Laborationsmaterial

Real-Time Systems - CT3200

Introduction

The labs consist of one lab and one project. The assignments are to be solved in groups of no more than 3 persons, and each group member must be able to present the solutions proposed by the group.

Laboration material:

The lab material consists of Lego Mindstorm ©, the developing environment, and the additional documentation.

LEGO Mindstorm ©: one box, Lego Mindstorm, can be obtained from the lab assistant. All group members must be registered to the course in order to obtain the Lego Mindstorm box. The inventory of the package will be checked both at delivery and return. The box shall be returned to the lab assistant no later than one week after the first course examination. If you are not done with the laborations within the scheduled time, you (the group) can loan a box for a limited period of time (one day). However, the requirement loan a box for one day, is to have all reports approved.

It is the group's responsibility to make sure that no material that should be included in the Lego box is missing when returning the box (otherwise, the group will be charged accordingly). For each group, the lab credits will not be reported by the lab assistant until the box is returned.

Developing environment can be downloaded from the course homepage as an archived file containing all documentation and tools needed for the lab-assignments. Detailed description of the development environment is available in section ***Set up the developing environment.***

Information necessary for solving the lab-assignment is available in the following documents:
Lab-instructions: available in "Laborationsmaterial.doc"
Asterix manual: available in "asterix.ppt" and "asterix.ps". These documents will not be updated. The latest information is always available in the lab-instructions.
Librarymanual för Lego Mindstorm: available in "librarymanual.doc".

For further details, please refer to the available technical reports about Asterix and Obelix.

Laboration

The target system for the laboration and the project is LEGO Mindstorm © in which a real-time operating system will be used to achieve the desired behavior of the system. The lab-instructions and the manuals contain detailed information about how to perform the programming and about the functionality available in the operating system.

The firs lab is an introduction to Asterix and LEGO Mindstorm ©. The purpose of the lab is to get familiar with Asterix and the available hardware. The basic idea about the first lab is to get used to the developing environment and to download the compiled system to the target platform. Take the opportunity to learn as much as possible about the system during the first lab as it will ease the work you will have to do in the project.

During the labs, you will even make use of the theoretical knowledge you have earned during the lectures. You will design and implement tasks that will lbe used in a real-time system. The tasks will be designed such that it should be possible to use them in an arbitrary real-time system designed for Asterix. In order to solve the assignment, you shall use the real-time operating system Asterix and LEGO Mindstorm © together with the sensors included in the box.

Project

In the project you shall build a car which must be able to find a track marked by a black line and, then, follow it. At the same time, a number of real-time requirements must be fulfilled as well. Building instructions for the car are available in the Lego Mindstorms © manual.

OBS! DO NOT FORGET TO READ THE FAQ ON THE COURSE HOMEPAGE !OBS

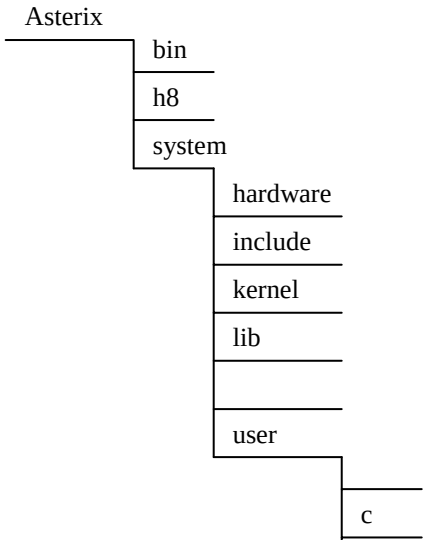
Read carefully the present document and the FAQ before you ask the lab-assistant.

Basics in Asterix

The scheme decides which tasks and resources are available in a particular system. Here, task attributes, e.g., priorities, are assigned to the tasks, or communication between tasks is specified, e.g., semaphores, etc.

Directory structure

The configuration of Asterix is built around a directory- and file-structure.



Config.obx

This file describes which tasks and resources will be used in asterix. The file **has to** reside in ***drive:\asterix\system\user*** , otherwise the compilation of the kernel will fail.

Scheduling strategy

OBS: Only Fixed Priority Scheduling (FPS) may be used as scheduling strategy in both lab and project (off-line scheduling strategy may not be used).

Defining and using resources in the configuration file

The syntax for defining resources which are to be used later on, is available in **asterix.ppt**. A detailed description of how to use already defined resources is presented below:

The following is used to read the content of a wait&lockfree channel:

```
buffertype_t    *buf=getReadPointerWF( wf_channel );
if(*buf > 42)
...
```

The following is used to read to a wait&lockfree channel:

```
buffertype_t    *ut=getWritePointerWF( wf_channel );
int value = 42;
...
writeChannel( ut, value, wf_channel );
```

The following call is used to get a semaphore:

```
getSemaphore(sem_1);
```

The following call is used to release a semaphore:

```
releaseSemaphore(sem_1);
```

The following function is used to raise a signal:

```
raiseSignal(sig_2);
```

Think about there is not possible to send any message to a task(s) which are being waked up the signal.

Task

Asterix is supporting only tasks defined with a deadling that must be guaranteed, i.e., tasks with *hard deadlines*. All tasks are defined in the configuration file: *system/user/config.obx*. The actual functionality of the task is coded as a function which has to have the following input:

```
void routine_A(void *param)
{

...

}
```

These functions are added only to the file *user\c\user.c*. It is even possible to add new files in the directory *user\c*. In that case, however, the header os_tasks.h must be included in each new file and in the make-file, such that the new file(s) are compiled and linked correctly. The recommendation is though to add all task routines in *user.c*.

In other words, the only 2 files you will have to work with are user.c (for task routines) and config.obx (for description of the tasks and resources)

Suported data types

Since the data types specified in the C-standard do not have the same size on a Intel-processor and a Hitachi-processor (built in Lego Mindstorm), the following data types are pre-defined:

Name	Type	supported by H8300
int8	8-bit signed	Yes
uint8	8-bit unsigned	Yes
int16	16-bit signed	Yes
uint16	16-bit unsigned	Yes
int32	32-bit signed	No
uint32	32-bit unsigned	No

However, it is possible to use standard c-types. Bear in mind that the sizes might not be the same as the ones you are used to.

Error handling

When the kernel detects a situation where a resource is used by another task than the one that has permission to use the resource, a routine associated to the task that caused the error will start. Similarly, when the kernel detects that a task has missed it's deadline, a routine associated to the task that caused the error will start. The default handler will start if no routine is present in the task specification.

The default handler will write the error code (see the file system\kernel\include\os_errno.h), i.e., the id of the task that caused the error and at which time point. After that, the processor will be reseted. The error code can be analyzed by pressing the *VIEW*-button on the RCX-unit (to 'buy' some time before the processor is reseted). The VIEW button must be pressed to obtain each new error code on the display.

Clock resolution

The clock resolution is specified in the RESOLUTION value before specifying the tasks. The value is given in microseconds (min=2 µs, max=130ms). However, for Hitachi H8/300, a suitable minimum value is 5ms. A clock resolution of 10 milliseconds should be specified as below:

```
RESOLUTION = 10000;
```

By specifying a resolution of 10 ms, the system will be 'driven forward' by a tick each 10 ms.

The hardware

Following files must be included to be able to have access to the different functions specified in the RCX Library Manual:

rcx_display.h	-	Functionality to write on the LCD-display
rcx_button.h	-	Access the buttons on the RCX-unit (view, prgm, run)
rcx_motor.h	-	Functionality to actuate motors
rcx_sensor.h	-	Functionality to handle (reading/activating) sensors

Basic schedule (config.obx)

The basic schedule is to be used in the laboration and project. The file is included in the developing environment in: ***asterix/system/user/config.obx***

```
SYSTEMMODE = NORMAL;
RAM = 1000;

MODE mode_1{

RESOLUTION = 10000;

/* System tasks, do not edit or remove */
HARD_TASK APERIODIC IRQ0{
        ACTIVATOR = IRQSIG0;
        OFFSET = 0;
        DEADLINE = 50;
        PRIORITY = 100;
        STACK = 50;
        ROUTINE = irq0_isr;
        ERR_ROUTINE = irqerror;
};
HARD_TASK APERIODIC IRQ1{
        ACTIVATOR = IRQSIG1;
        OFFSET = 0;
        DEADLINE = 50;
        PRIORITY = 99;
        STACK = 50;
        ROUTINE = irq1_isr;
        ERR_ROUTINE = irqerror;
};

/* Put user-tasks here */
HARD_TASK APERIODIC idle{
        ACTIVATOR = 0;
        OFFSET = 0;
        DEADLINE = 0;
        PRIORITY = 0;
        STACK = 50;
        ROUTINE = idletask;
};
/* Put waitfree-communication here */
/* default signals for irq, do not edit */
        SIGNAL IRQSIG0{
                USER = IRQ0;
        };

        SIGNAL IRQSIG1{
                USER = IRQ1;
        };
/* Put signals here */
/* Put semaphores here */
};
```

Setting up the environment

You have the freedom of choice of which editor or IDE to use in the lab/project. However, a short description on how to perform the assignments with a simple text editor and a command prompt in windows is presented below. The idea is to use a simple text editor for editing files, and a command window to compile and link the system. **Do not forget** to read *Troubleshooting* and *FAQ* on the course homepage.

Download files

1. Download Asterix (available at *http://www.idt.mdh.se/kurser/ct3200*).
2. Unzip the file (asterix.zip) to ***d:***
3. Check if there is a directory ***d:\asterix***

Set up the environment

1. Open the DOS-prompt
2. Go to ***d:\asterix***
3. Execute **b*in\init***
4. Execute ***compile***
5. If compile succeeded, a file, kernel.srec, should have been created in asterix\system. This is the file to be downloaded to the RCX-unit.
6. OBS! If you use 'make', the make-program do not always terminate! Make sure to check the 'task manager' and to end the old make process.

All files in all subdirectories to asterix\system have to be compiled and linked (see ***makefile*** and ***makefile.mk*** in asterix\system for detailes).

Downloading

Follow 1-3 in **Set up the environment**

4. Connect the IR-tower to COM1. If COM1 is not available, dl.bat has to be modified.
5. Check if there is a file kernel.srec in d:\asterix\system. (if the file is missing, execute ***compile***)
6. Start LEGO Mindstorm RCX.
7. Execute ***dl***

Troubleshooting

or : 'what to do when nothing works?'

1. The compilation fails!
   - Are all files located in d:\asterix ? If not, the makefiles have to be modified.
   - Are all environment variables (asterix and path) correct? (See asterix\bin\init.bat)
   - Is the compiler set up properly? (see ”Set up the environment”)
   - Is config.obx placed in the correct directory? (asterix\system\user\ )
   - Windows creates, sometimes, files capitals! Make sure that the files in asterix\system\kernel\c\tasks.c and asterix\system\kernel\include\tasks.h are created with small letters!

2. Downloading to RCX fails!
   - Is the cable connected? (the IR-tower has a led indicating when dl.bat is executed)
   - Old batteries?
   - Interference from the neighbor?
     ”Hide” tower and RCX in the Lego-box when downloading.

3. The behavior of the RCX is not the expected one!
   - Write down the expected behavior and compare to the config.obx-file.
   - Have you used the basic schedule as the start point?
     If not, the interrupts can act unexpected.
     Signals must be specified after the interrupts (irqsig)!!!

4. The RCX locks!
   - Remove batteries! This is the only way to restart.
   - Check eventual memory overwrites (stack sizes, pointer errors etc.).

5. Our code worked during previous lab. Why is it not working today?
   - Save always files you have changed. Usually only *user.c* and *config.obx*.
   - Download *asterix.zip* from the homepage and unzip it in the specified directory. Move your saved files to the asterix directory.

To think about:

- The make-program is a little bit unstable. Sometimes, no downloadable file is created. Run ***compile*** again. Sometimes, Make even creates processes that will not terminate. Make sure to check for those in the process list and to terminate old make-processes.
- It can be difficult to locate programming errors. If no downloadable file is created (kernel.srec), check for compiling errors. Eventual errors can be seen in *stderr* where your filename is the cause.
- Sometimes, the system has to be totally re-compiled. Run ***make clean*** or ***make realclean*** and the system will be entirely recompiled next time.

OBS:

- **Save your files to your home directory after each lab**
- **At the beginning of every new scheduled lab, you should download *asterix.zip* from the homepage and unzip it to the recommended directory. Move, then, you saved files from your home directory to the asterix directory.**

Ask the lab assistant only if you have not been able to solve the problem by using the suggestions above!!!

Laboration

Introduction

The purpose of this assignment is to get an insight of how the different parts of Asterix can be configured and used.

Presenting the solutions

The solutions to the assignment are presented to the lab-assistant by all group members. The presentation consists of a discussion about the implementation and a demonstration of the implementation functionality. The implementation is approved only if the group members can prove to the lab-assistant that the implementation has been performed according to the basic c- and real-time-programming rules, and if all group members can present the proposed solutions.

The lab will **not** be approved if:

- The code is hard to read and badly structured (”spaghetti-code”).
- The code is unnecessary complicated. If-statements, for example, are useful tools in c-programming, but can make the code hard to follow and unstructured. Ask yourselves in stead: can we expand our system with one or several tasks with simpler code that will solve the issue?
- Sensors are read and motors are actuated by more than one task.
- Task attributes are different from the attributes specified in the lab specification.
- Solutions can not be motivated by all group members.

Preparations

Read through the documentation about Asterix and Obelix, especially the parts describing the syntax used in the configuration files and services offered by the operating system. Read the section ”Basics in Asterix”. Set up the environment, compile and download the test program.

Always start from the basic schedule!

The file can be found in : ***asterix/system/user/config.obx***

A) The schedule

Create a schedule consisting of four tasks.

Task A shall write 111 on the display.
Task B shall write 222 on the display.
Task C shall write 333 on the display.
The idle task shall not perform any activity.

Task	Period	Deadline	Priority
A	10	10	30
B	20	20	20
C	30	30	10
Idle	0	0	0

The hardware clock resolution shall be 10 milliseconds.

1. Create the task routines with appropriate code.
2. Configure Asterix by adding what is needed to the basic schedule. Suitable stack size is 30-50. Think about what names you want to assign to the task routines when you configure the schedule.
3. Do not forget to include the required headers.
4. Compile the system.
5. Download the system to the RCX unit.

B) Deadline miss

Add an endless loop in Task C.

- What is happening?

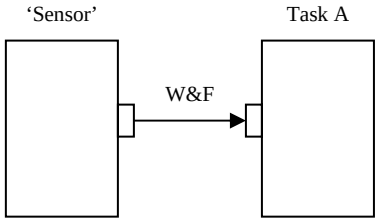
- What digit combination is displayed on the LCD? (Do not forget to use the *VIEW*-button)

- What means 1001? (Tips, in which file are the error codes defined?)

C) Wait&lockfree channels

Tasks can communicate with each other in different ways. There could be shared variables or advanced algorithms that guarantee tasks to read always the last written value. Asterix supports wait&lockfree channels (a buffer-based solution). Every time a task period starts, the task is assigned a buffer containing the last written value (if the task wants to read the data) or an empty buffer if the task wants to write data.

Create two tasks (plus the idle task) of which the writing task shall simulate a sensor. It is sufficient fif the actual sensor value is a static local variable which is increased each period. The sensor value shall be sent to the reading task through a wait&lockfree channel. Then, the reading task shall write the value on the LCD.



Task	Period	Deadline	Offset	Priority
Sensor	20	20	0	30
Task A	20	20	0	20
Idle	0	0	0	0

- Is there any problem by allowing the tasks to have the same period and offset?

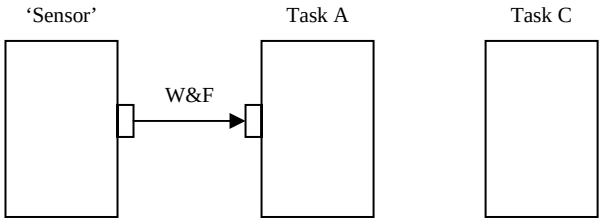
- How can it be solved?

(Tips: how are the buffers assigned in a channel?)

D) Synchronization

When tasks attempt to solve a common job, they have to be synchronized.. There are different techniques that can be used to achieve this.

Assume the same scenario as in the wait&lockfree-assignment above, i.e., one task reading one ‘sensor'. The sensor value will furthermore be used by a third task, C. All communication will take place through task A, since 'sensor' is a simulated sensor and do not allows more than one reader.
To make the assignment more realistic, we assume that only sensor values that are multiple of 10 are valid values, i.e., the only values accessible to task C. In that way, task A will become a device driver for the sensor.



Task	Period	Deadline	Offset	Priority
Sensor	20	20	0	30
Task A	20	20	0	20
Task C	20	20	0	10
Idle	0	0	0	0

Synchronize task C with respect to when a ‘valid sensor value’ has been read by A, by using different techniques.

1. Shared variable protected by a semaphore.

2. Wait&lockfree-channel

3. Aperiodic signals (Task C is a aperiodic task).

E) Design and implementation of tasks

The purpose of this assignment is to create different tools that can be used together with an arbitrary application in Asterix. The application will be the project you will complete but the tools should be able to use in an arbitrary application.

E.1) Clock

There is no clock handler in Asterix. Therefore, your assignment is to implement a clock. The updating procedure for the clock has to be such that ”right second” must be returned at any arbitrary reading. However, no software emulated clock can yield the exact time. Therefore, when presenting your solution, you have to motivate why your clock deviation is reasonable. At the same time, you have to motivate the choice you made when you assigned attributes to the clock, i.e., period and priority.

Answer the following question: A clock is implemented as a periodic task, with a period equal to 1000ms. Every time the task executes, a variable representing the seconds is increased by one unit. The clock will show the right time, but what is the worst case deviation of the clock compared to an exact one?

The minimum requirement is that the clock should be able to handle hours, minutes and seconds, i.e., you will need three integer variables to implement a 24 hours clock. However, only 4 digits can be printed on the LCD (ex: minutes and seconds). At the same time, in your clock implementation, you shall use the system call getTicks() that returns the number of ticks that “have passed” since the system startup (e.g., not only a periodic counter).

OBS! The procedure of reading the clock has to be **atomic**.

To think about:

- You have to protect shared resources.
- You will never get an exact clock. The real-time clock in a PC is updated aprox. 18 time per second.
- The largest data type is 16-bits integer (uint16, int16) even if the compiler supports larger data types, e.g., 32-bits integers.
- The system call getTicks() returns the number of ticks that “have passed” since the system startup. It is, however, only a 16-bits counter which starts all over again when it reaches 65535.
- Reading a 16-bits variable is not atomic. A single c-instruction generates several assembler instructions when the system is compiled.
- You may need a more accurate clock in your project, e.g., you may need an additional 16-bits integer to store milliseconds.

E.2) Device drivers for the display and the clock

An issue about the display is that only one task can write to it at a time. That means that the display has to be protected somehow. In a similar way, the clock updating must be protected from the tasks that want to read it. Create a device driver to handle writing tasks and to update the LCD in a correct and safe way, and a device driver for the clock.
It is sufficient if you implement the device driver as a routine that serves as an interface between the device (clock or display) and the user.

E.3) Sensors and motors

Even if the light sensor is designed to be used in a number of purposes, your assignment is to use it to primarily detect and identify different colors. Think about you have to turn the sensor passive before reading it. After reading the sensor you have to turn it active again. (setSensorPassive(), readSensor(), setSensorActive()).

Pressure sensors can be used to detect collisions. The main difference between light and pressure sensors is that the pressure sensor do not have to be turn passive before reading it; you can read it directly by using readSensor().

The sensors are connected to ports 1, 2 or 3.

The two engines are connected to ports A, B or C, and controlled with the command setMotor().

Try using the sensors to detect an activated pressure sensor or to distinguish between light and dark colors. Use the track included in the Lego Mindstorm © box to calibrate the light sensor. Start with some default value which can be adjusted later on, as the light sensor, in some cases, might difficult to read.

Activate the motors forward and backward depending on the different sensor values. Try to make you implementation modular (not everything in the same task). Sometimes it may be more important to avoid a collision than to follow the track.

The requirements for your tasks are:

- You have to activate the motors continuously with max 5 times/second
- You may not read sensors in the same task that actuates motors.
- One task may only read one sensor.
- No logics or calculations may be implemented in the motor- and sensor-tasks. You may, however, convert your in-data to a suitable format.
- The solutions must be reached by using scheduling techniques and/or the clock you implemented in E.1., i.e., solutions such as “turn during 3 periods” will not be accepted.

Besides the tasks you have created, you should add several temporary tasks to your system, in order to test the functionality and the timing constraints of the motor- and sensor-tasks.

Project assignment

In this project you will construct a car according to the instructions available in the Lego Mindstorms © manual or your own design ideas. There are a number of basic requirements your construction will have to fulfill in order to be approved.

- **The project report has to be written according to Thomas Larssons instructions in order to be approved.**
- **You have to be able to motivate why your system works (both functional and temporal behavior).**
- **You have to motivate (when presenting and in the report) what make your system a real-time system.**

Car project:

In this project you shall build a car which must be able to find a track marked by a black line and, then, follow it. Building instructions for the car are available in the Lego Mindstorms © manual.

Basic requirements for the car project:

- The tools you implemented in assignment E (in the lab) have to be used if possible, e.g., .if you want to turn your car for 50ms, you have to use the clock you implemented in the lab. No counters or “turn for n periods” strategies are allowed.
- Motor actuating: **max 5 times** per second.
- The car has to be able to find the black line from **up to one meter** distance.
- After finding the track, the car has to be able to drive a full loop on the track in **max 30 seconds** (of course, if no obstacles are in the way).
- The car must be able to **drive around an obstacle** placed on the track. It is not allowed to turn the car 180 degrees and to drive in the opposite direction.

In some cases, depending on the choice of construction design, you may find out that the requirements are not reasonable (due to, e.g., hardware limitations). In these cases, you must have a good motivation why the specified requirements can not be fulfilled, and, at the same time, you will have to define (and fulfill) your own requirements.

Tips: Think about building an appropriate transmission and to choose suitable tires since there is a time-requirement for the round- up of the track.

When demonstrating your project, do not assume that the lab assistant will place the car on the black track. After finding the track, the car should be able to fulfill the above mentioned requirements.

OBS! If you have other suggestions for the project, e.g., the robot arm or the lift, you will have to come up with your own requirements that your project must fulfill. However, think about you will have to match the complexity/difficulty specified in the car requirements. Please, contact the lab assistants for further information.

Demonstrating and presenting the project assignment.

The following requirements (**A**, **B**, **C** and **D**) must be fulfilled in order to pass the project assignment. (Think about the document has to be written as a report already from requirement A)

A.
Specification:

- A high level description.
- Functionality (description of what the construction can do).
- Description of the hardware used.
- Test case for functional and temporal behavior.

The lab assistant will not approve part B if part A has not been approved. You may, however, work further with part B, but be prepared to revise your part A.

B.

- Analysis of the requirements (software) and timing constraints in the system.
- Design, according to the RTT-model.
- Sections I-VII, page 166 in course literature. **OBS** you may only use Fixed Priority Scheduling. That will have impact on the choice of schedulability test.
- Temporal analysis based on estimated execution times.
- More test cases for functional and temporal behavior.

The lab assistant will not approve part C if part B has not been approved already. You may, however, work further with part C, but be prepared to revise your part B.

C

- Sections VII-IX, page 167 in the course literature.
- Temporal analysis based on measured execution times.
- Functional and temporal testing. Motivate why the system works (and not crashes)
- Conclusions containing (among other things):
    - Problems
    - What you have learned from the project

D

- Demonstration of the project.

The lab assistant will not approve part D if part C has not been approved already.

OBS! If you have to revise any part of the project, make sure you attach the old version as well. The old version is the one with comments from the lab assistant.

Floating point calculations:
Floats are not supported by Hitachi H8/300. If you want to use floats, you have to figure out “how to” by yourselves. (Tips! Fix Point Arithmetic).

Analysis execution times:
You analyze the temporal behavior of the real-time system with an analysis tool for WCET-analysis of real-time systems. More information about the tool is presented in a separate document describing temporal analysis of real-time systems.