Introduction to Lab 2
Programming in RTOS on LEGO Mindstorms
Syed Md Jakaria Abdullah

<jakaria.abdullah@it.uu.se>

10 September 2014
Lab 2: Programming in RTOS using LEGO Mindstorms

- **Lab goals:**
  - Basic programming on an embedded device
  - Using the API of an RTOS for concurrent tasks

- **Lab preparation:**
  - Work in your groups
  - *Get LEGO box* (next slide), charge battery
  - Possibly refresh your C knowledge
  - Lab will be done on Wed, 17.9. and Fri, 19.9. (both in 1515)
  - Have a look at the lab homepage
    [http://www.it.uu.se/edu/course/homepage/realtid/ht14/lab2](http://www.it.uu.se/edu/course/homepage/realtid/ht14/lab2)

- **Lab report:**
  - OIL file and C code to all 3 parts, well commented
  - Descriptions of what you did and why
  - To my mailbox, building 1, floor 2; *Deadline: Wed, 24.9. at 10:00*

- **Further:**
  - Demonstrate a working vehicle, participate in *car race on 24.9.*
  - Return all hardware you get to Karl (see next slide)
Lab 2: LEGO Mindstorms Boxes

- Each group gets one box

- All hardware issues are handled by Karl Marklund

- Office: 1440, mail: karl.marklund@it.uu.se

- Time schedule:
  - Today at 12:00: Boxes handed out (after lecture)
  - 24.9. at 10:00: Report deadline
  - 24.9. at 15:15: Car presentation, boxes handed back afterwords
Lab 2: Working At Home

- You may work at home (using Windows/Linux/Mac?)
- Toolchain installation is non-trivial
  - I can’t give support for that
  - Firmware upload, program compile, program upload
    - Windows: May need Cygwin
- Some hints at lab homepage
- Default: Work in the Solaris lab
LEGO Mindstorms

- Programmable LEGO brick with sensors and motors
- Comes in two generations:
  - RCX generation (1998)
  - NXT generation (2006)

- We will use the **NXT platform**
LEGO Mindstorms: Components

Package contents:

- **NXT unit:**
  - LCD matrix display
  - Sensor inputs 1 to 4
  - Motor outputs A, B, C
  - Speaker
  - USB, Bluetooth

- Three motors

- **Sensors:**
  - Light
  - Distance (Ultrasound)
  - Touch (2x)
  - Sound
  - (More from 3rd party vendors)

NXT Brick Internals:

- Atmel 32-bit ARM7 processor, 64k RAM, 256k Flash, 48MHz clock
RTOS: nxtOSEK

- We don’t use the standard firmware
- Instead: *nxtOSEK*
  - Real-time operating system
  - Based on OSEK (industry standard for automotive embedded systems)
  - Implements highest OSEK conformance class ECC2
  - Provides C/C++ development environment
  - Support for (concurrent) tasks, priorities, semaphores, events
  - Comprehensive API for low-level I/O accesses

- Rest of this introduction: How to
  - Flash the custom firmware
  - Compile/upload programs
  - Write programs/use nxtOSEK API
RTOS: nxtOSEK

- We don’t use the standard firmware
- Instead:  *nxtOSEK*
  - Real-time operating system
  - Based on OSEK (industry standard for automotive embedded systems)
  - Implements highest OSEK conformance class ECC2
  - Provides C/C++ development environment
  - Support for (concurrent) tasks, priorities, semaphores, events
  - Comprehensive API for low-level I/O accesses
- Rest of this introduction: How to
  - Flash the custom firmware
  - Compile/upload programs
  - Write programs/use nxtOSEK API
NXT Firmware Upload

1. Connect NXT unit to USB port (of SunRay)
2. Power up NXT unit
3. Put NXT into *reset mode*
4. Upload firmware:
   - Custom FW using `fwflash-jh`
   - Original FW using `fwflash-original`

Example Run: Firmware upload

```
$ /it/kurs/realtid/bin/fwflash-jh
...
Checking firmware... OK.
NXT device in reset mode located and opened.
Starting firmware flash procedure now...
Firmware flash complete.
New firmware started!
$
```
NXT Firmware Upload

1. Connect NXT unit to USB port (of SunRay)
2. Power up NXT unit
3. Put NXT into *reset mode*
4. Upload firmware:
   - Custom FW using `fwflash-jh`
   - Original FW using `fwflash-original`

Example Run: Firmware upload

```bash
$ /it/kurs/realtid/bin/fwflash-jh
... 
Checking firmware... OK.
NXT device in reset mode located and opened.
Starting firmware flash procedure now...
Firmware flash complete.
New firmware started!
$
```

<jakaria.abdullah@it.uu.se>
nxtOSEK: Program Compile/Upload

1. Use and *adjust* provided Makefile
2. Compile program (OIL+C) using `make all`
3. Upload program using `nxjupload`
   - NXT needs to be running and idle
   - .. and connected via USB

**Example Run: Program compile/upload**

```
$ make all
Compiling /it/kurs/realtid/nxt/nxtosek/ecrobot/.../
...
Generating binary image file: helloworld.rxe
$ /it/kurs/realtid/bin/nxjupload helloworld.rxe
Found NXT: NXT 0016530915 A7
leJOS NXJ> Connected to NXT
leJOS NXJ> Upload successful in 1750 milliseconds
$ make clean  # optional, but useful
...
$
```
nxtOSEK: Source Files

OIL Source File

```c
CPU ATMEL...
{
    ...
    TASK HelloWorld
    {
        ...
    }
};
```

C Source File

```c
#include <stdlib.h>
#include "kernel.h"
...

TASK(HelloWorld)
{
    display_string("Hello World!");
    ...
    TerminateTask();
}
```

Compilation, Linking, ...

NXTBINARY...

RXE Binary File
nxtOSEK: Source Files

OIL Source File

```c
CPU ATMEL...
{
  ...
  TASK HelloWorld
  {
    ...
  }
};
```

C Source File

```c
#include <stdlib.h>
#include "kernel.h"
...

TASK(HelloWorld)
{
  display_string("Hello World!");
  ...
  TerminateTask();
}
```

Compilation, Linking, ...

NXTBINARY...

RXE Binary File
nxtOSEK API

- You “program” two files:
  1. Systems description: *OIL Source File*
  2. Task implementations: *C Source File*

- **OIL File:**
  - Describe System: Scheduling and Task details
  - Counters, Alarms, Events, Resources, Task releases

- **C File: Task implementations**
  - Input/Output (orange Button/LCD)
  - Reading sensors (light/touch/distance/sound)
  - Controlling motors
  - Time functions (delay)
  - Generate/wait for events
  - Newlib (like libc, e.g., random numbers)

- Will do a short walk-through now

- See “nxtOSEK C API Reference” and “Newlib Reference” manuals!
You “program” two files:

1. Systems description: *OIL Source File*
2. Task implementations: *C Source File*

**OIL File:**
- Describe System: Scheduling and Task details
- Counters, Alarms, Events, Resources, Task releases

**C File: Task implementations**
- Input/Output (orange Button/LCD)
- Reading sensors (light/touch/distance/sound)
- Controlling motors
- Time functions (delay)
- Generate/wait for events
- Newlib (like libc, e.g., random numbers)

Will do a short walk-through now

See “nxtOSEK C API Reference” and “Newlib Reference” manuals!
nxtOSEK API

- You “program” two files:
  1. Systems description: *OIL Source File*
  2. Task implementations: *C Source File*

- OIL File:
  - Describe System: Scheduling and Task details
  - Counters, Alarms, Events, Resources, Task releases

- C File: Task implementations
  - Input/Output (orange Button/LCD)
  - Reading sensors (light/touch/distance/sound)
  - Controlling motors
  - Time functions (delay)
  - Generate/wait for events
  - Newlib (like libc, e.g., random numbers)

- Will do a short walk-through now

- See “nxtOSEK C API Reference” and “Newlib Reference” manuals!
nxtOSEK API: I/O

- Input via orange button and sensors
  - Initialize sensors before use
- Output via LCD (strings, integers), sound and motors
- Sensor and motor access via ports: NXT_PORT_S1, ..., NXT_PORT_A, ...
- See API reference!

Example: I/O via button, LCD and motors

```
#define LIGHTSENSOR NXT_PORT_S3
#define MOTOR NXT_PORT_B

if (ecrobot_is_ENTER_button_pressed()) { // Non-blocking
    display_clear(0);
    display_int(ecrobot_get_light_sensor(LIGHTSENSOR), 4);
    display_update();
    nxt_motor_set_speed(MOTOR, 100, 0); // full speed
}
```
nxtOSEK Tasks: Single Instance

OIL file

1 TASK RunOnce
2 {
3    AUTOSTART = TRUE
4    {
5        APPMODE = appmode1;
6    }
7    PRIORITY = 1; /* Low */
8    ACTIVATION = 1;
9    SCHEDULE = FULL;
10   STACKSIZE = 512;
11 }

C file

1 DeclareTask(RunOnce);
2 ...
3 TASK(RunOnce)
4 {
5    // This is executed
6    // just *once*
7    //
8    // (Use a loop?)
9    TerminateTask();
10 }

Note the declare statement in the C source
nxtOSEK Tasks: Single Instance

**OIL file**

```c
TASK RunOnce
{
    AUTOSTART = TRUE
    {
        APPMODE = appmode1;
    }
    PRIORITY = 1; /* Low */
    ACTIVATION = 1;
    SCHEDULE = FULL;
    STACKSIZE = 512;
}
```

**C file**

```c
DeclareTask(RunOnce);
...
TASK(RunOnce)
{
    // This is executed
    // just * once*
    //
    // (Use a loop?)
    TerminateTask();
}
```

- Note the *declare* statement in the C source.
nxtOSEK Tasks: Periodic

For periodic task releases every 100ms:

1. Declare a *counter*
   - Increased every ms

2. Declare an *alarm*
   - Activated when counter reaches specified value (100)
   - Can release a task

3. Declare and implement the *task*
   - Execute some code
   - Terminate cleanly with TerminateTask()

Counter and Task declarations also in C file
nxtOSEK Tasks: Periodic (cont.)

**OIL file: Counter declaration**

```plaintext
COUNTER SysTimerCnt {
  MINCYCLE = 1;
  MAXALLOWEDVALUE = 10000;
  TICKSPERBASE = 1;
};
```

**OIL file: Task declaration**

```plaintext
TASK PeriodicTask {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  STACKSIZE = 512;
};
```

**OIL file: Alarm declaration**

```plaintext
ALARM cyclic_alarm {
  COUNTER = SysTimerCnt;
  ACTION = ACTIVATETASK {
    TASK = PeriodicTask;
  };
  AUTOSTART = TRUE {
   ALARMTIME = 1;
    CYCLETIME = 100;
    APPMODE = appmode1;
  };
};
```
nxtOSEK Tasks: Periodic (cont.)

OIL file: Counter declaration

```c
COUNTER SysTimerCnt {
  MINCYCLE = 1;
  MAXALLOWEDVALUE = 10000;
  TICKSPERBASE = 1;
};
```

OIL file: Alarm declaration

```c
ALARM cyclic_alarm {
  COUNTER = SysTimerCnt;
  ACTION = ACTIVATETASK
  {
    TASK = PeriodicTask;
  }
  AUTOSTART = TRUE
  {
    ALARMTIME = 1;
    CYCLETIME = 100;
    APPMODE = appmode1;
  }
};
```

OIL file: Task declaration

```c
TASK PeriodicTask {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  STACKSIZE = 512;
};
```
**nxtOSEK Tasks: Periodic (cont.)**

### OIL file: Counter declaration

```c
COUNTER SysTimerCnt {
   MINCYCLE = 1;
   MAXALLOWEDVALUE = 10000;
   TICKSPERBASE = 1;
};
```

### OIL file: Task declaration

```c
TASK PeriodicTask {
   AUTOSTART = FALSE;
   PRIORITY = 1;
   ACTIVATION = 1;
   SCHEDULE = FULL;
   STACKSIZE = 512;
};
```

### OIL file: Alarm declaration

```c
ALARM cyclic_alarm {
   COUNTER = SysTimerCnt;
   ACTION = ACTIVATE_TASK
   {
      TASK = PeriodicTask;
   }
   AUTOSTART = TRUE
   {
      ALARMTIME = 1;
      CYCLETIME = 100;
      APPMODE = appmode1;
   }
};
```
C file: Periodic task

```
... 
1 DeclareCounter(SysTimerCnt);
2 DeclareTask(PeriodicTask);
... 
5 void user_1ms_isr_type2(){ SignalCounter(SysTimerCnt); }
... 
7 TASK(PeriodicTask) {

    // Executed just once
    //
    // DO NOT use an infinite loop!

    TerminateTask();
}
```
nxtOSEK: Synchronization Features

- Tasks can signal and wait for *events*
  - Declare in OIL file
  - ... and inside the Task in OIL file
  - ... and in the C file (using DeclareEvent())
  - Implemented as a bitmask
  - More details in lab description

- Tasks can use semaphores, called *resources*
  - Declare in OIL file
  - ... and inside the Task in OIL file
  - ... and in the C file (using DeclareResource())
  - More details in OSEK specification
nxtOSEK: Synchronization Features

- Tasks can signal and wait for *events*
  - Declare in OIL file
  - .. and inside the Task in OIL file
  - .. and in the C file (using DeclareEvent())
  - Implemented as a bitmask
  - More details in lab description

- Tasks can use semaphores, called *resources*
  - Declare in OIL file
  - .. and inside the Task in OIL file
  - .. and in the C file (using DeclareResource())
  - More details in OSEK specification
Lab Assignment

- **Part 1: ** *Warm-Up*
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- **Part 2: ** *Event-driven Scheduling*
  - Use OSEK’s event mechanism
  - Application: Four events with car on table
    1. Touch sensor is pressed/released
    2. Table edge is sensed (light sensor)

- **Part 3: ** *Periodic Scheduling*
  - Define different periodic tasks
  - Application: Distance and touch sensor sensing
    1. Drive (back off) while sensor pressed
    2. Otherwise, keep distance constant

- **Extra part: ** *LEGO Car Race*
  - Apply all you have learned
  - (See next slide)
Lab Assignment

- **Part 1:** *Warm-Up*
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- **Part 2:** *Event-driven Scheduling*
  - Use OSEK’s event mechanism
  - Application: Four events with car on table
    1. Touch sensor is pressed/released
    2. Table edge is sensed (light sensor)

- **Part 3:** *Periodic Scheduling*
  - Define different periodic tasks
  - Application: Distance and touch sensor sensing
    1. Drive (back off) while sensor pressed
    2. Otherwise, keep distance constant

- **Extra part:** *LEGO Car Race*
  - Apply all you have learned
  - (See next slide)
Lab Assignment

- **Part 1: Warm-Up**
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- **Part 2: Event-driven Scheduling**
  - Use OSEK’s event mechanism
  - Application: Four events with car on table
    1. Touch sensor is pressed/released
    2. Table edge is sensed (light sensor)

- **Part 3: Periodic Scheduling**
  - Define different periodic tasks
  - Application: Distance and touch sensor sensing
    1. Drive (back off) while sensor pressed
    2. Otherwise, keep distance constant

- **Extra part: LEGO Car Race**
  - Apply all you have learned
  - (See next slide)
Lab Assignment

- Part 1: *Warm-Up*
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- Part 2: *Event-driven Scheduling*
  - Use OSEK’s event mechanism
  - Application: Four events with car on table
    1. Touch sensor is pressed/released
    2. Table edge is sensed (light sensor)

- Part 3: *Periodic Scheduling*
  - Define different periodic tasks
  - Application: Distance and touch sensor sensing
    1. Drive (back off) while sensor pressed
    2. Otherwise, keep distance constant

- Extra part: *LEGO Car Race*
  - Apply all you have learned
  - (See next slide)
LEGO Car Race

- *Car demonstration* takes place on Thu, 24.9.
- Track looks roughly like this:

![](image.png)

- Procedure for each team:
  1. **1st phase:** Follow another car in constant distance (20cm) for 1 lap
  2. **2nd phase:** Be fastest on the next lap
- Fastest team wins! (*Prize award included*)
- 3 tries per team (otherwise: assignment failed, fix car)
- Keep in mind: Demo conditions might differ (different light etc.)
Questions?