #define GET_PAYLOAD_FRAME_ID(payload) (payload[1] & LO_MASK)
114
115 #define GET_PAYLOAD_PACK_ID(payload) ((payload[2] << 8) | payload[3])
116
117 #define GET_PAYLOAD_DST_BOARD(payload) ((payload[4] & HI_MASK) >> 4)
118 #define GET_PAYLOAD_DST_AGENT(payload) (payload[4] & LO_MASK)
```

```
119
120 #define GET_PAYLOAD_FRAME_LEN(payload) ((payload[5] << 8) | payload[6])
121
122 void recv_handler(uint8_t msg_length, uint8_t *msg_body);
123
124 char* serialize_agent(agent_t agent, uint16_t* agent_len);
125 agent_t deserialize_agent(uint8_t* packet);
126
127 #define MOBILITY_BYTE 0x55
128
129 #define HEADER_POS   0
130 #define ID_POS       1
131 #define PRIO_POS     2
132 #define STATUS_POS   3
133
134 #define FLAG_POS1    4
135 #define FLAG_POS2    5
136 #define FLAG_POS3    6
137 #define FLAG_POS4    7
138
139 #define PC_POS1      8
140 #define PC_POS2      9
141
142 #define REG0_POS1  10
143 #define REG0_POS2  11
144 #define REG1_POS1  12
145 #define REG1_POS2  13
146 #define REG2_POS1  14
147 #define REG2_POS2  15
148 #define REG3_POS1  16
149 #define REG3_POS2  17
150 #define REG4_POS1  18
151 #define REG4_POS2  19
152 #define REG5_POS1  20
153 #define REG5_POS2  21
154 #define REG6_POS1  22
155 #define REG6_POS2  23
156 #define REG7_POS1  24
157 #define REG7_POS2  25
158 #define REG8_POS1  26
159 #define REG8_POS2  27
160 #define REG9_POS1  28
161 #define REG9_POS2  29
162 #define REG10_POS1  30
163 #define REG10_POS2  31
164 #define REG11_POS1  32
165 #define REG11_POS2  33
166 #define REG12_POS1  34
167 #define REG12_POS2  35
168 #define REG13_POS1  36
169 #define REG13_POS2  37
170
171
172 #define CODE_LEN_POS1  38
173 #define CODE_LEN_POS2  39
174
175 #define REG0_STR_LEN_POS1  40
176 #define REG0_STR_LEN_POS2  41
```

```
177 #define REG1_STR_LEN_POS1 42
178 #define REG1_STR_LEN_POS2 43
179 #define REG2_STR_LEN_POS1 44
180 #define REG2_STR_LEN_POS2 45
181
182 #define REC_LEN_POS1 46
183 #define REC_LEN_POS2 47
184
185 #define FIXED_LEN 49
186 #define DYNAMIC_START 48
187
188 #define SET_MOBILITY_HEADER(packet) (packet[HEADER_POS] = MOBILITY_BYTE)
189 #define GET_MOBILITY_HEADER(packet) (packet[HEADER_POS])
190
191 #define SET_MOBILITY_END(packet, pos) (packet[pos] = MOBILITY_BYTE)
192 #define GET_MOBILITY_END(packet, pos) (packet[pos])
193
194 #define SET_AGENT_ID(packet, id) (packet[ID_POS] = id)
195 #define GET_AGENT_ID(packet) (packet[ID_POS])
196
197 #define SET_AGENT_PRIO(packet, prio) (packet[PRIO_POS] = prio)
198 #define GET_AGENT_PRIO(packet) (packet[PRIO_POS])
199
200 #define SET_AGENT_STATUS(packet, status) (packet[STATUS_POS] = status)
201 #define GET_AGENT_STATUS(packet) (packet[STATUS_POS])
202
203 #define SET_AGENT_FLAG_REG(packet, flag) \
204   packet[FLAG_POS1] = ((flag & 0xff000000) >> 24); \
205   packet[FLAG_POS2] = ((flag & 0x00ff0000) >> 16);  \
206   packet[FLAG_POS3] = ((flag & 0x0000ff00) >> 8);  \
207   packet[FLAG_POS4] = (flag & 0x00000000) ;
208
209 #define GET_AGENT_FLAG_REG(packet) \
210   (packet[FLAG_POS1] << 24) | (packet[FLAG_POS2] << 16) | (packet[↩
      FLAG_POS3] << 8) | packet[FLAG_POS4];
211
212
213 #define SET_AGENT_PC(packet, pc) \
214     packet[PC_POS1] = ((pc & 0xff00)>>8); \
215     packet[PC_POS2] = (pc & 0x00ff);
216
217 #define GET_AGENT_PC(packet) \
218     (packet[PC_POS1] << 8) | packet[PC_POS2];
219
220
221 #define SET_AGENT_REG(packet, agent, reg) \
222     packet[REG##reg##_POS1] = ((agent.regs[reg] & 0xff00) >> 8); \
223     packet[REG##reg##_POS2] = (agent.regs[reg] & 0x00ff);
224
225 #define GET_AGENT_REG(packet, reg) \
226     (packet[REG##reg##_POS1] << 8) | packet[REG##reg##_POS2];
227
228 #define SET_AGENT_CODE_LEN(packet, len) \
229     packet[CODE_LEN_POS1] = ((len & 0xff00)>>8); \
230     packet[CODE_LEN_POS2] = (len & 0x00ff);
231
232 #define GET_AGENT_CODE_LEN(packet) \
233     (packet[CODE_LEN_POS1] << 8) | packet[CODE_LEN_POS2];
```

```
234
235
236 #define SET_AGENT_REG_STR_LEN(packet, agent, reg) \
237     packet[REG##reg##_STR_LEN_POS1] = ((agent.regstr_len[reg] & 0xff00) ↩
        >> 8); \
238     packet[REG##reg##_STR_LEN_POS2] = (agent.regstr_len[reg] & 0x00ff);
239
240 #define GET_AGENT_REG_STR_LEN(packet, reg) \
241     (packet[REG##reg##_STR_LEN_POS1] << 8)| packet[REG##reg##↩
        _STR_LEN_POS2];
242
243
244 #define SET_AGENT_REC_MSG_LEN(packet, len) \
245     packet[REC_LEN_POS1] = ((len & 0xff00)>>8); \
246     packet[REC_LEN_POS2] = (len & 0x00ff);
247
248 #define GET_AGENT_REC_MSG_LEN(packet) \
249     (packet[REC_LEN_POS1] << 8) | packet[REC_LEN_POS2];
250
251
252 #endif /* COMM_LAYER_H_ */
```

A.1.9 comm_layer.c

```
1  /*
2   * comm_layer.c
3   *
4   *  Created on: 21.12.2012
5   *      Author: igor
6   */
7
8  #include "comm_layer.h"
9
10 frame_list_t frm_list;
11
12
13 uint8_t send_message(frame_t frame){
14
15   uint8_t res = 0;
16   packet_t packet;
17   uint16_t packet_id = 0;
18   uint8_t packet_ind =0;
19
20   while (frame.index < frame.frame_length){
21     uint8_t data_len = 0;
22     memset(packet.payload, 0, PACKET_LEN);
23     packet.header = 0;
24
25     SET_HEADER_NODE(packet.header, frame.dst_node);
26     SET_PAYLOAD_SRC_BOARD(packet.payload, frame.frame_id.src_board);
27     SET_PAYLOAD_SRC_NODE(packet.payload, frame.frame_id.src_node);
28     SET_PAYLOAD_FRAME_ID(packet.payload, frame.frame_id.id);
29
30     if (frame.index == 0){
31       //start package
32       SET_PAYLOAD_TYPE(packet.payload, START_PACKET);
33       SET_PAYLOAD_PACK_ID(packet.payload, packet_id);
```

```
34        SET_PAYLOAD_DST_BOARD(packet.payload, frame.dst_board);
35        SET_PAYLOAD_DST_AGENT(packet.payload, frame.dst_agent);
36        SET_PAYLOAD_FRAME_LEN(packet.payload, frame.frame_length);
37
38        if (frame.frame_length + START_PACKET_LEN > PACKET_LEN){
39          SET_HEADER_LEN(packet.header, PACKET_LEN);
40          data_len =  PACKET_LEN - START_PACKET_LEN;
41        } else {
42          SET_HEADER_LEN(packet.header, (START_PACKET_LEN + frame.↩
                frame_length));
43          data_len =  frame.frame_length;
44        }
45        packet_ind = START_PACKET_LEN;
46
47      } else {
48        //data packages
49        SET_PAYLOAD_TYPE(packet.payload, DATA_PACKET);
50        SET_PAYLOAD_PACK_ID(packet.payload, packet_id);
51
52        if ((frame.frame_length - frame.index) + DATA_PACKET_LEN > ↩
              PACKET_LEN){
53          SET_HEADER_LEN(packet.header, PACKET_LEN);
54          data_len =  PACKET_LEN - DATA_PACKET_LEN;
55        } else{
56          SET_HEADER_LEN(packet.header,  (DATA_PACKET_LEN + (frame.↩
                frame_length - frame.index)));
57          data_len =  frame.frame_length - frame.index;
58        }
59        packet_ind = DATA_PACKET_LEN;
60      }
61
62      memcpy((packet.payload+packet_ind), (frame.data+frame.index), ↩
           data_len);
63      frame.index+= data_len;
64
65      res += send_msg(packet.header, packet.payload);
66      //_delay_ms(3000);
67      packet_id += 1;
68    }
69
70    return res;
71  }
72
73
74
75  /*uint8_t send_msg(uint8_t message_header, uint8_t *msg_body){
76
77    printf("header %x\n", message_header);
78    uint8_t i = 0;
79    for (i = 0; i < (message_header & 0x0f); i++){
80      printf("%d: data: %x\n", i, msg_body[i]);
81    }
82    fflush(stdout);
83    recv_handler(message_header&0x0f, msg_body);
84  }*/
85
86  /* |  dst_node  | packet len  |
87   * |  start_type  |   src board |
```

```
88  * |  src_node  | frame id  |
89  * |    packet id hi      |
90  * |    packet id low     |
91  * |  dst board | dst agent |
92  * |    frame length hi   |
93  * |    frame length low  |
94  * |      data      |
95  * |      ...       |
96  * |      crc       |
97  *
98  *
99  * |  dst_node  | packet len  |
100 * |  data_type |   src board |
101 * |  src_node  | frame id  |
102 * |    packet id hi      |
103 * |    packet id low     |
104 * |      data      |
105 * |      ...       |
106 * |      crc       |
107 */
108 void recv_handler(uint8_t msg_length, uint8_t *msg_body){
109
110
111   if (GET_PAYLOAD_TYPE(msg_body) == START_PACKET){
112
113     //new frame received
114     uint8_t agent_id = GET_PAYLOAD_DST_AGENT(msg_body);
115     uint16_t frame_size = GET_PAYLOAD_FRAME_LEN(msg_body);
116
117     if (frame_size > (msg_length - START_PACKET_LEN)){
118       //we need to buffer
119       frame_t* new_frame = malloc(sizeof(frame_t));
120       memset(new_frame, 0, sizeof(frame_t));
121       new_frame->frame_id.id = GET_PAYLOAD_FRAME_ID(msg_body);
122       new_frame->frame_id.src_node = GET_PAYLOAD_SRC_NODE(msg_body);
123       new_frame->frame_id.src_board = GET_PAYLOAD_SRC_BOARD(msg_body);
124       new_frame->dst_agent = agent_id;
125       new_frame->frame_length = frame_size;
126       new_frame->data = (char*) malloc(frame_size+1);
127       memcpy(new_frame->data, msg_body+START_PACKET_LEN, msg_length - ↩
              START_PACKET_LEN);
128       new_frame->index = msg_length - START_PACKET_LEN;
129
130       if (frm_list.last != NULL){
131         frm_list.last->next_frame = new_frame;
132         frm_list.last = new_frame;
133         frm_list.size += 1;
134       } else {
135         frm_list.first = new_frame;
136         frm_list.last = new_frame;
137         frm_list.size = 1;
138       }
139
140     } else {
141       //write directly to agent
142       platform.agents[agent_id].rec_msg_content = (char*)realloc(platform↩
              .agents[agent_id].rec_msg_content, frame_size +1);
143       memset(platform.agents[agent_id].rec_msg_content, 0, frame_size + ↩
```

```
            1);
144       memcpy(platform.agents[agent_id].rec_msg_content, msg_body+↩
              START_PACKET_LEN, frame_size);
145       platform.agents[agent_id].rec_msg_len = frame_size;
146     }
147
148
149   } else {
150     //new packet received
151     uint8_t id = GET_PAYLOAD_FRAME_ID(msg_body);
152     uint8_t node  = GET_PAYLOAD_SRC_NODE(msg_body);
153     uint8_t board = GET_PAYLOAD_SRC_BOARD(msg_body);
154     uint8_t i;
155     // search for frame
156     frame_t* current = frm_list.first;
157     frame_t* prev = frm_list.first;
158
159     while (current != NULL){
160       if (current->frame_id.id == id && current->frame_id.src_node == ↩
              node && current->frame_id.src_board == board){
161         //found
162         memcpy(current->data+current->index, msg_body+DATA_PACKET_LEN, ↩
              msg_length - DATA_PACKET_LEN);
163         current->index += msg_length - DATA_PACKET_LEN;
164         if (current->frame_length == current->index){
165           //everything received
166
167           if (GET_MOBILITY_HEADER(current->data) == MOBILITY_BYTE && ↩
                GET_MOBILITY_END(current->data, current->frame_length - 1) ↩
                == MOBILITY_BYTE){
168             //code mobility message
169
170             for (i = 0; i < AGENT_MAX; i++){
171               if (platform.agents[i].status == stopped){
172                 agent_t agent = deserialize_agent(current->data);
173                 agent.id = i;
174                 agent.regs[REG_ACC] = 0;
175                 agent.pc+= 1;
176                 platform.agents[i] = agent;
177                 break;
178               }
179             }
180
181           } else {
182             //data message
183             uint8_t agent_id = current->dst_agent;
184             uint16_t frame_size = current->frame_length;
185
186             platform.agents[agent_id].rec_msg_content = (char*)realloc(↩
                  platform.agents[agent_id].rec_msg_content, frame_size +1);
187             memset(platform.agents[agent_id].rec_msg_content, 0, ↩
                  frame_size + 1);
188             memcpy(platform.agents[agent_id].rec_msg_content, current->↩
                  data, frame_size);
189             platform.agents[agent_id].rec_msg_len = frame_size;
190           }
191
192           prev->next_frame = current->next_frame;
```

```
193
194             //only one element
195             if (frm_list.last == frm_list.first){
196               frm_list.last = NULL;
197               frm_list.first = NULL;
198             }
199             else if (current == frm_list.first){
200               frm_list.first = current->next_frame;
201             } else if (current == frm_list.last){
202               frm_list.last = prev;
203             }
204             free(current->data);
205             free(current);
206             current = NULL;
207             frm_list.size -= 1;
208           } else {
209             current = NULL;
210           }
211         } else {
212           prev = current;
213           current = current->next_frame;
214         }
215       }
216     }
217
218 }
219
220 char* serialize_agent(agent_t agent, uint16_t* agent_len){
221
222   *agent_len = FIXED_LEN + (agent.code_len * sizeof(uint16_t)) + agent.↩
       regstr_len[0] + agent.regstr_len[1] + agent.regstr_len[2] + agent.↩
       rec_msg_len;
223
224   char* agent_str = (char*) malloc(*agent_len);
225
226   SET_MOBILITY_HEADER(agent_str);
227
228   SET_AGENT_ID(agent_str, agent.id);
229   SET_AGENT_PRIO(agent_str, agent.priority);
230   SET_AGENT_STATUS(agent_str, agent.status);
231
232   SET_AGENT_FLAG_REG(agent_str, agent.status_flag);
233   SET_AGENT_PC(agent_str, agent.pc);
234
235
236   SET_AGENT_REG(agent_str, agent, 0);
237   SET_AGENT_REG(agent_str, agent, 1);
238   SET_AGENT_REG(agent_str, agent, 2);
239   SET_AGENT_REG(agent_str, agent, 3);
240   SET_AGENT_REG(agent_str, agent, 4);
241   SET_AGENT_REG(agent_str, agent, 5);
242   SET_AGENT_REG(agent_str, agent, 6);
243   SET_AGENT_REG(agent_str, agent, 7);
244   SET_AGENT_REG(agent_str, agent, 8);
245   SET_AGENT_REG(agent_str, agent, 9);
246   SET_AGENT_REG(agent_str, agent, 10);
247   SET_AGENT_REG(agent_str, agent, 11);
248   SET_AGENT_REG(agent_str, agent, 12);
```

```
249     SET_AGENT_REG(agent_str, agent, 13);
250
251     SET_AGENT_CODE_LEN(agent_str, agent.code_len);
252
253     SET_AGENT_REG_STR_LEN(agent_str, agent, 0);
254     SET_AGENT_REG_STR_LEN(agent_str, agent, 1);
255     SET_AGENT_REG_STR_LEN(agent_str, agent, 2);
256
257     SET_AGENT_REC_MSG_LEN(agent_str, agent.rec_msg_len);
258
259     //copy code
260     memcpy(agent_str + DYNAMIC_START, agent.code, agent.code_len * sizeof(↩
            uint16_t));
261
262     //copy reg_str
263     uint32_t pos = DYNAMIC_START + (agent.code_len * sizeof (uint16_t));
264     memcpy(agent_str + pos, agent.reg_str[0], agent.regstr_len[0]);
265
266     pos += agent.regstr_len[0];
267     memcpy(agent_str + pos, agent.reg_str[1], agent.regstr_len[1]);
268
269     pos += agent.regstr_len[1];
270     memcpy(agent_str + pos, agent.reg_str[2], agent.regstr_len[2]);
271
272     pos += agent.regstr_len[2];
273     memcpy(agent_str + pos, agent.rec_msg_content, agent.rec_msg_len);
274
275     pos += agent.rec_msg_len;
276
277     SET_MOBILITY_END(agent_str, pos);
278
279
280     return agent_str;
281 }
282
283 agent_t deserialize_agent(uint8_t* packet){
284     agent_t agent;
285
286
287     agent.id = GET_AGENT_ID(packet);
288     agent.priority = GET_AGENT_PRIO(packet);
289     agent.status = GET_AGENT_STATUS(packet);
290     agent.status_flag = GET_AGENT_FLAG_REG(packet);
291     agent.pc = GET_AGENT_PC(packet);
292
293
294     agent.regs[0] = GET_AGENT_REG(packet, 0);
295     agent.regs[1] = GET_AGENT_REG(packet, 1);
296     agent.regs[2] = GET_AGENT_REG(packet, 2);
297     agent.regs[3] = GET_AGENT_REG(packet, 3);
298     agent.regs[4] = GET_AGENT_REG(packet, 4);
299     agent.regs[5] = GET_AGENT_REG(packet, 5);
300     agent.regs[6] = GET_AGENT_REG(packet, 6);
301     agent.regs[7] = GET_AGENT_REG(packet, 7);
302     agent.regs[8] = GET_AGENT_REG(packet, 8);
303     agent.regs[9] = GET_AGENT_REG(packet, 9);
304     agent.regs[10] = GET_AGENT_REG(packet, 10);
305     agent.regs[11] = GET_AGENT_REG(packet, 11);
```

```
306     agent.regs[12] = GET_AGENT_REG(packet, 12);
307     agent.regs[13] = GET_AGENT_REG(packet, 13);
308
309
310     agent.code_len = GET_AGENT_CODE_LEN(packet);
311
312     agent.regstr_len[0] = GET_AGENT_REG_STR_LEN(packet, 0);
313     agent.regstr_len[1] = GET_AGENT_REG_STR_LEN(packet, 1);
314     agent.regstr_len[2] = GET_AGENT_REG_STR_LEN(packet, 2);
315
316     agent.rec_msg_len = GET_AGENT_REC_MSG_LEN(packet);
317
318     agent.code = malloc(agent.code_len * sizeof(uint16_t));
319     memcpy(agent.code, packet+DYNAMIC_START, agent.code_len * sizeof(↩
            uint16_t));
320
321     agent.reg_str = (char**) malloc(STR_REG_MAX * sizeof(char*));
322
323     uint32_t pos = DYNAMIC_START + (agent.code_len * sizeof(uint16_t));
324     agent.reg_str[0] = malloc (agent.regstr_len[0]);
325     memcpy(agent.reg_str[0], packet+pos, agent.regstr_len[0]);
326
327     pos += agent.regstr_len[0];
328     agent.reg_str[1] = malloc( agent.regstr_len[1]);
329     memcpy(agent.reg_str[1], packet+pos, agent.regstr_len[1]);
330
331     pos += agent.regstr_len[1];
332     agent.reg_str[2] = malloc(agent.regstr_len[2]);
333     memcpy(agent.reg_str[2], packet+pos, agent.regstr_len[2]);
334
335     pos += agent.regstr_len[2];
336     agent.rec_msg_content = malloc(agent.rec_msg_len);
337     memcpy(agent.rec_msg_content, packet+pos, agent.rec_msg_len);
338
339     return agent;
340 }
```

A.2 Agent assembler tool

A.2.1 asm_agent

```
1  #!/usr/bin/python
2  import re
3  import sys
4
5
6  LABEL = r"[A-Za-z][A-Za-z0-9]{0,5}"
7  MNEMONIC = r"add|sub|div|mul|jmpgr|jmpeq|jmpls|compare|convert|move|clr|↩
       clone|die|sendmsg|pullmsg|storecr|ldh|ldl|mv|wait|priority|setservice|↩
       getservice"
8  REG = r"reg_10|reg_11|reg_12|reg_0|reg_1|reg_2|reg_3|reg_4|reg_5|reg_6|↩
       reg_7|reg_8|reg_9|reg_str_0|reg_str_1|reg_str_2"
9  VALUE = r"0x[0-9A-F]{1,2}|0b[01]{1,8}|[+-]?[01]?[0-9]?[0-9]"
10 SERVICE = r"temp|bargraph|fan|heater|led_matrix|lcd|button0|button1"
11 CHAR = r"[-_:;!,.A-Za-z0-9]"
12 OPERAND = REG + "|" + VALUE + "|" + SERVICE
13 OPERAND_OR_CHAR = OPERAND + "|" + CHAR
```

```
14 LEGAL_LABEL = "((?!" + MNEMONIC + "|" + OPERAND + ")" + LABEL +")"
15 LABEL_LOOKUP = "(?:" + LEGAL_LABEL + "\s*:)?" + "\s*(" + MNEMONIC + ")" +↩
       "(?:\s+" + LEGAL_LABEL + ")?"
16 OPERAND_LOOKUP = "(?:" + LEGAL_LABEL + "\s*:)?" + "\s*(" + MNEMONIC + ")"↩
       + "(?:\s+(" + OPERAND_OR_CHAR  + "))?\s*(?:,\s*(" + OPERAND_OR_CHAR +↩
       "))?\s*(?:,\s*(" + OPERAND_OR_CHAR + "))?\s*"
17
18 reg = {"reg_0":"0000",\
19             "reg_1":"0001",\
20             "reg_2":"0010",\
21             "reg_3":"0011",\
22             "reg_4":"0100",\
23             "reg_5":"0101",\
24             "reg_6":"0110",\
25      "reg_7":"0111",\
26      "reg_8":"1000",\
27      "reg_9":"1001",\
28      "reg_10":"1010",\
29      "reg_11":"1011",\
30      "reg_12":"1100",\
31            }
32
33 reg_str = {"reg_str_0":"1101",\
34                "reg_str_1":"1110",\
35                "reg_str_2":"1111"}
36
37 service = {"temp":"1",\
38                 "bargraph":"0",\
39                 "fan":"2",\
40                 "heater":"3",\
41                 "led_matrix":"4",\
42                 "lcd":"5",\
43          "button0":"6",\
44          "button1":"7",\
45                 }
46
47 command_opcode = {"add":"00000011",\
48                 "addv":"0011",\
49                 "sub":"00000110",\
50                 "subv":"0110",\
51                 "div":"00001001",\
52                 "divv":"1001",\
53                 "mul":"00001100",\
54                 "mulv":"1100",\
55                 "jmpgr":"11110011",\
56                 "jmpeq":"11110110",\
57                 "jmpls":"11111100",\
58                 "compare":"00001010",\
59                 "comparev":"1010",\
60                 "move":"11110001",\
61                 "clone":"1111001000000000",\
62                 "die":"1111010000000000",\
63                 "sendmsg":"11111000",\
64                 "pullmsg":"111110100000",\
65                 "ldh":"1101",\
66                 "ldl":"0100", \
67                 "storecr":"1011",\
68                 "clr":"000000100000",\
```

```
69                  "mv":"00001101",\
70                  "wait":"00000101",\
71                  "priority":"00001000",\
72                  "setservice":"0111",\
73                  "getservice":"00000111",\
74              "convert":"11111111"}
75
76  #LEGAL LINE:
77  #LABEL:? MNEMONIC (?: OPERAND1? (?:, OPERAND2?)) | (LABEL2)
78  FILENAME = r""
79  SOURCE_FILE = r""
80  FIRST_PARS = {}
81
82  def asm_tool():
83      try:
84          check_files(sys.argv[1])
85          first_pass()
86          second_pass()
87      except (IOError, IndexError):
88          print "\tSpecify correct filename"
89
90
91
92  def first_pass():
93      source = open(SOURCE_FILE , "r")
94      num_lines = sum(1 for line in open(SOURCE_FILE))
95      i,k=1,0
96      while i <= num_lines:
97          source_line = source.readline()
98          label_lookup = re.compile(LABEL_LOOKUP).match(source_line)
99          operand_lookup = re.compile(OPERAND_LOOKUP).match(source_line)
100         if label_lookup != None:
101             label_1, label_2 = label_lookup.group(1), label_lookup.group↩
                    (3)
102             mnemonic = label_lookup.group(2)
103             operand_1, operand_2, operand_3 = operand_lookup.group(3), ↩
                    operand_lookup.group(4), operand_lookup.group(5)
104             FIRST_PARS[k] = {"LABEL1":label_1,"MNEMONIC":mnemonic,"LABEL2↩
                    ":label_2,\
105                                 "OPERAND1":operand_1, "OPERAND2":↩
                                        operand_2, "OPERAND3":operand_3, "↩
                                        LINE":i}
106             k+=1
107         i+=1
108     source.close()
109
110 def second_pass():
111     binary = open(FILENAME + ".bin", "w+")
112     listing = open(FILENAME + ".lst", "w+")
113     i = 0
114
115     while i < len(FIRST_PARS):
116         #assemble line
117         opcode = ""
118         mnem = FIRST_PARS[i]["MNEMONIC"].lower()
119         label1 = None if FIRST_PARS[i]["LABEL1"] == None else FIRST_PARS[↩
                i]["LABEL1"]
120         label2 =  None if FIRST_PARS[i]["LABEL2"] == None else FIRST_PARS↩
```

```
        [i]["LABEL2"]
121     operand1 = None if FIRST_PARS[i]["OPERAND1"] == None else ↩
        FIRST_PARS[i]["OPERAND1"].lower()
122     operand2 =  None if FIRST_PARS[i]["OPERAND2"] == None else ↩
        FIRST_PARS[i]["OPERAND2"].lower()
123     operand3 =  None if FIRST_PARS[i]["OPERAND3"] == None else ↩
        FIRST_PARS[i]["OPERAND3"].lower()
124     line =  None if FIRST_PARS[i]["LINE"] == None else FIRST_PARS[i][↩
        "LINE"]
125
126     # 2 operand reg-reg or reg-value commands
127     if  mnem == "add" or mnem == "sub" or mnem == "mul" or mnem == "↩
        div" or mnem == "compare":
128        if label2 != None:
129            print "Error in line %s, illegal label " %line
130        elif  operand1 not in reg or operand2 == None or operand3 != ↩
           None:
131            print "Error in line %s, illegal operand" %line
132        elif operand2 in reg:
133            opcode = command_opcode[mnem] + reg[operand1]+reg[operand2↩
               ]
134        elif re.compile(VALUE).match(operand2) != None:
135            opcode = command_opcode[mnem + "v"] + reg[operand1] + ↩
               convert_to_binary(operand2)
136     #0 operand 1 label commands
137     if mnem == "jmpeq" or mnem == "jmpls" or mnem == "jmpgr":
138         label = []
139         for lines in FIRST_PARS.iterkeys():
140                 if FIRST_PARS[lines]["LABEL1"] == label2:
141                     label.append(lines)
142         if len(label) != 1 or label2 == None:
143             print " Unresolved labels in line %s" %line
144         else:
145             offset = label[0] - i -1
146             opcode = command_opcode[mnem] + convert_to_binary(str(↩
                   offset))
147
148     #0 operand commands
149     if mnem == "clone" or mnem == "die":
150         if operand1 != None or operand2 != None or operand3 != None:
151             print "Error in line %s, illegal operand" %line
152         elif label2 != None:
153             print "Error in line %s, illegal label " %line
154         else:
155             opcode = command_opcode[mnem]
156     #1 operand commands
157     if mnem == "move" or mnem == "pullmsg" or mnem == "wait" or mnem ↩
        == "priority" or mnem == "getservice" or mnem == "clr":
158         if operand1 == None or operand2 != None or operand3 != None:
159             print "Error in line %s, illegal operand" %line
160         elif label2 != None:
161             print "Error in line %s, illegal label " %line
162         elif mnem == "move" or mnem == "getservice":
163             if operand1 not in service:
164                 print  "Error in line %s, illegal operand" %line
165             else:
166                 opcode = command_opcode[mnem] + convert_to_binary(↩
                       service[operand1])
```

```
167             elif mnem == "pullmsg":
168                 if operand1 in reg:
169                     opcode = command_opcode[mnem] + reg[operand1]
170                 elif operand1 in reg_str:
171                     opcode = command_opcode[mnem] + reg_str[operand1]
172                 else:
173                     print  "Error in line %s, illegal message destination↩
                           " %line
174             elif mnem == "wait":
175                 if re.compile(VALUE).match(operand1) != None:
176                     opcode = command_opcode[mnem] + convert_to_binary(↩
                           operand1)
177                 else:
178                     print  "Error in line %s, illegal delay value" %line
179             elif mnem == "priority":
180                 if re.compile("[0-3]").match(operand1) != None:
181                     opcode = command_opcode[mnem] + convert_to_binary(↩
                           operand1)
182                 else:
183                     print "Error in line %s, illegal priority value" %↩
                           line
184             elif mnem == "clr":
185                 if operand1 not in reg_str:
186                     print "Error in line %s, illegal operand" %line
187                 else:
188                     opcode = command_opcode[mnem] + reg_str[operand1]
189 
190         if mnem == "ldl" or mnem == "ldh" or mnem == "storecr":
191             if label2 != None:
192                print "Error in line %s, illegal label " %line
193             elif mnem == "ldl" or mnem == "ldh" and operand1 in reg:
194                 opcode = command_opcode[mnem] + reg[operand1] + ↩
                        convert_to_binary(operand2)
195             elif mnem == "storecr" and operand1 in reg_str:
196                 opcode = command_opcode[mnem] + reg_str[operand1] + ↩
                        convert_to_binary(str(ord(operand2)))
197             else:
198                 print "Error in line %s, operands do not match mnemonic "↩
                        %line
199         if mnem == "setservice":
200     if label2 != None:
201       print "Error in line %s, illegal label " %line
202     elif operand1 in service:
203       if operand2 in reg:
204         opcode = command_opcode[mnem] + reg[operand2] + convert_to_binary↩
              (service[operand1])
205       elif operand2 in reg_str:
206         opcode = command_opcode[mnem] + reg_str[operand2] + ↩
              convert_to_binary(service[operand1])
207     else:
208       print  "Error in line %s, illegal operand" %line
209         if mnem == "sendmsg":
210             if label2 != None:
211                  print "Error in line %s, illegal label " %line
212             elif re.compile("[0-3]").match(operand2) != None and \
213                     re.compile("[0-3]").match(operand3) != None:
214                 agent =  bin(int(operand2))[2:]
215                 dest_agent = agent if len(agent) == 2 else "0" + agent
```

```
216                 platform = bin(int(operand3))[2:]
217                 dest_platform = platform if len(platform) == 2 else "0" +↩
                        platform
218     if operand1 in reg:
219       opcode = command_opcode[mnem] + reg[operand1] + dest_agent + ↩
              dest_platform
220     elif operand1 in reg_str:
221       opcode = command_opcode[mnem] + reg_str[operand1] + dest_agent + ↩
              dest_platform
222             else:
223                 print  "Error in line %s, illegal operand" %line
224
225         if mnem == "mv":
226     if label2 != None:
227       print "Error in line %s, illegal label " %line
228     elif (operand1 in reg) and (operand2 in reg):
229       opcode = command_opcode[mnem] + reg[operand1] + reg[operand2]
230     elif (operand1 in reg) and (operand2 in reg_str):
231       opcode = command_opcode[mnem] + reg[operand1] + reg_str[operand2]
232     elif (operand1 in reg_str) and (operand2 in reg):
233       opcode = command_opcode[mnem] + reg_str[operand1] + reg[operand2]
234     elif (operand1 in reg_str) and (operand2 in reg_str):
235       opcode = command_opcode[mnem] + reg_str[operand1] + reg_str[↩
              operand2]
236     else:
237       print "Error in line %s, illegal operand" %line
238   if mnem == "convert":
239     if label2 != None:
240       print "Error in line %s, illegal label " %line
241     elif (operand1 in reg_str) and (operand2 in reg):
242       opcode = command_opcode[mnem] + reg_str[operand1] + reg[operand2]
243     else:
244       print "Error in line %s, illegal operand" %line
245
246         binary.write(opcode)
247         listing.write(xstr(FIRST_PARS[i]["LABEL1"]) + "\t\t" + opcode + "↩
              \t\t" + xstr(FIRST_PARS[i]["MNEMONIC"]) + "\t\t" + \
248                         xstr(operand1) + "\t\t" + xstr(operand2) + "\t\t↩
                              " + \
249                          xstr(operand3) + "\t\t" + xstr(label2) + "\n")
250         i+=1
251
252     binary.close()
253     listing.close()
254
255
256 def convert_to_binary(value):
257     bin_value = ""
258     if re.compile("^0x([0-9a-fA-F]{1,2})$").match(value) != None:
259           bin_value = bin(int(re.compile("^0x([0-9a-fA-F]{1,2})$").match(↩
                value).group(1), 16))[2:]
260     elif re.compile("^0b([01]{1,8})$").match(value) != None:
261           bin_value = re.compile("^0b([01]{1,8})$").match(value).group(1)
262     elif re.compile("^(-[01]?[0-9]?[0-9])$").match(value) != None:
263           bin_value = bin(2**9 + int(re.compile("^(-[01]?[0-9]?[0-9])$").↩
                match(value).group(1)))[3:]
264     elif re.compile("^([+]?[01]?[0-9]?[0-9])$").match(value) != None:
265           bin_value = bin(int(re.compile("^([+]?[01]?[0-9]?[0-9])$").↩
```

```
                match(value).group(1)))[2:]
266     else:
267           print "invalid value"
268           return None
269     while len(bin_value) < 8:
270           bin_value = "0" + bin_value
271     return bin_value
272
273
274 def check_files(filename):
275     """check agent file for .ma extension, extract filename for listing, ↩
          hex, binary generation"""
276     SOURCE_REGEX = re.compile("([A-Za-z0-9_]+)\.ma").match(filename)
277     if SOURCE_REGEX == None:
278         print "\tInvalid file!\n\tShould be with .ma extension"
279         return None
280     global SOURCE_FILE, FILENAME
281     SOURCE_FILE = SOURCE_REGEX.group(0)
282     FILENAME = SOURCE_REGEX.group(1)
283     return SOURCE_FILE
284
285 def xstr(s):
286     if s == None:
287         return ""
288     else:
289         return s
290
291 asm_tool()
```

A.3 Communication protocol

A.3.1 protocol0.h

```
1  /*=========================================================================↩
       */
2  /* Application: Header file for Protocoll B                */
3  /*                                    */
4  /* File: protocoll.h                          */
5  /* Revision: 1.0                            */
6  /* Author: Pelesic Igor                          */
7  /* e0006828@stud3.tuwien.ac.at                        */
8  /*=========================================================================↩
       */
9
10
11 #ifndef __PROTOCOLL_H__
12 #define __PROTOCOLL_H__
13
14 #include <stdio.h>
15 #include <avr/io.h>
16 #include <avr/interrupt.h>
17
18 #define F_CPU 14745600
19 #include <util/delay.h>
20
21 #include "timer2.h"
22
```

```
23 #if defined (__AVR_ATmega128__)
24
25 # define CPU_CLK 14745600
26
27 # define HW0_PORT PORTE
28 # define HW0_PIN PINE
29 # define HW0_DDR DDRE
30 # define HW0_RX PE0
31 # define HW0_TX PE1
32
33 # define HW1_PORT PORTD
34 # define HW1_PIN PIND
35 # define HW1_DDR DDRD
36 # define HW1_RX PD2
37 # define HW1_TX PD3
38
39 # define SW_PORT PORTE
40 # define SW_PIN PINE
41 # define SW_DDR DDRE
42 # define SW_RX PE7
43 # define SW_TX PE5
44
45 #define SET_BUS0_HIGH (HW0_PORT |= (1<<HW0_TX))
46 #define SET_BUS0_LOW (HW0_PORT &= ~(1<<HW0_TX))
47
48 #define SET_BUS1_HIGH (HW1_PORT |= (1<<HW1_TX))
49 #define SET_BUS1_LOW (HW1_PORT &= ~(1<<HW1_TX))
50
51 #define PROTCL_MAX_MESSAGE 15
52 #define BAUD 57600
53 #define BAUD1 115200
54 #define MODE_8E1 0x26 /*8 bits, even parity, 1 stop bit*/
55
56 #define ON 1
57 #define OFF 0
58
59 extern void protocol_init(uint8_t timeout, void (*receive_msg)(uint8_t ↩
      msg_header, uint8_t *msg_body));
60 extern int8_t send_msg(uint8_t message_header, uint8_t *msg_body);
61
62 void disable_int7(void);
63 void enable_int7(void);
64 void init_timer3(uint8_t bit_time);
65 void stop_timer3(void);
66
67
68 #else
69 # error "Wrong Target!\n"
70 #endif
71
72 #endif /* ifndef __PROTOCOLL__ */
73
74
75 extern void protocol_init(uint8_t timeout, void (*receive_msg)(uint8_t ↩
      msg_header, uint8_t *msg_body));
76 extern int8_t send_msg(uint8_t msg_length, uint8_t *msg_body);
77 void (*receive_handler)(uint8_t msg_length, uint8_t *msg_body);
```

A.3.2 protocol0.c

```
1  /*============================================================================
   */
2  /* Application: Protokoll B                                                  
   */
3  /*                                                                           
   */
4  /*                                                                           
   */
5  /*    Revision: 1.0                                                          
   */
6  /*      Author: Pelesic Igor                                                 
   */
7  /*              e0006828@stud3.tuwien.ac.at                                  
   */
8  /*============================================================================
   */
9
10 #include "protocol0.h"
11 #include "global.h"
12
13 volatile uint8_t send_timeout = 0;
14
15
16 volatile uint8_t timestamp = 0;
17 volatile uint8_t msg_header = 0; // used instead of message_length
18 volatile uint8_t checksum=0;
19 volatile uint8_t help;
20 volatile uint8_t length;
21 volatile uint8_t msg_id = 0;
22 volatile uint8_t message[PROTCL_MAX_MESSAGE];
23 volatile uint8_t *msg_pointer;
24 volatile uint8_t msg_index;
25 volatile uint8_t node_id=0;
26 volatile uint8_t send_check=0;
27 volatile uint8_t timer3_done=0;
28 volatile uint8_t int_occured=0;
29 volatile uint8_t init_state=0;
30
31
32 SIGNAL(SIG_UART0_RECV) {
33
34   PORTF &= ~(1 << PF3);
35
36   if ((UCSR0A & (1 << FE0) ) || (UCSR0A & (1 << DOR0)) || (UCSR0A & (1 <<
       FE0))  ) {
37     help=UDR0;
38     return;
39   } else {
40     //PORTF &= ~(1 << PF3);
41   }
42
43
44   if (timestamp==0)
45   { /* Anfangsbyte?*/
46     help=UDR0;
```

```
47      checksum = help;
48
49      length=(help&0x0f)
50
51      help=(help&0xf0);
52      msg_id = (help>>4);
53
54      timestamp=length+1; /* Wieviel Bytes zu Erwarten sind*/
55      msg_index=0;
56
57    }
58
59    else if ((timestamp>1)&&(node_id==msg_id))
60    {
61
62      message[msg_index]=UDR0;
63      checksum^=message[msg_index];
64      msg_index++;
65      timestamp--;
66
67    }
68
69    /* der falsche targetnode?? wir tun nur warten; timestamp verringern*/
70    /*  if false destination, wait */
71    else if ((timestamp>1)&&(node_id!=msg_id))
72    { help=UDR0;
73      timestamp--;
74
75    }
76
77    /*das Checksummenbyte?? wenn am richtigen node berprfen wir die*/
78    /* die checksumme, und falls sie bereinstimmt rufen wir die */
79    /* recieve function, und wir enablen den transmitter*/
80    /* if at end of message, check crc byte */
81    else if (timestamp==1)
82    { help=UDR0;
83
84      if (node_id==msg_id)
85      {
86        if (checksum==help)
87        { /* checksummen check*/
88          PORTF|=(1<<PF3);
89          msg_pointer = message;
90          receive_handler(length, (uint8_t *) msg_pointer);
91        }
92
93      }
94
95      timestamp--; /* letztes byte, andere drfen wieder*/
96    }
97
98    //PORTF|=(1<<PF3);
99  }
100
101
102
103
104
```

```
105 ISR(SIG_OUTPUT_COMPARE3A)
106 {
107  disable_int7();
108  stop_timer3();
109  timer3_done=1;
110 }
111
112 ISR(SIG_INTERRUPT7)
113 {
114   int_occured=1;
115 }
116
117 ISR(SIG_OUTPUT_COMPARE2) /*TIMER2 overflow interrupt*/
118 {
119   timer2_done=1; /* call the handler*/
120
121   TIMER_MASK&=~(1<<OCIE2);
122   TIMER2_CON&=~(1<<CS22);
123   TIMER2_CON&=~(1<<CS21);
124   TIMER2_CON&=~(1<<CS20);
125 }
126
127 void wait_ms(uint8_t timeout) {
128
129   configure_timer2(ONESHOT, timeout);
130   while (timer2_done != 1);
131 }
132
133
134 uint8_t send_byte(uint8_t byte) {
135
136   while (!(UCSR0A & (1 << UDRE0)));
137   UDR0 = byte;
138   return byte;
139 }
140
141 void init_timer3(uint8_t bit_time)
142 {
143       timer3_done=0;
144       ETIMSK|=(1<<OCIE3A);
145       OCR3A=TCNT3+bit_time;
146       TCCR3B|=(1<<CS32);
147 }
148
149 void stop_timer3(void)
150 {
151   ETIMSK&=~(1<<OCIE3A);
152   TCCR3B&=~(1<<CS32);
153 }
154
155 void enable_int7(void)
156 {
157  int_occured=0;
158  EICRB|=(1<<ISC71);              /*falling edge of int7*/
159  EIMSK|=(1<<INT7);              /* enable int 7*/
160 }
161
162 void disable_int7(void)
```

```
163 {
164  EIMSK&=~(1<<INT7);              /* disable int 7*/
165  EICRB&=~(1<<ISC71);
166 }
167
168 /**
169    Function:
170    send_msg
171    - Sends data to a node over serial line.
172    - Global interrupt flag needs to be enabled.
173    Parameters:
174    message_header - format 0b rrrr llll
175    - rrrr -destination node id
176    - llll  body length;
177    msg_body - payload; maximum length is 14;
178  */
179 int8_t send_msg(uint8_t message_header, uint8_t *msg_body) {
180   uint8_t anzahl_versuche = 0;
181   int i;
182   uint8_t ready = OFF;
183   init_state = 0;
184   PORTF &= ~(1 << PF2);
185
186
187   while (ready == OFF) {
188     anzahl_versuche++;
189
190     switch (init_state) {
191
192     case 0:
193       while (!(SW_PIN & (1 << SW_RX)))
194         ;
195       init_state = 1;
196       break;
197
198     case 1:
199       enable_int7();
200       init_timer3(12);
201       while (timer3_done != 1)
202         ;
203
204       if (int_occured == 1) {
205         init_state = 0;
206         break;
207       } else {
208         init_state = 2;
209       }
210
211     case 2:
212       SET_BUS0_LOW;
213       wait_ms(send_timeout);
214       SET_BUS0_HIGH;
215       init_timer3(1);
216       while (timer3_done != 1)
217         ;
218       if (SW_PIN & (1 << SW_RX)) {
219         ready = ON;
220       } else {
```

```
221           init_state = 0;
222         }
223         break;
224
225
226       default:
227         break;
228
229       }
230     }
231     UCSR0B |= (1 << TXEN0);
232
233     send_byte(message_header);
234
235     send_check = message_header;
236
237     for (i = 0; i < (message_header & 0x0F); i++) {
238
239       send_byte(msg_body[i]);
240       send_check ^= msg_body[i];
241     }
242     send_byte(send_check);
243     UCSR0B &= ~(1 << TXEN0);
244     init_timer3(2);
245     while (timer3_done != 1)
246       ;
247     PORTF |= (1 << PF2);
248     return anzahl_versuche;
249 }
250
251 void protocol_init(uint8_t timeout, void(*receive_msg)(uint8_t msg_header↩
      ,
252       uint8_t *msg_body)) {
253
254     uint16_t baudrate;
255
256     HW0_PORT |= (1 << HW0_TX) | (1 << HW0_RX);
257     HW0_DDR |= (1 << HW0_TX);
258     HW0_DDR &= ~(1 << HW0_RX);
259
260     SW_PORT |= (1 << SW_TX) | (1 << SW_RX);
261     SW_DDR &= ~((1 << SW_TX) | (1 << SW_RX));
262
263     PORTF |= (1 << PF3) | (1 << PF2);
264     DDRF |= (1 << PF3) | (1 << PF2);
265
266     baudrate = ((CPU_CLK >>4) / BAUD) - 1;
267     UBRR0H = (baudrate >> 8);
268     UBRR0L = (baudrate & 0xff);
269     UCSR0C = MODE_8E1;
270     UCSR0B |= (1 << RXCIE0) | (1 << RXEN0);
271
272     timestamp = 0;
273     msg_header = 0;
274     send_timeout = timeout;
275     node_id = timeout;
276     receive_handler = receive_msg;
277
```

```
278   }
```